TIME SERIES FORECASTING

Group Assignment

(Sparkling Wine dataset)

Lavanya

Nikhil

Rekha

Rajiv

Submission Dt:

June 28th,2021

About Us

This section will cover brief introduction about us. We are seasoned professionals with diverse experience.

Rajiv

Am a CA and currently working as **Vice President & Group Head of Internal Audits** for Flipkart, Myntra, PhonePe, e-Kart and Walmart India. Prior to that I have worked in different audit and governance roles in PwC, Coke and Diageo.

Rekha

Overall **20+ years** of corporate and management experience. Currently (from 2018) working as **Strategy Consultant (Independently)** for firms and companies that range from hospitality, manufacturing, and retail industries. Before moving to an independent role was with **KPMG Global Services for 12 years**. Hands-on manager with expertise in accounting systems development, fiscal management. and financial reporting. Proven record of developing and implementing financial and operational controls that improve P&L scenario and competitively position firms.

Nikhil

Overall **13+ years of cross functional experience** in FMCG and Alcobev industry. Working with **Diageo PLC** since 2017, looking after **Data Analytics CoE for Global Audit & Risk department**. Before to that worked with **Coca-Cola for 10 years** and completed stint in operations, supply chain (direct & indirect), Master data maintenance (SAP cross functional role) and Data analytics (Internal Audit).

Lavanya

Working with **Novo Nordisk** as a Manager, having **13 years of techno-functional experience** in SAP Fiori, Embedded Analytics,SCP, ABAP on HANA and also having SAP Consulting experience on UX design strategy, RFP's/Pursuits and solutioning. During this tenure, worked in **various SAP ERP implementation projects spanning across Manufacturing, Healthcare, Retail and FMCG domains**. Prior to this, I have worked with **Accenture, Caterpillar and Wipro**.

Table of Contents

Problem Statement: 4

1 Import relevant libraries, read the data as an appropriate Time Series data and plot the data 4

Read the dataset 4

Plot the timeseries 4

2 Perform appropriate Exploratory Data Analysis to understand the data and also perform decomposition. **4**

Missing data check 5

Basic measure of Descriptive Statistics 5

Plot a year on year boxplot 5

Month wise plots 6

Lag Plot 6

Decompose the time series 7

3 Split the data into training and test. The test data should start in 1991 .... **9**

4 Model Building using various Time series techniques **9**

Model1: Regression on Time 10

Model2: Regression on Time With Seasonal Components 10

Model3: Naive Approach: $\hat{y}\ t{+}1{=}yt$ 12

Model 4: Simple Average 12

Model 5: Moving Average (MA) 13

Model 6: Simple Exponential Smoothing (SES) 15

Model 7: Simple Exponential Smoothing (SES) with least RMSE 16

Model 8: Holt's Linear Trend Method (Double Exponential Smoothing) 17

Model 9: DES model - Best alpha and beta with least RMSE 18

Model 10: Holt-Winters Method - Additive seasonality 18

Model 11: Holt-Winters Method - Multiplicative seasonality 20

5 Check for the stationarity of the data **20**

ADF Test (Full data and Train data) 21

Convert non-stationary to stationary (Full data and Train data) 22

6 Build Automated version of ARIMA & SARIMA model with least AIC value **22**

Model12: Auto ARIMA model – least AIC value 22

Model 13: Auto SARIMA model – least AIC value 24

7 Build ARIMA & SARIMA models based on the cut-off points of ACF & PACF **26**

Model 14: Manual ARIMA model – using ACF and PACF value 26

Model 15: Manual SARIMA model – using ACF and PACF value 27

8 Build a table with all the models 29

9 Run the best model on full dataset & predict 12 months into the future with appropriate CI (confidence intervals) 30

Run best model on full dataset 30

Predict 12 months into the future with appropriate CI 31

10 Comment on the model thus built and report your findings and suggest the measures that the company should be taking for future sales 32

Key Findings 32

Recommendations 33

Problem Statement:

For this particular assignment, the data of different types of wine sales in the 20th century is to be analysed. Both of these data are from the same company but of different wines. As an analyst in the ABC Estate Wines, you are tasked to analyse and forecast Wine Sales in the 20th century.

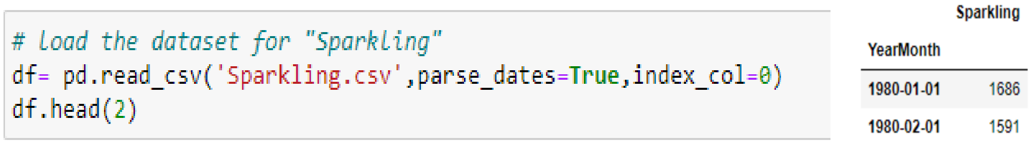
Data set for the Problem: **Sparkling.csv**

Remarks: All coding performed in Jupyter notebook, named **"Time Series_GA_Grp1_Sparkling" for sparkling**, has been duly attached.

1 Import relevant libraries, read the data as an appropriate Time Series data and plot the data

We imported all necessary libraries before we load the dataset.

Read the dataset

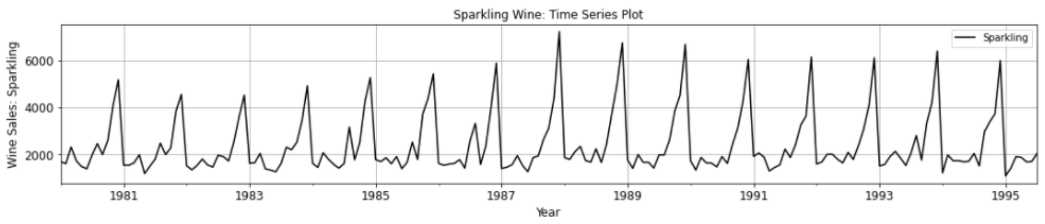


```
# Load the dataset for "Sparkling"
df= pd.read_csv('Sparkling.csv',parse_dates=True,index_col=0)
df.head(2)
```

	Sparkling
YearMonth	
1980-01-01	1686
1980-02-01	1591

Using pandas library we have read the '.csv' file, as a time-stamped file by enabling the parameter 'parse_dates'. Further, time Series in a lot of cases is easier to work if the index is Time stamped. Thus, we are making the zeroth column as index and making 'index_col' is ZERO.

Plot the timeseries



Insights: By looking at the graph, it seems:

- ❖ There are no null values in the data. We will double check the same during EDA.
- ❖ There is seasonality in the data.

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

EDA is performed to understand the data first and try to gather as many insights from it. EDA is a critical process for performing initial investigations on data so as to discover patterns, to spot anomalies and to check assumptions with the help of summary statistics and graphical representations.

Below is the Exploratory Data Analysis for the data sets of our problem statement.

Missing data check

```
df.isnull().sum()
```

```
Sparkling    0
dtype: int64
```

Insights: There are no null values in the dataset.

Basic measure of Descriptive Statistics

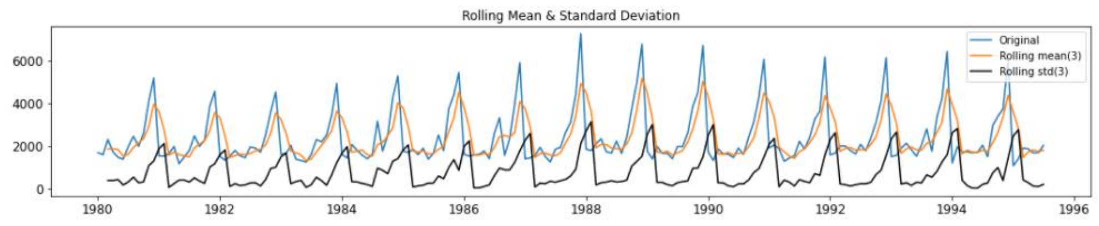
- We have used '**describe'** function to see the descriptive summary of the dataset.
- Further, we have checked **skewness** of the dataset.
- To check **mean and std deviation** across time we have checked rolling mean and rolling std dev. Using rolling months as 3.

```
df.describe().T
```

	count	mean	std	min	25%	50%	75%	max
Sparkling	187.0	2402.417112	1295.11154	1070.0	1605.0	1874.0	2549.0	7242.0

```
df.skew()
```

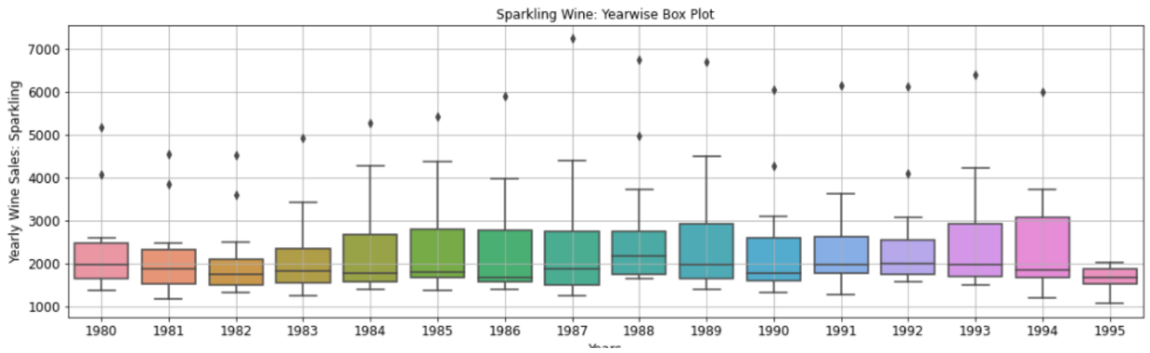
```
Sparkling    1.817612
dtype: float64
```



Insights:

- **Mean value is higher than the median value**. It means, **data is skewed**.
- We could see **huge variation/deviation across the years**.

Plot a year on year boxplot

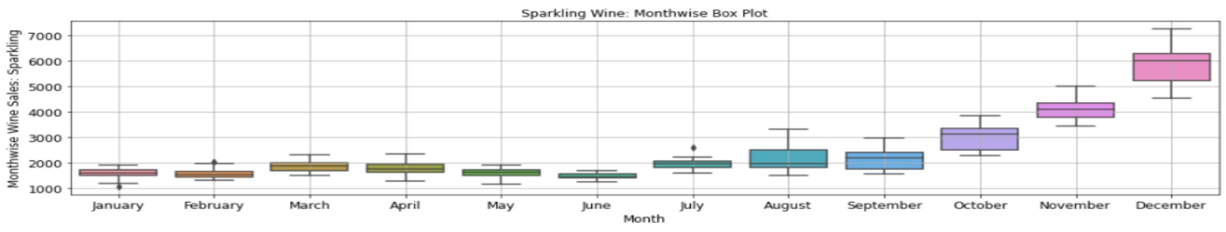


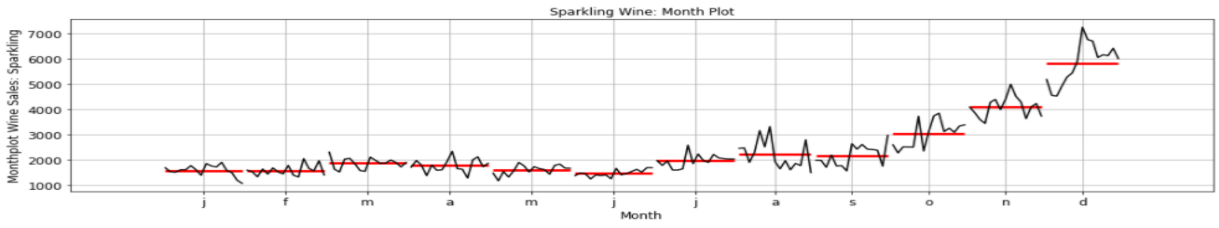
A boxplot for different years aggregating over different months gives us an indication of the TREND.

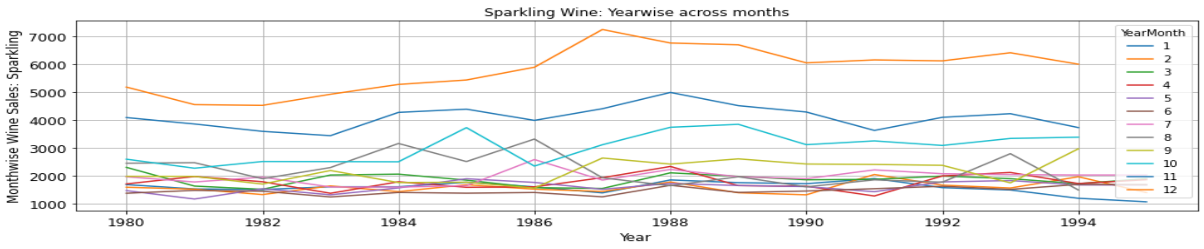
Insights:

- Trend over years seems almost flat.
- Median sales value of each year is almost same. Further, Year 1987 and 1988 are the highest sales years amongst other years.

Month wise plots







A boxplot for different months aggregating over different years, gives us an indication of SEASONALITY. All three plots are the different way of looking at the month wise sales across the years.

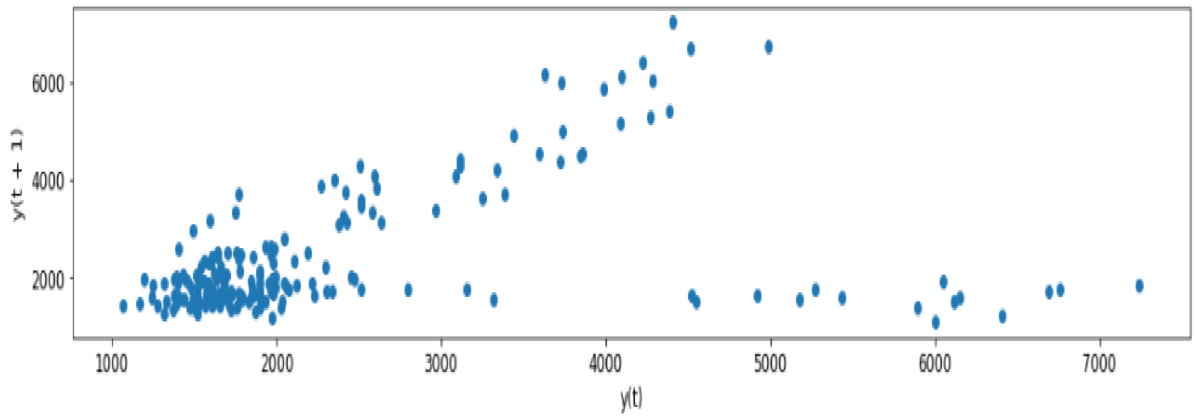
Insights:

- All 3 plots show that the sparkling wine is **seasonal towards 2nd half of the year,** and
- The **demand peaks in December** of every year.

Lag Plot

If the points cluster along a diagonal line from the bottom-left to the top-right of the plot, it suggests a positive correlation relationship. If the points cluster along a diagonal line from the top-left to the bottom-right, it suggests a negative correlation relationship.

More points tighter in to the diagonal line suggests a stronger relationship and more spread from the line suggests a weaker relationship. A ball in the middle or a spread across the plot suggests weak or no relationship.



Insights:

- Since the points are clustering along a diagonal line from the bottom-left to the top-right of the plot, it suggests a **positive correlation** relationship.
- Besides, points being more spread from the line are suggesting a **weaker relationship** between different periods/months.

Decompose the time series

Time series decomposition involves thinking of a series as a combination of level, trend, seasonality, and noise components.

Decomposition provides a useful abstract model for thinking about time series generally and for better understanding problems during time series analysis and forecasting.

Time Series Components:

A useful abstraction for selecting forecasting methods is to break a time series down into systematic and unsystematic components.

Systematic: Components of the time series that have consistency or recurrence and can be described and modelled.
Non-Systematic: Components of the time series that cannot be directly modelled.

A given time series is thought to consist of three systematic components including level, trend, seasonality, and one non-systematic component called noise.

Component	Description
Level	The average value in the series
Trend	The increasing or decreasing value in the series
Seasonality	The repeating short-term cycle in the series
Noise	The random variation in the series

There are mainly two types of time series decomposition i) Additive and ii) Multiplicative

Additive Decomposition:

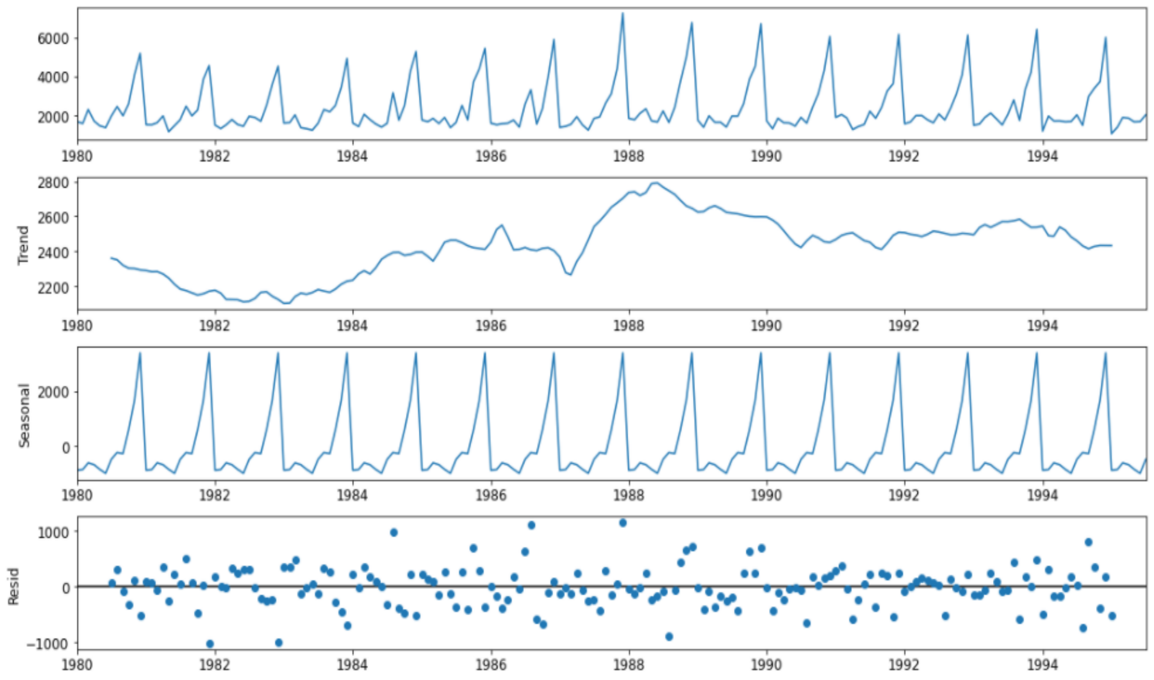
- An additive model suggests that the components are added together.
- An additive model is linear where changes over time are consistently made by the same amount.
- A linear seasonality has the same frequency (width of the cycles) and amplitude (height of the cycles).

Multiplicative Decomposition:

- An additive model suggests that the components are multipled together.
- An additive model is non-linear such as quadratic or exponential.
- Changes increase or decrease over time.
- A non-linear seasonality has an increasing or decreasing frequency (width of the cycles) and / or amplitude (height of the cycles) over time.

Let’s decompose the time series and inference the same.

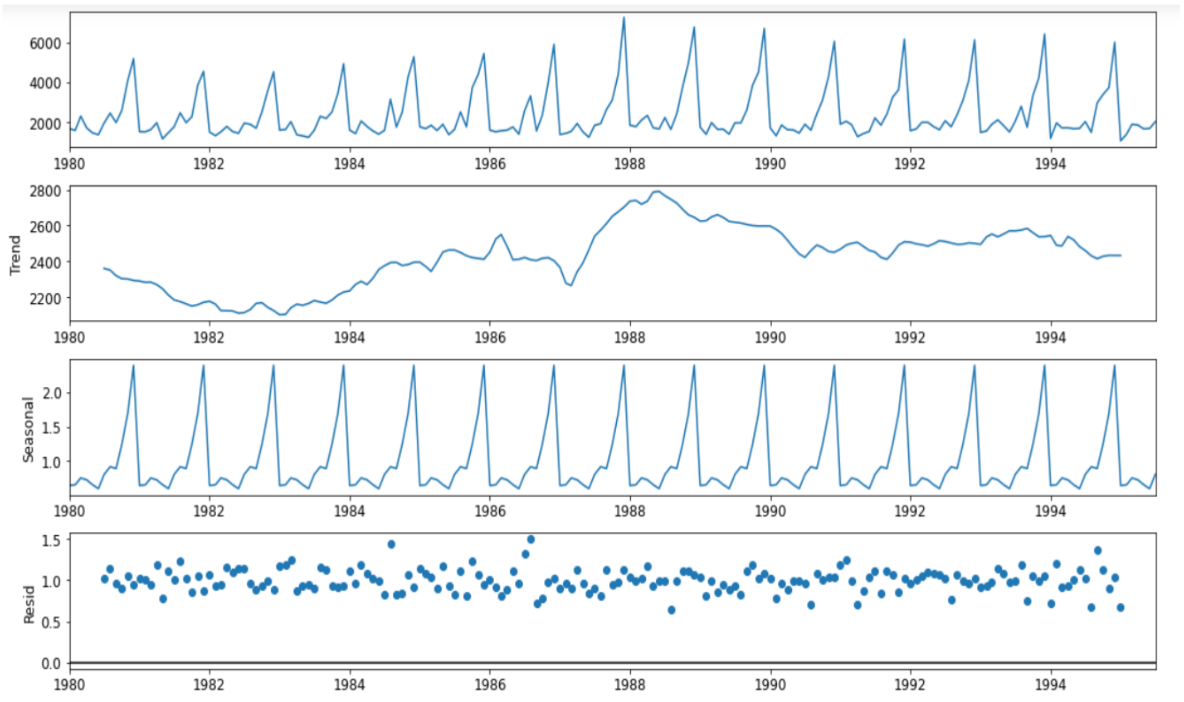
Additive Decomposition:



Insights: In this additive decomposition, we could see that:

- Trend value is moving from 2400 to 2200 to 2800 and back to 2400. so eventually no clear trend coming out.
- Seasonality, out of mean of 2400, around 2200 is the seasonality which means, this dataset has the high seasonality.
- Residuals, the range of the residual is also around +/- 1000 (which leads to noise).

Multiplicative Decomposition:



Insights: In this multiplicative decomposition, we could see that:

- Trend value is moving from 2400 to 2200 to 2800 and back to 2400. so eventually no clear trend coming out.
- Seasonality is as high as 200%, as well.
- Residuals are from 0.5 times to 1.5 times (which leads to noise).

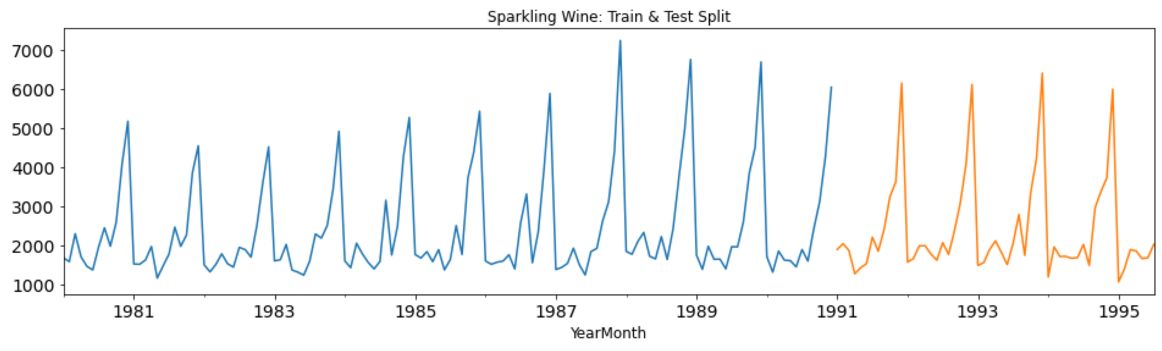
3 Split the data into training and test. The test data should start in 1991

Let's split the dataset into train and test. There are total 185 months data available. Mentioned below is the conversion of the same.

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

```
Train Dataset: Sparkling
            Sparkling
YearMonth
1980-01-01       1686
1980-02-01       1591
            Sparkling
YearMonth
1990-11-01       4286
1990-12-01       6047


Train Dataset Shape: (132, 1)
```

```
Test Dataset: Sparkling
            Sparkling
YearMonth
1991-01-01       1902
1991-02-01       2049
            Sparkling
YearMonth
1995-06-01       1688
1995-07-01       2031


Test Dataset Shape: (55, 1)
```



- After splitting data into train and test we have 132 months details in train dataset and 55 months details (i.e., from year 1991) in test dataset.
- Further, time series plot also indicate trains and test split.

4 Model Building using various Time series techniques

As a part of this exercise, we have tried to build as many models which we learned during our time series forecasting classes.

The common rule for each model:

- We will build the model and fit the model on train data.
- Predict the values on test data.
- Further, on test data we will be checking the performance using RMSE parameter.

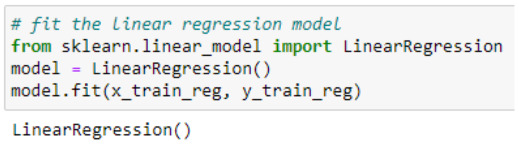
Let's go ahead and build models one by one.

Model1: Regression on Time

For this model, we are going to regress the 'Sparkling sales' variable against the order of the occurrence(monthly).

Let’s follow below steps:

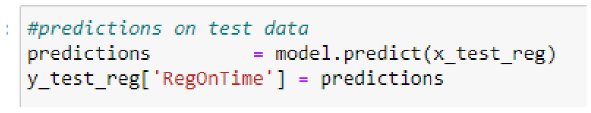
- Create the train and test data copy to use for this model
- Create time variable for train data and test data which would be useful for this model
- Train and test split (x_train, x_test, y_train, y_test) based on times series variable
- Import the linear regression library and fit the model in train data.



```
# fit the linear regression model
from sklearn.linear_model import LinearRegression
model = LinearRegression()
model.fit(x_train_reg, y_train_reg)
```

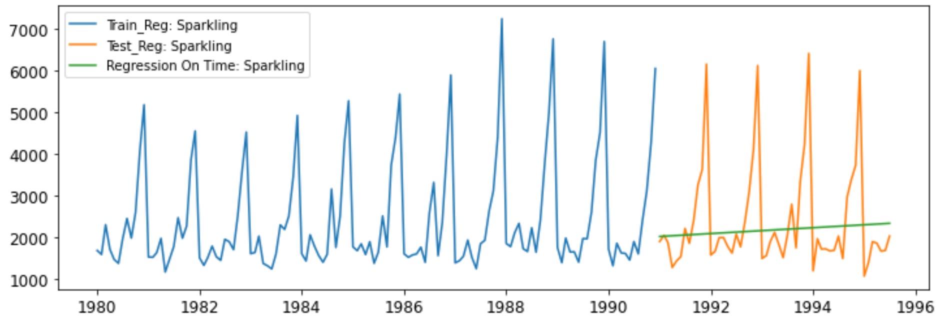
```
LinearRegression()
```

- Predict the values on test data

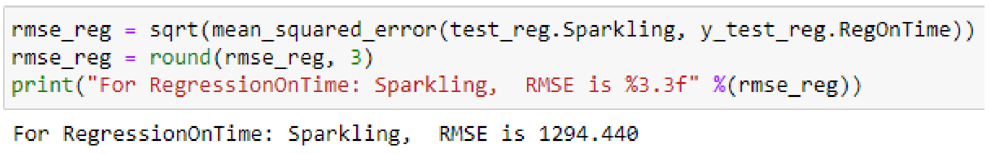


```
#predictions on test data
predictions            = model.predict(x_test_reg)
y_test_reg['RegOnTime'] = predictions
```

- Plot the view of train, test data with predictions



- Calculate RMSE value on test data



```
rmse_reg = sqrt(mean_squared_error(test_reg.Sparkling, y_test_reg.RegOnTime))
rmse_reg = round(rmse_reg, 3)
print("For RegressionOnTime: Sparkling,  RMSE is %3.3f" %(rmse_reg))
```

```
For RegressionOnTime: Sparkling,  RMSE is 1294.440
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

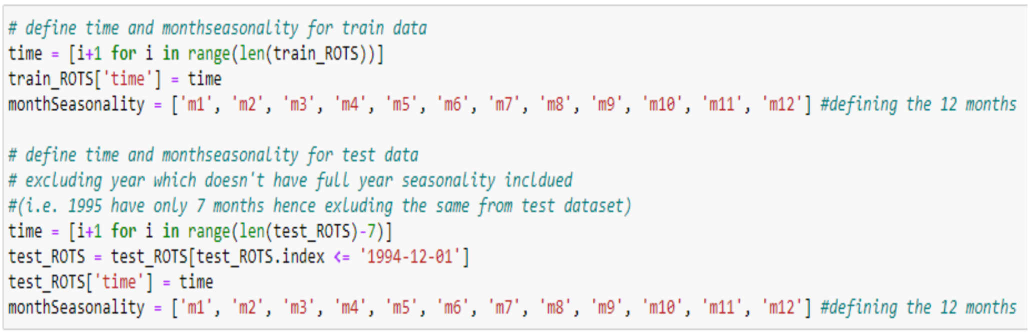
Model2: Regression on Time With Seasonal Components

In this model, we are adding seasonal component in regression on time and check the test RMSE value.

Steps:

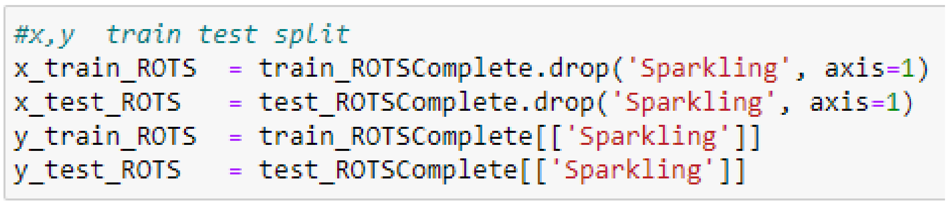
- Create the train and test data copy to use for this model

- Create **time variable and month seasonality (m1 to m12)** for train data and test data which would be useful for this model. This divides the dataset into m1 to m12 and repeats the pattern.



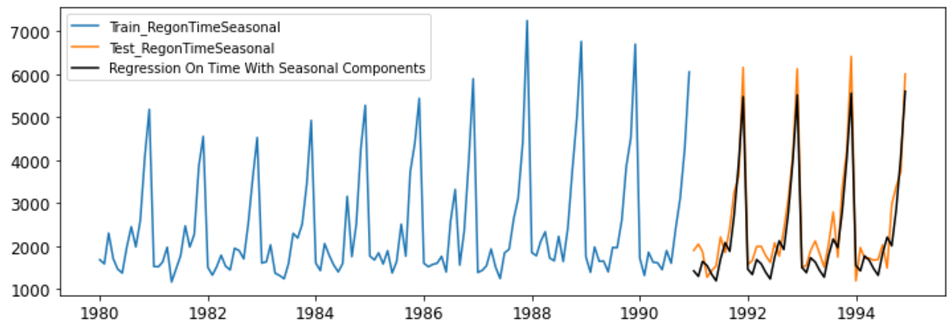
```
# define time and monthseasonality for train data
time = [i+1 for i in range(len(train_ROTS))]
train_ROTS['time'] = time
monthSeasonality = ['m1', 'm2', 'm3', 'm4', 'm5', 'm6', 'm7', 'm8', 'm9', 'm10', 'm11', 'm12'] #defining the 12 months

# define time and monthseasonality for test data
# excluding year which doesn't have full year seasonality incldued
#(i.e. 1995 have only 7 months hence exluding the same from test dataset)
time = [i+1 for i in range(len(test_ROTS)-7)]
test_ROTS = test_ROTS[test_ROTS.index <= '1994-12-01']
test_ROTS['time'] = time
monthSeasonality = ['m1', 'm2', 'm3', 'm4', 'm5', 'm6', 'm7', 'm8', 'm9', 'm10', 'm11', 'm12'] #defining the 12 months
```

- Use "get_dummies" function to convert each m1 to m12 into columns for train and test data
- Train and test split (x_train, x_test, y_train, y_test) based on times series variable

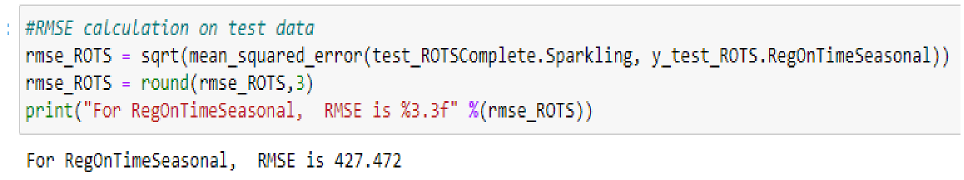


```
#x,y  train test split
x_train_ROTS  = train_ROTSComplete.drop('Sparkling', axis=1)
x_test_ROTS   = test_ROTSComplete.drop('Sparkling', axis=1)
y_train_ROTS  = train_ROTSComplete[['Sparkling']]
y_test_ROTS   = test_ROTSComplete[['Sparkling']]
```

- fit the linear regression model in train data.
- Predict the values on test data
- Plot the view of train, test data with predictions



- Calculate RMSE value on test data



```
#RMSE calculation on test data
rmse_ROTS = sqrt(mean_squared_error(test_ROTSComplete.Sparkling, y_test_ROTS.RegOnTimeSeasonal))
rmse_ROTS = round(rmse_ROTS,3)
print("For RegOnTimeSeasonal,  RMSE is %3.3f" %(rmse_ROTS))
```

```
For RegOnTimeSeasonal,  RMSE is 427.472
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

Model3: Naive Approach: $\hat{y}\ t+1=yt$

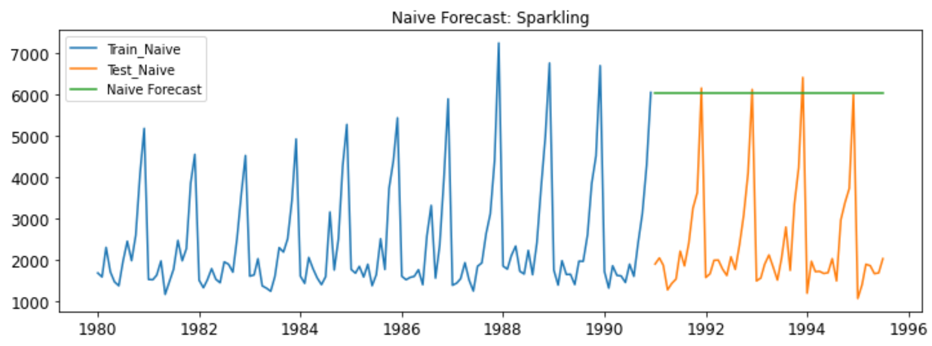
A naive forecast involves using the previous observation directly as the forecast without any change. It is often called the persistence forecast as the prior observation is persisted. This simple approach can be adjusted slightly for seasonal data.

Steps:

- Create the train and test data copy to use for this model
- Create naïve model using previous observation

```
y_hat_s['naive'] = dd_s[len(dd_s)-1]
```

- Plot the view of train, test data with naive



- Calculate RMSE value on test data

```
# RMSE calculation on test data
rmse_Naive = sqrt(mean_squared_error(test.Sparkling, y_hat_s.naive))
rmse_Naive = round(rmse_Naive, 3)
print("For Naive model,  RMSE is %3.3f" %(rmse_Naive))
```

```
For Naive model,  RMSE is 3864.279
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

Model 4: Simple Average

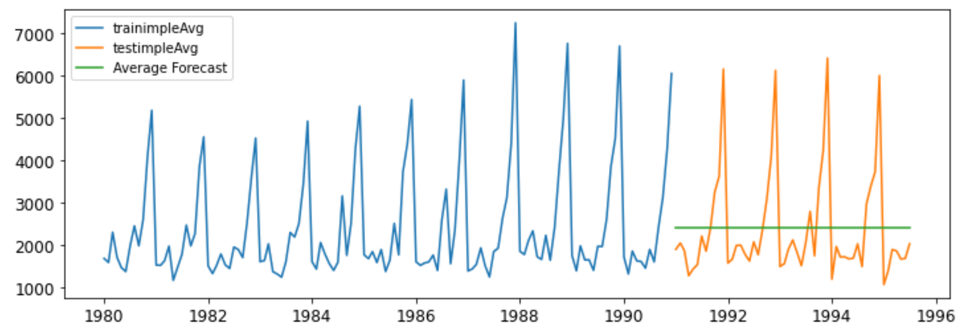
The model is very simple. Average the data by months and calculate the average for the period. Then plot the same so see the variation.

Steps:

- Create the train and test data copy to use for this model
- Create time variable **average mean** for train data and test data which would be useful for this model

```
#create varible with putting train data mean
y_hat_avg_s['avg_forecast'] = train['Sparkling'].mean()
```

- Plot the view of train, test data with simple mean details



- Calculate RMSE value on test data

```
#RMSE calculation on test data
rmse_SA = sqrt(mean_squared_error(test.Sparkling, y_hat_avg_s.avg_forecast))
rmse_SA = round(rmse_SA, 3)
print("For Simple Average model,  RMSE is %3.3f" %(rmse_SA))
```

```
For Simple Average model,  RMSE is 1275.082
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

Model 5: Moving Average (MA)

In the method of moving average, successive arithmetic averages are computed from overlapping groups of successive values of a time series. Each group includes all the observations in a given time interval, termed as the period of moving average.

In this dataset we are going to take rolling mean for 3 months, 4 months, 5 months, 6 months, 8 months and 12 months and will see at which rolling mean we are getting better results.

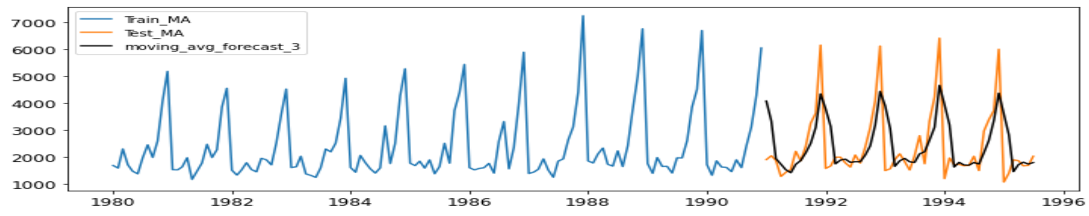
Steps:

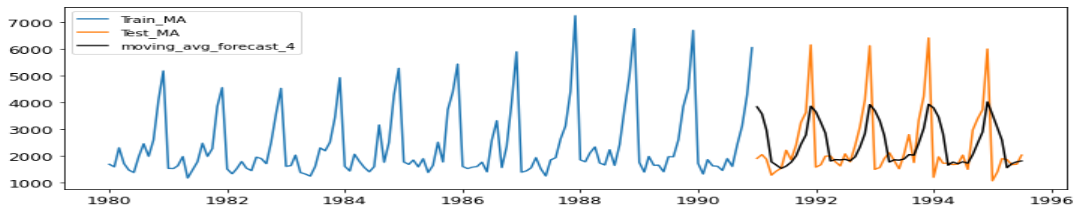
- Create the copy of full dataset to use for this model
- Create moving averages (rolling mean) for said moths as above.

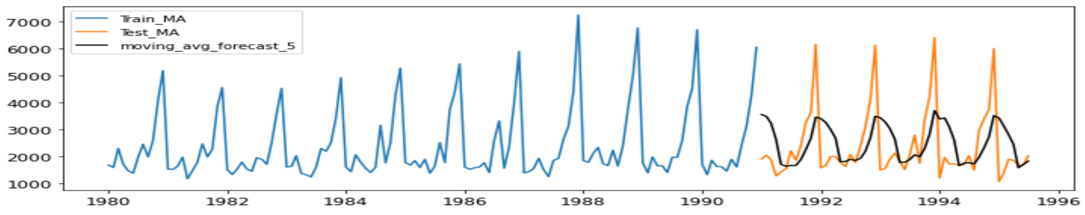
```
# rolling MA details
df_MA['moving_avg_forecast_3']  = df_MA['Sparkling'].rolling(3).mean()
df_MA['moving_avg_forecast_4']  = df_MA['Sparkling'].rolling(4).mean()
df_MA['moving_avg_forecast_5']  = df_MA['Sparkling'].rolling(5).mean()
df_MA['moving_avg_forecast_6']  = df_MA['Sparkling'].rolling(6).mean()
df_MA['moving_avg_forecast_8']  = df_MA['Sparkling'].rolling(8).mean()
df_MA['moving_avg_forecast_12'] = df_MA['Sparkling'].rolling(12).mean()
```

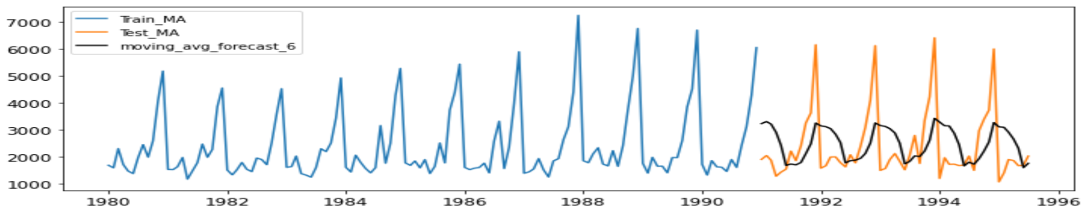
- Create the columns related to same defined moving average months
- Split the dataset into train and test

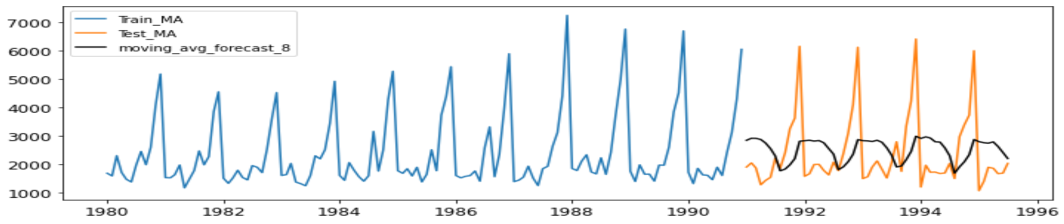
- Plot the view of train, test data with each rolling means

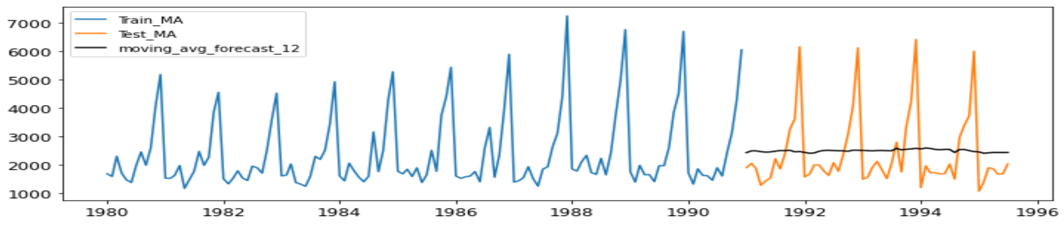












We could see here, as the rolling month increases accuracy is dropping off.

- Calculate RMSE value on test data

```
For Moving Average model, moving_avg_forecast_3  RMSE is 1028.606
For Moving Average model, moving_avg_forecast_4  RMSE is 1156.590
For Moving Average model, moving_avg_forecast_5  RMSE is 1234.045
For Moving Average model, moving_avg_forecast_6  RMSE is 1283.927
For Moving Average model, moving_avg_forecast_8  RMSE is 1342.568
For Moving Average model, moving_avg_forecast_12  RMSE is 1267.925
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

Model 6: Simple Exponential Smoothing (SES)

Single Exponential Smoothing (SES), is a time series forecasting method for univariate data without a trend or seasonality. It requires a single parameter called alpha (a), also called the smoothing factor or smoothing coefficient. This parameter controls the rate at which the influence of the observations at prior time steps decay exponentially. **Alpha is often set to a value between 0 and 1 and considered as Smoothing factor for the level**

A value close to 1 indicates fast learning (that is, only the most recent values influence the forecasts), whereas a value close to 0 indicates slow learning (past observations have a large influence on forecasts).

Let's build the model using below steps:

- Import SimpleExpSmoothing library from statsmodels.tsa.api
- Fit the model using train data using optimised equals to True.

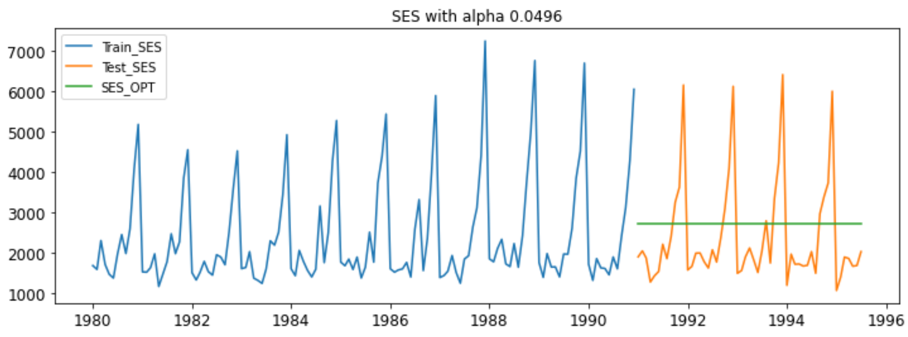
```
# create class
model_SES = SimpleExpSmoothing(train['Sparkling'])
```

```
#fit the model
model_SES_fit = model_SES.fit(optimized = True)
```

- Print the SES parameter and its initial level

```
== Simple Exponential Smoothing (SES)


Smoothing Level 0.0496
Initial Level 1818.5048
```

- Create the copy of test dataset before we do prediction
- Do the prediction on test data
- Plot the view of train, test data with predictions



The alpha value is close to 0 indicates slow learning (past observations have a large influence on forecasts).

- Calculate RMSE value on test data

```
rmse_SES = sqrt(mean_squared_error(test['Sparkling'], y_hat_avg_SES.SES))
rmse_SES = round(rmse_SES, 3)
print("For SES model: For alpha = %1.2f,  RMSE is %3.3f" %(alpha_value,rmse_SES))
```

```
For SES model: For alpha = 0.05,  RMSE is 1316.035
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

Model 7: Simple Exponential Smoothing (SES) with least RMSE

Let's build the model using below steps:

- Create table which helps to store alpha and RMSE value
- Create the copy of test dataset for this model
- Create the for loop function using alpha value from 0.01 to 1 and interval is 0.01 and put all the below steps in the loop to get the alpha value which has least RMSE value.
  - Fit the model using train data using optimised equals to True.
  - Do the prediction on test data
  - Calculate RMSE value on test data
  - Store the alpha value and RMSE value and sort the same by least RMSE value

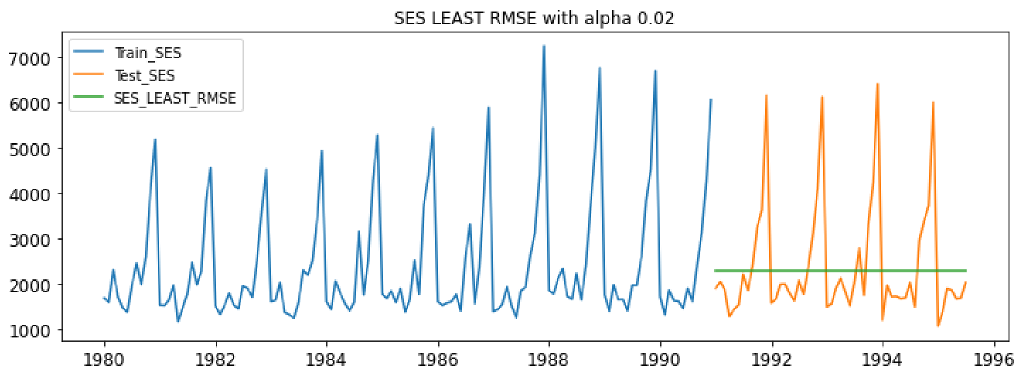
```
SES_least_rmse = resultsDf_model.sort_values(by=['Test RMSE'],ascending=True)
SES_least_rmse.head(2)
```

	Alpha Values	Test RMSE
1	0.02	1279.495201
0	0.01	1281.032699

```
print("For SES model(least RMSE): For alpha = %3.2f,  RMSE is %3.3f"
    %(SES_least_rmse.iloc[0].values[0],SES_least_rmse.iloc[0].values[1]))
```

```
For SES model(least RMSE): For alpha = 0.02,  RMSE is 1279.495
```

- Plot the view of train, test data with predictions



The alpha value is close to 0 indicates slow learning (past observations have a large influence on forecasts).

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

Model 8: Holt's Linear Trend Method (Double Exponential Smoothing)

Double Exponential Smoothing is an extension to Exponential Smoothing that explicitly adds support for trends in the univariate time series. In addition to the **alpha parameter for controlling smoothing factor for the level, an additional smoothing factor is added to control the decay of the influence of the change in trend called beta (b).**

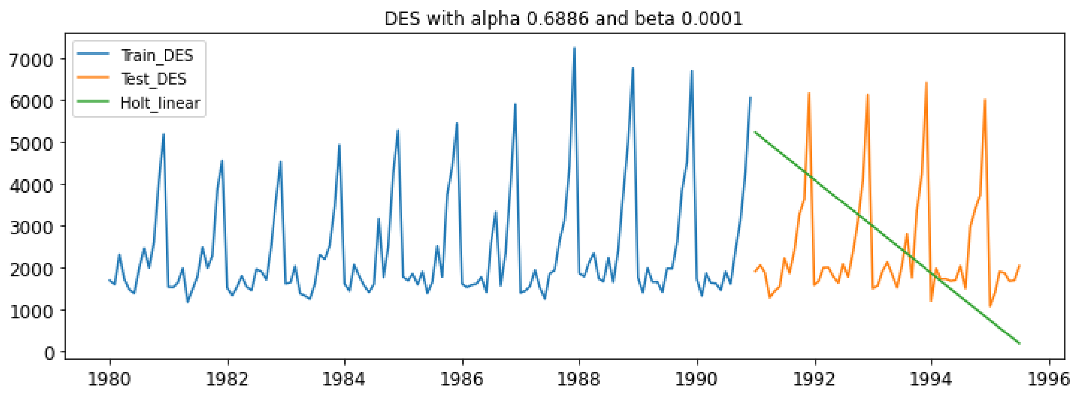
Double Exponential Smoothing with an additive trend is classically referred to as Holt's linear trend model, named for the developer of the method Charles Holt.

Let's build the model using below steps:

- Import Holt library from statsmodels.tsa.api
- Create the copy of test dataset
- Fit the model using train dataset.
- Do the prediction on test data
- Print the DES parameters and its initial level

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

Smoothing Level 0.6886
Smoothing Trend 0.0001
Initial Level 1686.0
```

- Plot the view of train, test data with predictions



The alpha value is close to 1 which indicates fast learning (that is, only the most recent values influence the forecasts) and beta value is close 0 which indicates slow learning (past observations have a large influence on forecasts).

- Calculate RMSE value on test data

```
rmse_DES          =  np.sqrt(mean_squared_error(test['Sparkling'], y_hat_avg_DES['Holt_linear']))

print("For alpha = %1.2f, For beta = %1.2f,  RMSE is %4.2f"
      %(alpha_DES_value, beta_DES_value, rmse_DES))
```

```
For alpha = 0.69, For beta = 0.00,  RMSE is 2007.24
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

Model 9: DES model - Best alpha and beta with least RMSE

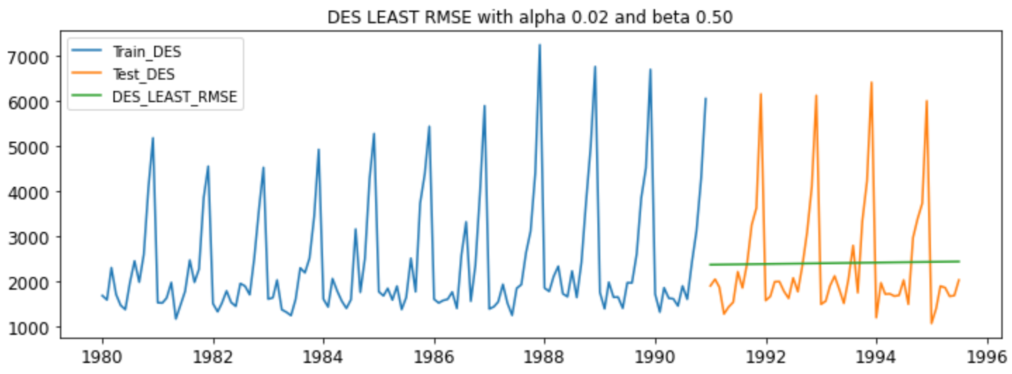
Let's build the model using below steps:

- Create table which helps to store alpha, beta and RMSE value
- Create the for loop function using alpha value from 0.01 to 1 and interval is 0.01 and put all the below steps in the loop to get the alpha value which has least RMSE value.
    - Fit the model using train data using optimised equals to false.
    - Do the prediction on test data
    - Calculate RMSE value on test data
    - Store the alpha, beta and RMSE value and sort the same by least RMSE value

```
DES_least_rmse = resultsDfDES_model.sort_values(by=['Test RMSE'],ascending=True)
DES_least_rmse.head(2)
```

	Alpha Values	Beta Values	Test RMSE
148	0.02	0.50	1274.630824
115	0.02	0.17	1275.105310

- Plot the view of train, test data with predictions



The alpha value is close 0 which indicates slow learning (past observations have a large influence on forecasts) and beta value is close to 1 which indicates fast learning (that is, only the most recent values influence the forecasts).

- Calculate RMSE value on test data

```
In [77]: print("For DES model(least RMSE): For alpha = %1.2f, beta = %1.2f, RMSE is %3.2f"
            %(DES_least_rmse.iloc[0].values[0],DES_least_rmse.iloc[0].values[1],DES_least_rmse.iloc[0].values[2]))

         For DES model(least RMSE): For alpha = 0.02, beta = 0.50, RMSE is 1274.63
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

Model 10: Holt-Winters Method - Additive seasonality

Triple Exponential Smoothing is an extension of Exponential Smoothing that explicitly adds support for seasonality to the univariate time series. This method is sometimes called Holt-Winters Exponential Smoothing, named for two contributors to the method: Charles Holt and Peter Winters. In addition to **the alpha and beta smoothing factors, a new parameter is added called gamma (g) that controls the influence on the seasonal component**.

As with the trend, the seasonality may be modeled as either an additive or multiplicative process for a linear or exponential change in the seasonality.

Additive Seasonality: Triple Exponential Smoothing with a linear seasonality.
Multiplicative Seasonality: Triple Exponential Smoothing with an exponential seasonality.

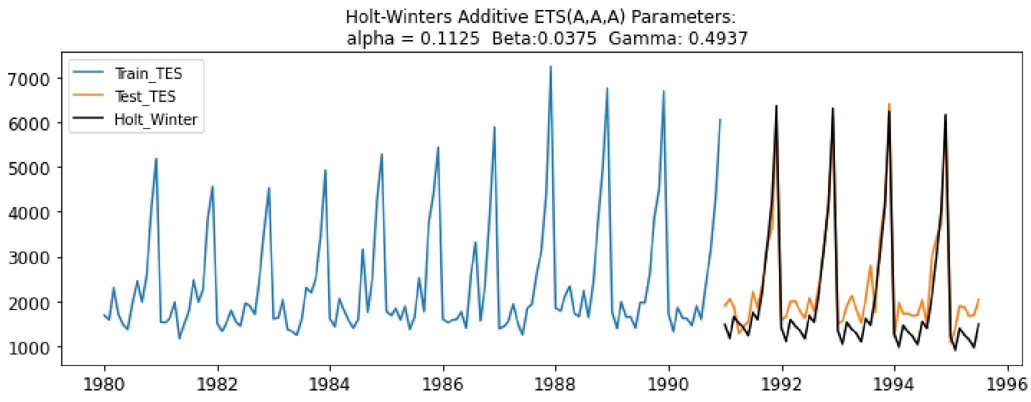
In summary, *Being an adaptive method, Holt-Winter's exponential smoothing allows the level, trend and seasonality patterns to change over time*

Let's build the model using below steps:

- Import ExponentialSmoothing library from statsmodels.tsa.api
- Create the copy of test dataset for this model
- Fit the model using train dataset using **hyperparameters as "seasonal_periods" = 12, trend = "additive", seasonal = "additive".**
- Do the prediction on test data
- Print the TES parameters and its initial level

```
== Holt-Winters Additive ETS(A,A,A) Parameters ==

Smoothing Level:  0.1125
Smoothing Trend:  0.0375
Smoothing Seasonal:  0.4937
Initial Level:  1640.1903
Initial Trend:  -2.8837
Initial Seasons:  [  45.9036   -48.989    662.9357    72.6896 -168.8846 -262.4524   326.0668
   813.2342   344.3313   956.0857 2446.8137 3538.46    ]
```

- Plot the view of train, test data with predictions



The alpha and beta value are close 0 which indicates slow learning (past observations have a large influence on forecasts). gamma value is close to 1 which indicates fast learning (that is, only the most recent values influence the forecasts)

- Calculate RMSE value on test data

```
rmse_TES_a              =  np.sqrt(mean_squared_error(test['Sparkling'], y_hat_TES_a_avg['Holt_Winter']))

print("For TES(Additive) model: alpha = %1.2f, beta = %1.2f, gamma = %1.2f, RMSE is %3.2f"
      %(alpha_TES_a_value, beta_TES_a_value, gamma_TES_a_value, rmse_TES_a))
```

```
For TES(Additive) model: alpha = 0.11, beta = 0.04, gamma = 0.49, RMSE is 473.95
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

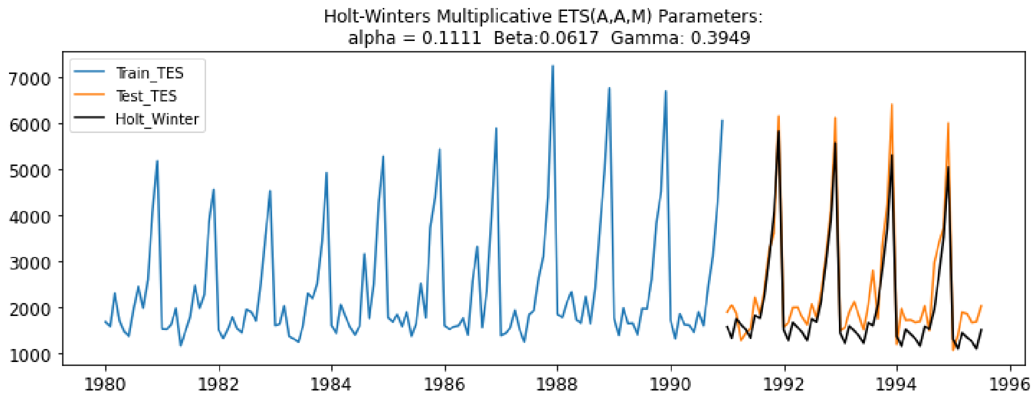
Model 11: Holt-Winters Method - Multiplicative seasonality

Let's build the model using below steps:

- Create the copy of test dataset for this model
- Fit the model using train dataset using **hyperparameters as "seasonal_periods" = 12, trend = "additive", seasonal = "multiplicative".**
- Do the prediction on test data
- Print the TES parameters and its initial level

```
== Holt-Winters Multiplicative ETS(A,A,M) Parameters ==

Smoothing Level:  0.1111
Smoothing Trend:  0.0617
Smoothing Seasonal:  0.3949
Initial Level:  1639.5306
Initial Trend:  -13.8037
Initial Seasons:  [1.0441 1.001  1.4046 1.2091 0.9641 0.9675 1.3048 1.6984 1.3703 1.8166
 2.8471 3.6246]
```

- Plot the view of train, test data with predictions



The alpha, beta and gamma value are close 0 which indicates slow learning (past observations have a large influence on forecasts).

- Calculate RMSE value on test data

```
rmse_TES_m              =  np.sqrt(mean_squared_error(test['Sparkling'], y_hat_TES_m_avg['Holt_Winter']))

print("For TES(Multiplicative) alpha = %1.2f, beta = %1.2f, gamma = %1.2f, RMSE is %3.2f"
      %(alpha_TES_m_value, beta_TES_m_value, gamma_TES_m_value, rmse_TES_m))
```

```
For TES(Multiplicative) alpha = 0.11, beta = 0.06, gamma = 0.39, RMSE is 469.43
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

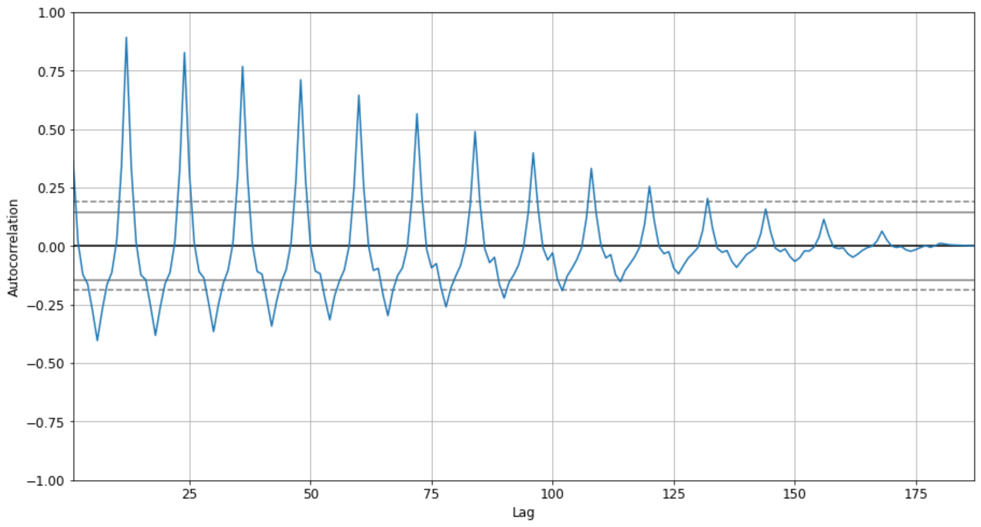
5 Check for the stationarity of the data

A stationary time series is one whose statistical properties **such as mean, variance, autocorrelation, etc. are all constant over time.** Such statistics are useful as descriptors of future behaviour only if the series is stationary. Mentioned below is the hypothesis to test the stationarity.

H0 : Time series is non-stationary and H1 : Time series is stationary
(if p value is less than 0.05 then the series is called stationary).

Note: If the series is non-stationary, stationarise the time series by taking a difference of the time series. Then we can use this particular differenced series to train the models. We do not need to worry about stationarity for the Test Data because we are not building any models on the test Data, we are only evaluating our models over there.

Auto correlation plot helps to see the same.



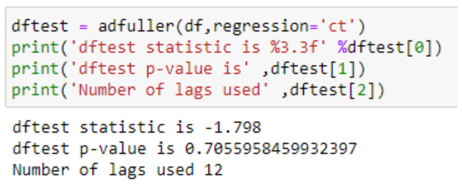
Insights:
We can quantify the strength and type of relationship between observations and their lags. In statistics, this is called correlation, and when calculated against lag values in time series it is called autocorrelation (self-correlation).

A correlation value calculated between two groups of numbers, such as observations and their lag1 values, results in a number between -1 and 1. The sign of this number indicates a negative or positive correlation respectively. A value close to zero suggests a weak correlation, whereas a value closer to -1 or 1 indicates a strong correlation. Dotted lines are provided that indicate any correlation values above those lines are statistically significant (meaningful).

Using ADF (Augmented Dickey–Fuller) test, we will be able to find out whether the time series is stationary or not. Let's run ADF test on full data and train dataset.

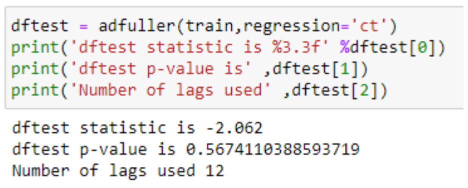
ADF Test (Full data and Train data)

- Import adfuller library from the statsmodels.tsa.stattool
- Run the adfuller test on full dataset and train dataset



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

```
dftest statistic is -1.798
dftest p-value is 0.7055958459932397
Number of lags used 12
```



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

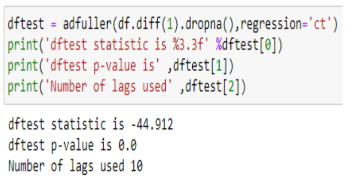
```
dftest statistic is -2.062
dftest p-value is 0.5674110388593719
Number of lags used 12
```

Both full data and train data p value is more then 0.05. hence, series in non-stationary.

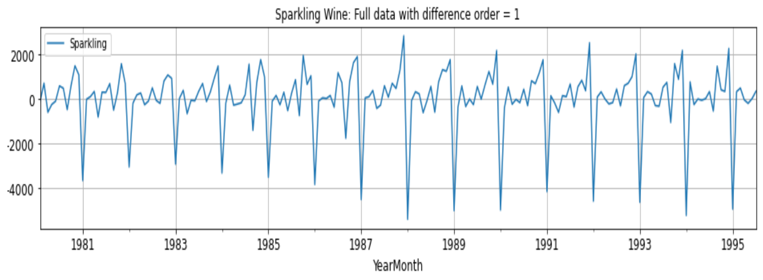
Convert non-stationary to stationary (Full data and Train data)

A non-stationary process with a deterministic trend becomes stationary after removing the trend, or detrending. Using differencing method we remove non-stationarity from the dataset.
As such, the process of differencing can be repeated more than once until all temporal dependence has been removed. The number of times that differencing is performed is called the difference order.

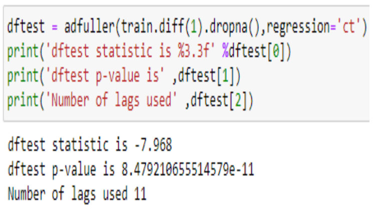
Full dataset after difference order (1) and p value less than 0.05 (stationary) series.



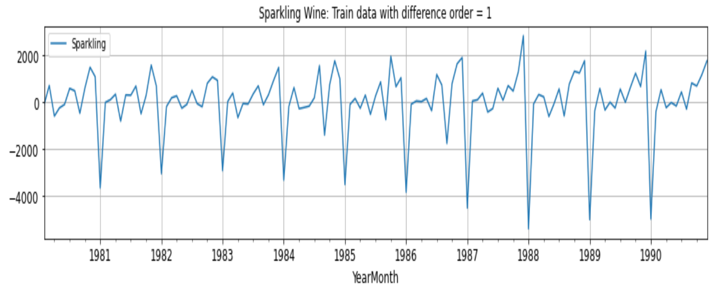
```
dftest = adfuller(df.diff(1).dropna(),regression='ct')
print('dftest statistic is %3.3f' %dftest[0])
print('dftest p-value is' ,dftest[1])
print('Number of lags used' ,dftest[2])

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



Train dataset after difference order (1) and p value less than 0.05 (stationary) series.



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

dftest statistic is -7.968
dftest p-value is 8.479210655514579e-11
Number of lags used 11
```



6 Build Automated version of ARIMA & SARIMA model with least AIC value

In point5, we have concluded that original series both (full and train) dataset were non-stationary and after doing difference order of (1) we were able to convert series into stationary.

While building ARIMA and SARIMA we will consider this point. Let's build these models.

Model12: Auto ARIMA model – least AIC value

An autoregressive integrated moving average (ARIMA), is a statistical analysis model that uses time series data to either better understand the data set or to predict future trends.

It is a form of regression analysis that gauges the strength of one dependent variable relative to other changing variables. The model's goal is to predict future securities or financial market moves by examining the differences between values in the series instead of through actual values. An ARIMA model can be understood by outlining each of its components as follows:

Autoregression (AR): refers to a model that shows a changing variable that regresses on its own lagged, or prior, values.
Integrated (I): represents the differencing of raw observations to allow for the time series to become stationary (i.e., data values are replaced by the difference between the data values and the previous values).

Moving average (MA): incorporates the dependency between an observation and a residual error from a moving average model applied to lagged observations.

ARIMA Parameters: Each component in ARIMA functions as a parameter with a standard notation. For ARIMA models, a standard notation would be ARIMA with p, d, and q, where integer values substitute for the parameters to indicate the type of ARIMA model used. The parameters can be defined as:

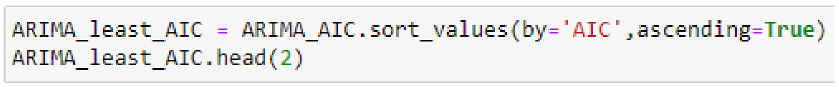
p: the number of lag observations in the model; also known as the lag order.
d: the number of times that the raw observations are differenced; also known as the degree of differencing.
q: the size of the moving average window; also known as the order of the moving average.

Auto ARIMA: The module auto.arima fits the best ARIMA model to univariate time series according to either AIC, AIC or BIC value. This function conducts a search over possible model within the order constraints provided.

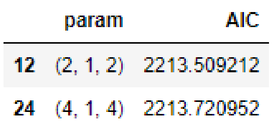
The Akaike information criterion (AIC) is an estimator of the relative quality of statistical models for a given set of data. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection.

Let's build the model using below steps:

- Define "p", "q" and "d" value and create a for loop to create the combinations which would be used for the model. We already have "d" value as (1) based on differencing order.
- Create table to store "p", "d" and "q" combination and AIC value
- Import ARIMA library from statsmodels.tsa.arima.model
- Create the for loop within the pdq parameters defined by itertools to get the least AIC value.
  - Fit the model using train data using all the combinations.
  - Print the p,d,q combinations with AIC value
  - Store these values into table which we have created in step2



```
ARIMA_least_AIC = ARIMA_AIC.sort_values(by='AIC',ascending=True)
ARIMA_least_AIC.head(2)
```



	param	AIC
12	(2, 1, 2)	2213.509212
24	(4, 1, 4)	2213.720952

- Create ARIMA model using order (p,d,q) which has lowest AIC value (here order is (2,1,2)).
- Fit the model on train dataset.
- Print the ARIMA result summary

```
                               SARIMAX Results
==============================================================================
Dep. Variable:              Sparkling   No. Observations:                  132
Model:                 ARIMA(2, 1, 2)   Log Likelihood               -1101.755
Date:                Sun, 27 Jun 2021   AIC                           2213.509
Time:                        22:01:59   BIC                           2227.885
Sample:                    01-01-1980   HQIC                          2219.351
                         - 12-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          1.3121      0.046     28.782      0.000       1.223       1.401
ar.L2         -0.5593      0.072     -7.741      0.000      -0.701      -0.418
ma.L1         -1.9917      0.109    -18.217      0.000      -2.206      -1.777
ma.L2          0.9999      0.110      9.109      0.000       0.785       1.215
sigma2      1.099e+06   1.99e-07   5.51e+12      0.000     1.1e+06     1.1e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.19   Jarque-Bera (JB):                14.46
Prob(Q):                              0.67   Prob(JB):                         0.00
Heteroskedasticity (H):               2.43   Skew:                             0.61
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.08
===================================================================================
```

Here, we could see all p value are less than 0.05. further, all the coeff value are making sense towards the model.

- Do the prediction on test data
- Calculate RMSE value on test data

```
ARIMA_RMSE = metrics.mean_squared_error(test['Sparkling'],predictions,squared=False)

print("AutoARIMA Model(least AIC)(2,1,2):, RMSE is %3.2f"
      %(ARIMA_RMSE))
```

```
AutoARIMA Model(least AIC)(2,1,2):, RMSE is 1299.98
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

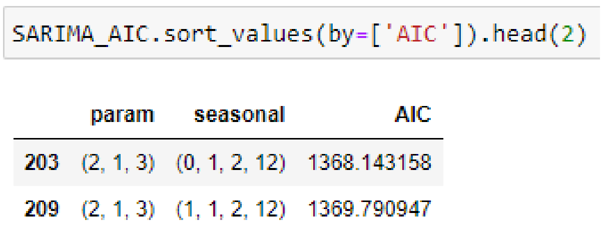
Model 13: Auto SARIMA model – least AIC value

The seasonal part of an ARIMA model has the same structure as the non-seasonal part: it may have an AR factor, an MA factor, and/or an order of differencing.
In the seasonal part of the model, all of these factors operate across multiples of lag s (the number of periods in a season).
A seasonal ARIMA model is classified as an ARIMA(p,d,q)x(P,D,Q) model,
Where:
P=number of seasonal autoregressive (SAR) terms,
D=number of seasonal differences,
Q=number of seasonal moving average (SMA) terms
S = seasonal terms (here, series is monthly. So, S = 12)

Let's build the model using below steps:

- Define (p, d ,q) (P, D, Q) value and create a for loop to create the combinations which would be used for the model. We already have "d" value as (1) based on differencing order.
- Create table to store (p, d ,q) (P, D, Q) combination and AIC value
- Create the for loop within the pdq and PDQ parameters defined by itertools to get the least AIC value.

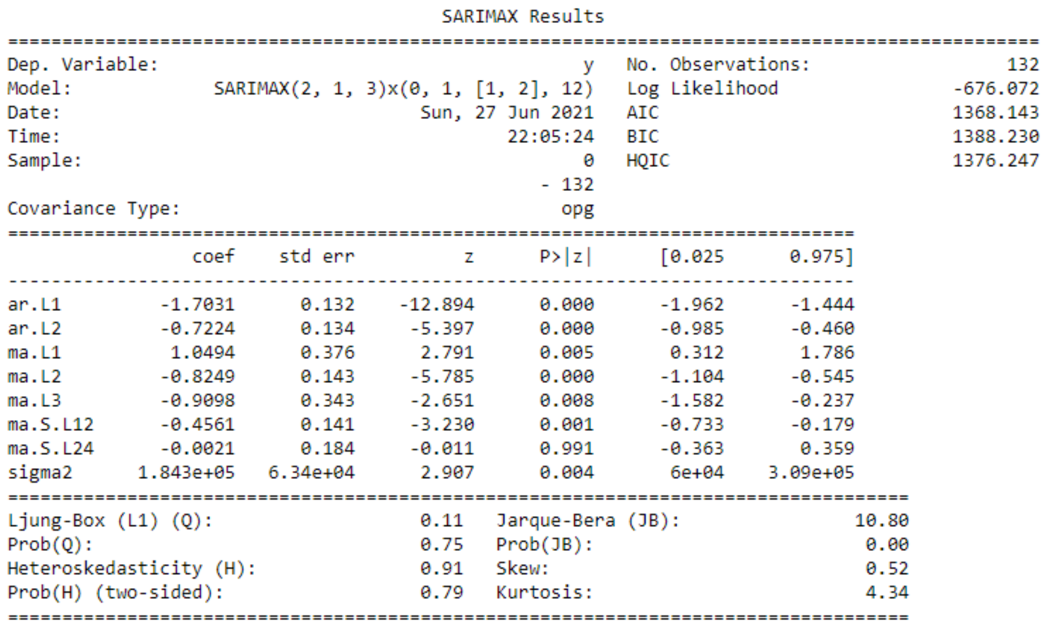
- Fit the model using train data using all the combinations using hypers parameters as order = “pdq combination”, seasonal order = “PDQ combination”, enforce_stationarity and enforce_invertibility equals to false
- Print the (p,d,q) and (P,D,Q) combinations with AIC value
- Store these values into table which we have created in step2



```
SARIMA_AIC.sort_values(by=['AIC']).head(2)
```

	param	seasonal	AIC
203	(2, 1, 3)	(0, 1, 2, 12)	1368.143158
209	(2, 1, 3)	(1, 1, 2, 12)	1369.790947

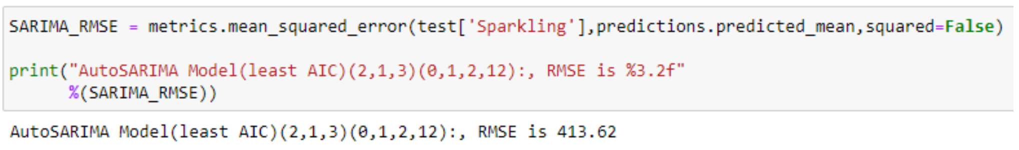
- Create SARIMA model using order (p,d,q) (P,D,Q) which has lowest AIC value (here order = (2,1,3) and seasonal order = (0,1,2,12)).
- Fit the model on train dataset.
- Print the SARIMA result summary



```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                  y   No. Observations:                  132
Model:             SARIMAX(2, 1, 3)x(0, 1, [1, 2], 12)   Log Likelihood              -676.072
Date:                            Sun, 27 Jun 2021   AIC                           1368.143
Time:                                    22:05:24   BIC                           1388.230
Sample:                                         0   HQIC                          1376.247
                                            - 132
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.7031      0.132    -12.894      0.000      -1.962      -1.444
ar.L2         -0.7224      0.134     -5.397      0.000      -0.985      -0.460
ma.L1          1.0494      0.376      2.791      0.005       0.312       1.786
ma.L2         -0.8249      0.143     -5.785      0.000      -1.104      -0.545
ma.L3         -0.9098      0.343     -2.651      0.008      -1.582      -0.237
ma.S.L12      -0.4561      0.141     -3.230      0.001      -0.733      -0.179
ma.S.L24      -0.0021      0.184     -0.011      0.991      -0.363       0.359
sigma2      1.843e+05   6.34e+04      2.907      0.004       6e+04    3.09e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.11   Jarque-Bera (JB):                10.80
Prob(Q):                              0.75   Prob(JB):                         0.00
Heteroskedasticity (H):               0.91   Skew:                             0.52
Prob(H) (two-sided):                  0.79   Kurtosis:                         4.34
===================================================================================
```

Here, we could see all p value are less than 0.05 except seasonal point of MA at lag2. further, all the coeff value are making sense towards the model.

- Do the prediction on test data
- Calculate RMSE value on test data



```
SARIMA_RMSE = metrics.mean_squared_error(test['Sparkling'],predictions.predicted_mean,squared=False)

print("AutoSARIMA Model(least AIC)(2,1,3)(0,1,2,12):, RMSE is %3.2f"
      %(SARIMA_RMSE))
```

```
AutoSARIMA Model(least AIC)(2,1,3)(0,1,2,12):, RMSE is 413.62
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

7 Build ARIMA & SARIMA models based on the cut-off points of ACF & PACF

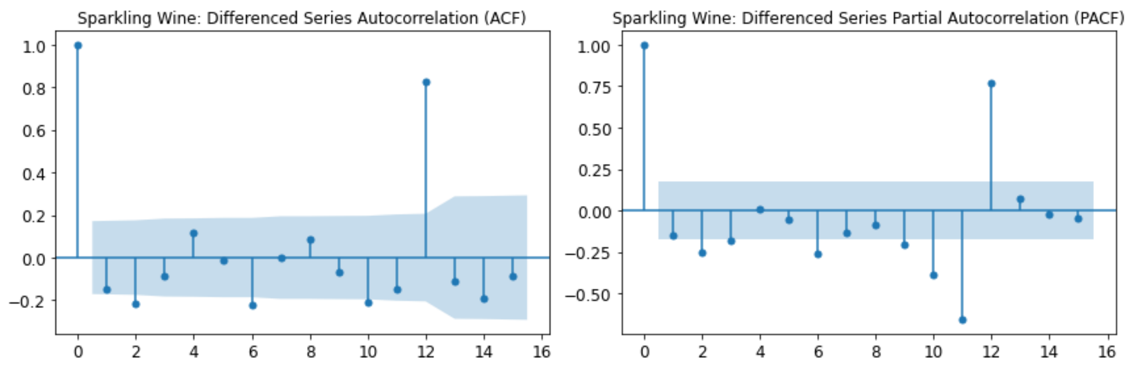
ACF (Auto Correlation function): it is merely a bar chart of the coefficients of correlation between a time series and lags of itself. Auto correlation of different orders gives inside information regarding the time series. ACF make sense only if time series is stationary. Significant auto correlation imply observations of long past influences current observations. ACF represents the q value.
PACF (Partial auto Correlation function): it is a plot of the partial correlation coefficients between the series and lags of itself. PACF represents the p value.

ACF and PACF together to be considered for identification of order of auto regression.

Model 14: Manual ARIMA model – using ACF and PACF value

Let's build the model using below steps:

- First let's plot ACF and PACF at the order difference of (1).



  Based on graph, let's evaluate p and q value. Here, we have taken alpha=0.05.
  - The Auto-Regressive parameter in an ARIMA model is 'p' which comes from the significant lag before which the PACF plot cuts-off to 0.
  - The Moving-Average parameter in an ARIMA model is 'q' which comes from the significant lag before the ACF plot cuts-off to 0.

  By looking at the above plots, we will take the value of p and q to be 1. Hence, let's take **ACF(q)(MA) value as 1 and PACF(p)(AR) value as 1**
- Create ARIMA model using order (p,d,q) based on ACF and PACF plot {order = (1,1,1)}.
- Fit the model on train dataset.
- Print the ARIMA result summary

```
                               SARIMAX Results
==============================================================================
Dep. Variable:              Sparkling   No. Observations:                  132
Model:                 ARIMA(1, 1, 1)   Log Likelihood               -1114.878
Date:                Sun, 27 Jun 2021   AIC                           2235.755
Time:                        22:05:25   BIC                           2244.381
Sample:                    01-01-1980   HQIC                          2239.260
                         - 12-01-1990
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4494      0.043     10.366      0.000       0.364       0.534
ma.L1         -0.9996      0.102     -9.811      0.000      -1.199      -0.800
sigma2      1.401e+06   7.57e-08   1.85e+13      0.000     1.4e+06     1.4e+06
===================================================================================
Ljung-Box (L1) (Q):                   0.50   Jarque-Bera (JB):                10.42
Prob(Q):                              0.48   Prob(JB):                         0.01
Heteroskedasticity (H):               2.64   Skew:                             0.46
Prob(H) (two-sided):                  0.00   Kurtosis:                         4.03
===================================================================================
```

Here, we could see all p value are less than 0.05. further, all the coeff value are making sense towards the model.

- Do the prediction on test data
- Calculate RMSE value on test data

```
ARIMA_Manual_RMSE = metrics.mean_squared_error(test['Sparkling'],predictions,squared=False)

print("ManualARIMA Model(ACF-PACF)(1,1,1):, RMSE is %3.2f"
     %(ARIMA_Manual_RMSE))
```

```
ManualARIMA Model(ACF-PACF)(1,1,1):, RMSE is 1319.94
```

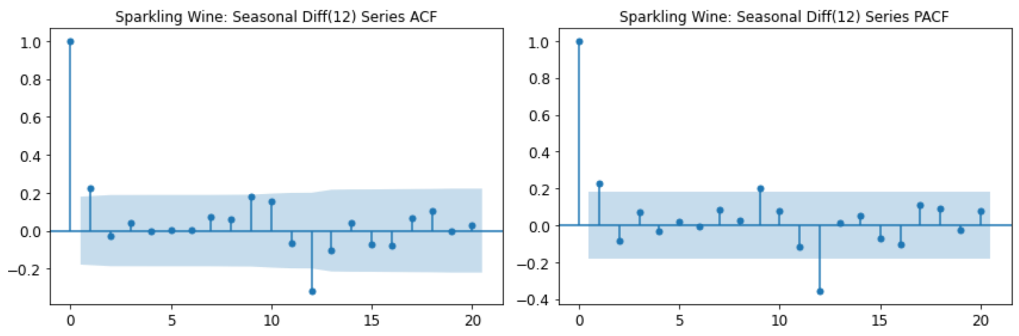
- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

Model 15: Manual SARIMA model – using ACF and PACF value

In SARIMA, we will be incorporating seasonal component for which we need to take seasonal order difference as (12) because this is a monthly time series.

Let's build the model using below steps:

- First let's plot ACF and PACF at the order difference of (12) to get P and Q value. D will be 1 as we will be taking seasonal difference

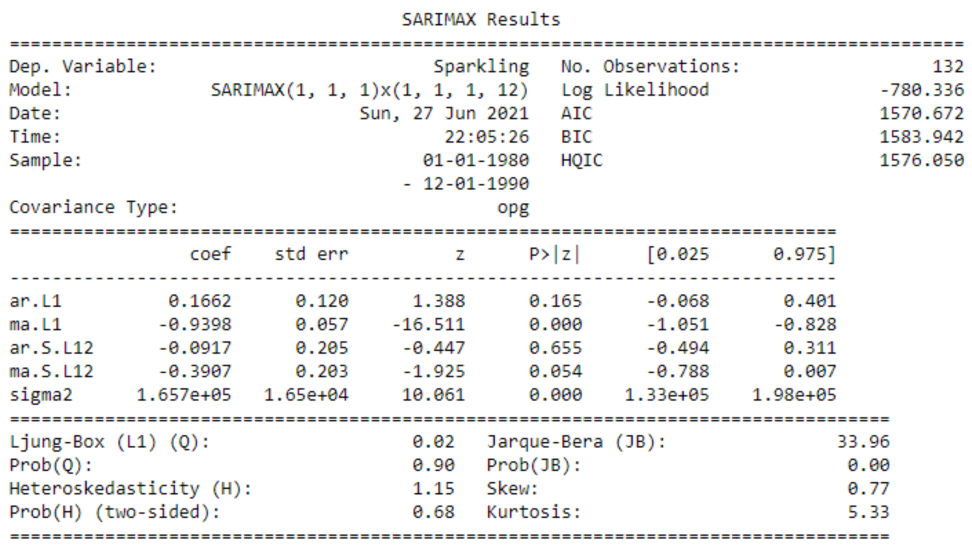


Based on graph, let's evaluate P and Q value. Here, we have taken alpha=0.05.

    - The Auto-Regressive parameter in an ARIMA model is 'P' which comes from the significant seasonal lag before which the seasonal PACF plot cuts-off to 0.
    - The Moving-Average parameter in an ARIMA model is 'Q' which comes from the significant seasonal lag before the seasonal ACF plot cuts-off to 0.

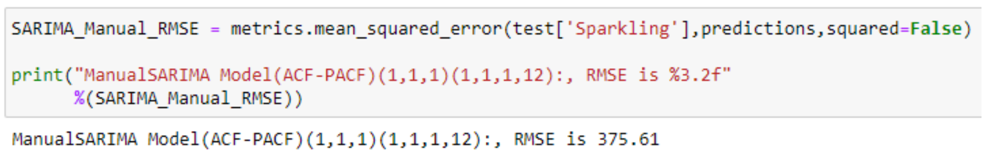
  By looking at the above plots, we will take the value of P and Q to be 1. Hence, let's take **Seasonal ACF(Q)(MA) value as 1 and Seasonal PACF(P)(AR) value as 1**

- Create SARIMA model using order (p,d,q) (P,D,Q,S) based on seasonal ACF and seasonal PACF plot {order = (1,1,1) and seasonal order = (1,1,1,12)}.
- Fit the model on train dataset.
- Print the SARIMA result summary



```
                                      SARIMAX Results
==========================================================================================
Dep. Variable:                           Sparkling   No. Observations:                  132
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 12)   Log Likelihood                -780.336
Date:                             Sun, 27 Jun 2021   AIC                           1570.672
Time:                                     22:05:26   BIC                           1583.942
Sample:                                 01-01-1980   HQIC                          1576.050
                                      - 12-01-1990
Covariance Type:                               opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1662      0.120      1.388      0.165      -0.068       0.401
ma.L1         -0.9398      0.057    -16.511      0.000      -1.051      -0.828
ar.S.L12      -0.0917      0.205     -0.447      0.655      -0.494       0.311
ma.S.L12      -0.3907      0.203     -1.925      0.054      -0.788       0.007
sigma2      1.657e+05   1.65e+04     10.061      0.000    1.33e+05    1.98e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):                33.96
Prob(Q):                              0.90   Prob(JB):                         0.00
Heteroskedasticity (H):               1.15   Skew:                             0.77
Prob(H) (two-sided):                  0.68   Kurtosis:                         5.33
===================================================================================
```

- Do the prediction on test data
- Calculate RMSE value on test data



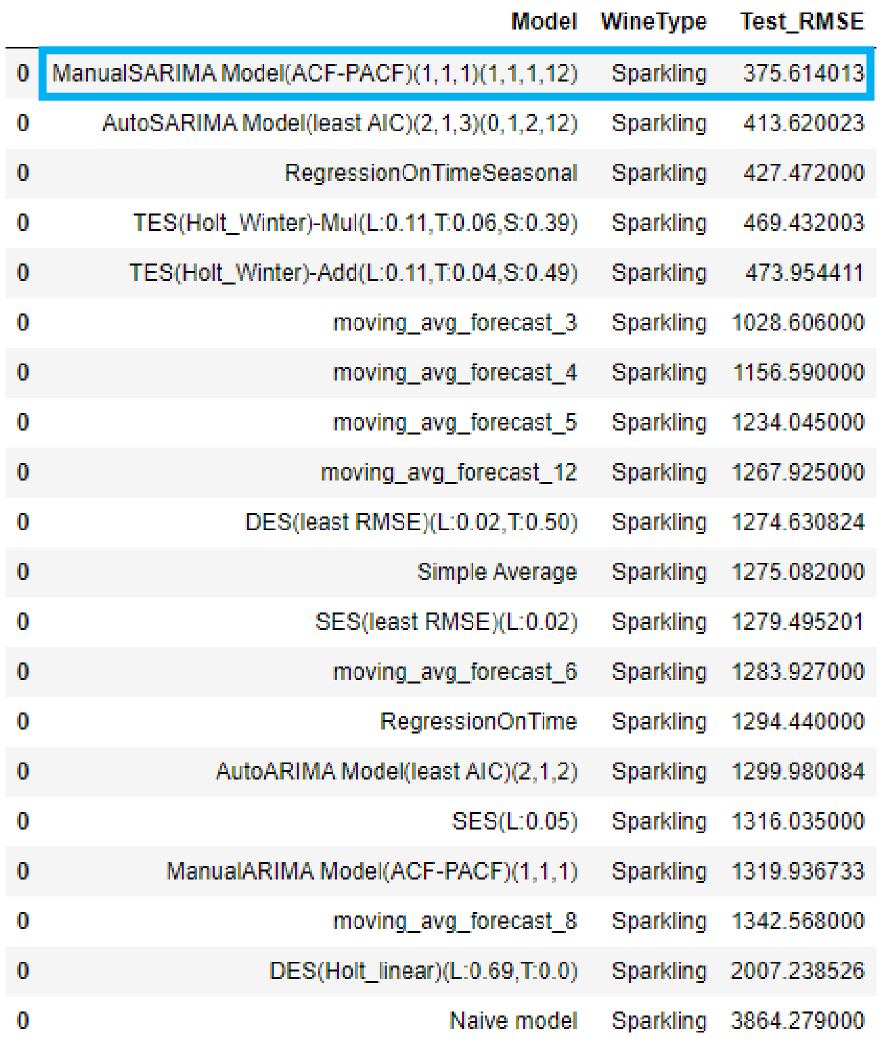
```
SARIMA_Manual_RMSE = metrics.mean_squared_error(test['Sparkling'],predictions,squared=False)

print("ManualSARIMA Model(ACF-PACF)(1,1,1)(1,1,1,12):, RMSE is %3.2f"
      %(SARIMA_Manual_RMSE))
```

```
ManualSARIMA Model(ACF-PACF)(1,1,1)(1,1,1,12):, RMSE is 375.61
```

- Store the test RMSE values in the table. This would help us to compare the models once we build all the models.

8 Build a table with all the models

We have built total of 15 models and mentioned below is table which consist of model name, its corresponding parameters and respective RMSE values (in ascending order).



	Model	WineType	Test_RMSE
0	ManualSARIMA Model(ACF-PACF)(1,1,1)(1,1,1,12)	Sparkling	375.614013
0	AutoSARIMA Model(least AIC)(2,1,3)(0,1,2,12)	Sparkling	413.620023
0	RegressionOnTimeSeasonal	Sparkling	427.472000
0	TES(Holt_Winter)-Mul(L:0.11,T:0.06,S:0.39)	Sparkling	469.432003
0	TES(Holt_Winter)-Add(L:0.11,T:0.04,S:0.49)	Sparkling	473.954411
0	moving_avg_forecast_3	Sparkling	1028.606000
0	moving_avg_forecast_4	Sparkling	1156.590000
0	moving_avg_forecast_5	Sparkling	1234.045000
0	moving_avg_forecast_12	Sparkling	1267.925000
0	DES(least RMSE)(L:0.02,T:0.50)	Sparkling	1274.630824
0	Simple Average	Sparkling	1275.082000
0	SES(least RMSE)(L:0.02)	Sparkling	1279.495201
0	moving_avg_forecast_6	Sparkling	1283.927000
0	RegressionOnTime	Sparkling	1294.440000
0	AutoARIMA Model(least AIC)(2,1,2)	Sparkling	1299.980084
0	SES(L:0.05)	Sparkling	1316.035000
0	ManualARIMA Model(ACF-PACF)(1,1,1)	Sparkling	1319.936733
0	moving_avg_forecast_8	Sparkling	1342.568000
0	DES(Holt_linear)(L:0.69,T:0.0)	Sparkling	2007.238526
0	Naive model	Sparkling	3864.279000

Around total of 15 models we have built for "Sparkling Wine" dataset and the best model which is coming out is

SARIMA Model (ACF and PACF) with parameters (p=1,d=1,q=1)(P=1,D=1,Q=1,F=12) and test RMSE value ~376

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

9 Run the best model on full dataset & predict 12 months into the future with appropriate CI (confidence intervals)

Here, we have both training and full data both were non-stationary hence, nothing to be done in terms of stationarity. Let's run the best model (i.e. Manual SARIMA with order (1,1,1) and seasonal order (1,1,1,12))

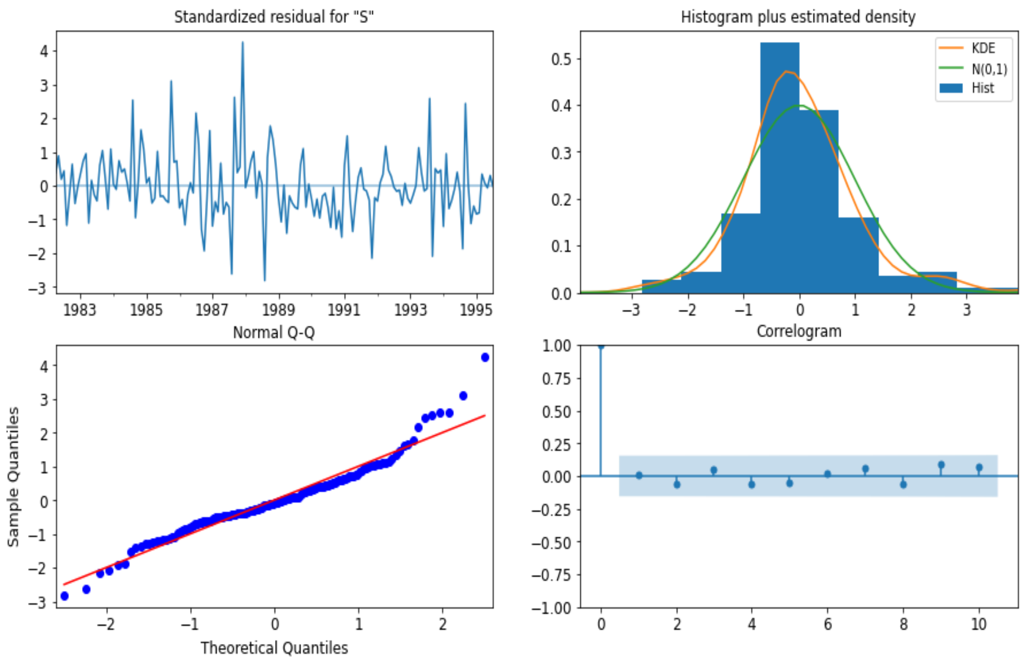
Run best model on full dataset

Let's build the model using below steps:

- Create SARIMA model using order = (1,1,1) and seasonal order = (1,1,1,12).
- Fit the model on **full dataset**.
- Print the SARIMA result summary

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                            Sparkling   No. Observations:                  187
Model:             SARIMAX(1, 1, 1)x(1, 1, 1, 12)   Log Likelihood               -1180.382
Date:                              Sun, 27 Jun 2021   AIC                           2370.765
Time:                                      22:05:27   BIC                           2386.141
Sample:                                  01-01-1980   HQIC                          2377.009
                                       - 07-01-1995
Covariance Type:                                opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.1151      0.082      1.398      0.162      -0.046       0.276
ma.L1         -0.9655      0.033    -29.501      0.000      -1.030      -0.901
ar.S.L12      -0.0903      0.122     -0.743      0.457      -0.329       0.148
ma.S.L12      -0.4995      0.113     -4.430      0.000      -0.720      -0.278
sigma2      1.477e+05   1.17e+04     12.575      0.000    1.25e+05    1.71e+05
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):                52.61
Prob(Q):                              0.92   Prob(JB):                         0.00
Heteroskedasticity (H):               0.92   Skew:                             0.69
Prob(H) (two-sided):                  0.75   Kurtosis:                         5.44
===================================================================================
```

- Run the plot diagnostic for full dataset on SARIMA result



Predict 12 months into the future with appropriate CI

Let's predict the values using below steps:

- Do the prediction on 12 months after the date at which time series ends. So forecast will be from Aug'1995 to Jul'1996.
- Use summary_frame to get predicted values for next 12 months with confidence interval of 95%

```
prediction_nxt12mnths = results_model_fulldata.get_forecast(steps=12)
```

```
prediction_nxt12mnths.summary_frame(alpha=0.05)
```

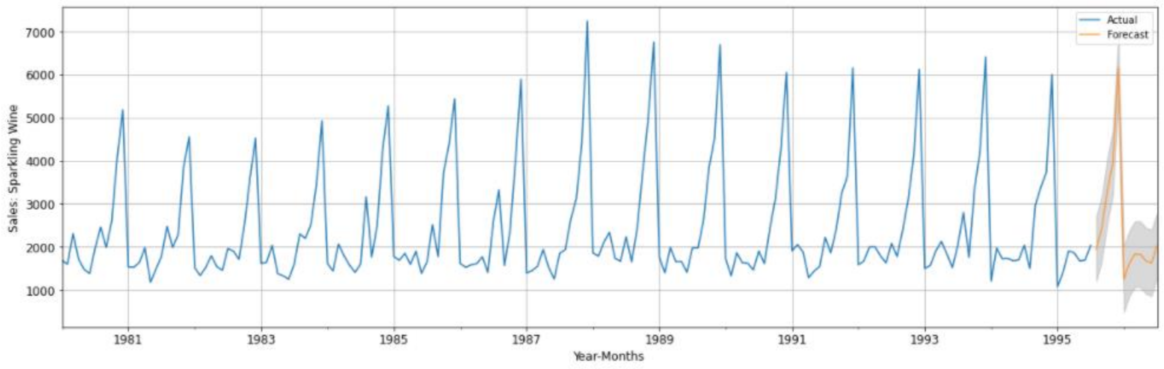
Sparkling	mean	mean_se	mean_ci_lower	mean_ci_upper
1995-08-01	1946.522299	384.306578	1193.295246	2699.749352
1995-09-01	2417.781910	388.581362	1656.176435	3179.387384
1995-10-01	3292.400711	389.089260	2529.799774	4055.001647
1995-11-01	3950.757308	389.399739	3187.547845	4713.966772
1995-12-01	6151.543256	389.690425	5387.764058	6915.322453
1996-01-01	1238.842795	389.978688	474.498611	2003.186979
1996-02-01	1613.898807	390.266484	848.990554	2378.807061
1996-03-01	1828.004545	390.554039	1062.532695	2593.476394
1996-04-01	1820.785331	390.841378	1054.750306	2586.820356
1996-05-01	1670.366615	391.128506	903.768830	2436.964400
1996-06-01	1622.236282	391.415423	855.076150	2389.396415
1996-07-01	2018.270769	391.702130	1250.548700	2785.992837

- Calculate RMSE value on full data

```
RMSE_fulldata = metrics.mean_squared_error(df['Sparkling'],results_model_fulldata.fittedvalues,squared=False)

print("ManualSARIMA Model(ACF-PACF)(3,1,2)(1,1,1,12):, RMSE is %3.2f"
      %(RMSE_fulldata))
```

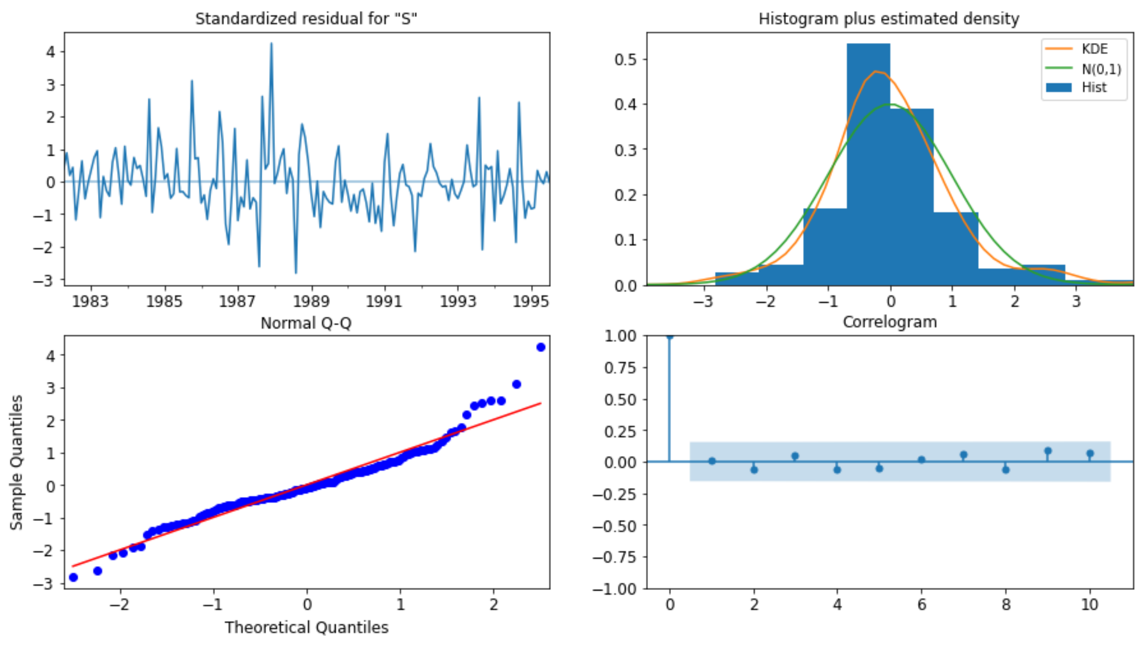
```
ManualSARIMA Model(ACF-PACF)(3,1,2)(1,1,1,12):, RMSE is 544.98
```

- Plot the full dataset with forecast value and confidence interval band.



10 Comment on the model thus built and report your findings and suggest the measures that the company should be taking for future sales

Key Findings



Plot diagnostics summary charts gives promising result and below are point that justifies the same.

- **(Top left chart):** Residual chart says it is ranges between 4 to -2 which is good. The residuals over time don't display any obvious seasonality and appear to be white noise.

- **(Bottom right chart):** This is confirmed by the autocorrelation (i.e. correlogram) plot, which shows that the time series residuals have low correlation with lagged versions of itself. Autocorrelation (correlogram) plot of the residuals doesn't show the unexplained correlation which is left in the data.

- **(Bottom left chart):** QQ (Normality) plot have the linear relationship. Further, it shows that the ordered distribution of residuals (blue dots) follows the linear trend of the samples taken from a standard normal distribution with N(0, 1). Again, this is a strong indication that the residuals are normally distributed.

- **(Top right chart):** Distribution plot is seeming to be normal with mean 0 and std as 1 with in no real outliers.

So, Overall we can conclude that that residuals are random with no information or juice in them and our model produces a satisfactory fit that could help us understand our time series data and forecast future values.

Recommendations

Time series forecasting helps businesses make informed business decisions because it can be based on historical data patterns. It can be used to forecast future conditions and events. mentioned below are four key characteristics which time series forecasting has.

1. **Reliability:** Time series forecasting is most reliable, especially when the data represents a broad time period such as large numbers of observations for longer time periods. Information can be extracted by measuring data at various intervals.

2. **Seasonal patterns:** Data points variances measured can reveal seasonal fluctuation patterns that serve as the basis for forecasts. Such information is of particular importance to markets whose products fluctuate seasonally because it helps them plan for production and delivery requirements.

3. **Trend estimation:** Time series method can also be used to identify trends because data tendencies from it can be useful to managers when measurements show a decrease or an increase in sales for a particular product.

4. **Growth:** Time series method is useful to measure both endogenous and financial growth. Endogenous growth is the development from within an organization's internal human capital that leads to economic growth. For example, the impact of policy variables can be evidenced through time series analysis.

Based on the Sparkling Wine dataset, it is coming out that this timeseries has **almost no trend, high seasonality(~95% of the volume is happening in the month of November and December){hence, median sales across the years doesn't have any changes but month wise it is giving major impact}**.

According to us, mentioned below are the **ways through which we could manage the business seasonality**.

- **Look for ways to diversify:** we can offset seasonality by being as innovative as possible. Whether that means diversifying product lines or by pre-empting what your customers want. Market survey is the best way to understand the business/customer needs.
- **Promotional campaign:** Due to strong presence of seasonality, Company should start stock up the wine from July and ensure the availability for December sales (Christmas and New Year eve) and during non-peak seasons Jan, Feb should run promotions to catch up the sales
- **Develop sales, inventory and staffing plans:** Planning inventory and sales at least six months ahead will helps pinpoint where business needs to build in a cash cushion to tide over during quieter periods. Plan everything from stock levels to staff. That way business has the supplies and all material required when demand picks up again. A cash flow forecast can also help with this.
- **Think ahead**: how will this new seasonal business develop? Will it simply become an addition to your current business, or has it got strong enough legs to stand on its own?