TSF PROJECT - GUIDED

DSBA

Krishnabhamini Sinha

Contents

Data Dictionary for Problem : (gold_prices)

Data: the date that contains the price for a particular date point.
Price: prices of gold pertaining to certain dates.

Problem 1.1 - Define the problem and perform Exploratory Data Analysis

- Read the data as an appropriate time series data - Plot the data - Perform EDA - Perform Decomposition

Problem Definition:

Context

In the dynamic landscape of the financial markets, the demand for accurate predictions of gold prices is crucial for investors, traders, and stakeholders. Similar to the challenges faced by business communities in the United States in identifying and attracting the right talent, predicting the future values of precious metals poses a significant challenge. The inherent volatility and complex dynamics influencing gold prices require a reliable forecasting tool.

The goal is to provide a valuable asset for individuals navigating the financial markets, ensuring they can approach the unpredictability of gold prices with confidence and strategic insight.

Objective

In the realm of financial markets, predicting the future value of gold and silver holds significant importance for investors, traders, and stakeholders. The challenge lies in the inherent volatility and complex dynamics influencing gold prices. To address this, the objective is to develop an accurate time series forecasting model that can predict the future prices of gold. The model should leverage historical gold and silver price data. The anticipated outcome is a robust forecasting tool that empowers stakeholders to make informed decisions and navigate the dynamic landscape of the gold market with confidence.

Solution:

Reading the dataset as a Time Series:

```
df = pd.read_csv('gold_prices.csv', parse_dates=True, index_col=0)
```

```
df.head() #Complete the code to view first 5 rows of the data
```

Date	Price
2023-08-17	1915.2
2023-08-16	1928.3
2023-08-15	1935.2
2023-08-14	1944.0
2023-08-11	NaN

```
df.tail() #Complete the code to view last 5 rows of the data
```

Date	Price
2013-08-23	NaN
2013-08-22	1370.8
2013-08-21	1370.1
2013-08-20	1372.6
2013-08-19	1365.7

Shape of the dataset:

```
df.shape
```

```
(2539, 1)
```

Data Types:

The data type for each column is enlisted as below in the figure given below.

```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 2539 entries, 2023-08-17 to 2013-08-19
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype  
---  ------  --------------  -----  
 0   Price   2460 non-null   float64
dtypes: float64(1)
memory usage: 39.7 KB
```

Checking for null values:

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

```
Price    79
dtype: int64
```

There are 79 rows where the prices of gold have not been entered.

Missing Value Treatment

Although this is instructed in the second question, we will go with this first, because a Time Series cannot be plotted in a broken format.

The process we will use is interpolation. The reason being this process fills the missing values with the immediate past 1 or 2 values. That brings familiarity and relatability to the data points.

```
df['Price'] = df['Price'].interpolate(method='linear')
```

```
df.head()
```

Date	Price	Year	Month
2023-08-17	1915.20	2023	8
2023-08-16	1928.30	2023	8
2023-08-15	1935.20	2023	8
2023-08-14	1944.00	2023	8
2023-08-11	1946.45	2023	8

We can see after completing the interpolation process, we can see that there are no missing values now:

```
df['Price'].isnull().sum()
```

```
0
```

Plotting the data:

The prices of gold have been plotted according to the year. We can see an increasing trend in the prices of gold. Seasonality also seems to be present in the data. We will finalize it after the decomposition of the time series.

Resampling of time series:

Resampling is a fundamental technique in time series analysis, enabling us to adjust the frequency of our data by aggregating (downsampling) or interpolating (upsampling) values. We are going to resample our dataset before we apply any further processes on it.

```
df = df.resample('M').mean()
df.head()
```

Date	Price	Year	Month
2013-08-31	1390.225000	2013.0	8.0
2013-09-30	1348.461905	2013.0	9.0
2013-10-31	1317.193478	2013.0	10.0
2013-11-30	1273.433333	2013.0	11.0
2013-12-31	1223.186364	2013.0	12.0

We can see that the first five after resampling have different values than the original dataset's first five values. Let us check the shape of the two datasets.

Shape of original dataset:

```
df.shape
```

```
(2539, 1)
```

Shape of resampled dataset:

```
df.shape
```

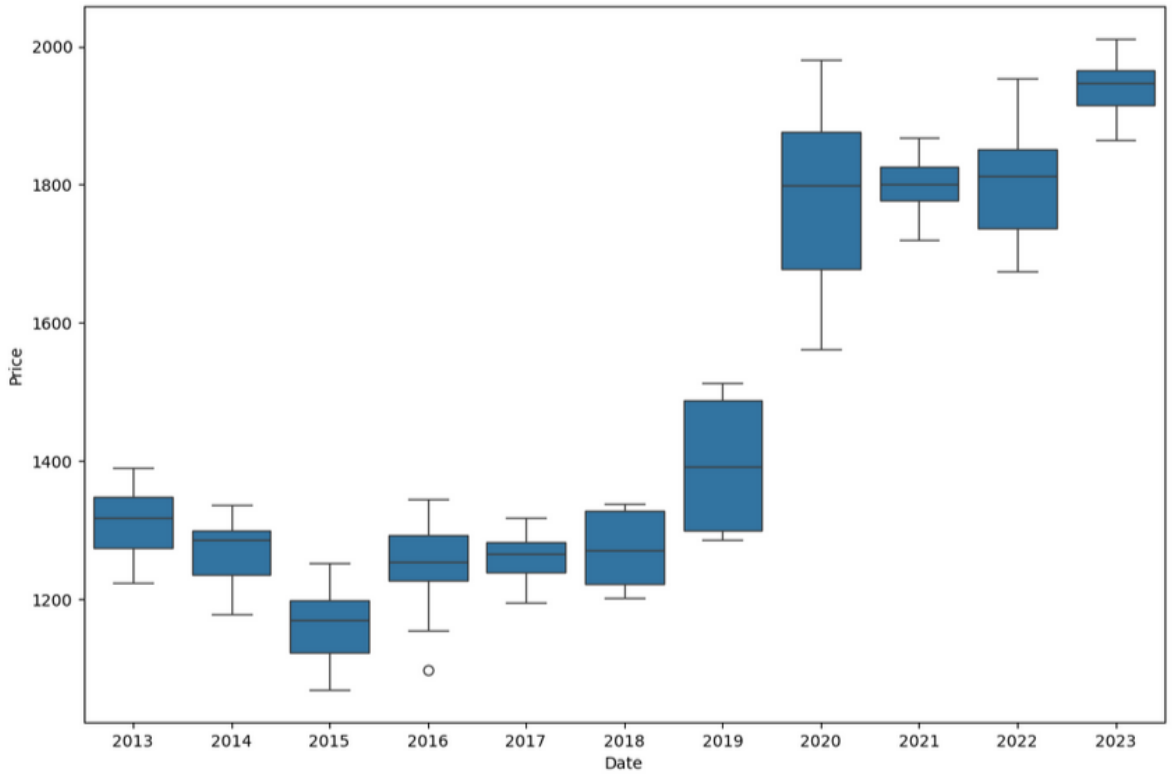
```
(121, 3)
```

EDA (Exploratory Data Analysis):

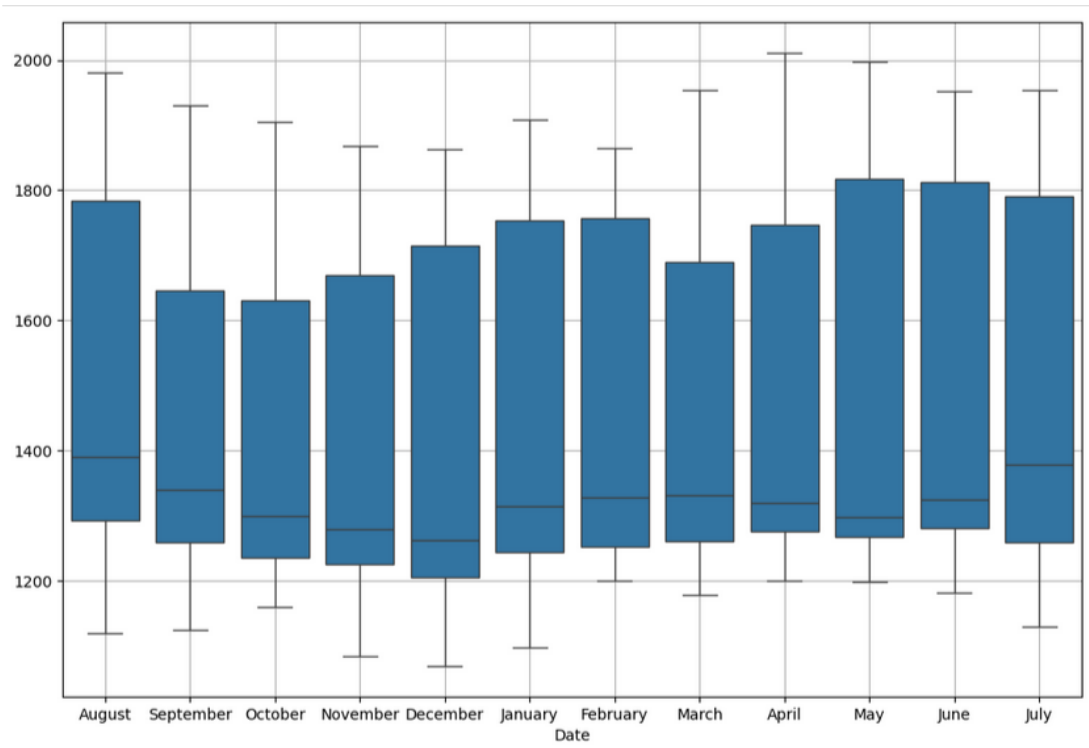
Trend

The given chart shows us data in forms of boxplots year-wise – cumulatively displaying the trend of the gold prices.

We see that prices started peaking from the year 2020 and the price ranges continued to vary in a similar price range till 2023.

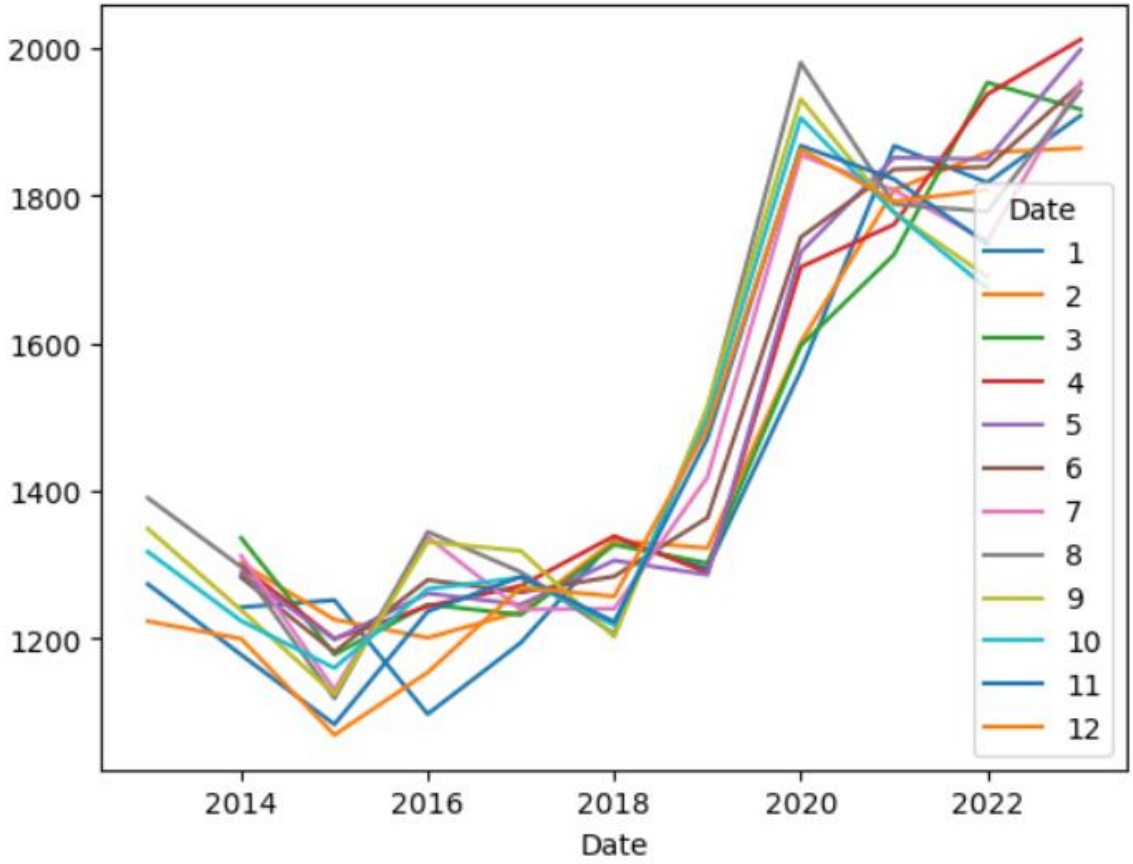


Seasonality



In the above chart we can see that there us a clear pattern going on through all years – concluding that there is a factor of seasonality present in the time series.

Also, to accompany the previous figure, given below is the monthly gold prices across all years – setting the trend of each month across the years. To create this chart we have created a pivot table that helps us pick cumulative prices of each month across the years.



Decomposition

Time series decomposition is the process of separating a time series into its constituent components, such as trend, seasonality, and noise. In this article, we will explore various time series decomposition techniques, their types, and provide code samples for each.

Time series decomposition helps us break down a time series dataset into three main components:

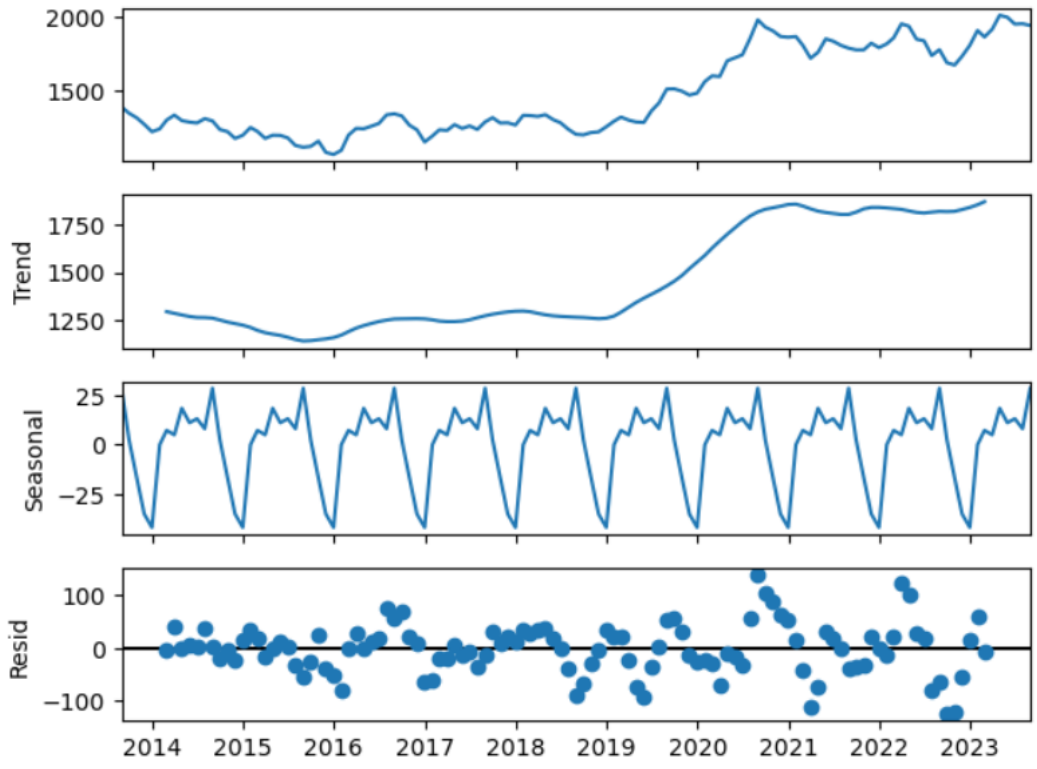
Trend: The trend component represents the long-term movement in the data, representing the underlying pattern.

Seasonality: The seasonality component represents the repeating, short-term fluctuations caused by factors like seasons or cycles.

Residual (Noise): The residual component represents random variability that remains after removing the trend and seasonality.

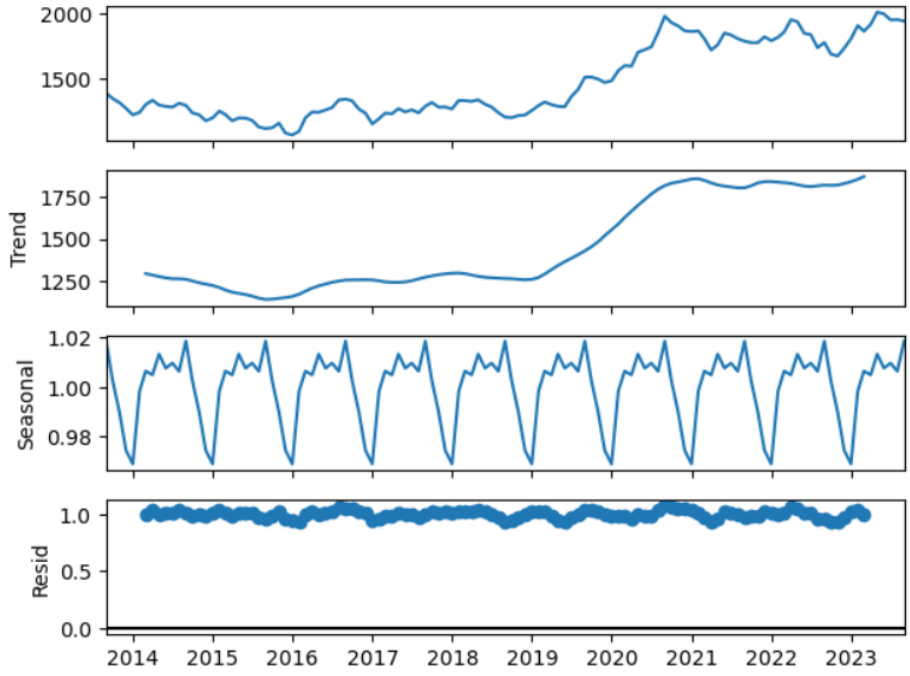
We will perform both **addtive and multiplicative decompostion** on the time series to finalise the type of time series we hsave at hand.

```
#Decompose the time series additively
df_add_decompose = seasonal_decompose(df, model = 'additive')  # Complete the code to decompose
df_add_decompose.plot() # Complete the code to plot
plt.show()
```



We can see the presence of both trend and seasonality in the time series. Let us look the decomposition of the time series in a multiplicative mode:

```
#Decompose the time series multiplicative
df_mul_decompose = seasonal_decompose(df, model='multiplicative') #Complete the code to decompose multiplicatively
df_mul_decompose.plot() #Complte the code to plot
plt.show()
```



There is no change in the trends or seasonalities in both the additive and multiplicative models of decomposition – indicating that the time series given to us is additive in nature.

Problem 1.2 - Data Preprocessing

- Missing value treatment - Visualize the processed data - Train-test split

Solution:

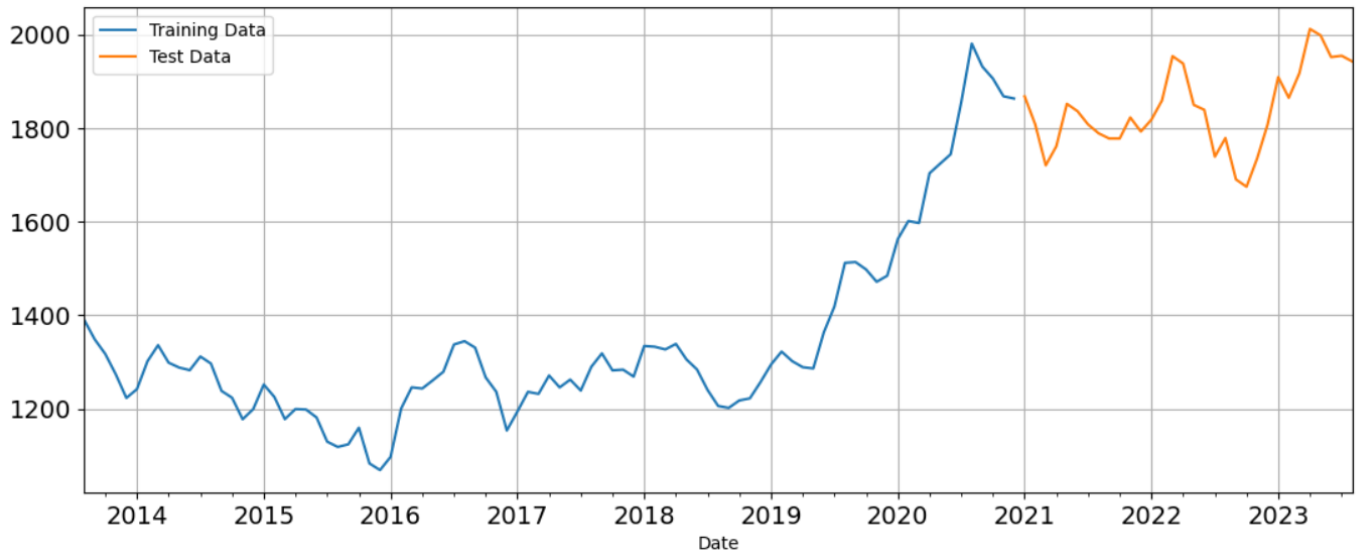
We already saw the missing value treatment in the previous question.

Splitting the data into train and test datasets:

```
#Complete the code to split train set consisting data until 2020 and test set from 2021 onwards
train = df[df.index.year < 2021]
test = df[df.index.year >= 2021]
```

We have alloted the date-time indexes before 2021 to train data and equal to and after 2021 to test data.

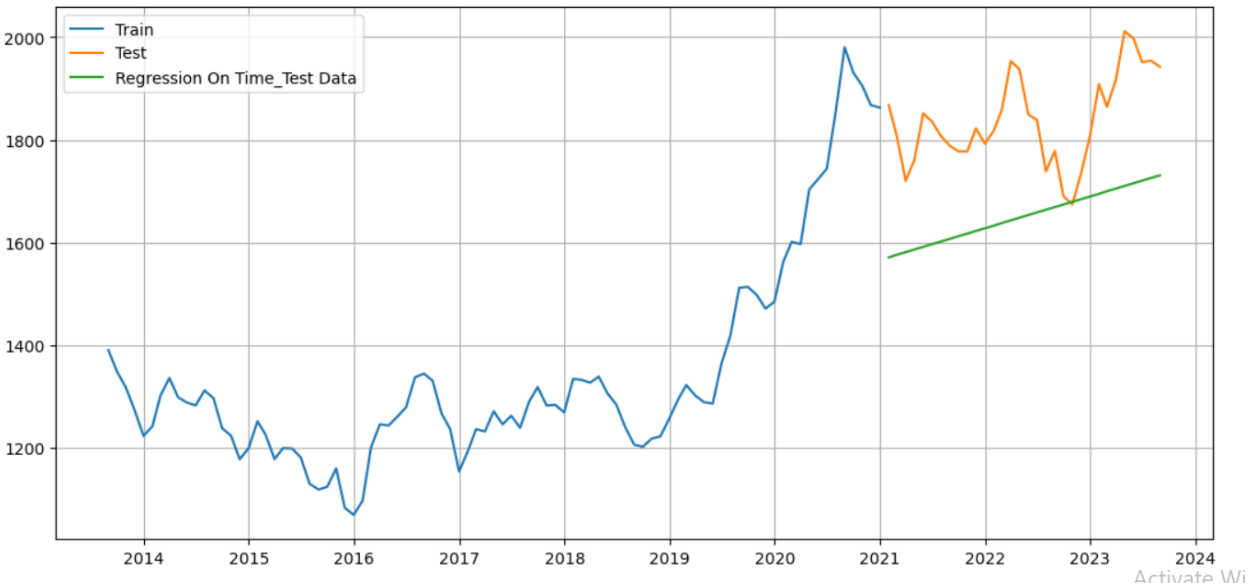
Visualizing the processed data:



Problem 1.3 - Model Building - Original Data

- Build forecasting models - Linear regression - Moving Average - Exponential Models (Single, Double, Triple) - Check the performance of the models built.

Linear Regression Model:



The linear regression model does not take trend and seasonality into account and produces a quite flat forecast. We cannot rely much on it to take various necessary dynamics into account for producing a robust forecast.

Let us look at the performance of this model with the RMSE metrics on the test data. Root mean squared error (RMSE) is the square root of the mean of the square of all of the error. The use of RMSE is very common, and it is considered an excellent general purpose error metric for numerical predictions.

	Test RMSE
Linear Regression	202.993632

Moving Averages Model

For this model we will take random rolling means of 2, 4, 6 and 9. The best interval can be determined by the maximum accuracy (or the minimum error) over here. For Moving Average, we are going to average over the entire data.

We can see that moving average with trailing point 2 i.e., prediction with a moving average with rolling means of 2 is closest to the original trend. We will look at the RMSE values as well to finalize our model.

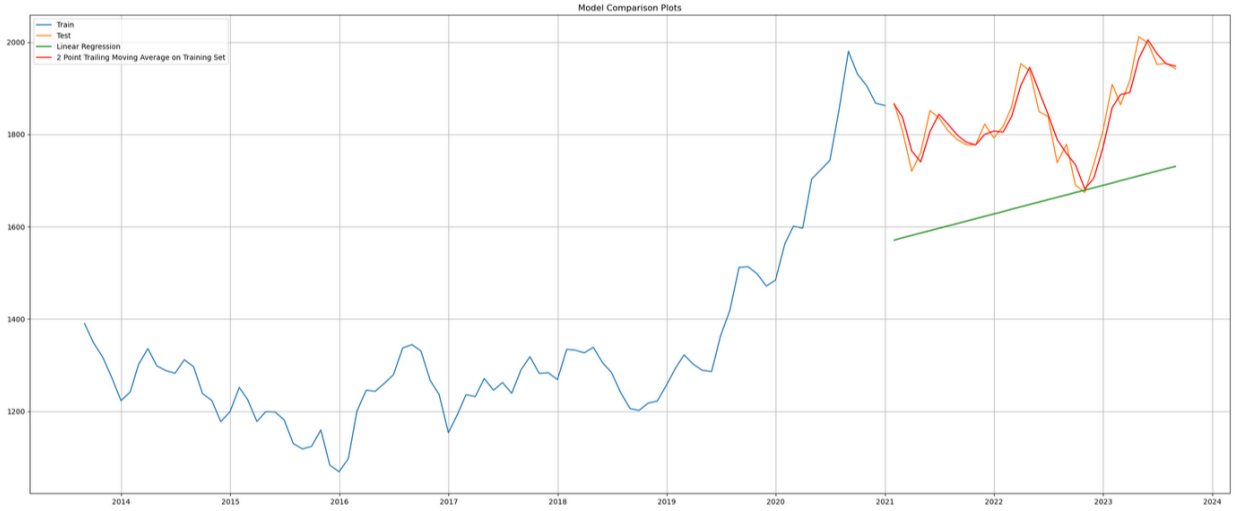
Now, we will divide the data into train and test datasets and apply the Moving Averages model to the same.

```
#Creating train and test set
trailing_MovingAverage_train= MovingAverage[MovingAverage.index.year < 2021]
trailing_MovingAverage_test= MovingAverage[MovingAverage.index.year >= 2021]
```

The RMSE scores of test data are as follows:

	Test RMSE
Linear Regression	202.993632
2pointTrailingMovingAverage	27.958204
4pointTrailingMovingAverage	54.547384
6pointTrailingMovingAverage	70.810413
9pointTrailingMovingAverage	85.434070

We find out that the moving averages model with rolling means of 2 has the lowest RMSE score i.e., it has the highest performance.



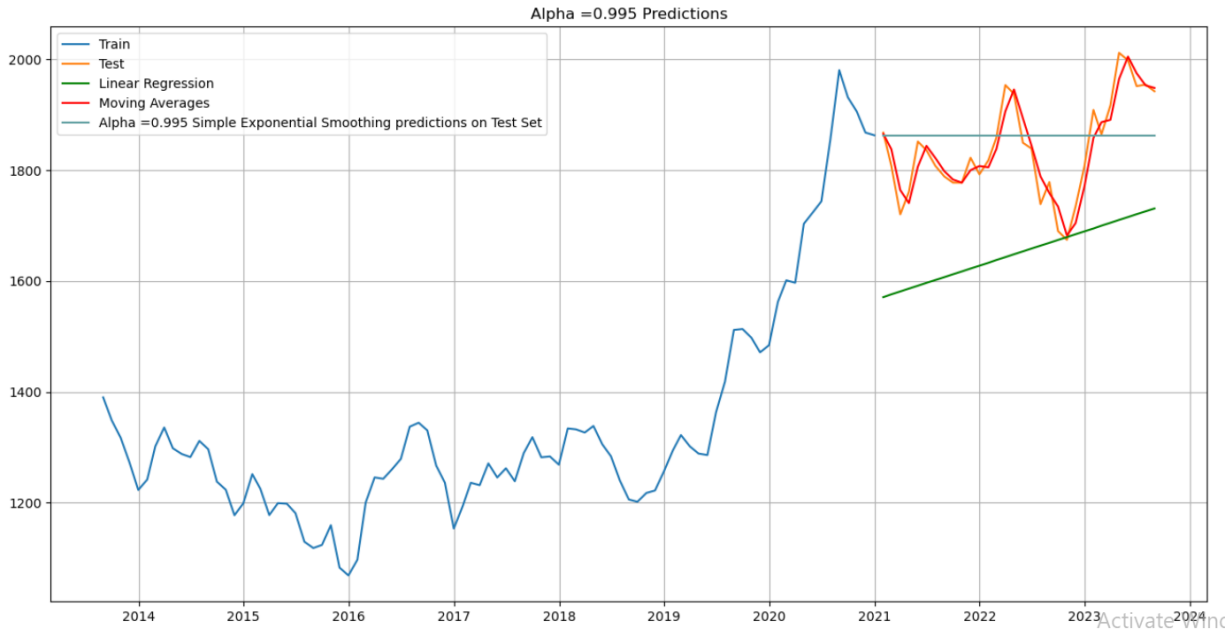
Exponential Models (Single, Double, Triple)

We will look at how close the exponential models are predicting and performing based on both the plots and the RMSE scores. Let us begin with the Simple Exponential Smoothing Model. We will take parameters generated through autofitting of the model:

```
model_SES_autofit.params
```

```
{'smoothing_level': 0.995,
 'smoothing_trend': nan,
 'smoothing_seasonal': nan,
 'damping_trend': nan,
 'initial_level': 1390.225,
 'initial_trend': nan,
 'initial_seasons': array([], dtype=float64),
 'use_boxcox': False,
 'lamda': None,
 'remove_bias': False}
```

For a level at alpha = 0.995, the plot and RMSE scores are as follows:

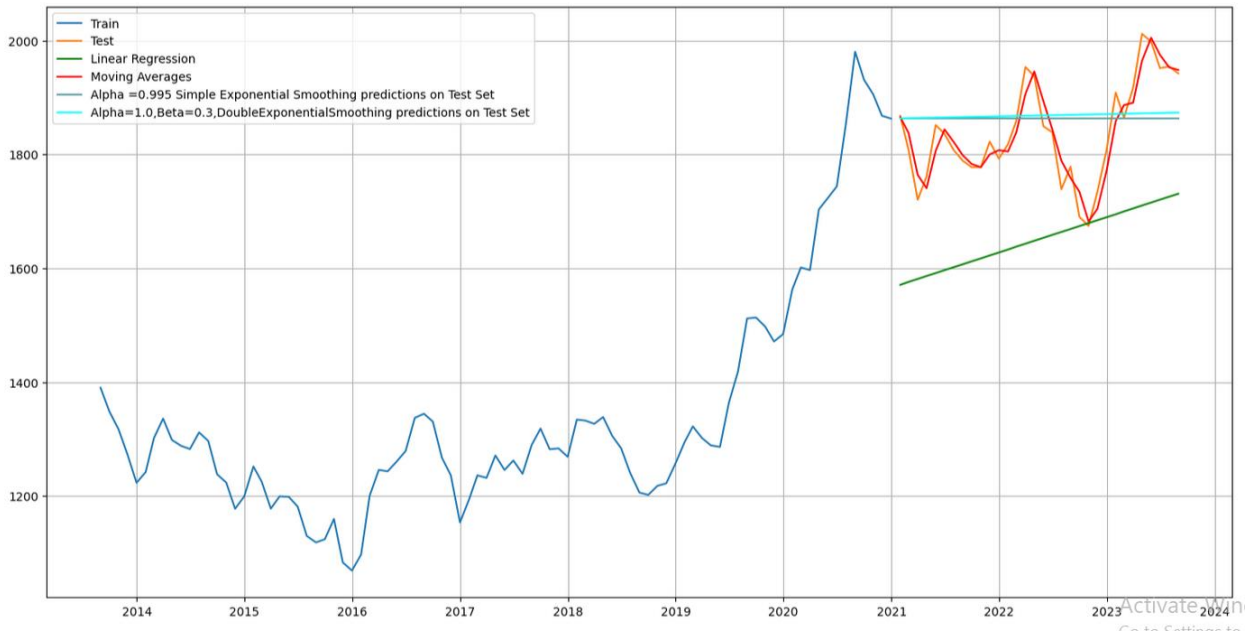


	Test RMSE
Linear Regression	202.993632
2pointTrailingMovingAverage	27.958204
4pointTrailingMovingAverage	54.547384
6pointTrailingMovingAverage	70.810413
9pointTrailingMovingAverage	85.434070
Alpha=0.995,SimpleExponentialSmoothing	89.506086

The simple exponential smoothing model gives us a better prediction than the linear regression model but its performance is still lower than that of moving averages with rolling means of 2.

Double Exponential Smoothing

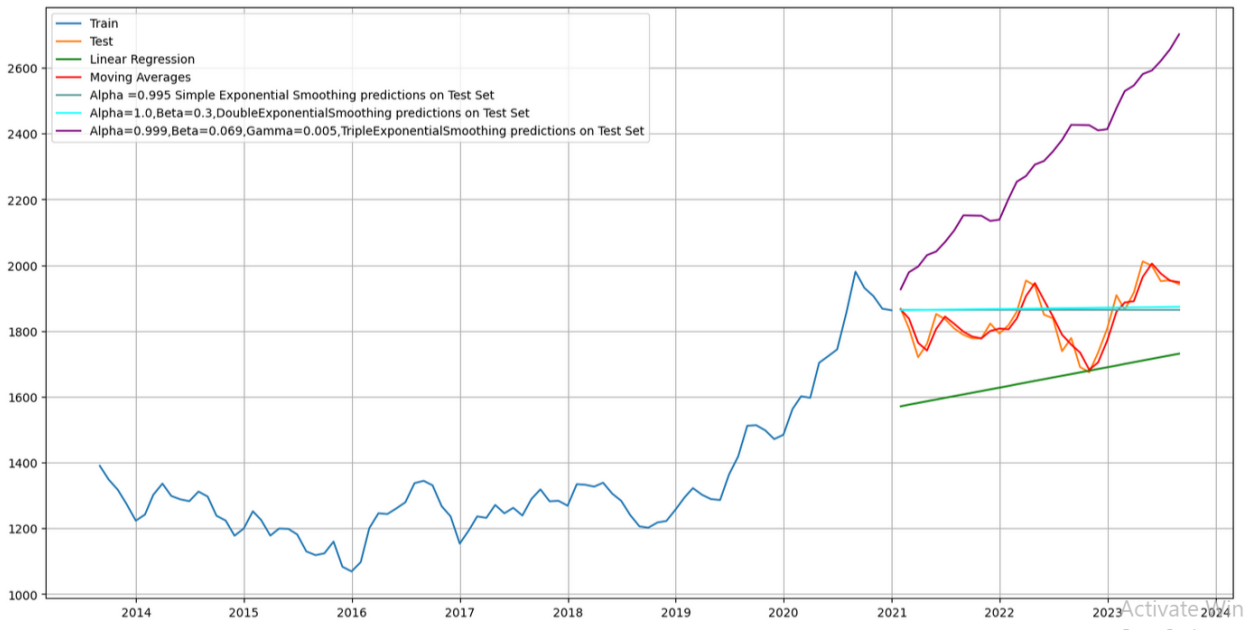
For the lowest RMSE score at alpha (level) = 1.0 and beta (trend) = 0.3, the plot and RMSE score are as follows:



	Test RMSE
Linear Regression	202.993632
2pointTrailingMovingAverage	27.958204
4pointTrailingMovingAverage	54.547384
6pointTrailingMovingAverage	70.810413
9pointTrailingMovingAverage	85.434070
Alpha=0.995,SimpleExponentialSmoothing	89.506086
Alpha=1.0,Beta=0.3,DoubleExponentialSmoothing	44.907310

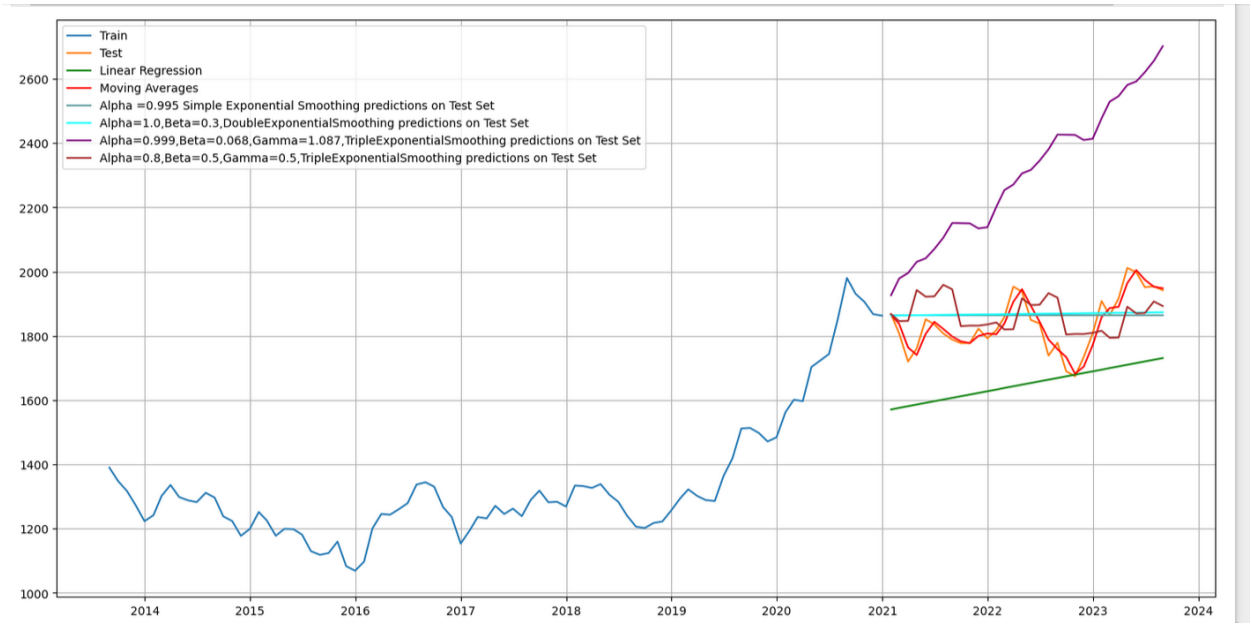
Triple Exponential Smoothing

For the RMSE score at alpha (level) = 0.99 and beta (trend) = 0.069 and gamma (seasonality) = 0.005 after autofitting on the model, the plot and RMSE score are as follows:



	Test RMSE
Linear Regression	202.993632
2pointTrailingMovingAverage	27.958204
4pointTrailingMovingAverage	54.547384
6pointTrailingMovingAverage	70.810413
9pointTrailingMovingAverage	85.434070
Alpha=0.995,SimpleExponentialSmoothing	89.506086
Alpha=1.0,Beta=0.3,DoubleExponentialSmoothing	44.907310
Alpha=0.999,Beta=0.069,Gamma=0.005,TripleExponentialSmoothing	503.699889

We can see that the RMSE score is too high and hence, might not be a good model. We now try to iterate through various of alpha, beta and gamma and for the lowest RMSE score at alpha = 0.8, beta = 0.5 and gamma = 0.5 the plot and score are as follows:



	Test RMSE
Linear Regression	202.993632
2pointTrailingMovingAverage	27.958204
4pointTrailingMovingAverage	54.547384
6pointTrailingMovingAverage	70.810413
9pointTrailingMovingAverage	85.434070
Alpha=0.995,SimpleExponentialSmoothing	89.506086
Alpha=1.0,Beta=0.3,DoubleExponentialSmoothing	44.907310
Alpha=0.999,Beta=0.069,Gamma=0.005,TripleExponentialSmoothing	503.699889
Alpha=0.8,Beta=0.5,Gamma=0.5,TripleExponentialSmoothing	98.000340

Performance of models – Till now we can see that the model with most good performance is the moving average with rolling means of 2 followed by the double exponential model at alpha = 1.0, beta = 0.3.

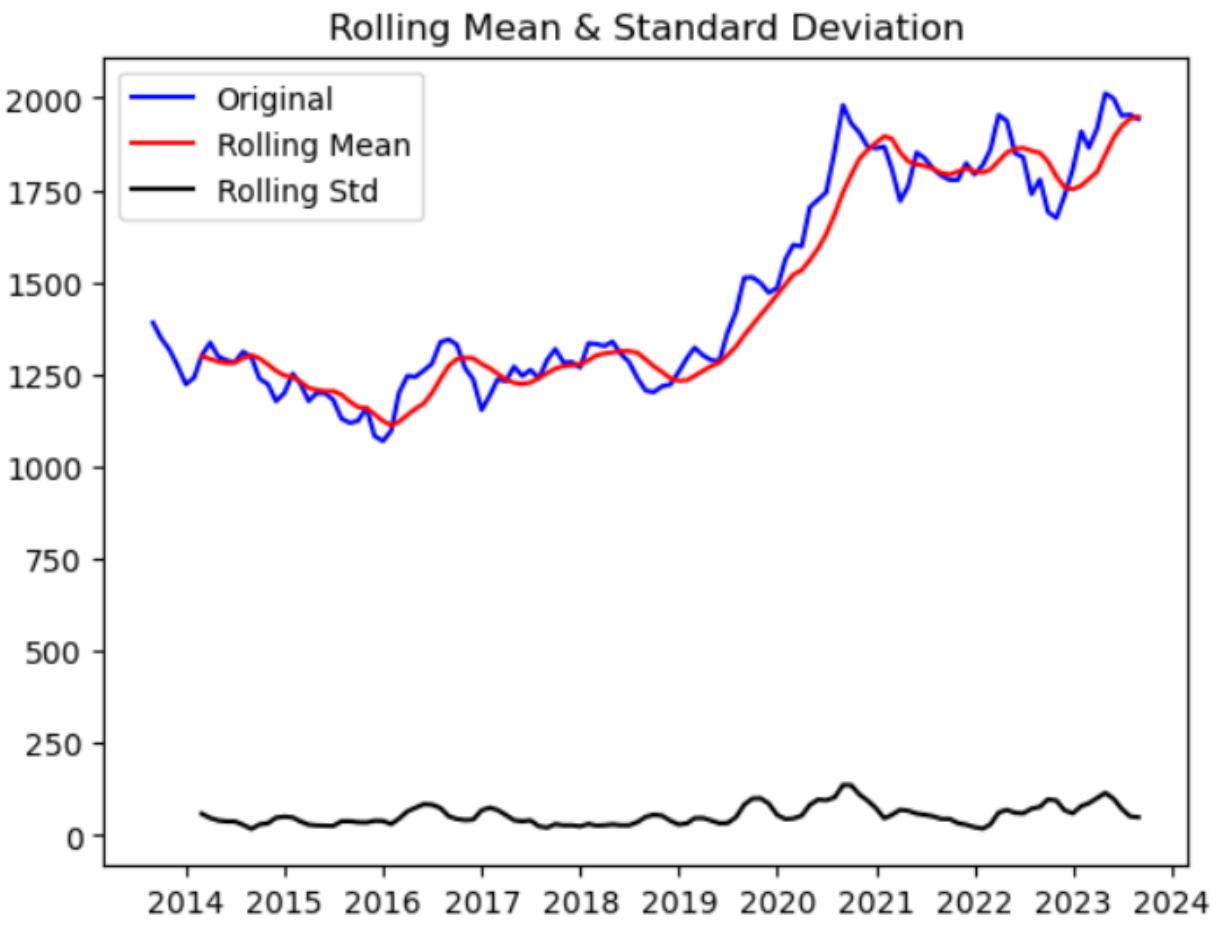
Problem 1.4 - Check for Stationarity

- Check for stationarity - Make the data stationary (if needed)

Solution:

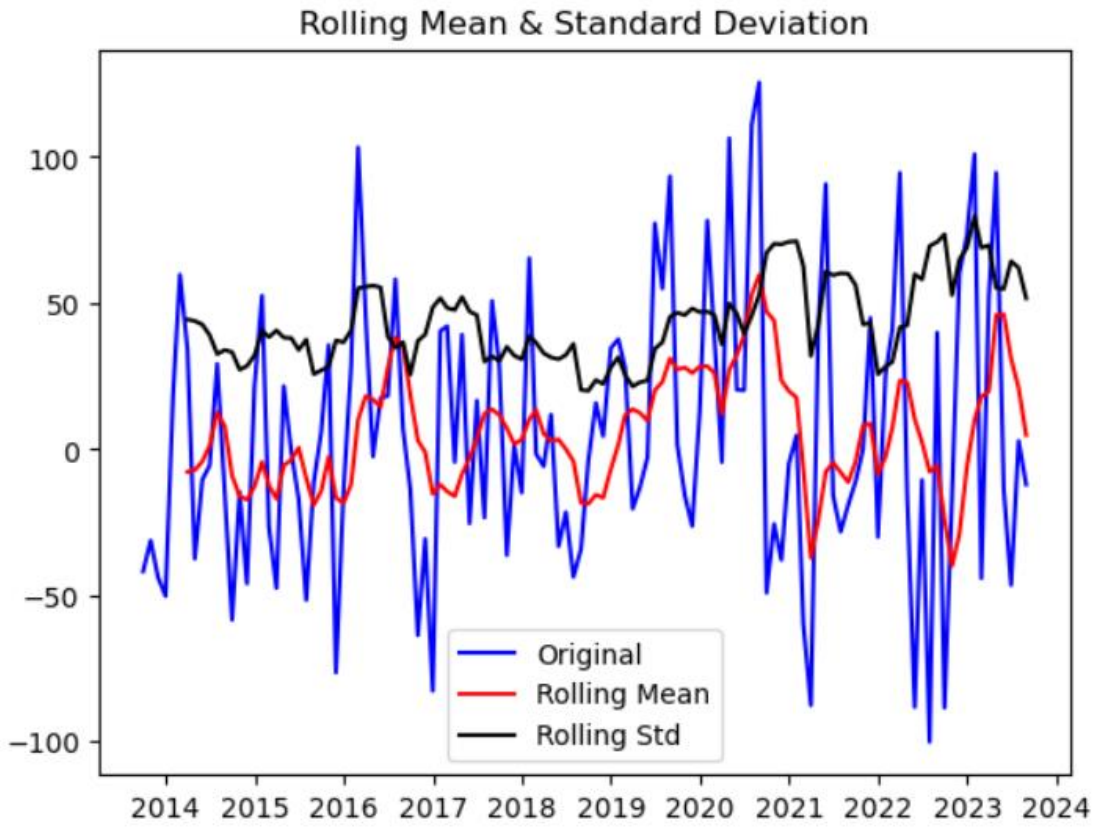
We use the Dickey-Fuller test to check the stationarity of the given time series. The p-value is seen to be >0.05 indicating that the time series is not stationary.

```
Results of Dickey-Fuller Test:
Test Statistic                  -0.568954
p-value                          0.877844
#Lags Used                       1.000000
Number of Observations Used    119.000000
Critical Value (1%)             -3.486535
Critical Value (5%)             -2.886151
Critical Value (10%)            -2.579896
dtype: float64
```



We will use a differencing level of 1 to make the data stationary. Let us look at the plot and the Dickey-Fuller test:

```
Results of Dickey-Fuller Test:
Test Statistic                -8.692706e+00
p-value                        4.000533e-14
#Lags Used                     0.000000e+00
Number of Observations Used    1.190000e+02
Critical Value (1%)           -3.486535e+00
Critical Value (5%)           -2.886151e+00
Critical Value (10%)          -2.579896e+00
dtype: float64
```



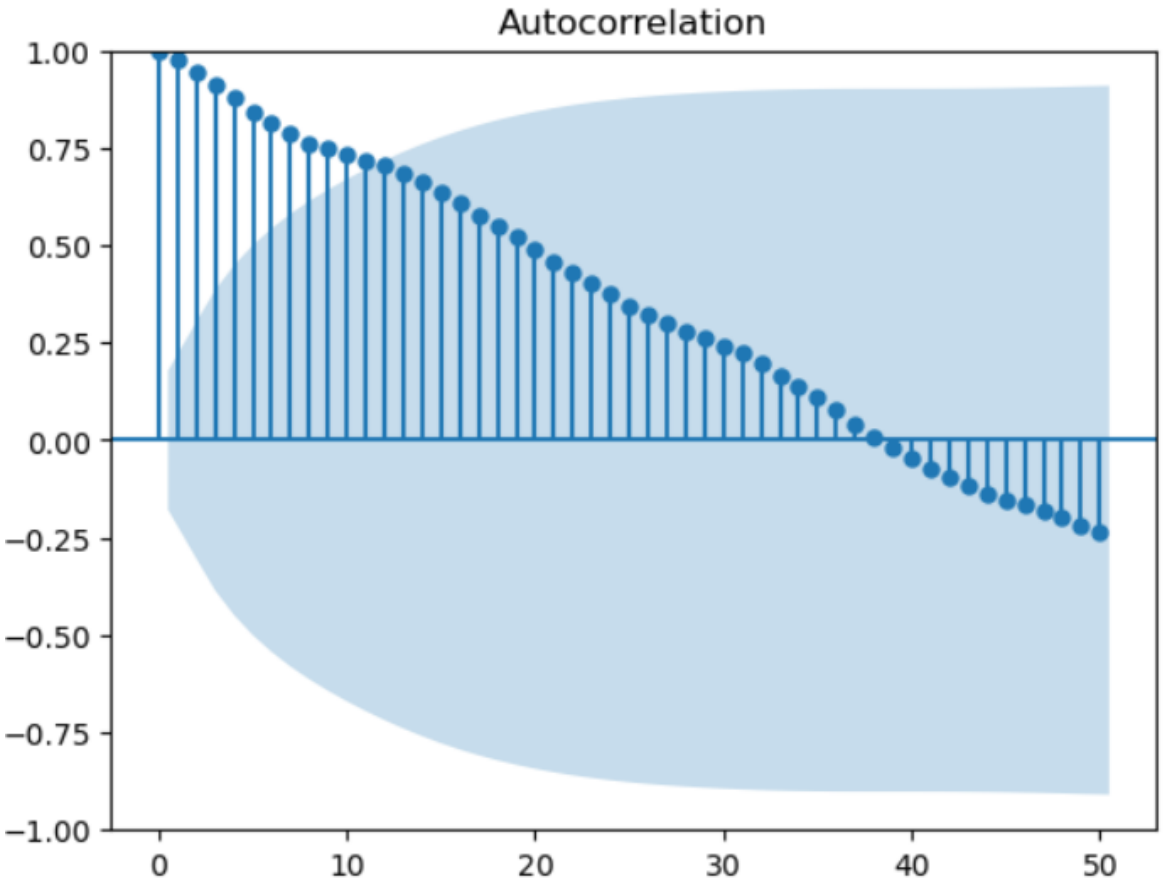
The p-value is now less than 0.05 – indicating that the time series is stationary.

Problem 1.5 - Model Building - Stationary Data

- Generate ACF & PACF Plot and find the AR, MA values. - Build different ARIMA models - Auto ARIMA - Manual ARIMA - Check the performance of the models built.

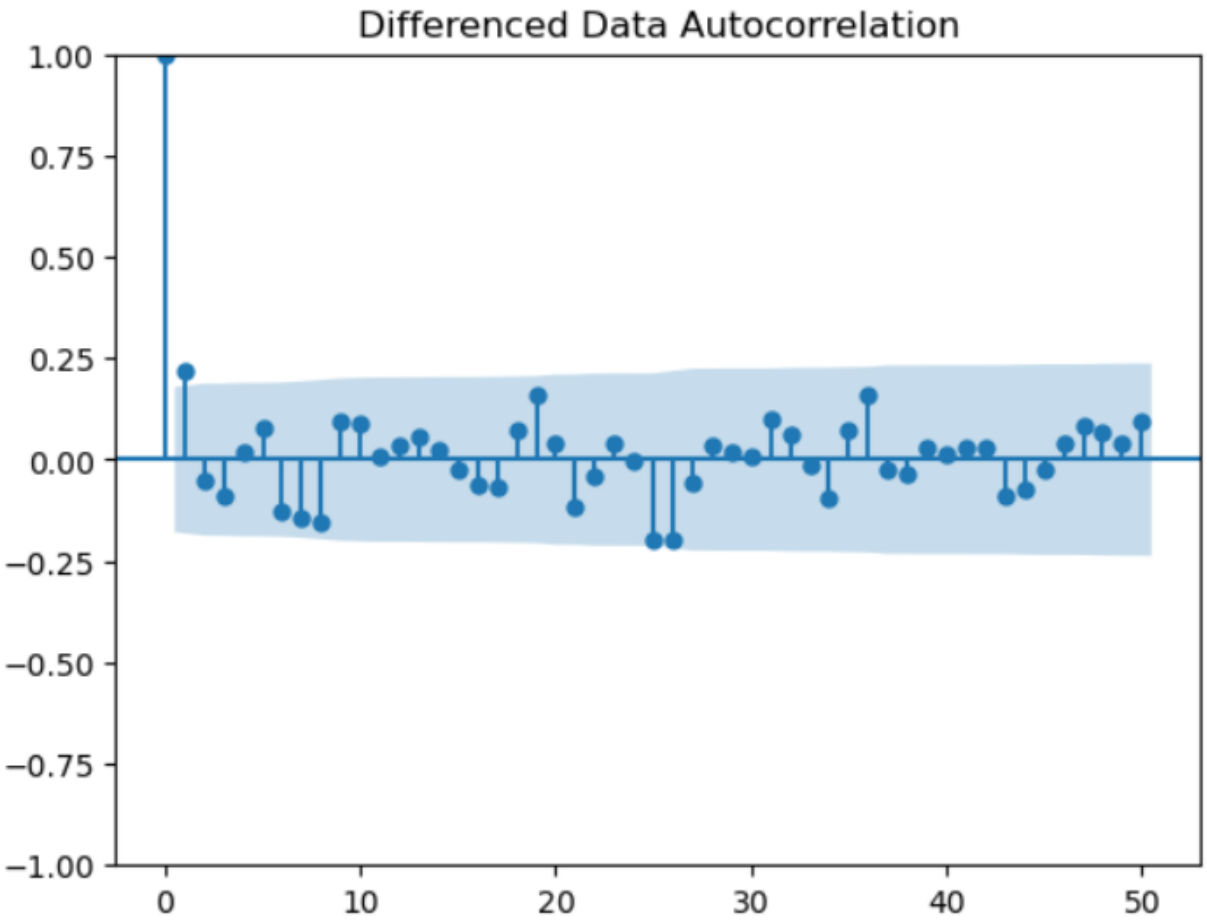
Solution:

ACF plot:



The ACF plot of the original data gives us an idea about the AR and MA values. We can take AR=0 since we are plotting on the original data. MA value as can be seen from the graph is = 11 which is quite impractical.

PACF plot:

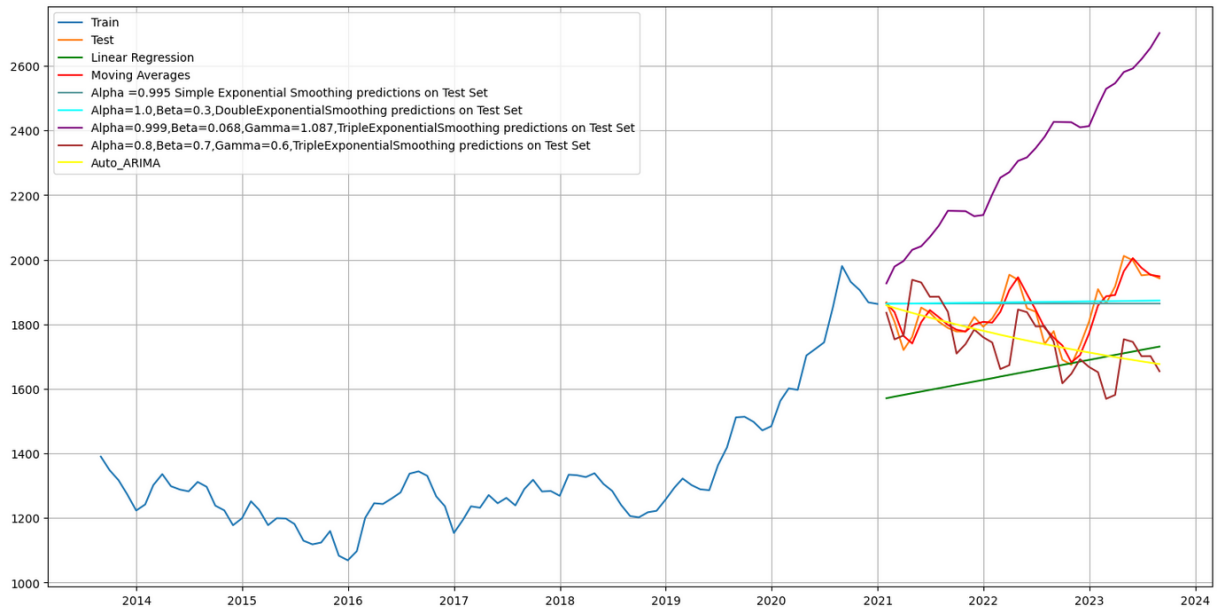


PACF plot contains the differenced data and we can easily take the AR (auto-regressive) value as 1 since we conducted a differencing of level 1 on it. The MA (moving averages) value seems to be 0.

Auto ARIMA Model:

The auto ARIMA model is created with the parameters being generated by the autofit function on the model. As we choose the best parameters by filtering out the lowest AIC we find alpha = 1, beta=0 and gamma = 1. We apply the model created by using the training data on the test data.

The plot and RMSE score of the same are as follows:



RMSE score -

```
rmse = mean_squared_error(test['Price'],predicted_auto_ARIMA,squared=False)
print(rmse)
```

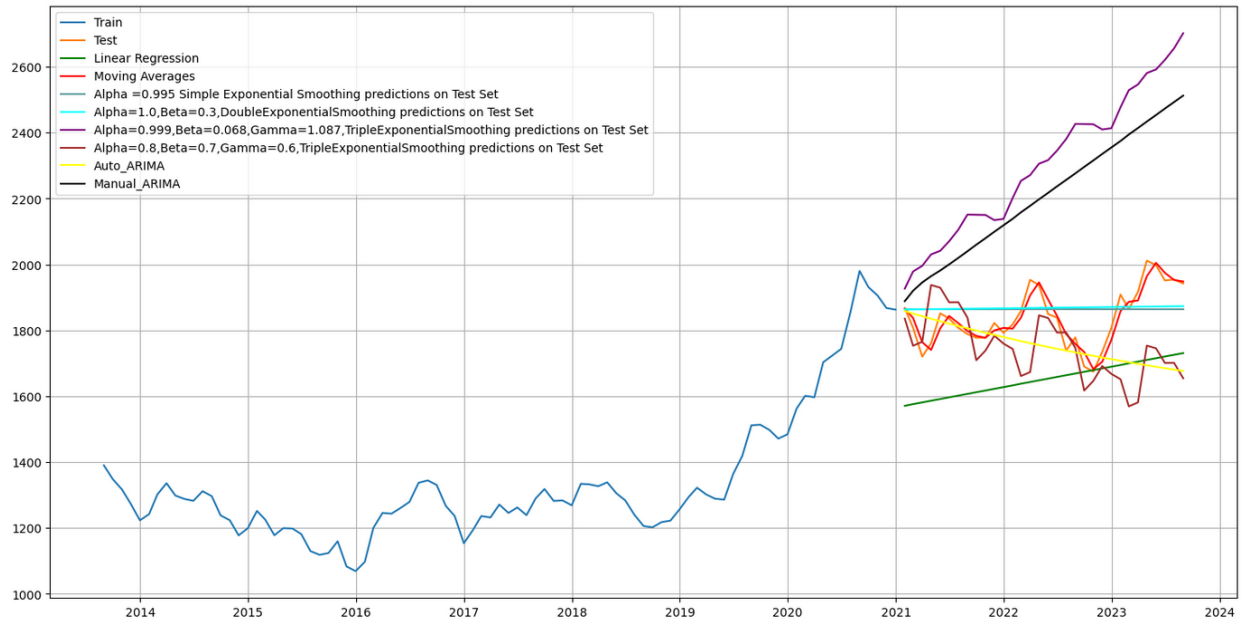
```
144.36208163638477
```

Manual ARIMA Model:

We define the values of p, d and q while creating the model via ARIMA.

p = (0,3) : defining the range of AR values based on the ACF & PACF plots
q = (0,1) : defining the range of AR values based on the ACF & PACF plots
d= (1,2) : defining the value to make the data stationary

Using parameters as alpha = 3, beta=2 and gamma = 1 at lowest AIC score, we build the model and apply it on the test dataset. The plot and the score of the same are as follows:



RMSE score:

```
rmse1 = mean_squared_error(test['Price'],predicted_ARIMA,squared=False)
print(rmse1)
```

```
402.7049689135044
```

The auto ARIMA model seems to have better performance than the manual ARIMA model. The mean squared error in the auto ARIMA model seems to the least among the two.

Problem 1.6 - Compare the performance of the models

Solution:

Given below is the RMSE scores of all models built till now with their RMSE scores sorted from lowest to highest. Our goal is to pick the model that will have the least error i.e., least RMSE score to guarantee a higher performance and good forecast.

	Test RMSE
2pointTrailingMovingAverage	27.958204
Alpha=1.0,Beta=0.3,DoubleExponentialSmoothing	44.907310
4pointTrailingMovingAverage	54.547384
6pointTrailingMovingAverage	70.810413
9pointTrailingMovingAverage	85.434070
Alpha=0.995,SimpleExponentialSmoothing	89.506086
Alpha=0.8,Beta=0.5,Gamma=0.5,TripleExponentialSmoothing	98.000340
Linear Regression	202.993632
Alpha=0.999,Beta=0.069,Gamma=0.005,TripleExponentialSmoothing	503.699889

We can see that the moving averages model has the lowest RMSE followed by the Double Exponential Smoothing model.

Final Model: We will pick the Double Exponential Smoothing model as our final model.

Rationale: although the moving averages model has a significantly lower RMSE score, we will void this model because it only takes the adjacent values around the data points to train and predict a forecast.

Thus, it fails to take into account a crucial factor while building the model – the cyclical nature of the times series of gold prices presented to us. The trend is based on the seasonality of how gold prices are projected throughout the months each year. Hence, the forecast will not be much reliable.

Rebuilding the model using entire data:

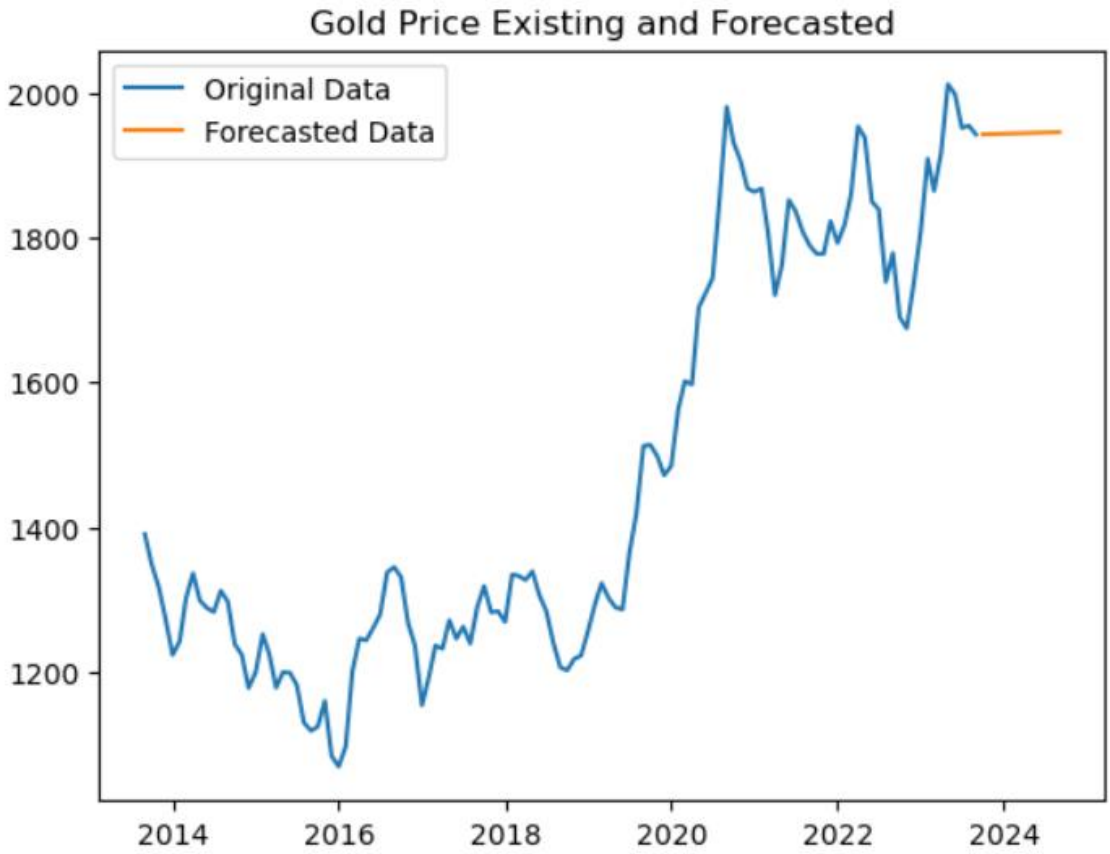
```
model_DES = Holt(df['Price'], initialization_method='estimated') # Complete the code to build the best model to forecast

model_DES_alpha_i_j = model_DES.fit(smoothing_level=1.0000000000000002,smoothing_trend=0.3,optimized=False,use_brute=True)

predicted_results_final = model_DES_alpha_i_j.forecast(steps=12) # Complete the code to forecast for the next 12 months

plt.plot(df['Price'], label = 'Original Data')
plt.plot(predicted_results_final, label = 'Forecasted Data')
plt.legend(loc ='best')
plt.title('Gold Price Existing and Forecasted');
```

The output comes out like this:



Although we cannot see the seasonality detailed in the above plot, the trend gives us an idea that just the gold prices have been fluctuating from 1700-2000 units the future 12 months in 2024 will do the same.

Problem 1.7 - Actionable Insights & Recommendations

- Conclude with the key takeaways (actionable insights and recommendations) for the business

Solution:

Based on the chart showing historical and forecasted gold prices, here are actionable insights and recommendations:

Insights:

Historical Growth Trend:

- Gold prices have shown significant volatility from 2013 to 2023, with steep increases, particularly around 2020-2021.
- The general trend reflects an upward movement, despite intermittent dips.

Stabilization in 2024:

- Forecasted prices for 2024 indicate stability or minimal growth compared to the sharp increases in earlier years.
- This could suggest a plateau after the rapid growth seen in prior years.

Market Context:

- Stabilized prices may indicate reduced uncertainty or global economic factors leading to a steady demand for gold.

Recommendations:

For Investors:

- With gold prices stabilizing, it may be prudent to diversify investments into other asset classes like equities or bonds to balance returns.
- Consider leveraging the current stability for short-term trading opportunities if volatility is low.
- Retain gold as a hedge against potential economic downturns or inflationary pressures, even if growth seems limited.

For Jewelers and Retailers:

- Utilize the price stability forecast for bulk purchases to lock in costs and plan promotions during predictable price points.
- Highlight the stability of gold prices to attract customers who view gold as a safe and reliable investment.

For Policy Makers and Analysts:

- Monitor the factors contributing to price stabilization, such as interest rates, inflation, or geopolitical stability, to inform economic policies.

- Educate citizens and investors about gold's long-term value and its role during periods of price stabilization.

For Industrial Users:

- Industries relying on gold (e.g., electronics) should leverage stable prices for cost-effective procurement.

- Explore substitutes for gold in manufacturing if the stabilized prices don't meet cost-efficiency targets.