Excercise_8.2_Time_Series_Modeling_RiazAhmedTamimAnsari

October 20, 2024

[ ]:
```
from google.colab import files
uploaded = files.upload()
```

```
<IPython.core.display.HTML object>
```

```
Saving us_retail_sales.csv to us_retail_sales.csv
```

Importing the required pandas and reading the input file.
It is a dataset with 13 columns and 30 rows of monthly sales data.

[ ]:
```
```

[ ]:
```
import pandas as pd

# loaded file is a CSV (for example, 'data.csv')
sales = pd.read_csv('us_retail_sales.csv')

# Display the first few rows of the DataFrame
print ("First few rows of dataframe\n")
print (sales.head())
print (sales)
```

```
First few rows of dataframe

   YEAR     JAN     FEB     MAR     APR     MAY     JUN       JUL       AUG  \
0  1992  146925  147223  146805  148032  149010  149800  150761.0  151067.0
1  1993  157555  156266  154752  158979  160605  160127  162816.0  162506.0
2  1994  167518  169649  172766  173106  172329  174241  174781.0  177295.0
3  1995  182413  179488  181013  181686  183536  186081  185431.0  186806.0
4  1996  189135  192266  194029  194744  196205  196136  196187.0  196218.0

        SEP       OCT       NOV       DEC
0  152588.0  153521.0  153583.0  155614.0
1  163258.0  164685.0  166594.0  168161.0
2  178787.0  180561.0  180703.0  181524.0
3  187366.0  186565.0  189055.0  190774.0
4  198859.0  200509.0  200174.0  201284.0
```

[ ]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG
0	1992	146925	147223	146805	148032	149010	149800	150761.0	151067.0
1	1993	157555	156266	154752	158979	160605	160127	162816.0	162506.0
2	1994	167518	169649	172766	173106	172329	174241	174781.0	177295.0
3	1995	182413	179488	181013	181686	183536	186081	185431.0	186806.0
4	1996	189135	192266	194029	194744	196205	196136	196187.0	196218.0
5	1997	202371	204286	204990	203399	201699	204675	207014.0	207635.0
6	1998	209666	209552	210832	213633	214639	216337	214841.0	213636.0
7	1999	223997	226250	227417	229037	231235	231903	233948.0	236566.0
8	2000	243436	247133	249825	245831	246201	248160	247176.0	247576.0
9	2001	252654	252704	250328	254763	255218	254022	252997.0	254560.0
10	2002	256307	257670	257059	261333	257573	259786	262769.0	265043.0
11	2003	267230	263188	267820	267197	267362	270396	273352.0	277965.0
12	2004	278913	280932	286209	282952	288252	284133	287358.0	287941.0
13	2005	296696	300557	301308	303760	301776	310989	313520.0	310046.0
14	2006	322348	320171	320869	322561	321794	323184	324204.0	325324.0
15	2007	327181	327953	330579	329560	334202	331076	332342.0	334169.0
16	2008	337412	334584	335193	334843	337947	338311	336771.0	334045.0
17	2009	298673	297631	292300	293614	296501	302169	302802.0	309023.0
18	2010	308299	308628	316003	318707	315604	314925	315632.0	317408.0
19	2011	332357	334710	338007	339884	339303	341600	341373.0	342288.0
20	2012	352862	357379	358719	356849	356018	352043	353891.0	358450.0
21	2013	367009	372291	369081	367514	369493	371041	373554.0	372489.0
22	2014	373033	378581	382601	386689	387100	388106	388359.0	391305.0
23	2015	385648	385157	391420	391356	394718	395464	398193.0	398105.0
24	2016	394749	398105	396911	398190	400143	404756	403730.0	403968.0
25	2017	416081	415503	414620	416889	414540	416505	416744.0	417179.0
26	2018	432148	434106	433232	435610	439996	438191	440703.0	439278.0
27	2019	440751	439996	447167	448709	449552	450927	454012.0	456500.0
28	2020	460586	459610	434281	379892	444631	476343	481627.0	483716.0
29	2021	520162	504458	559871	562269	548987	550782	NaN	NaN

	SEP	OCT	NOV	DEC
0	152588.0	153521.0	153583.0	155614.0
1	163258.0	164685.0	166594.0	168161.0
2	178787.0	180561.0	180703.0	181524.0
3	187366.0	186565.0	189055.0	190774.0
4	198859.0	200509.0	200174.0	201284.0
5	208326.0	208078.0	208936.0	209363.0
6	215720.0	219483.0	221134.0	223179.0
7	237481.0	237553.0	240544.0	245485.0
8	251837.0	251221.0	250331.0	250658.0
9	249845.0	267999.0	260514.0	256549.0
10	260626.0	261953.0	263568.0	265930.0
11	276430.0	274764.0	278298.0	277612.0
12	293139.0	295115.0	296177.0	299763.0
13	310673.0	310479.0	313303.0	313473.0

```
14  323236.0  322678.0  323343.0  326849.0
15  335442.0  337530.0  341133.0  336189.0
16  328343.0  314830.0  301332.0  294025.0
17  301033.0  304154.0  306675.0  308413.0
18  320080.0  323900.0  327745.0  329627.0
19  345496.0  347924.0  349304.0  349744.0
20  361470.0  361991.0  362876.0  364488.0
21  372505.0  373663.0  373914.0  377032.0
22  389860.0  390506.0  391805.0  388569.0
23  396248.0  394503.0  396240.0  397052.0
24  405958.0  407395.0  406061.0  412610.0
25  426501.0  426933.0  431158.0  433282.0
26  438985.0  444038.0  445242.0  434803.0
27  452849.0  455486.0  457658.0  458055.0
28  493327.0  493991.0  488652.0  484782.0
29       NaN       NaN       NaN       NaN
```

As we were able to find NAN values in the last row, mean of that year (2021) has been found out and NAN values were filled up.

[ ]:
```
print ("The number of NA values in the dataset are\n")
print (sales.isna().sum())
#df['A'].fillna(df['A'].mean(), inplace=True)
sales_mean=sales.fillna(sales.mean(axis=1)[29:][29])
sales_mean
```

```
The number of NA values in the dataset are

YEAR    0
JAN     0
FEB     0
MAR     0
APR     0
MAY     0
JUN     0
JUL     1
AUG     1
SEP     1
OCT     1
NOV     1
DEC     1
dtype: int64
```

[ ]:
```
    YEAR     JAN     FEB     MAR     APR     MAY     JUN            JUL  \
0   1992  146925  147223  146805  148032  149010  149800  150761.000000
1   1993  157555  156266  154752  158979  160605  160127  162816.000000
2   1994  167518  169649  172766  173106  172329  174241  174781.000000
```

3	1995	182413	179488	181013	181686	183536	186081	185431.000000
4	1996	189135	192266	194029	194744	196205	196136	196187.000000
5	1997	202371	204286	204990	203399	201699	204675	207014.000000
6	1998	209666	209552	210832	213633	214639	216337	214841.000000
7	1999	223997	226250	227417	229037	231235	231903	233948.000000
8	2000	243436	247133	249825	245831	246201	248160	247176.000000
9	2001	252654	252704	250328	254763	255218	254022	252997.000000
10	2002	256307	257670	257059	261333	257573	259786	262769.000000
11	2003	267230	263188	267820	267197	267362	270396	273352.000000
12	2004	278913	280932	286209	282952	288252	284133	287358.000000
13	2005	296696	300557	301308	303760	301776	310989	313520.000000
14	2006	322348	320171	320869	322561	321794	323184	324204.000000
15	2007	327181	327953	330579	329560	334202	331076	332342.000000
16	2008	337412	334584	335193	334843	337947	338311	336771.000000
17	2009	298673	297631	292300	293614	296501	302169	302802.000000
18	2010	308299	308628	316003	318707	315604	314925	315632.000000
19	2011	332357	334710	338007	339884	339303	341600	341373.000000
20	2012	352862	357379	358719	356849	356018	352043	353891.000000
21	2013	367009	372291	369081	367514	369493	371041	373554.000000
22	2014	373033	378581	382601	386689	387100	388106	388359.000000
23	2015	385648	385157	391420	391356	394718	395464	398193.000000
24	2016	394749	398105	396911	398190	400143	404756	403730.000000
25	2017	416081	415503	414620	416889	414540	416505	416744.000000
26	2018	432148	434106	433232	435610	439996	438191	440703.000000
27	2019	440751	439996	447167	448709	449552	450927	454012.000000
28	2020	460586	459610	434281	379892	444631	476343	481627.000000
29	2021	520162	504458	559871	562269	548987	550782	464078.571429

	AUG	SEP	OCT	NOV	DEC
0	151067.000000	152588.000000	153521.000000	153583.000000	155614.000000
1	162506.000000	163258.000000	164685.000000	166594.000000	168161.000000
2	177295.000000	178787.000000	180561.000000	180703.000000	181524.000000
3	186806.000000	187366.000000	186565.000000	189055.000000	190774.000000
4	196218.000000	198859.000000	200509.000000	200174.000000	201284.000000
5	207635.000000	208326.000000	208078.000000	208936.000000	209363.000000
6	213636.000000	215720.000000	219483.000000	221134.000000	223179.000000
7	236566.000000	237481.000000	237553.000000	240544.000000	245485.000000
8	247576.000000	251837.000000	251221.000000	250331.000000	250658.000000
9	254560.000000	249845.000000	267999.000000	260514.000000	256549.000000
10	265043.000000	260626.000000	261953.000000	263568.000000	265930.000000
11	277965.000000	276430.000000	274764.000000	278298.000000	277612.000000
12	287941.000000	293139.000000	295115.000000	296177.000000	299763.000000
13	310046.000000	310673.000000	310479.000000	313303.000000	313473.000000
14	325324.000000	323236.000000	322678.000000	323343.000000	326849.000000
15	334169.000000	335442.000000	337530.000000	341133.000000	336189.000000
16	334045.000000	328343.000000	314830.000000	301332.000000	294025.000000
17	309023.000000	301033.000000	304154.000000	306675.000000	308413.000000

18	317408.000000	320080.000000	323900.000000	327745.000000	329627.000000
19	342288.000000	345496.000000	347924.000000	349304.000000	349744.000000
20	358450.000000	361470.000000	361991.000000	362876.000000	364488.000000
21	372489.000000	372505.000000	373663.000000	373914.000000	377032.000000
22	391305.000000	389860.000000	390506.000000	391805.000000	388569.000000
23	398105.000000	396248.000000	394503.000000	396240.000000	397052.000000
24	403968.000000	405958.000000	407395.000000	406061.000000	412610.000000
25	417179.000000	426501.000000	426933.000000	431158.000000	433282.000000
26	439278.000000	438985.000000	444038.000000	445242.000000	434803.000000
27	456500.000000	452849.000000	455486.000000	457658.000000	458055.000000
28	483716.000000	493327.000000	493991.000000	488652.000000	484782.000000
29	464078.571429	464078.571429	464078.571429	464078.571429	464078.571429

The dataset is in wide form.
For applying the model, it has been converted to tall form by using melt function and mapping function to give the numerical values to months.
A new derived variable by the name Date has been created by combining the month and year.

[ ]:
```
# Melt the DataFrame

month_map = {'JAN': 1, 'FEB': 2, 'MAR': 3, 'APR': 4, 'MAY': 5, 'JUN': 6,
             'JUL': 7, 'AUG': 8, 'SEP': 9, 'OCT': 10, 'NOV': 11, 'DEC': 12}


sales_melted = pd.melt(sales_mean, id_vars=['YEAR'], var_name='Month',␣
 ↪value_name='Value')
sales_melted['Month'] = sales_melted['Month'].map(month_map)


print("\nMelted DataFrame:")
print(sales_melted)
# Combine YEAR and Month to create a datetime column
sales_melted['Date'] = pd.to_datetime(sales_melted[['YEAR', 'Month']].
 ↪assign(DAY=1))
sales_melted
```

```
Melted DataFrame:
     YEAR  Month          Value
0    1992      1  146925.000000
1    1993      1  157555.000000
2    1994      1  167518.000000
3    1995      1  182413.000000
4    1996      1  189135.000000
..    …    …            …
355  2017     12  433282.000000
356  2018     12  434803.000000
357  2019     12  458055.000000
```

```
358  2020     12  484782.000000
359  2021     12  464078.571429

[360 rows x 3 columns]
```

[ ]:
```
      YEAR  Month          Value       Date
0     1992      1  146925.000000 1992-01-01
1     1993      1  157555.000000 1993-01-01
2     1994      1  167518.000000 1994-01-01
3     1995      1  182413.000000 1995-01-01
4     1996      1  189135.000000 1996-01-01
..     …    …             …         …
355   2017     12  433282.000000 2017-12-01
356   2018     12  434803.000000 2018-12-01
357   2019     12  458055.000000 2019-12-01
358   2020     12  484782.000000 2020-12-01
359   2021     12  464078.571429 2021-12-01

[360 rows x 4 columns]
```

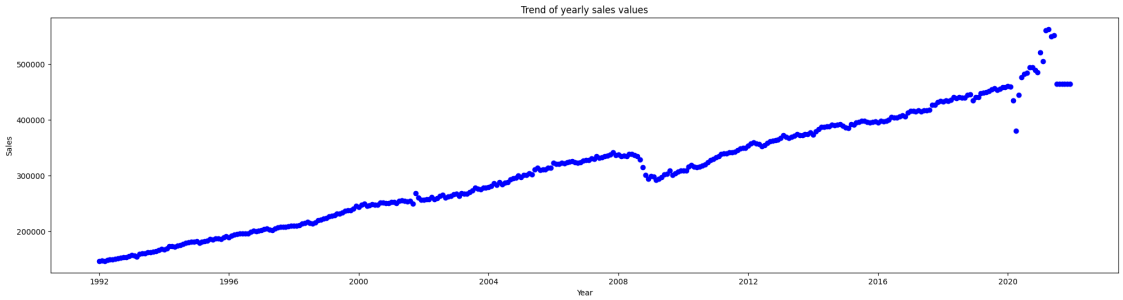
1 Visualization

Below plot shows the trend of the sales. Clearly, it is an increasing trend.

[ ]:
```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd

plt.figure(figsize=(25, 6))
plt.plot(sales_melted['Date'], sales_melted['Value'], marker='o',␣
 ↪label='Monthly Values', color='blue',linestyle='None')
plt.title('Trend of yearly sales values')
plt.xlabel('Year')
plt.ylabel('Sales')
```

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



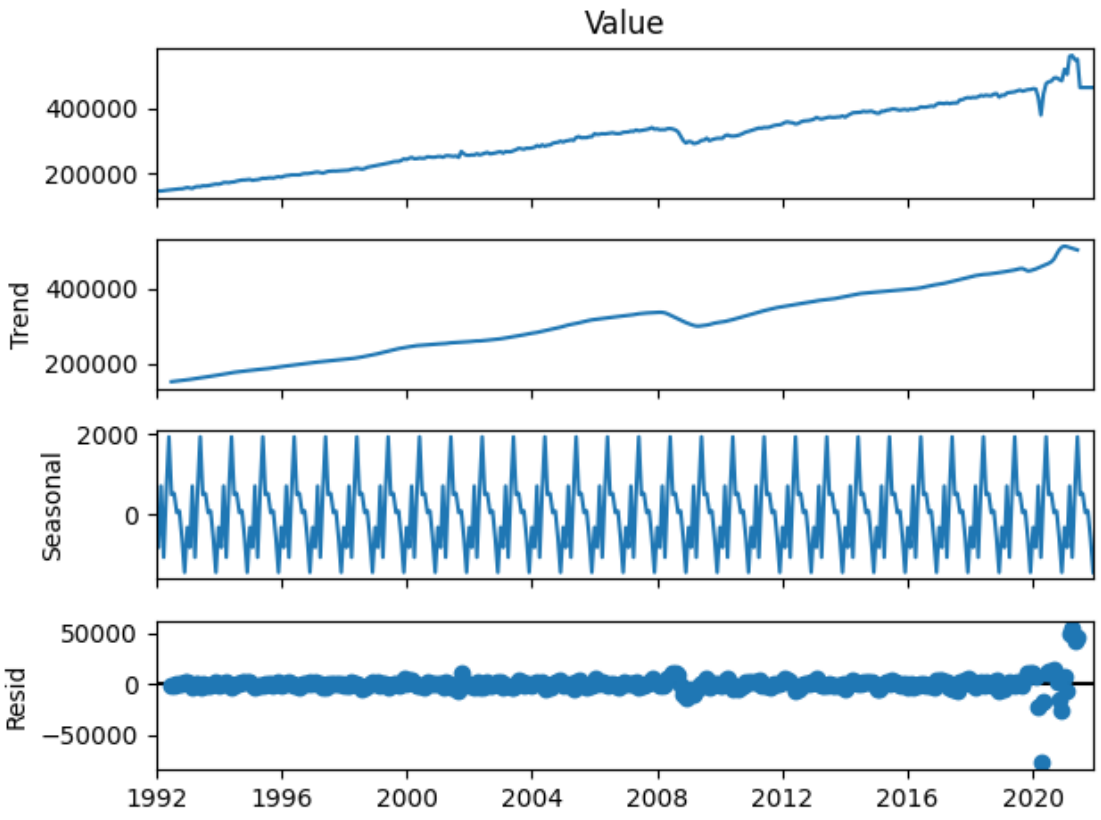
The dataset has also been checked for seasonality by using the seasonal_decompose function.
As the seasonality is simple and does not change in magnitude for year on year, it has additive seasonality component.

[ ]:
```
sales_reformat=sales_melted.set_index('Date')
sales_reformatted = sales_reformat.drop(columns=['YEAR','Month'])

sales_sorted=sales_reformatted.sort_index()

from statsmodels.tsa.seasonal import seasonal_decompose

# Perform Seasonal Decomposition
result = seasonal_decompose(sales_sorted['Value'],period=12)
result.plot()
plt.show()
```



From the above Trend, Seasonality and Residual graph, it is clear that there is a upward additive Trend and a additive seasonality observed .
The reason for choosing additive for both the components is the Trend curve is linear, not exponential.

Same case is seen for seasonality curve, the magnitude is constant.

2 Model Selection

The Holt-Winters Exponential Smoothing (Triple Exponential Smoothing) is being chosen here as both the trend and seasonality is not complex. The data has been split based on the index label for training and testing.

[ ]:
```
sales_sorted_train=sales_sorted.loc[:'2020-07-01']
sales_sorted_test=sales_sorted.loc['2020-08-01':'2021-06-01']

from statsmodels.tsa.holtwinters import ExponentialSmoothing

model = ExponentialSmoothing(sales_sorted_train, trend="add", seasonal="add",␣
 ↪seasonal_periods=12)
fit = model.fit()
print (fit.summary())
print (sales_sorted_train)
print (sales_sorted_test)
```

```
/usr/local/lib/python3.10/dist-packages/statsmodels/tsa/base/tsa_model.py:473:
ValueWarning: No frequency information was provided, so inferred frequency MS
will be used.
  self._init_dates(dates, freq)

                       ExponentialSmoothing Model Results
==============================================================================
Dep. Variable:                      Value   No. Observations:                  343
Model:             ExponentialSmoothing   SSE                 11838494513.971
Optimized:                         True   AIC                          5985.414
Trend:                         Additive   BIC                          6046.818
Seasonal:                      Additive   AICC                         5987.525
Seasonal Periods:                    12   Date:                 Sat, 19 Oct 2024
Box-Cox:                          False   Time:                         13:57:25
Box-Cox Coeff.:                    None
==============================================================================
=
                            coeff                 code              optimized
------------------------------------------------------------------------------
-
smoothing_level            0.9596429              alpha
True
smoothing_trend               0.0001               beta
True
smoothing_seasonal         0.0242143              gamma
True
initial_level             1.4979e+05                1.0
True
```

```
initial_trend                 887.75783                 b.0
True
initial_seasons.0            -77.327257                 s.0
True
initial_seasons.1            -758.83767                 s.1
True
initial_seasons.2            -488.40017                 s.2
True
initial_seasons.3             28.901910                 s.3
True
initial_seasons.4             94.120660                 s.4
True
initial_seasons.5             110.56858                 s.5
True
initial_seasons.6            -28.493924                 s.6
True
initial_seasons.7             33.870660                 s.7
True
initial_seasons.8             154.00608                 s.8
True
initial_seasons.9             8.7560764                 s.9
True
initial_seasons.10            181.30816                s.10
True
initial_seasons.11            741.52691                s.11
True
--------------------------------------------------------------------------------
-
               Value
Date
1992-01-01  146925.0
1992-02-01  147223.0
1992-03-01  146805.0
1992-04-01  148032.0
1992-05-01  149010.0
…              …
2020-03-01  434281.0
2020-04-01  379892.0
2020-05-01  444631.0
2020-06-01  476343.0
2020-07-01  481627.0

[343 rows x 1 columns]
               Value
Date
2020-08-01  483716.0
2020-09-01  493327.0
2020-10-01  493991.0
```

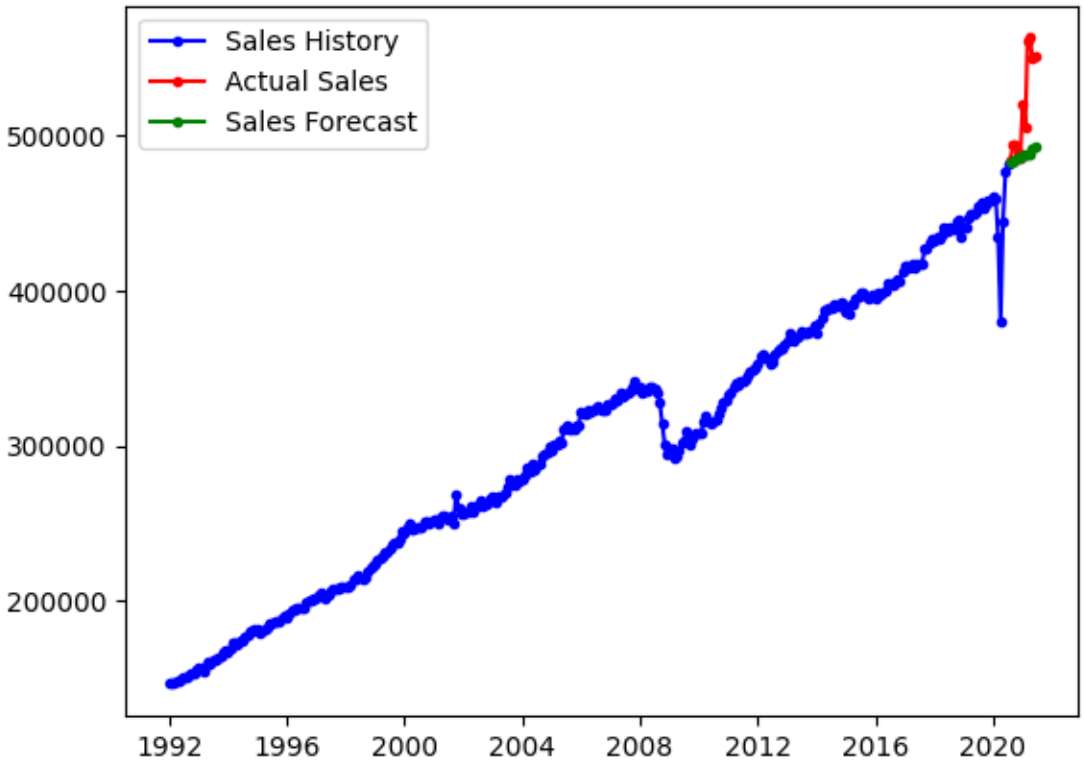
```
2020-11-01  488652.0
2020-12-01  484782.0
2021-01-01  520162.0
2021-02-01  504458.0
2021-03-01  559871.0
2021-04-01  562269.0
2021-05-01  548987.0
2021-06-01  550782.0
```

```
/usr/local/lib/python3.10/dist-
packages/statsmodels/tsa/holtwinters/model.py:918: ConvergenceWarning:
Optimization failed to converge. Check mle_retvals.
  warnings.warn(
```

After fitting the model, forecasting is done and the graphs are plotted.

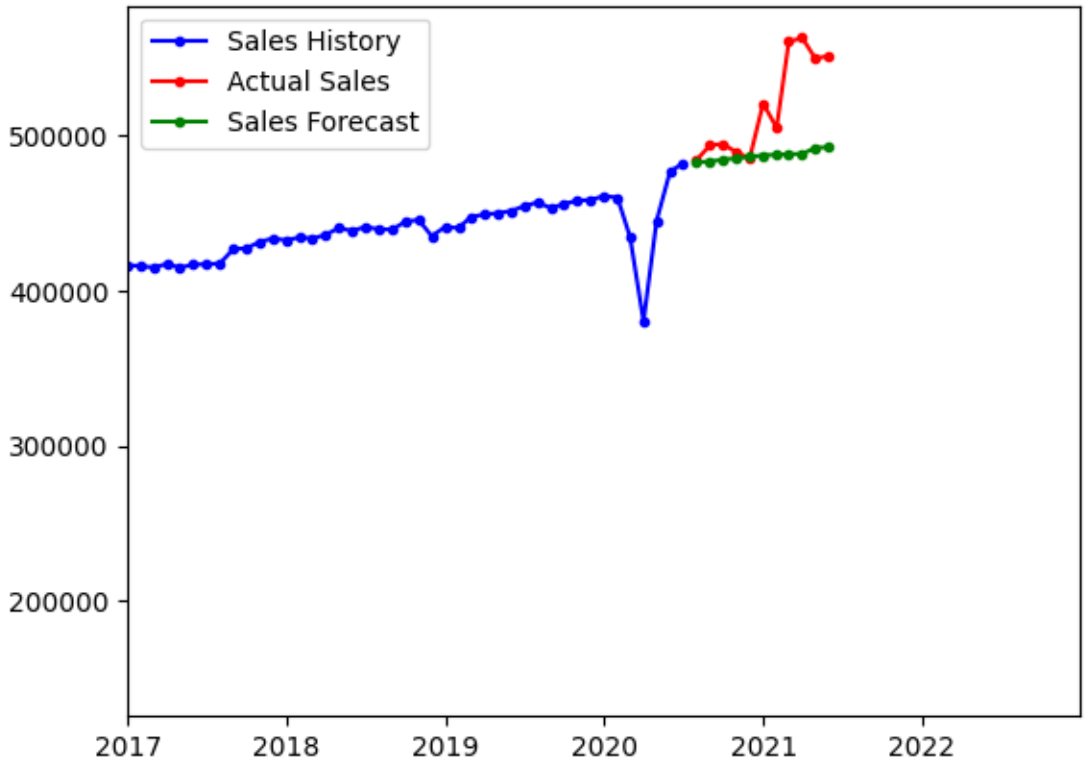
[ ]:
```
sales_forecast = fit.forecast(steps=11)

fig = plt.figure()
past, = plt.plot(sales_sorted_train.index, sales_sorted_train, 'b.-',␣
 ↪label='Sales History')
future, = plt.plot(sales_sorted_test.index, sales_sorted_test, 'r.-',␣
 ↪label='Actual Sales')
predicted_future, = plt.plot(sales_sorted_test.index, sales_forecast, 'g.-',␣
 ↪label='Sales Forecast')
plt.legend(handles=[past, future, predicted_future])
plt.show()
```



The zoomed in graph is also plotted to get a closer look into the forecasting.
As seen from the below legend, blue is the sales history, red actual sales and green being sales forecast. There is a slight difference between the sales forecast and the actual sales. Actual sales is found to be a lot higher than what got forecasted by the model.

[ ]:
```
fig = plt.figure()
past, = plt.plot(sales_sorted_train.index, sales_sorted_train, 'b.-',␣
 ↪label='Sales History')
future, = plt.plot(sales_sorted_test.index, sales_sorted_test, 'r.-',␣
 ↪label='Actual Sales')
predicted_future, = plt.plot(sales_sorted_test.index, sales_forecast, 'g.-',␣
 ↪label='Sales Forecast')
plt.legend(handles=[past, future, predicted_future])
plt.xlim('2017-01-01', '2022-12-31')
plt.show()
```



Following are the results of MAE and RMSE

[ ]:
```
import pandas as pd
from statsmodels.tsa.holtwinters import ExponentialSmoothing
from sklearn.metrics import mean_absolute_error, mean_squared_error
import numpy as np

# Calculate MAE, RMSE, and MAPE for both models
mae_add = mean_absolute_error(sales_sorted_test, sales_forecast)

rmse_add = np.sqrt(mean_squared_error(sales_sorted_test, sales_forecast))

# Print the evaluation metrics
print(f"Additive Model - MAE: {mae_add}, RMSE: {rmse_add}")
```

```
Additive Model - MAE: 30963.968037992352, RMSE: 41769.420692112646
```

3 Conclusion

Data has been imported to panda and NAN values were filled up with the mean of sales values of that year.
Visualization has been done to find the trend and seasonality components. The data has been split

to train and test.
Holts Winters method has been chosen and prediction made.
The visualization graphs are plotted and the MAE/RMSE values are also computed.
Additive Model - MAE: 30963.968037992352, RMSE: 41769.420692112646