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 [39]:

```
# 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
```

```
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 [208…
```
# To read the data and parse_dates to automatically infer datetime format for a dat

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

In [209…
```
df.index.name = 'Time_Stamp' # Renaming index name
df = df.rename(columns={'Rose': 'Rose_Wine_Sales'}) # Renaming column name
```

Data Overview

In [210…
```
df.head() # To view first 5 rows of the data
```

Out[210…

	Rose_Wine_Sales
Time_Stamp	
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 [211…
```
df.tail() # To view last 5 rows of the data
```

Out[211...

	Rose_Wine_Sales
Time_Stamp	
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 [212...

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

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

In [213...

```
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_Wine_Sales  185 non-null    float64
dtypes: float64(1)
memory usage: 2.9 KB
```

In [214...

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

Out[214...

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

Missing value treatment

In [217...

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

Out[217...

```
Rose_Wine_Sales    2
dtype: int64
```

Impute missing values using LOCF (Last Observation Carried Forward) method

In [218…
```
# Impute missing values using LOCF (Last Observation Carried Forward) method

df['Rose_Wine_Sales']= df['Rose_Wine_Sales'].interpolate(method='ffill')
```

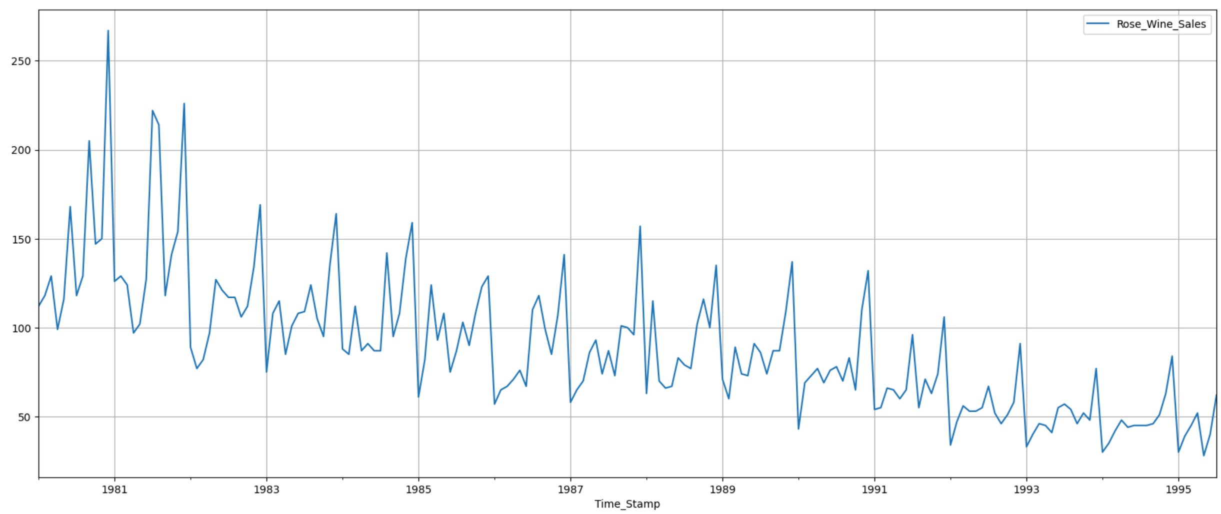
In [220…
```
df.isnull().sum() # Check for null values
```

Out[220…
```
Rose_Wine_Sales    0
dtype: int64
```

Exploratory Data Analysis

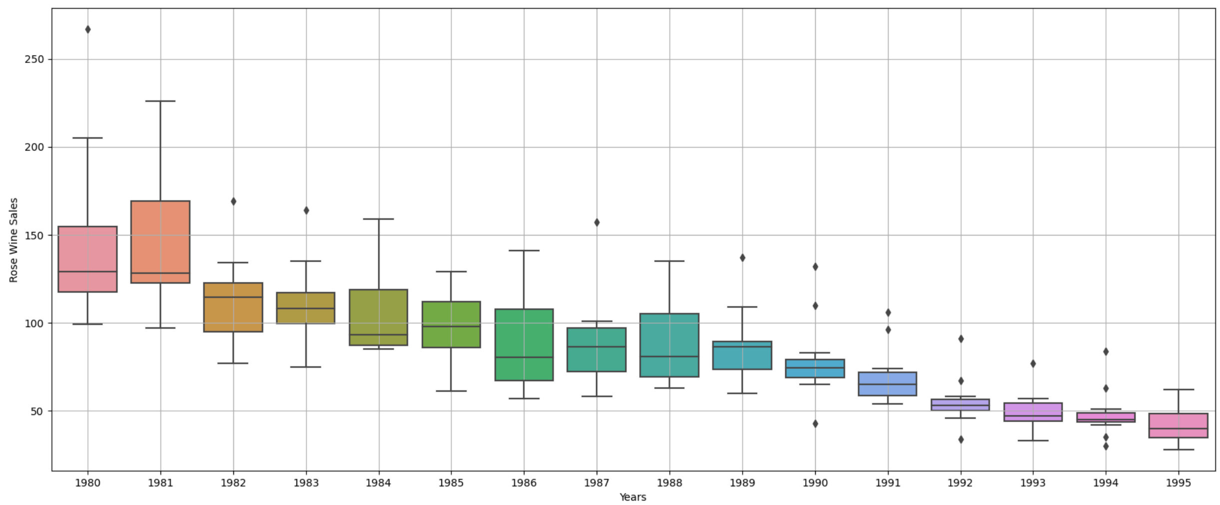
Plot to find trend of data

In [221…
```
# To find trend of data

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



Yearly Boxplot

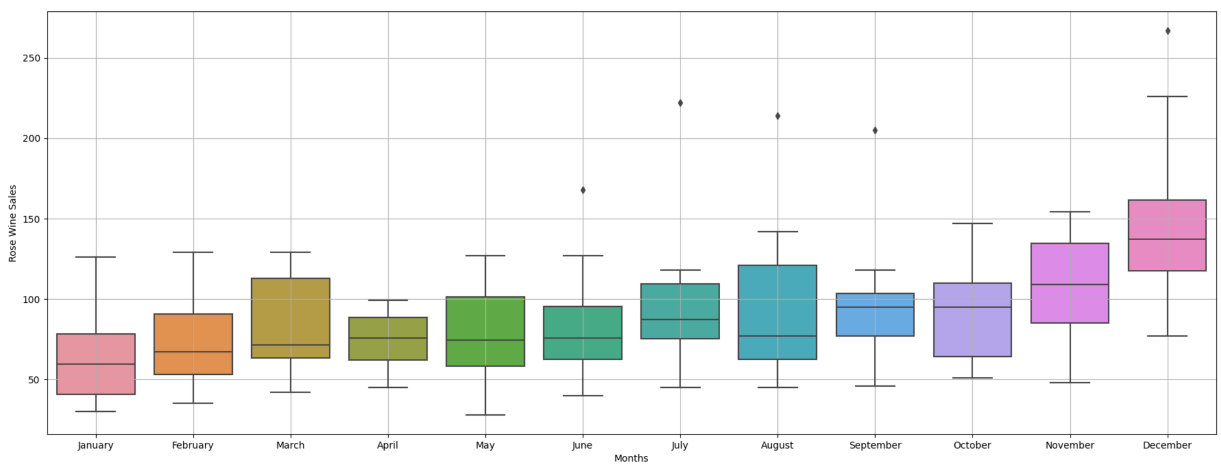
In [222…
```
_, ax = plt.subplots(figsize=(20,8))
sns.boxplot(x = df.index.year,y = df.values[:,0],ax=ax)
plt.grid()
plt.xlabel('Years')
plt.ylabel('Rose Wine Sales')
plt.show()
```



Monthly Boxplot

In [223…

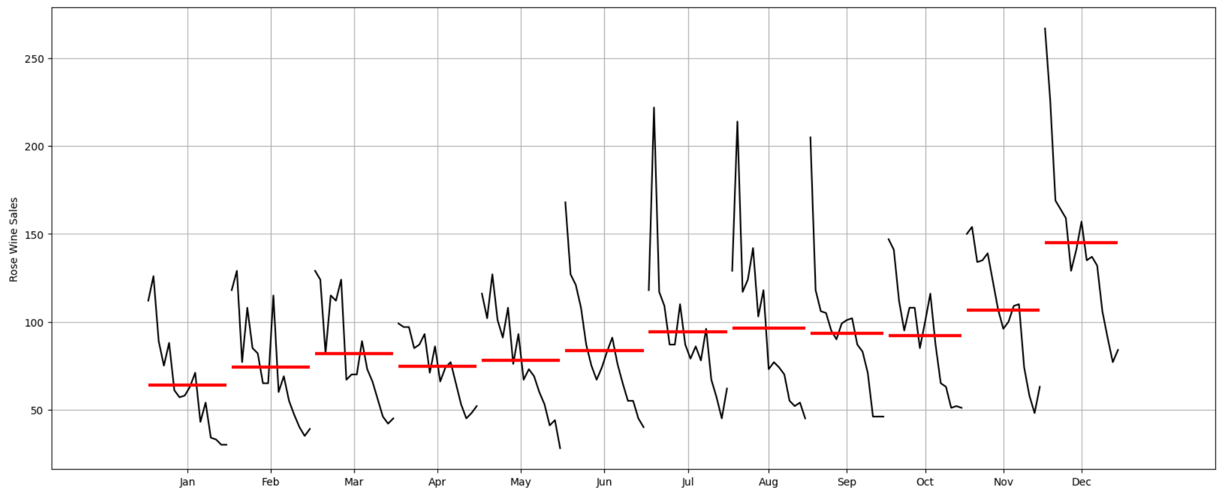
```
_, ax = plt.subplots(figsize=(22,8))
sns.boxplot(x = df.index.month_name(),y = df.values[:,0],ax=ax)
plt.grid()
plt.xlabel('Months')
plt.ylabel('Rose Wine Sales')
plt.show()
```



Time series monthplot

In [224…

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

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



Plot of monthly Rose sales across years

In [225…

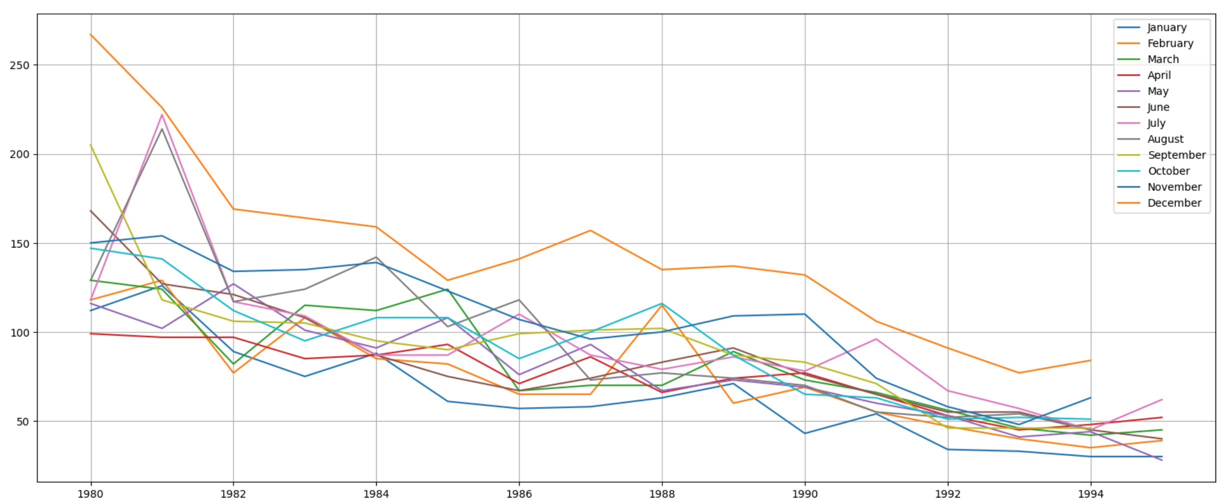
```
monthly_sales_across_years = pd.pivot_table(df, values = 'Rose_Wine_Sales', columns

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

monthly_sales_across_years
```

Out[225...

Time_Stamp	January	February	March	April	May	June	July	August	September	Oc
Time_Stamp										
1980	112.0	118.0	129.0	99.0	116.0	168.0	118.0	129.0	205.0	
1981	126.0	129.0	124.0	97.0	102.0	127.0	222.0	214.0	118.0	
1982	89.0	77.0	82.0	97.0	127.0	121.0	117.0	117.0	106.0	
1983	75.0	108.0	115.0	85.0	101.0	108.0	109.0	124.0	105.0	
1984	88.0	85.0	112.0	87.0	91.0	87.0	87.0	142.0	95.0	
1985	61.0	82.0	124.0	93.0	108.0	75.0	87.0	103.0	90.0	
1986	57.0	65.0	67.0	71.0	76.0	67.0	110.0	118.0	99.0	
1987	58.0	65.0	70.0	86.0	93.0	74.0	87.0	73.0	101.0	
1988	63.0	115.0	70.0	66.0	67.0	83.0	79.0	77.0	102.0	
1989	71.0	60.0	89.0	74.0	73.0	91.0	86.0	74.0	87.0	
1990	43.0	69.0	73.0	77.0	69.0	76.0	78.0	70.0	83.0	
1991	54.0	55.0	66.0	65.0	60.0	65.0	96.0	55.0	71.0	
1992	34.0	47.0	56.0	53.0	53.0	55.0	67.0	52.0	46.0	
1993	33.0	40.0	46.0	45.0	41.0	55.0	57.0	54.0	46.0	
1994	30.0	35.0	42.0	48.0	44.0	45.0	45.0	45.0	46.0	
1995	30.0	39.0	45.0	52.0	28.0	40.0	62.0	NaN	NaN	

In [226...

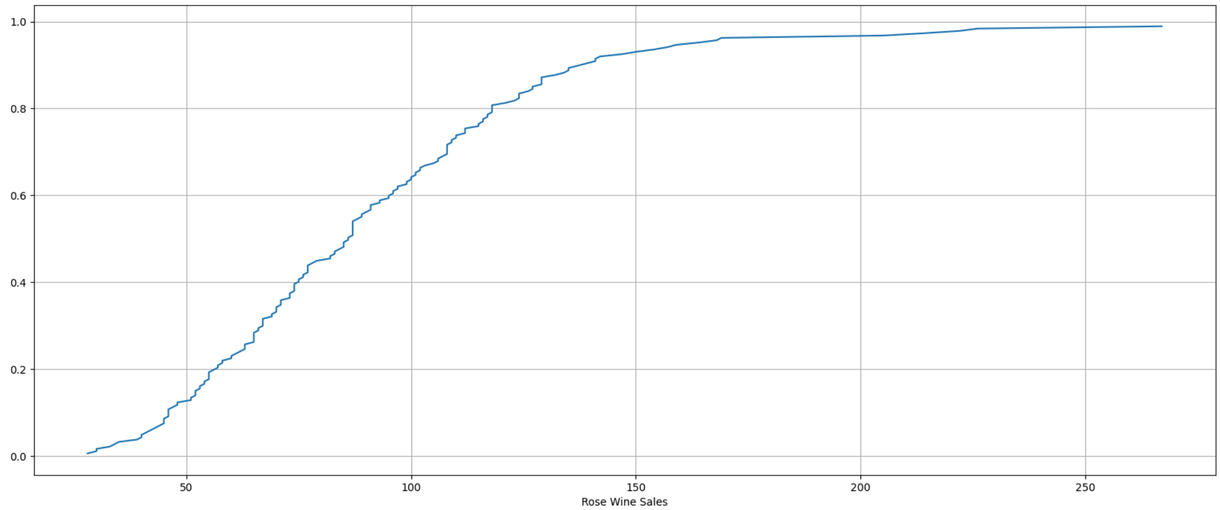
```
monthly_sales_across_years.plot(figsize=(20,8))
plt.grid()
plt.legend(loc='best')
plt.show()
```



Empirical Cumulative Distribution plot

In [73]:

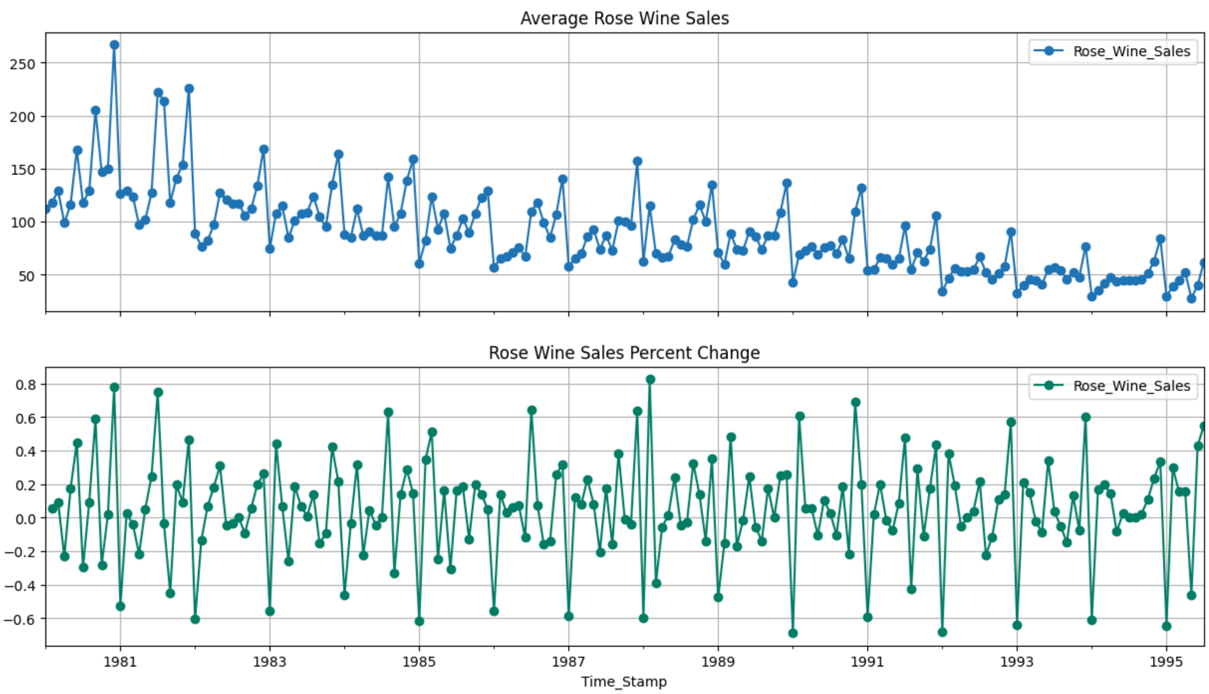
```
plt.figure(figsize = (20, 8))
cdf = ECDF(df['Rose_Wine_Sales'])
plt.plot(cdf.x, cdf.y, label = "statmodels");
plt.grid()
plt.xlabel('Rose Wine Sales');
```



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

In [228…

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

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

# plot average Rose Sales over time(year-month)
ax1 = average.plot(legend=True,ax=axis1,marker='o',title="Average Rose Wine Sales",
ax1.set_xticks(range(len(average)))
ax1.set_xticklabels(average.index.tolist())

# plot precent change for Rose Sales over time(year-month)
ax2 = pct_change.plot(legend=True,ax=axis2,marker='o',colormap="summer",title="Rose
```

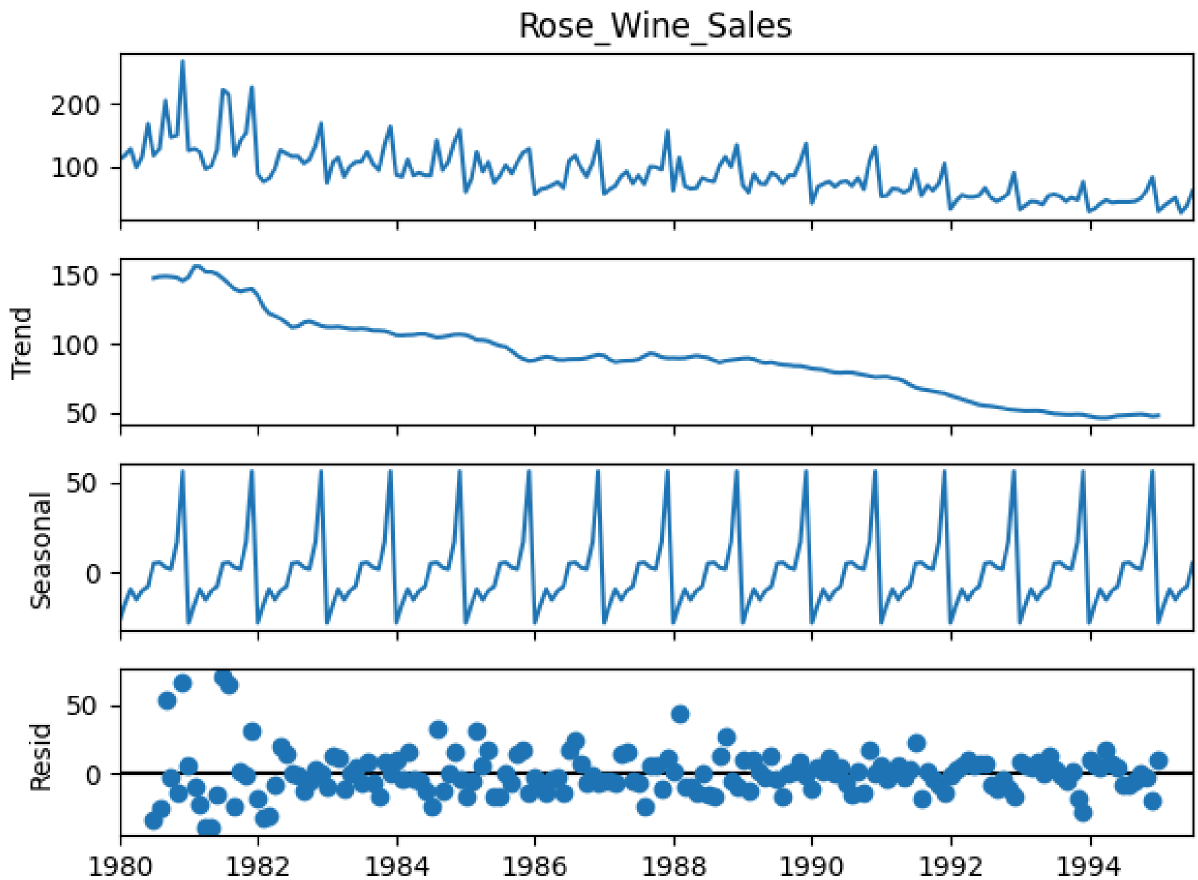


Decomposition

Additive Decomposition

In [230…

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



In [231…

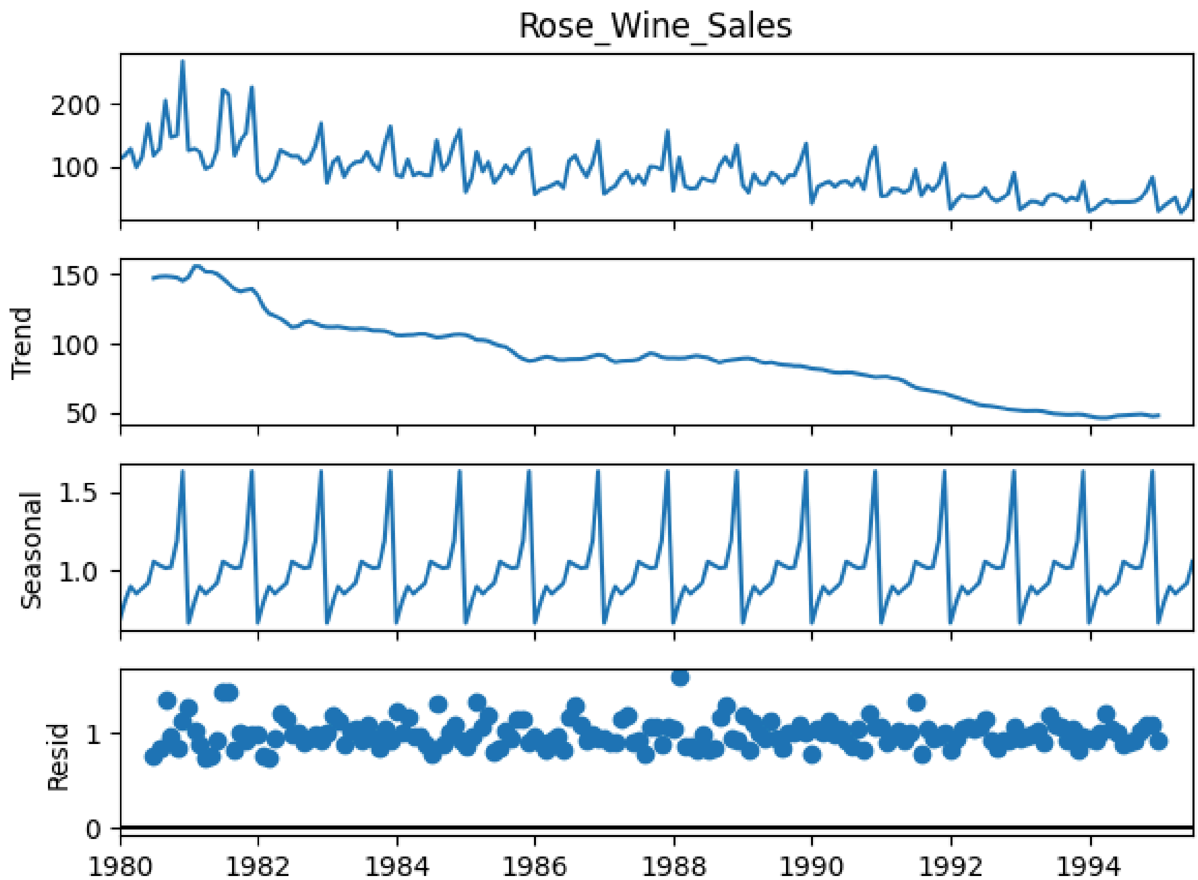
```
trend = decomposition_additive.trend
seasonality = decomposition_additive.seasonal
residual = decomposition_additive.resid

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    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
 Time_Stamp
1980-01-01   -27.903092
1980-02-01   -17.431663
1980-03-01    -9.279878
1980-04-01   -15.092378
1980-05-01   -10.190592
1980-06-01    -7.672735
1980-07-01     4.880241
1980-08-01     5.460797
1980-09-01     2.780241
1980-10-01     1.877464
1980-11-01    16.852464
1980-12-01    55.719130
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   -33.963575
1980-08-01   -24.585797
1980-09-01    53.844759
1980-10-01    -2.960797
1980-11-01   -14.269130
1980-12-01    66.155870
Name: resid, dtype: float64
```

Multiplicative Decomposition

In [232…

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



In [233…

```
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    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
 Time_Stamp
1980-01-01    0.670182
1980-02-01    0.806224
1980-03-01    0.901278
1980-04-01    0.854154
1980-05-01    0.889531
1980-06-01    0.924099
1980-07-01    1.057682
1980-08-01    1.035066
1980-09-01    1.017753
1980-10-01    1.022688
1980-11-01    1.192494
1980-12-01    1.628848
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    0.758514
1980-08-01    0.841382
1980-09-01    1.357534
1980-10-01    0.970661
1980-11-01    0.853274
1980-12-01    1.129506
Name: resid, dtype: float64
```

Data Pre-processing

Train-Test split

In [234…

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

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

In [235…

```
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	Rose_Wine_Sales
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
```

Time_Stamp	Rose_Wine_Sales
1990-06-01	76.0
1990-07-01	78.0
1990-08-01	70.0
1990-09-01	83.0
1990-10-01	65.0

In [236…

```
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	Rose_Wine_Sales
1990-11-01	110.0
1990-12-01	132.0
1991-01-01	54.0
1991-02-01	55.0
1991-03-01	66.0

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

	Rose_Wine_Sales
Time_Stamp	
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

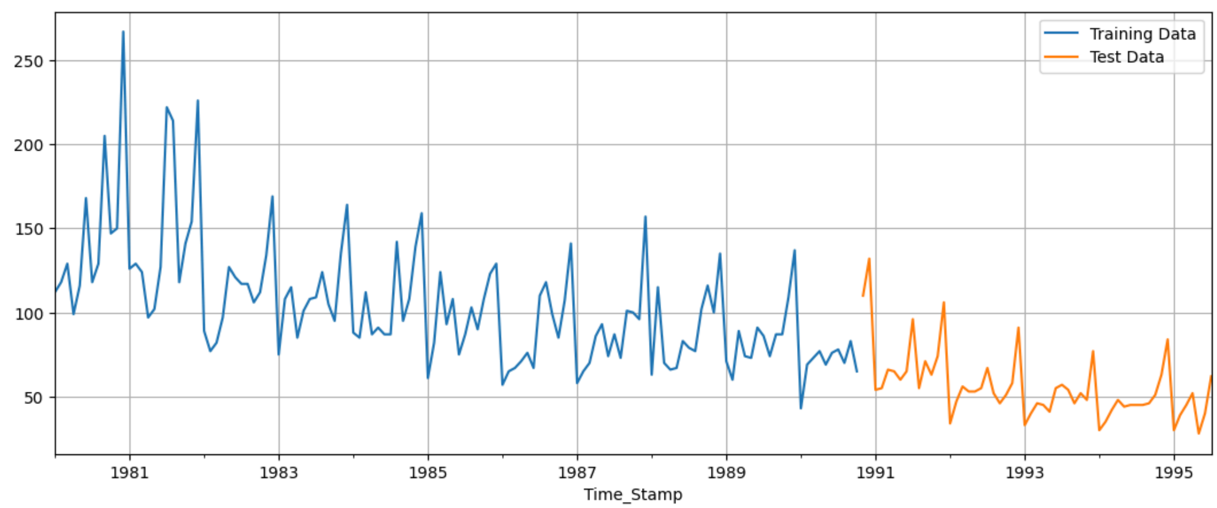
In [237...

```
# 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 [238...

```
train['Rose_Wine_Sales'].plot(figsize=(13,5), fontsize=10)
test['Rose_Wine_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 [239...

```
# 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 [240…

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

In [241…

```
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
            Rose_Wine_Sales  time
Time_Stamp
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_Wine_Sales  time
Time_Stamp
1990-06-01             76.0   126
1990-07-01             78.0   127
1990-08-01             70.0   128
1990-09-01             83.0   129
1990-10-01             65.0   130

First few rows of Test Data
            Rose_Wine_Sales  time
Time_Stamp
1990-11-01            110.0   131
1990-12-01            132.0   132
1991-01-01             54.0   133
1991-02-01             55.0   134
1991-03-01             66.0   135

Last few rows of Test Data
            Rose_Wine_Sales  time
Time_Stamp
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 [242…

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

In [243…

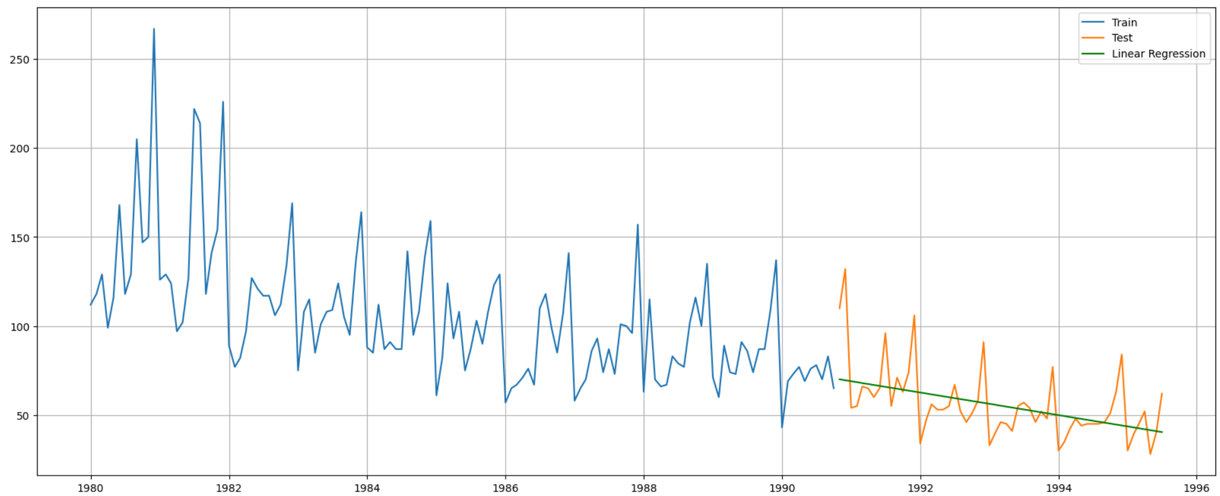
```
lr.fit(LinearRegression_train[['time']],LinearRegression_train['Rose_Wine_Sales'].v
```

Out[243…

```
▾ LinearRegression
LinearRegression()
```

In [245…

```
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['Rose_Wine_Sales'], label='Train')
plt.plot(test['Rose_Wine_Sales'], label='Test')
plt.plot(LinearRegression_test['RegOnTime'], label='Linear Regression', color = 'gr
plt.legend(loc='best')
plt.grid()
```



Model Evaluation

In [247…

```
# Test Data - RMSE

rmse_model1_test = metrics.mean_squared_error(test['Rose_Wine_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 17.36
```

In [248…

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

Out[248…

	Test RMSE
Linear Regression	17.356924

Moving Average (MA) Model

In [249…

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

Out[249…

	Rose_Wine_Sales
Time_Stamp	
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 [250…

```
#Trailing Moving Average

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

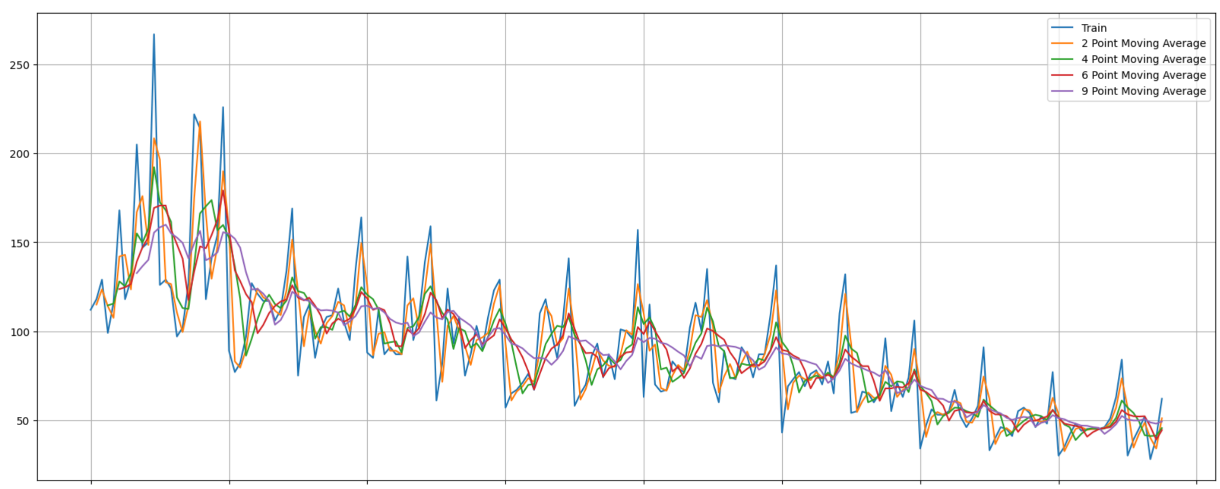
```
MovingAverage['Trailing_4'] = MovingAverage['Rose_Wine_Sales'].rolling(4).mean() #
MovingAverage['Trailing_6'] = MovingAverage['Rose_Wine_Sales'].rolling(6).mean() #
MovingAverage['Trailing_9'] = MovingAverage['Rose_Wine_Sales'].rolling(9).mean() #

MovingAverage.head()
```

Out[250…

	Rose_Wine_Sales	Trailing_2	Trailing_4	Trailing_6	Trailing_9
Time_Stamp					
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	114.5	NaN	NaN
1980-05-01	116.0	107.5	115.5	NaN	NaN

In [251…

```
# Plotting on the whole data

plt.figure(figsize=(20,8))
plt.plot(MovingAverage['Rose_Wine_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();
```



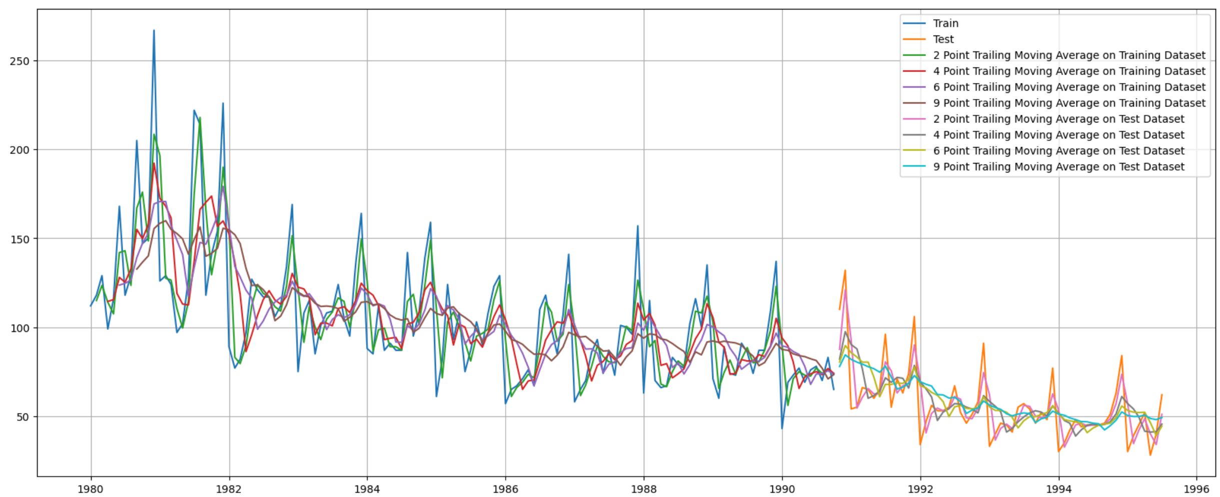
In [252…

```
# 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 [253…

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

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

```
plt.plot(trailing_MovingAverage_train['Rose_Wine_Sales'], label='Train')
plt.plot(trailing_MovingAverage_test['Rose_Wine_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 [254…

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

rmse_model4_test_2 = metrics.mean_squared_error(test['Rose_Wine_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['Rose_Wine_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['Rose_Wine_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['Rose_Wine_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 11.801
For 4 point Moving Average Model forecast on the Test Data, RMSE is 15.371
For 6 point Moving Average Model forecast on the Test Data, RMSE is 15.867
For 9 point Moving Average Model forecast on the Test Data, RMSE is 16.345
```

In [255…

```
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
```

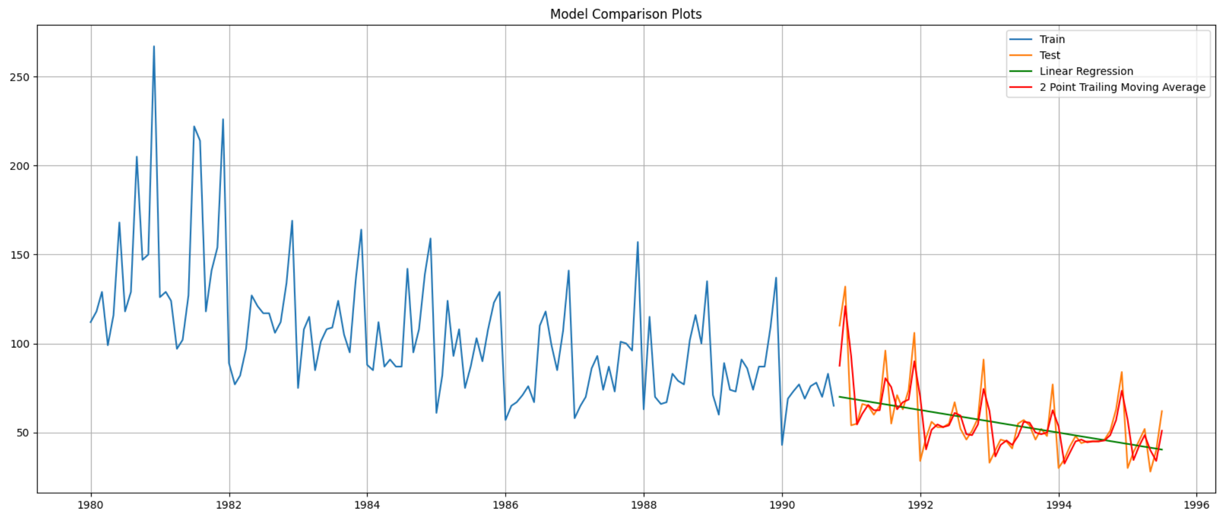
Out[255…

	Test RMSE
Linear Regression	17.356924
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032

Model Comparison Plots

In [256…

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

plt.figure(figsize=(20,8))
plt.plot(train['Rose_Wine_Sales'], label='Train')
plt.plot(test['Rose_Wine_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 [257…
```
SES_train = train.copy()
SES_test = test.copy()
```

In [258…
```
model_SES = SimpleExpSmoothing(SES_train['Rose_Wine_Sales']) # Define the simple ex
```

In [259…
```
model_SES_autofit = model_SES.fit(optimized=True) # Fit the simple exponential smoo
```

In [260…
```
model_SES_autofit.params
```

Out[260…
```
{'smoothing_level': 0.1277774057492626,
 'smoothing_trend': nan,
 'smoothing_seasonal': nan,
 'damping_trend': nan,
 'initial_level': 112.0,
 'initial_trend': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

In [261…
```
SES_test['predict'] = model_SES_autofit.forecast(steps=len(test))
SES_test.head()
```

Out[261…

	Rose_Wine_Sales	predict
Time_Stamp		
1990-11-01	110.0	77.599284
1990-12-01	132.0	77.599284
1991-01-01	54.0	77.599284
1991-02-01	55.0	77.599284
1991-03-01	66.0	77.599284

In [262…

```
## Test Data

rmse_model5_test_1 = metrics.mean_squared_error(SES_test['Rose_Wine_Sales'],SES_tes
print("For Alpha = 0.127 Simple Exponential Smoothing Model forecast on the Test Da
```

```
For Alpha = 0.127 Simple Exponential Smoothing Model forecast on the Test Data, RMSE
is 29.243
```

In [263…

```
resultsDf_5 = pd.DataFrame({'Test RMSE': [rmse_model5_test_1]},index=['Alpha=0.127,

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

Out[263…

	Test RMSE
Linear Regression	17.356924
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032
Alpha=0.127,Simple Exponential Smoothing	29.243074

Setting different alpha (α) values

In [264…

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

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

Out[264…

Alpha Values	Train RMSE	Test RMSE

In [265…

```
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['Rose_Wine_Sales'],S

    rmse_model5_test_i = metrics.mean_squared_error(SES_test['Rose_Wine_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 [266…

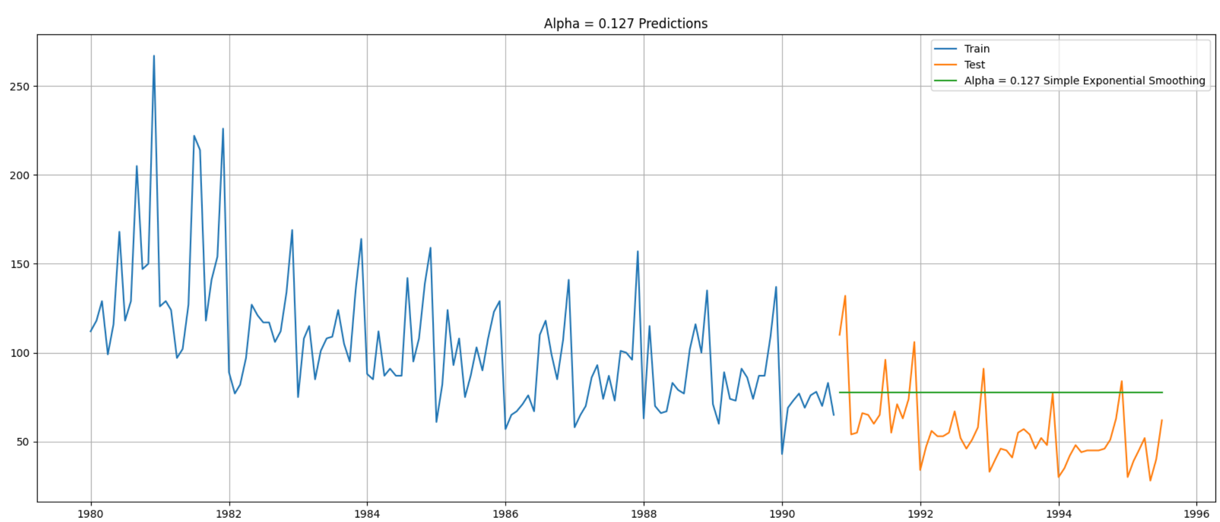
```
resultsDf_6.sort_values(by=['Test RMSE'],ascending=True)
```

Out[266...

	Alpha Values	Train RMSE	Test RMSE
7	1.0	38.833273	21.782820
6	0.9	37.507371	22.513502
5	0.8	36.330954	23.230049
4	0.7	35.288467	23.912646
3	0.6	34.372651	24.547868
2	0.5	33.578304	25.127923
1	0.4	32.893017	25.676296
0	0.3	32.292266	26.329097

In [267...

```
# Plotting on both Training and Test dataset

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

# To plot the predictions made by simple exponential smoothening model
plt.plot(SES_test['predict'], label='Alpha = 0.127 Simple Exponential Smoothing')

plt.legend(loc='best')
plt.grid()
plt.title('Alpha = 0.127 Predictions')
plt.show()
```



In [268...

```
# 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=1.0,Simple Exponential Smoothing'])

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

resultsDf
```

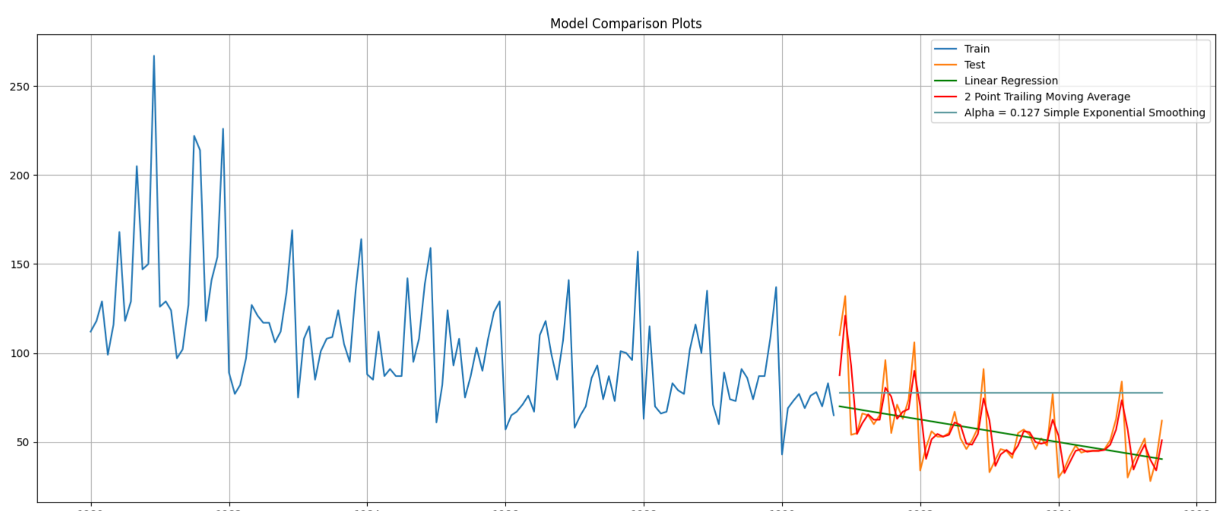
Out[268...

	Test RMSE
Linear Regression	17.356924
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032
Alpha=0.127,Simple Exponential Smoothing	29.243074
Alpha=1.0,Simple Exponential Smoothing	21.782820

Model Comparison Plots

In [269...

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

plt.figure(figsize=(20,8))
plt.plot(train['Rose_Wine_Sales'], label='Train')
plt.plot(test['Rose_Wine_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 = 0.127 Simple Exponential Smoothing', c

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



Double Exponential Smoothing (Holt's Model)

In [270…
```
DES_train = train.copy()
DES_test = test.copy()
```

In [271…
```
model_DES = Holt(DES_train['Rose_Wine_Sales'])
```

In [272…
```
model_DES_autofit = model_DES.fit()
```

In [273…
```
model_DES_autofit.params
```

Out[273…
```
{'smoothing_level': 0.15194196175832653,
 'smoothing_trend': 0.15194196172434046,
 'smoothing_seasonal': nan,
 'damping_trend': nan,
 'initial_level': 112.0,
 'initial_trend': 6.0,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

In [274…
```
DES_test['predict'] = model_DES_autofit.forecast(steps=len(test))
DES_test.head()
```

Out[274…

	Rose_Wine_Sales	predict
Time_Stamp		
1990-11-01	110.0	71.319113
1990-12-01	132.0	70.164359
1991-01-01	54.0	69.009605
1991-02-01	55.0	67.854852
1991-03-01	66.0	66.700098

In [276…
```
## Test Data

rmse_model6_test_1 = metrics.mean_squared_error(DES_test['Rose_Wine_Sales'],DES_tes
print("For Alpha=0.151,Beta=0.151, Double Exponential Smoothing Model forecast on t
```

```
For Alpha=0.151,Beta=0.151, Double Exponential Smoothing Model forecast on the Test
Data, RMSE is 26.032
```

In [277…
```
resultsDf_7 = pd.DataFrame({'Test RMSE': [rmse_model6_test_1]},index=['Alpha=0.151,

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

Out[277…

	Test RMSE
Linear Regression	17.356924
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032
Alpha=0.127,Simple Exponential Smoothing	29.243074
Alpha=1.0,Simple Exponential Smoothing	21.782820
Alpha=0.151,Beta=0.151 Double Exponential Smoothing	26.032338

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

In [278…

```
## 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[278…

Alpha Values	Beta Values	Train RMSE	Test RMSE

In [279…

```
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['Rose_Wine_Sales

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

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

Model Evaluation

In [280…

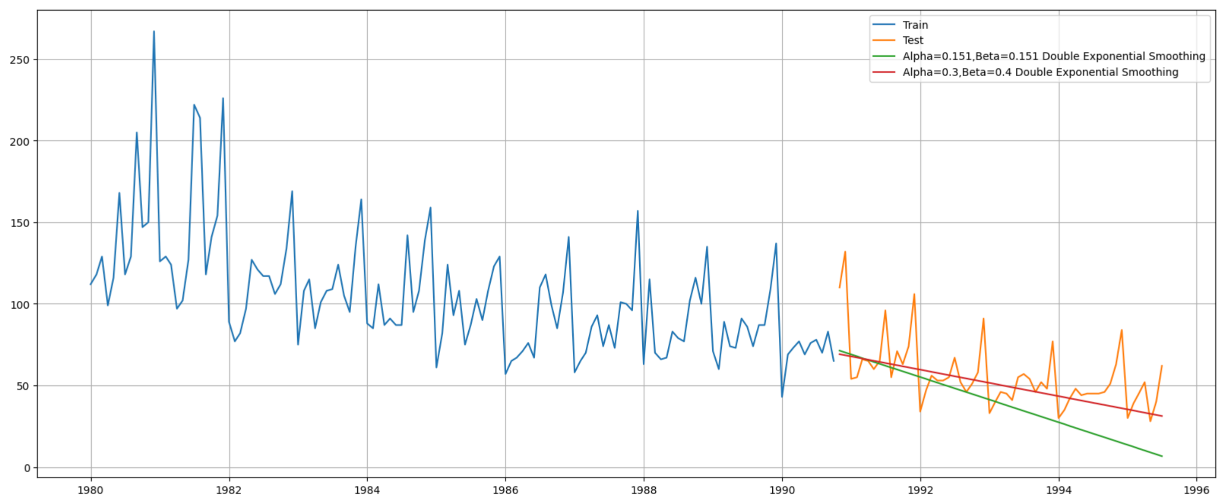
```
resultsDf_8.sort_values(by=['Test RMSE']).head()
```

Out[280…

	Alpha Values	Beta Values	Train RMSE	Test RMSE
1	0.3	0.4	37.287813	18.337178
12	0.4	0.7	40.744796	18.985480
9	0.4	0.4	37.990913	19.144553
17	0.5	0.4	38.598226	19.187818
8	0.4	0.3	36.682435	19.759466

In [281…

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

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

plt.plot(DES_test['predict'], label='Alpha=0.151,Beta=0.151 Double Exponential Smoo

plt.plot(DES_test['predict', 0.3, 0.4], label='Alpha=0.3,Beta=0.4 Double Exponentia

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



In [282…

```
resultsDf_8_1 = pd.DataFrame({'Test RMSE': [resultsDf_8.sort_values(by=['Test RMSE'
                            ,index=['Alpha=0.3,Beta=0.4,Double Exponential Smoothing

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

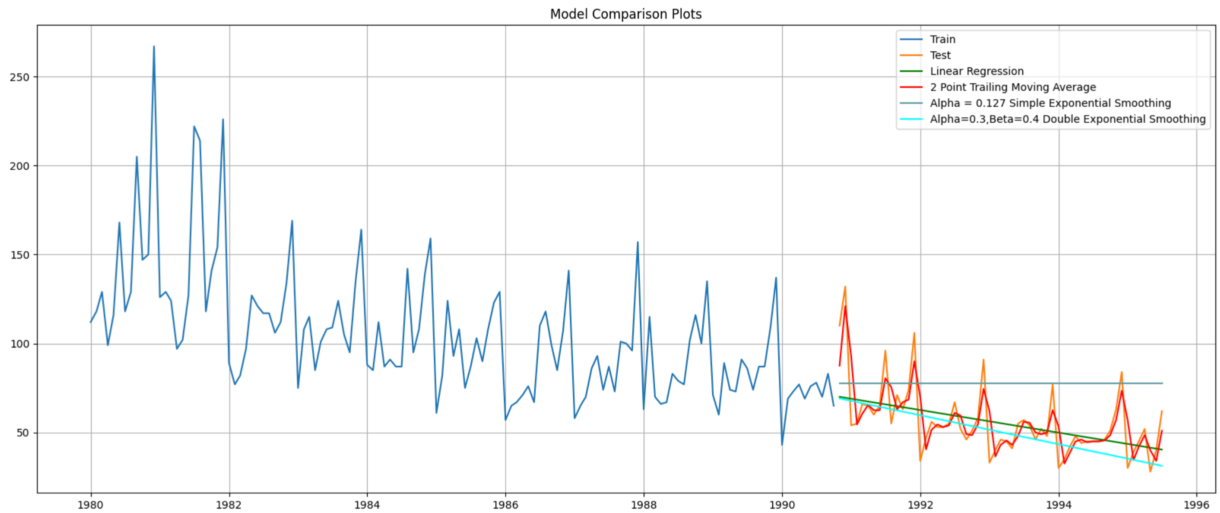
Out[282...

	Test RMSE
Linear Regression	17.356924
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032
Alpha=0.127,Simple Exponential Smoothing	29.243074
Alpha=1.0,Simple Exponential Smoothing	21.782820
Alpha=0.151,Beta=0.151 Double Exponential Smoothing	26.032338
Alpha=0.3,Beta=0.4,Double Exponential Smoothing	18.337178

Model Comparison Plots

In [283...

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

plt.figure(figsize=(20,8))
plt.plot(train['Rose_Wine_Sales'], label='Train')
plt.plot(test['Rose_Wine_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'], label='Alpha = 0.127 Simple Exponential Smoothing', c

# To plot the predictions made by double exponential smoothing model
plt.plot(DES_test['predict', 0.3, 0.4], label='Alpha=0.3,Beta=0.4 Double Exponentia

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



Triple Exponential Smoothing (Holt - Winter's Model)

In [284…

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

In [285…

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

In [286…

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

In [287…

```
model_TES_autofit.params
```

Out[287…

```
{'smoothing_level': 0.0999080139189177,
 'smoothing_trend': 1.9932826568022853e-06,
 'smoothing_seasonal': 0.00017683239767298466,
 'damping_trend': nan,
 'initial_level': 109.16836143052193,
 'initial_trend': -0.44137924420686336,
 'initial_seasons': array([1.0049411 , 1.13565754, 1.2416344 , 1.08896356, 1.22239
28 ,
        1.31686195, 1.44959601, 1.55043078, 1.45169973, 1.42782318,
        1.64159637, 2.26353792]),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

In [288…

```
## Prediction on the test dataset

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

Out[288...

Time_Stamp	Rose_Wine_Sales	auto_predict
1990-11-01	110.0	86.291902
1990-12-01	132.0	117.979447
1991-01-01	54.0	51.933830
1991-02-01	55.0	58.193935
1991-03-01	66.0	63.075288

In [290...

```
## Test Data

rmse_model7_test_1 = metrics.mean_squared_error(TES_test['Rose_Wine_Sales'],TES_tes
print("For Alpha=0.099,Beta=1.993,Gamma=0.0001, Triple Exponential Smoothing Model
```

```
For Alpha=0.099,Beta=1.993,Gamma=0.0001, Triple Exponential Smoothing Model forecast
on the Test Data, RMSE is 9.337
```

In [291...

```
resultsDf_9 = pd.DataFrame({'Test RMSE': [rmse_model7_test_1]},index=['Alpha=0.099,

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

Out[291...

	Test RMSE
Linear Regression	17.356924
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032
Alpha=0.127,Simple Exponential Smoothing	29.243074
Alpha=1.0,Simple Exponential Smoothing	21.782820
Alpha=0.151,Beta=0.151 Double Exponential Smoothing	26.032338
Alpha=0.3,Beta=0.4,Double Exponential Smoothing	18.337178
Alpha=0.099,Beta=1.993,Gamma=0.0001 Triple Exponential Smoothing	9.337258

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

In [292...

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

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

Out[292…

Alpha Values	Beta Values	Gamma Values	Train RMSE	Test RMSE

In [293…

```
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['Rose_Wine_S

            rmse_model7_test_i = metrics.mean_squared_error(TES_test['Rose_Wine_Sal

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

Model Evaluation

In [294…

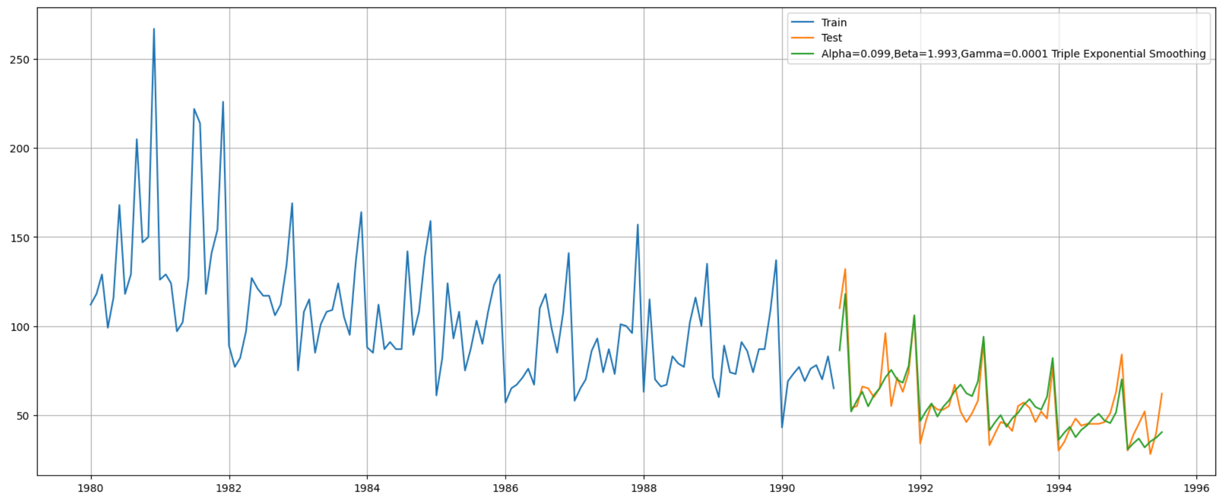
```
resultsDf_10.sort_values(by=['Test RMSE']).head()
```

Out[294…

	Alpha Values	Beta Values	Gamma Values	Train RMSE	Test RMSE
33	0.3	0.7	0.4	29.968505	28.341124
177	0.5	0.9	0.4	41.232290	28.840273
25	0.3	0.6	0.4	27.743621	39.634578
78	0.4	0.4	0.9	43.001123	51.414131
135	0.5	0.3	1.0	47.353331	62.140816

In [295…

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

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

plt.plot(TES_test['auto_predict'], label='Alpha=0.099,Beta=1.993,Gamma=0.0001 Tripl

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



In [296…
```
resultsDf_10_1 = pd.DataFrame({'Test RMSE': [resultsDf_10.sort_values(by=['Test RMS
                                ,index=['Alpha=0.3,Beta=0.7,Gamma=0.4,Triple Exponential

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

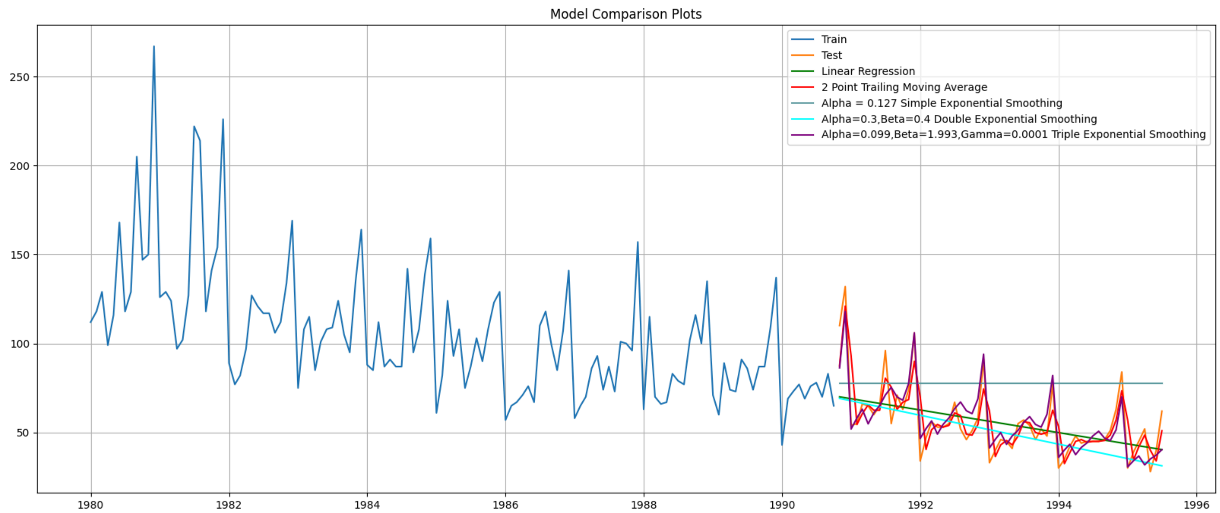
Out[296…

	Test RMSE
Linear Regression	17.356924
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032
Alpha=0.127,Simple Exponential Smoothing	29.243074
Alpha=1.0,Simple Exponential Smoothing	21.782820
Alpha=0.151,Beta=0.151 Double Exponential Smoothing	26.032338
Alpha=0.3,Beta=0.4,Double Exponential Smoothing	18.337178
Alpha=0.099,Beta=1.993,Gamma=0.0001 Triple Exponential Smoothing	9.337258
Alpha=0.3,Beta=0.7,Gamma=0.4,Triple Exponential Smoothing	28.341124

Model Comparison Plots

In [297…
```
# Plotting on both the Training and Test data

plt.figure(figsize=(20,8))
plt.plot(train['Rose_Wine_Sales'], label='Train')
plt.plot(test['Rose_Wine_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'], label='Alpha = 0.127 Simple Exponential Smoothing', c

# To plot the predictions made by double exponential smoothing model
plt.plot(DES_test['predict', 0.3, 0.4], label='Alpha=0.3,Beta=0.4 Double Exponentia

# To plot the predictions based on the triple exponential smoothing model
plt.plot(TES_test['auto_predict'], label='Alpha=0.099,Beta=1.993,Gamma=0.0001 Tripl

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



Check for Stationarity

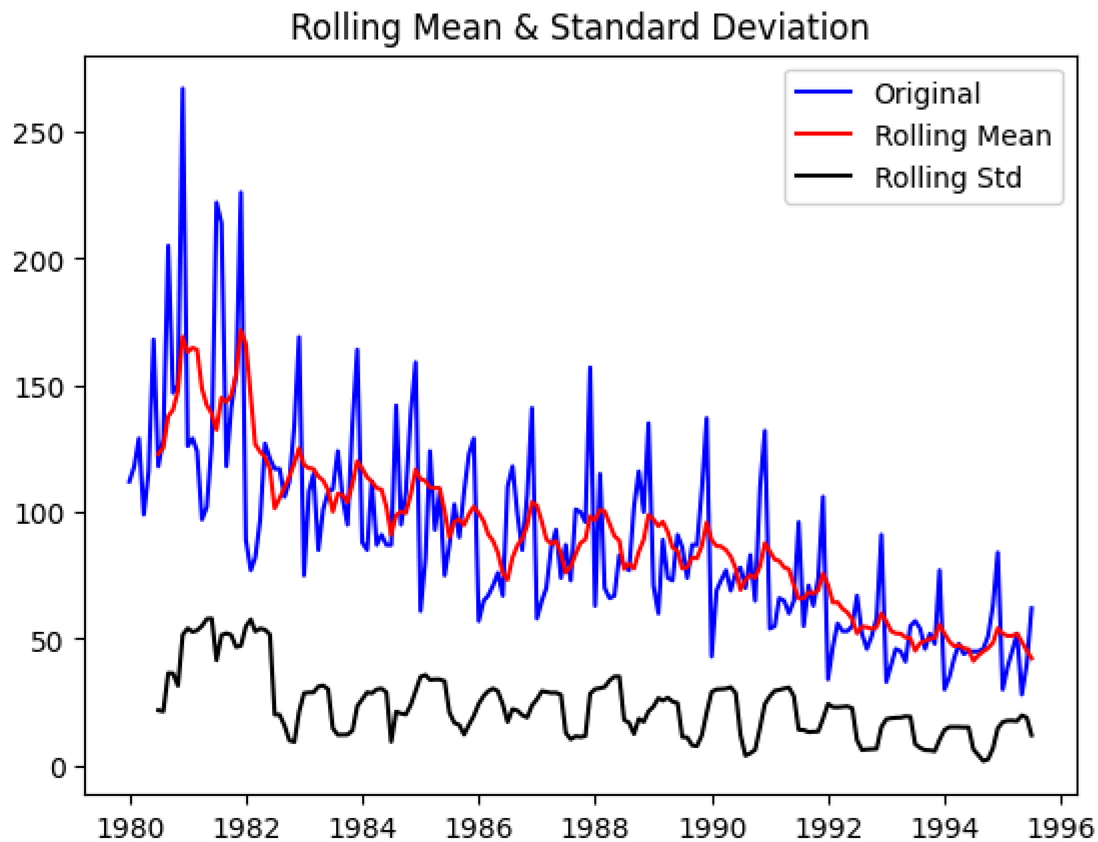
In [298…

```
## 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 [299…

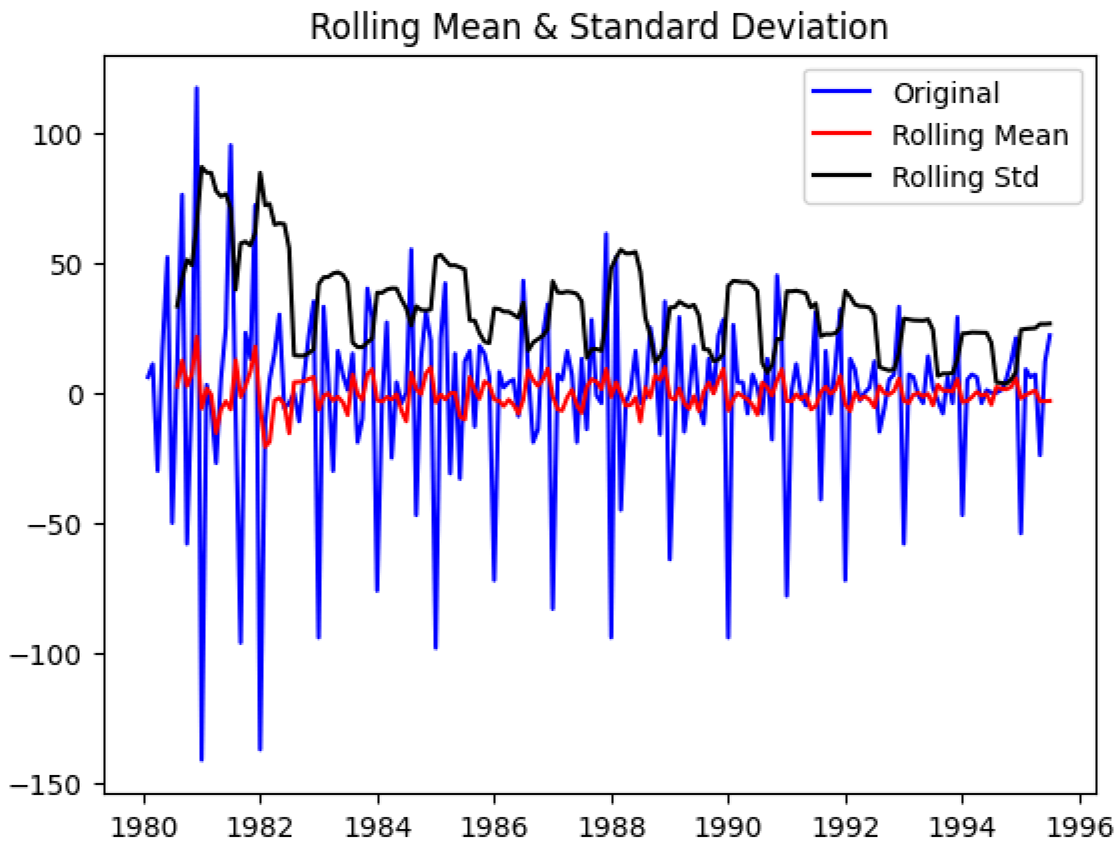
```
test_stationarity(df['Rose_Wine_Sales'])
```




```
Results of Dickey-Fuller Test:
Test Statistic                  -1.874856
p-value                          0.343981
#Lags Used                      13.000000
Number of Observations Used    173.000000
Critical Value (1%)             -3.468726
Critical Value (5%)             -2.878396
Critical Value (10%)            -2.575756
dtype: float64
```

In [300…

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



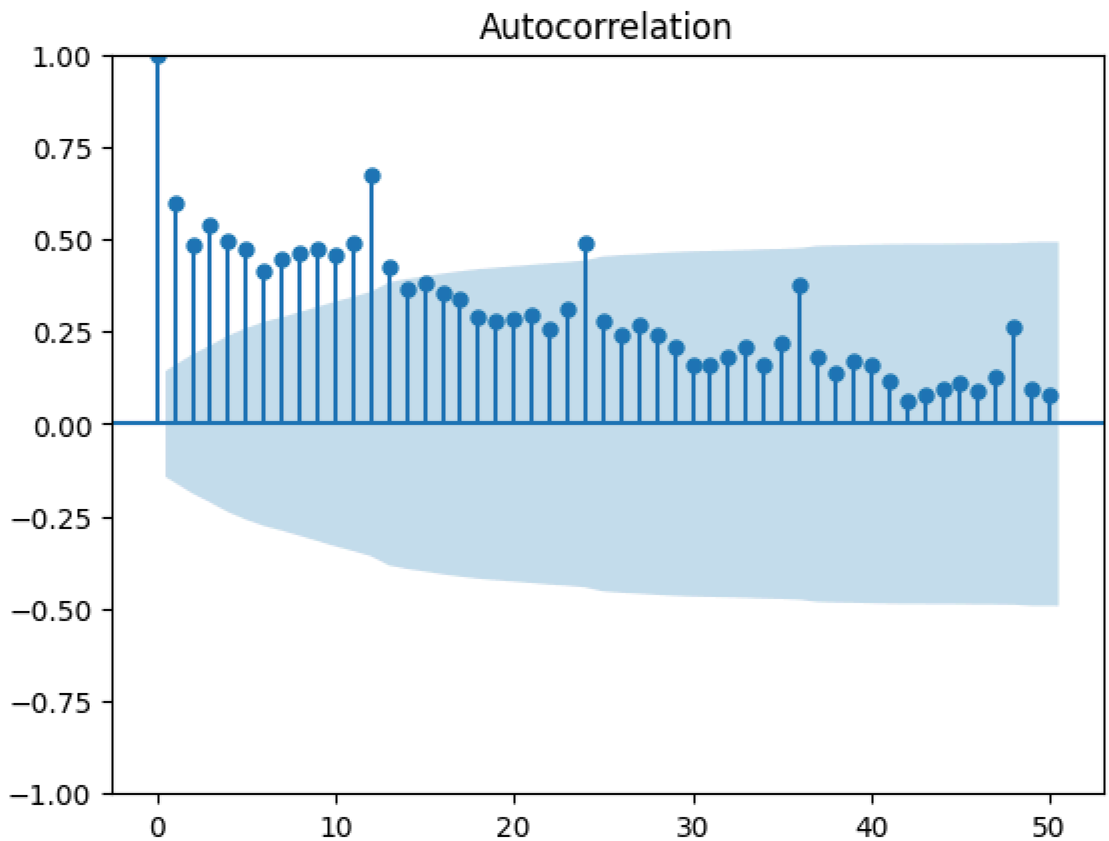

```
Results of Dickey-Fuller Test:
Test Statistic                 -8.044139e+00
p-value                         1.813580e-12
#Lags Used                      1.200000e+01
Number of Observations Used     1.730000e+02
Critical Value (1%)            -3.468726e+00
Critical Value (5%)            -2.878396e+00
Critical Value (10%)           -2.575756e+00
dtype: float64
```

Model Building - Stationary Data

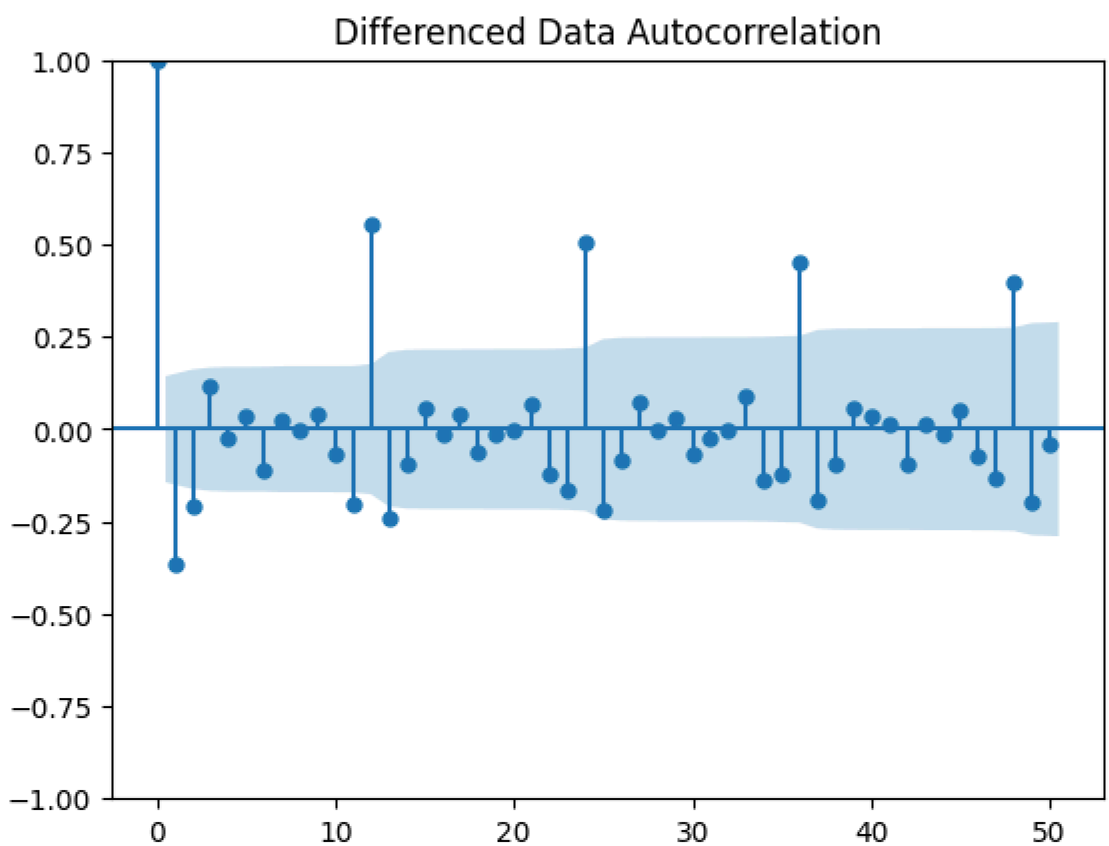
Autocorrelation and Partial Autocorrelation function plots

In [301…

```
plot_acf(df['Rose_Wine_Sales'],lags=50)
plot_acf(df['Rose_Wine_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 [302…

```
## 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 [303…

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

Out[303…

param	AIC

In [304…

```
for param in pdq:
    ARMA_model = ARIMA(train['Rose_Wine_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:1313.1758613526422
ARIMA(0, 1, 1) - AIC:1261.3274438405824
ARIMA(0, 1, 2) - AIC:1259.2477803151232
ARIMA(1, 1, 0) - AIC:1297.0772943848556
ARIMA(1, 1, 1) - AIC:1260.0367627035926
ARIMA(1, 1, 2) - AIC:1259.4732049501208
ARIMA(2, 1, 0) - AIC:1278.1352807484318
ARIMA(2, 1, 1) - AIC:1261.0140762916876
ARIMA(2, 1, 2) - AIC:1261.4720006568955
```

In [305…

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

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

Out[305...

	param	AIC
2	(0, 1, 2)	1259.247780
5	(1, 1, 2)	1259.473205
4	(1, 1, 1)	1260.036763
7	(2, 1, 1)	1261.014076
1	(0, 1, 1)	1261.327444
8	(2, 1, 2)	1261.472001
6	(2, 1, 0)	1278.135281
3	(1, 1, 0)	1297.077294
0	(0, 1, 0)	1313.175861

In [306...

```
auto_ARIMA = ARIMA(train['Rose_Wine_Sales'], order=(0,1,0),freq='MS')

results_auto_ARIMA = auto_ARIMA.fit()

print(results_auto_ARIMA.summary())
```

```
                               SARIMAX Results                                
==============================================================================
Dep. Variable:        Rose_Wine_Sales   No. Observations:                  130
Model:                 ARIMA(0, 1, 0)   Log Likelihood                -655.588
Date:                Sun, 17 Mar 2024   AIC                           1313.176
Time:                        13:05:17   BIC                           1316.036
Sample:                    01-01-1980   HQIC                          1314.338
                         - 10-01-1990                                         
Covariance Type:                  opg                                         
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
sigma2      1519.7229    123.131     12.342      0.000    1278.391    1761.054
===================================================================================
Ljung-Box (L1) (Q):                  17.38   Jarque-Bera (JB):                57.81
Prob(Q):                              0.00   Prob(JB):                         0.00
Heteroskedasticity (H):               0.36   Skew:                            -0.94
Prob(H) (two-sided):                  0.00   Kurtosis:                         5.69
===================================================================================

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

Model Evaluation

In [307...

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

In [308...

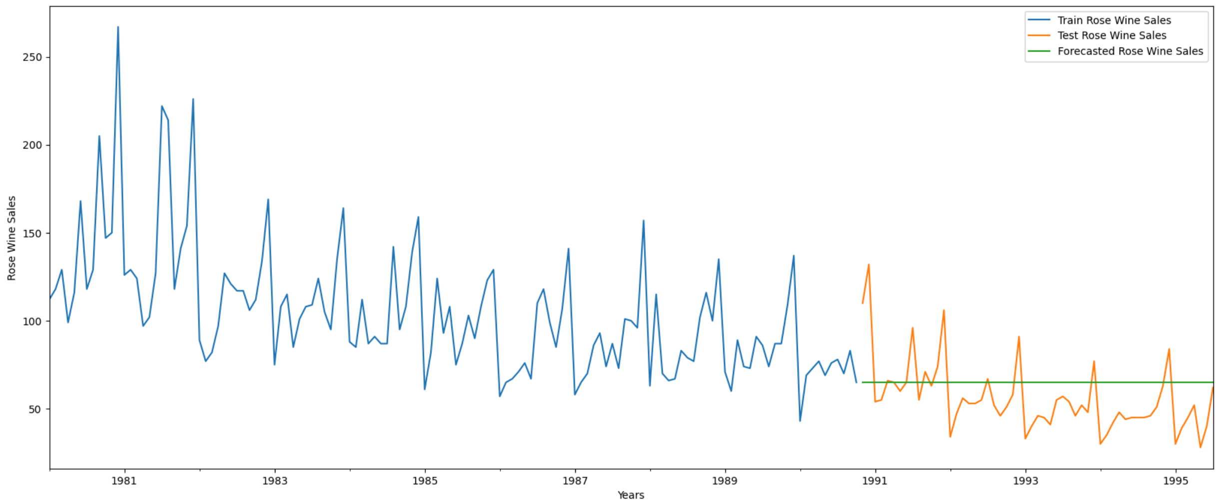
```
predicted_auto_ARIMA = results_auto_ARIMA.get_forecast(steps=len(test))
```

In [309…

```
Rose_Wine_Sales_Forecasted = pred_dynamic.predicted_mean
testCopy1 = test.copy()
testCopy1['Rose_Wine_Sales_Forecasted'] = predicted_auto_ARIMA.predicted_mean
```

In [310…

```
axis = train['Rose_Wine_Sales'].plot(label='Train Rose Wine Sales', figsize=(20, 8)
testCopy1['Rose_Wine_Sales'].plot(ax=axis, label='Test Rose Wine Sales')
testCopy1['Rose_Wine_Sales_Forecasted'].plot(ax=axis, label='Forecasted Rose Wine S
axis.set_xlabel('Years')
axis.set_ylabel('Rose Wine Sales')
plt.legend(loc='best')
plt.show()
plt.close()
```



In [311…

```
rmse_arima = mean_squared_error(test['Rose_Wine_Sales'],predicted_auto_ARIMA.predic
print("For order=(0,1,0), Auto ARIMA Model forecast on the Test Data,  RMSE is %3.3
```

```
For order=(0,1,0), Auto ARIMA Model forecast on the Test Data,  RMSE is 21.783
```

In [312…

```
resultsDf_11 = pd.DataFrame({'Test RMSE': [rmse_arima]},index=['order=(0,1,0) ARIMA

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

Out[312...

	Test RMSE
Linear Regression	17.356924
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032
Alpha=0.127,Simple Exponential Smoothing	29.243074
Alpha=1.0,Simple Exponential Smoothing	21.782820
Alpha=0.151,Beta=0.151 Double Exponential Smoothing	26.032338
Alpha=0.3,Beta=0.4,Double Exponential Smoothing	18.337178
Alpha=0.099,Beta=1.993,Gamma=0.0001 Triple Exponential Smoothing	9.337258
Alpha=0.3,Beta=0.7,Gamma=0.4,Triple Exponential Smoothing	28.341124
order=(0,1,0) ARIMA	21.782820

SARIMA Model

In [313...

```
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 [314...

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

Out[314...

param	seasonal	AIC

In [316...

```
for param in pdq:
    for param_seasonal in model_pdq:
        SARIMA_model = sm.tsa.statespace.SARIMAX(train['Rose_Wine_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:1303.984314159292
SARIMA(0, 1, 0)x(0, 0, 1, 12) - AIC:1127.0323185122152
SARIMA(0, 1, 0)x(0, 0, 2, 12) - AIC:956.4131665692037
SARIMA(0, 1, 0)x(1, 0, 0, 12) - AIC:1121.3977282304234
SARIMA(0, 1, 0)x(1, 0, 1, 12) - AIC:1097.1665795246952
SARIMA(0, 1, 0)x(1, 0, 2, 12) - AIC:950.6998497228892
SARIMA(0, 1, 0)x(2, 0, 0, 12) - AIC:941.2946512703301
SARIMA(0, 1, 0)x(2, 0, 1, 12) - AIC:943.2558966541914
SARIMA(0, 1, 0)x(2, 0, 2, 12) - AIC:936.3148887533573
SARIMA(0, 1, 1)x(0, 0, 0, 12) - AIC:1242.5766056799896
SARIMA(0, 1, 1)x(0, 0, 1, 12) - AIC:1079.9832204946422
SARIMA(0, 1, 1)x(0, 0, 2, 12) - AIC:904.3132399734159
SARIMA(0, 1, 1)x(1, 0, 0, 12) - AIC:1078.2285176554356
SARIMA(0, 1, 1)x(1, 0, 1, 12) - AIC:1035.7241510279464
SARIMA(0, 1, 1)x(1, 0, 2, 12) - AIC:901.6481142950458
SARIMA(0, 1, 1)x(2, 0, 0, 12) - AIC:897.5837355743923
SARIMA(0, 1, 1)x(2, 0, 1, 12) - AIC:898.6607999294712
SARIMA(0, 1, 1)x(2, 0, 2, 12) - AIC:884.3850768411747
SARIMA(0, 1, 2)x(0, 0, 0, 12) - AIC:1231.2314145388955
SARIMA(0, 1, 2)x(0, 0, 1, 12) - AIC:1065.389179967098
SARIMA(0, 1, 2)x(0, 0, 2, 12) - AIC:894.4419226275869
SARIMA(0, 1, 2)x(1, 0, 0, 12) - AIC:1071.6440642890716
SARIMA(0, 1, 2)x(1, 0, 1, 12) - AIC:1026.7446561818697
SARIMA(0, 1, 2)x(1, 0, 2, 12) - AIC:888.1231053550697
SARIMA(0, 1, 2)x(2, 0, 0, 12) - AIC:895.8772183601822
SARIMA(0, 1, 2)x(2, 0, 1, 12) - AIC:897.330095946717
SARIMA(0, 1, 2)x(2, 0, 2, 12) - AIC:871.0752383372954
SARIMA(1, 1, 0)x(0, 0, 0, 12) - AIC:1287.8863498975584
SARIMA(1, 1, 0)x(0, 0, 1, 12) - AIC:1117.0161467241387
SARIMA(1, 1, 0)x(0, 0, 2, 12) - AIC:943.5830348969492
SARIMA(1, 1, 0)x(1, 0, 0, 12) - AIC:1106.4720677346959
SARIMA(1, 1, 0)x(1, 0, 1, 12) - AIC:1086.8367200388916
SARIMA(1, 1, 0)x(1, 0, 2, 12) - AIC:939.0945779695301
SARIMA(1, 1, 0)x(2, 0, 0, 12) - AIC:919.9038293903151
SARIMA(1, 1, 0)x(2, 0, 1, 12) - AIC:921.8570502213606
SARIMA(1, 1, 0)x(2, 0, 2, 12) - AIC:923.485535697853
SARIMA(1, 1, 1)x(0, 0, 0, 12) - AIC:1241.6300492575135
SARIMA(1, 1, 1)x(0, 0, 1, 12) - AIC:1076.159275257069
SARIMA(1, 1, 1)x(0, 0, 2, 12) - AIC:903.9456130482074
SARIMA(1, 1, 1)x(1, 0, 0, 12) - AIC:1066.1584467973262
SARIMA(1, 1, 1)x(1, 0, 1, 12) - AIC:1035.7723005493954
SARIMA(1, 1, 1)x(1, 0, 2, 12) - AIC:899.5130609044788
SARIMA(1, 1, 1)x(2, 0, 0, 12) - AIC:888.7495145600985
SARIMA(1, 1, 1)x(2, 0, 1, 12) - AIC:890.3875310824516
SARIMA(1, 1, 1)x(2, 0, 2, 12) - AIC:883.6660397652686
SARIMA(1, 1, 2)x(0, 0, 0, 12) - AIC:1231.5587519588084
SARIMA(1, 1, 2)x(0, 0, 1, 12) - AIC:1067.3841149203404
SARIMA(1, 1, 2)x(0, 0, 2, 12) - AIC:896.4380127094355
SARIMA(1, 1, 2)x(1, 0, 0, 12) - AIC:1064.1976586919916
SARIMA(1, 1, 2)x(1, 0, 1, 12) - AIC:1024.1455749178697
SARIMA(1, 1, 2)x(1, 0, 2, 12) - AIC:890.0396976551013
SARIMA(1, 1, 2)x(2, 0, 0, 12) - AIC:889.0946821104656
SARIMA(1, 1, 2)x(2, 0, 1, 12) - AIC:890.642161694707
SARIMA(1, 1, 2)x(2, 0, 2, 12) - AIC:873.1687054186847
SARIMA(2, 1, 0)x(0, 0, 0, 12) - AIC:1259.7833248707425
SARIMA(2, 1, 0)x(0, 0, 1, 12) - AIC:1110.474174173972
```

```
SARIMA(2, 1, 0)x(0, 0, 2, 12) - AIC:938.0326522143368
SARIMA(2, 1, 0)x(1, 0, 0, 12) - AIC:1081.4099214593275
SARIMA(2, 1, 0)x(1, 0, 1, 12) - AIC:1057.6713554217959
SARIMA(2, 1, 0)x(1, 0, 2, 12) - AIC:932.3204745301107
SARIMA(2, 1, 0)x(2, 0, 0, 12) - AIC:905.5948860010933
SARIMA(2, 1, 0)x(2, 0, 1, 12) - AIC:907.2330623206876
SARIMA(2, 1, 0)x(2, 0, 2, 12) - AIC:909.1483851063575
SARIMA(2, 1, 1)x(0, 0, 0, 12) - AIC:1242.7200244071344
SARIMA(2, 1, 1)x(0, 0, 1, 12) - AIC:1075.9999714658704
SARIMA(2, 1, 1)x(0, 0, 2, 12) - AIC:904.009052225023
SARIMA(2, 1, 1)x(1, 0, 0, 12) - AIC:1054.365902101651
SARIMA(2, 1, 1)x(1, 0, 1, 12) - AIC:1034.3832672124836
SARIMA(2, 1, 1)x(1, 0, 2, 12) - AIC:899.8125434693609
SARIMA(2, 1, 1)x(2, 0, 0, 12) - AIC:879.7923634513132
SARIMA(2, 1, 1)x(2, 0, 1, 12) - AIC:881.2073386963306
SARIMA(2, 1, 1)x(2, 0, 2, 12) - AIC:882.9435022269579
SARIMA(2, 1, 2)x(0, 0, 0, 12) - AIC:1233.5045954992506
SARIMA(2, 1, 2)x(0, 0, 1, 12) - AIC:1065.6395070815809
SARIMA(2, 1, 2)x(0, 0, 2, 12) - AIC:897.3204257747524
SARIMA(2, 1, 2)x(1, 0, 0, 12) - AIC:1056.2515450532733
SARIMA(2, 1, 2)x(1, 0, 1, 12) - AIC:1033.413633688881
SARIMA(2, 1, 2)x(1, 0, 2, 12) - AIC:890.6376679226937
SARIMA(2, 1, 2)x(2, 0, 0, 12) - AIC:880.7638572027852
SARIMA(2, 1, 2)x(2, 0, 1, 12) - AIC:882.1078735147555
SARIMA(2, 1, 2)x(2, 0, 2, 12) - AIC:874.2139611370644
```

In [317…

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

Out[317…

	param	seasonal	AIC
107	(0, 1, 2)	(2, 0, 2, 12)	871.075238
26	(0, 1, 2)	(2, 0, 2, 12)	871.075238
134	(1, 1, 2)	(2, 0, 2, 12)	873.168705
53	(1, 1, 2)	(2, 0, 2, 12)	873.168705
80	(2, 1, 2)	(2, 0, 2, 12)	874.213961

In [318…

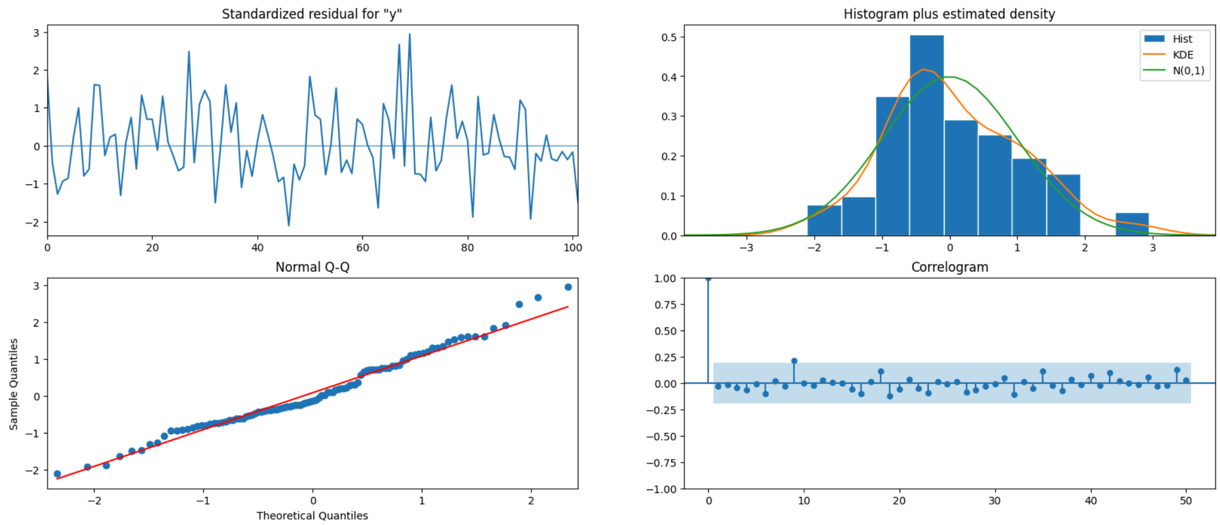
```
auto_SARIMA = sm.tsa.statespace.SARIMAX(train['Rose_Wine_Sales'].values,
                                order=(0, 1, 2),
                                seasonal_order=(2, 0, 2, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)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

```
                                      SARIMAX Results
==========================================================================================
======
Dep. Variable:                                  y   No. Observations:                      
130
Model:             SARIMAX(0, 1, 2)x(2, 0, 2, 12)   Log Likelihood                        -4
28.538
Date:                            Sun, 17 Mar 2024   AIC                                    8
71.075
Time:                                    13:09:23   BIC                                    8
89.450
Sample:                                         0   HQIC                                   8
78.516
                                            - 130
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.8367    239.178     -0.003      0.997    -469.617     467.943
ma.L2         -0.1633     39.038     -0.004      0.997     -76.677      76.350
ar.S.L12       0.3494      0.079      4.408      0.000       0.194       0.505
ar.S.L24       0.3067      0.075      4.103      0.000       0.160       0.453
ma.S.L12       0.0454      0.134      0.338      0.735      -0.218       0.309
ma.S.L24      -0.0912      0.145     -0.628      0.530      -0.376       0.193
sigma2       250.7786      6e+04      0.004      0.997   -1.17e+05    1.18e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.09   Jarque-Bera (JB):                 3.10
Prob(Q):                              0.76   Prob(JB):                         0.21
Heteroskedasticity (H):               0.88   Skew:                             0.43
Prob(H) (two-sided):                  0.71   Kurtosis:                         3.05
===================================================================================

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

Plot ACF and PACF for residuals of SARIMA

In [330…

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



Model Evaluation

In [319…

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

In [320…

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

```
For order=(0,1,2),seasonal_order=(2, 0, 2, 12) Auto SARIMA Model forecast on the Tes
t Data, RMSE is 25.364
```

In [321…

```
resultsDf_12 = pd.DataFrame({'Test RMSE': [rmse_sarima]},index=['order=(0,1,2),seas

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

Out[321...

	Test RMSE
Linear Regression	17.356924
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032
Alpha=0.127,Simple Exponential Smoothing	29.243074
Alpha=1.0,Simple Exponential Smoothing	21.782820
Alpha=0.151,Beta=0.151 Double Exponential Smoothing	26.032338
Alpha=0.3,Beta=0.4,Double Exponential Smoothing	18.337178
Alpha=0.099,Beta=1.993,Gamma=0.0001 Triple Exponential Smoothing	9.337258
Alpha=0.3,Beta=0.7,Gamma=0.4,Triple Exponential Smoothing	28.341124
order=(0,1,0) ARIMA	21.782820
order=(0,1,2),seasonal_order=(2, 0, 2, 12) SARIMA	25.363821

Compare the performance of the models

In [322...

```
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[322...

	Test RMSE
Alpha=0.099,Beta=1.993,Gamma=0.0001 Triple Exponential Smoothing	9.337258
2 point Trailing Moving Average	11.801167
4 point Trailing Moving Average	15.370676
6 point Trailing Moving Average	15.867384
9 point Trailing Moving Average	16.345032
Linear Regression	17.356924
Alpha=0.3,Beta=0.4,Double Exponential Smoothing	18.337178
Alpha=1.0,Simple Exponential Smoothing	21.782820
order=(0,1,0) ARIMA	21.782820
order=(0,1,2),seasonal_order=(2, 0, 2, 12) SARIMA	25.363821
Alpha=0.151,Beta=0.151 Double Exponential Smoothing	26.032338
Alpha=0.3,Beta=0.7,Gamma=0.4,Triple Exponential Smoothing	28.341124
Alpha=0.127,Simple Exponential Smoothing	29.243074

Building the most optimum model on the full dataset

Triple Exponential Smoothing (Holt - Winter's Model)

In [323...

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

In [324...

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

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

```
RMSE of most optimum model on the full dataset: 16.107657115519423
```

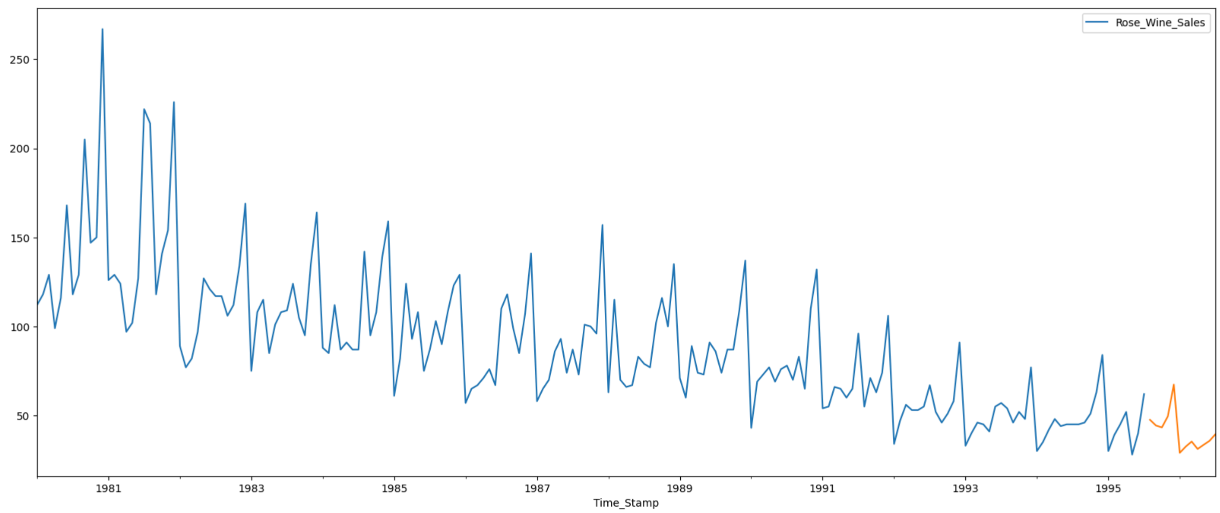
Forecast for the next 12 months

In [325...

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

In [326...

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



In [327…

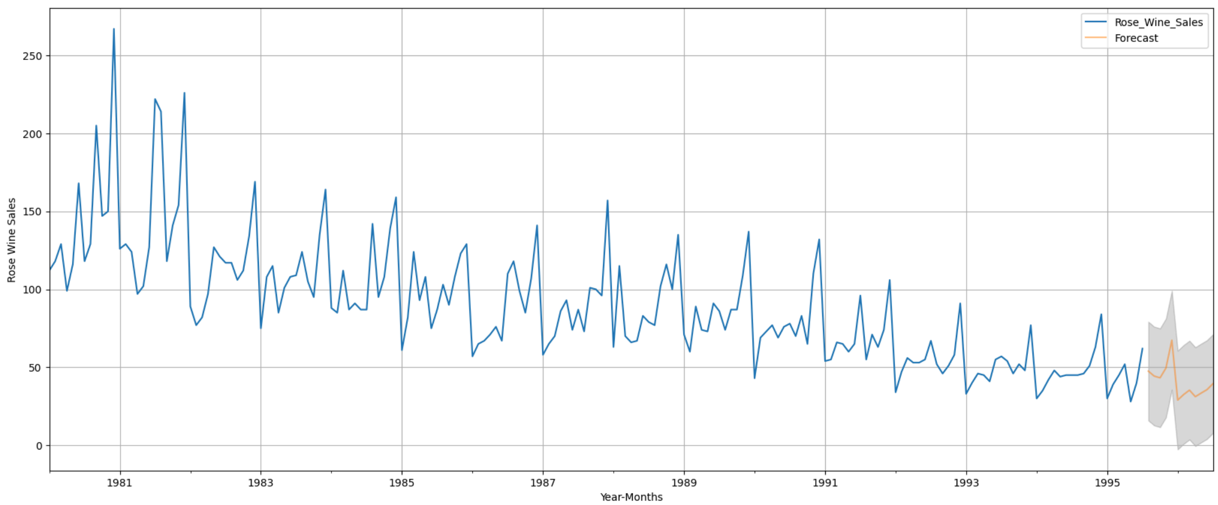
```
# Calculating the upper and lower confidence bands at 95% confidence level

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[327…

	lower_CI	prediction	upper_ci
1995-08-01	15.896024	47.551785	79.207546
1995-09-01	12.695793	44.351554	76.007316
1995-10-01	11.615091	43.270852	74.926614
1995-11-01	17.888679	49.544440	81.200201
1995-12-01	35.701828	67.357590	99.013351

In [328…

```
# 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('Rose Wine Sales')
plt.legend(loc='best')
plt.grid()
plt.show()
```



Rose_Wine_Sales
Forecast
Rose Wine Sales
Year-Months
0
50
100
150
200
250
1981
1983
1985
1987
1989
1991
1993
1995

In [ ]: