[1]:

Stock Prediction Analysis

July 21, 2024

[2]:
```
# Imports required packages

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
import datetime as datetime
import plotly.express as px
import statsmodels.api as sm



pd.set_option('display.max_columns', 50)
pd.set_option('display.max_colwidth', None)
pd.set_option("display.max_rows", 100)
import warnings
warnings.filterwarnings('ignore')
```

[3]:
```
sales_df = pd.read_csv("us_retail_sales.csv")

sales_df.head()
```

[3]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV	DEC
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				

```
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
```

[4]:
```
sales_df.tail()
```

[4]:
```
    YEAR     JAN     FEB     MAR     APR     MAY     JUN       JUL       AUG  \
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
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
```

[5]:
```
sales_df.YEAR.min(), sales_df.YEAR.max()
```

[5]:
```
(1992, 2021)
```

[6]:
```
sales_df.describe()
```

[6]:
```
              YEAR            JAN            FEB            MAR  \
count    30.000000      30.000000      30.000000      30.000000
mean   2006.500000  304803.833333  305200.900000  307533.566667
std       8.803408   97687.399232   96682.043053  100002.422696
min    1992.000000  146925.000000  147223.000000  146805.000000
25%    1999.250000  228856.750000  231470.750000  233019.000000
50%    2006.500000  303486.000000  304592.500000  308655.500000
75%    2013.750000  371527.000000  377008.500000  379221.000000
max    2021.000000  520162.000000  504458.000000  559871.000000

                APR            MAY            JUN            JUL  \
count     30.000000      30.000000      30.000000      29.000000
mean  306719.600000  309205.633333  311406.966667  304375.448276
std    98207.161171   99541.010078  101057.212178   92471.103673
min   148032.000000  149010.000000  149800.000000  150761.000000
25%   233235.500000  234976.500000  235967.250000  233948.000000
50%   311233.500000  308690.000000  312957.000000  313520.000000
75%   376797.500000  382698.250000  383839.750000  373554.000000
max   562269.000000  548987.000000  550782.000000  481627.000000
```

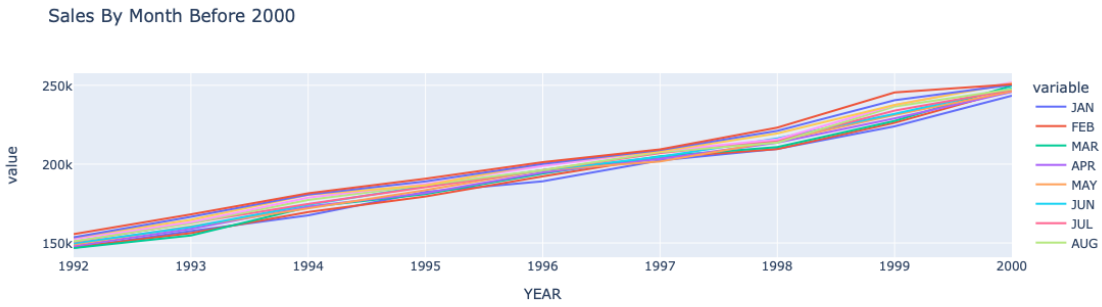
```
                AUG            SEP           OCT            NOV  \
count     29.000000      29.000000      29.00000      29.000000
mean   305451.965517  306078.206897  307310.62069  307794.896552
std     92504.808195   93008.417392   92836.64419   92702.551770
min    151067.000000  152588.000000  153521.00000  153583.000000
25%    236566.000000  237481.000000  237553.00000  240544.000000
50%    310046.000000  310673.000000  310479.00000  306675.000000
75%    372489.000000  372505.000000  373663.00000  373914.000000
max    483716.000000  493327.000000  493991.00000  488652.000000

                DEC
count     29.000000
mean   308099.620690
std     91784.061634
min    155614.000000
25%    245485.000000
50%    308413.000000
75%    377032.000000
max    484782.000000
```

[7]:
```
sales_df.columns
```

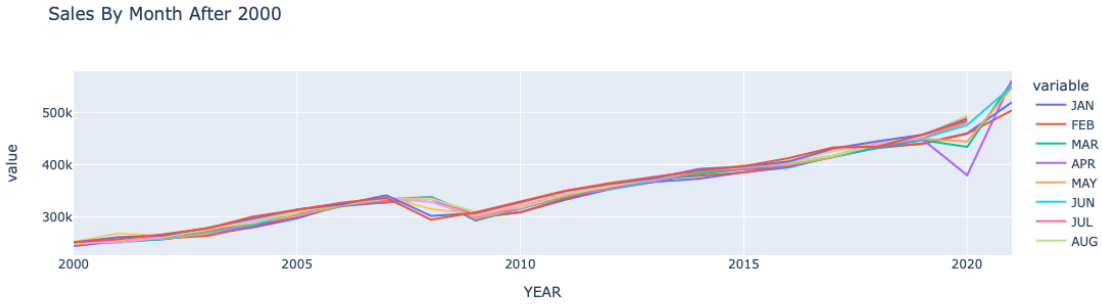
[7]:
```
Index(['YEAR', 'JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP',
       'OCT', 'NOV', 'DEC'],
      dtype='object')
```

1 - Plot the data with proper labeling and make some observations on the graph.

[8]:
```
# Displays the sales average of each month through the first half of the dataset

fig = px.line(sales_df[sales_df.YEAR <=2000], x='YEAR',
y = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT',␣
 ↪'NOV', 'DEC'],
title = "Sales By Month Before 2000")

fig.show()
```



[9]:
```
# Displays the sales average of each month through the second half of the dataset

fig = px.line(sales_df[sales_df.YEAR >=2000], x='YEAR',
y = ['JAN', 'FEB', 'MAR', 'APR', 'MAY', 'JUN', 'JUL', 'AUG', 'SEP', 'OCT', 'NOV', 'DEC'],
title = "Sales By Month After 2000")

fig.show()
```



The first graph shows a steady increase over time throughout 1992-2000 where profits began just over 150 thousand and finished just under the 250 thousand mark. It is also noticeable that the month of December is a mainstay as the most profitable month throughout the graph. The second graph showcases a much less predictable pattern. The period of 2007-2008 saw a sizable dip in the market that didn't correct back to an upward trajectory until 2010. Overall in the second graph profits doubled as they started near the 250k mark but ended closer to the 500-575k mark. One particular outlier noticed is April 2020 which is the only month throughout the entire dataset that saw an average of 100k below market value compared to other months of the same year.

2- Split this data into a training and test set. Use the last year of data (July 2020 – June 2021) of data as your test set and the rest as your training set.

[10]:
```
# Selects the columns to utilize (all except for YEAR)

melted_df = pd.melt(sales_df, id_vars = 'YEAR', value_vars = sales_df.columns[1:
 ↪13])

melted_df.head()
```

[10]:

	YEAR	variable	value
0	1992	JAN	146925.0
1	1993	JAN	157555.0
2	1994	JAN	167518.0
3	1995	JAN	182413.0
4	1996	JAN	189135.0

[11]:
```
# Creates the DATE column in month/day/year format

melted_df['Date'] = pd.to_datetime(melted_df['YEAR'].astype(str) +␣
 ↪melted_df['variable'], format = '%Y%b')
melted_df['Date'] = melted_df['Date'].dt.strftime('%m%d%Y')

melted_df.head()
```

[11]:

	YEAR	variable	value	Date
0	1992	JAN	146925.0	01011992
1	1993	JAN	157555.0	01011993
2	1994	JAN	167518.0	01011994
3	1995	JAN	182413.0	01011995
4	1996	JAN	189135.0	01011996

[12]:
```
# Drops the original columns from the melted set

melted_df = melted_df.drop(["YEAR", "variable"], axis = 1)

melted_df.head()
```

[12]:

	value	Date
0	146925.0	01011992
1	157555.0	01011993
2	167518.0	01011994
3	182413.0	01011995
4	189135.0	01011996

[13]:
```
# Displays the datatype of both columns

melted_df = melted_df.sort_values(by = 'Date')
melted_df.dtypes
```

[13]:
```
value    float64
Date      object
dtype: object
```

[14]:
```
# Creates the test and train columns for the split model

melted_test = pd.DataFrame(columns = ['Date', 'value'])
melted_train = pd.DataFrame(columns = ['Date', 'value'])
counter = 0
```

[15]:
```
# Stops the Date value at July of 2020 and established both sets

from datetime import datetime # I have to re-import 'datetime' or this wont run␣
 ↪for some reason?

Max_Date = datetime(2020, 7, 1)

for index, row in melted_df.iterrows():
    if(datetime.strptime(row.Date, '%m%d%Y') >= Max_Date):
        melted_test.loc[len(melted_test.index)] = [datetime.strptime(row.Date,␣
 ↪'%m%d%Y'),  row.value]
        counter = counter + 1
    else:
        melted_train.loc[len(melted_train.index)] = [datetime.strptime(row.
 ↪Date, '%m%d%Y'),  row.value]
```

[16]:
```
# Establishes the new dataframs values and means

melted_train = melted_train.sort_values(by = "Date")
melted_test = melted_test.sort_values(by = "Date")

melted_test = melted_test.fillna(melted_test['value'].mean())
```

[17]:
```
#confirms the attributes for the train and test sets

melted_test.shape, melted_train.shape
```

[17]:
```
((18, 2), (342, 2))
```

[18]:
```
melted_train.isnull().values.any()
```

[18]:
```
False
```

[19]:
```
melted_test.isnull().values.any()
```

[19]:
```
False
```

[20]:
```
melted_test[melted_test.value.isna()==True]
```

[20]:
```
Empty DataFrame
Columns: [Date, value]
Index: []
```

Shows that the values stop at the end of the dataset

3 - Use the training set to build a predictive model for the monthly retail sales.

[21]:
```
# Builds the model with 'Date' as the index

melted_train = melted_train.set_index('Date')
melted_train.index = pd.to_datetime(melted_train.index)
```

4 - Use the model to predict the monthly retail sales on the last year of data.

[22]:
```
# Creates the sarima model to forecast the final months

import warnings

warnings.filterwarnings("ignore")

sarima_model = sm.tsa.SARIMAX(melted_train, trend='n', order=(0,1,0),␣
 ↪seasonal_order=(1,1,1,12))
model_fit = sarima_model.fit()
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            3     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  9.94755D+00    |proj g|=  4.23213D-01

At iterate    5    f=  9.82501D+00    |proj g|=  3.11834D-03
 This problem is unconstrained.

At iterate   10    f=  9.82469D+00    |proj g|=  6.28829D-04

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
```

```
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    3     13     16      1     0     0   1.032D-05   9.825D+00
  F =   9.8246805908693808

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

[23]:
```
# Using the model to forecast the final months

forecast = model_fit.predict(start = pd.to_datetime('2020-07-01'), end = pd.
 ↪to_datetime('2021-12-01'), dynamic = True)
```

5 - Report the RMSE of the model predictions on the test set.

[24]:
```
# Calculates the RMSE of the model

import numpy as np
from sklearn.metrics import mean_squared_error

melted_test = melted_test.set_index('Date')

test_forecast = model_fit.predict(start=melted_test.index[0], end = melted_test.
 ↪index[-1], dynamic=False)

rmse = np.sqrt(mean_squared_error(melted_test['value'], forecast))

print("RMSE:", rmse)
```

```
RMSE: 44787.19374582108
```

The RMSE that has been calculated shows that it is safe to predict an increase in sales over the final few months of the year that is without data.