Predictive Analytics: Assignment 8.2

Joshua Greenert

DSC630-T301 Predictive Analytics

1/29/2023

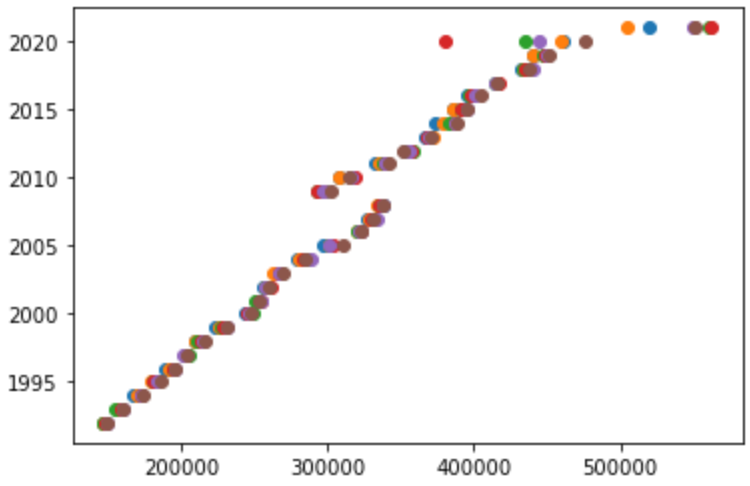
In [410…
```
# Import the required libaries.
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Pull in the data to begin preparation.
df_sales = pd.read_csv('us_retail_sales.csv')
df_sales.head(5)
```

Out[410]:

	YEAR	JAN	FEB	MAR	APR	MAY	JUN	JUL	AUG	SEP	OCT	NOV
0	1992	146925	147223	146805	148032	149010	149800	150761.00	151067.00	152588.00	153521.00	153583.00
1	1993	157555	156266	154752	158979	160605	160127	162816.00	162506.00	163258.00	164685.00	166594.00
2	1994	167518	169649	172766	173106	172329	174241	174781.00	177295.00	178787.00	180561.00	180703.00
3	1995	182413	179488	181013	181686	183536	186081	185431.00	186806.00	187366.00	186565.00	189055.00
4	1996	189135	192266	194029	194744	196205	196136	196187.00	196218.00	198859.00	200509.00	200174.00

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

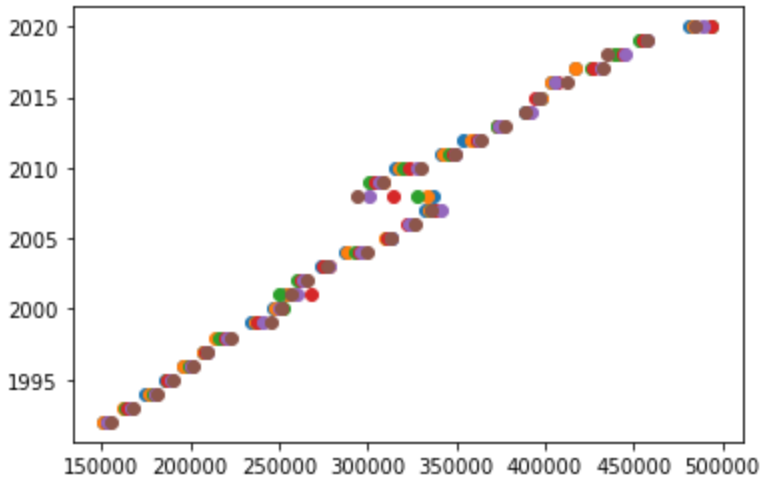
In [411…
```
# Plot 6 months of data simultaneously.
plt.scatter(x = df_sales["JAN"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["FEB"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["MAR"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["APR"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["MAY"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["JUN"], y = df_sales["YEAR"])
plt.show()
```



The data displayed from the first 6 months appears almost identical. However, the biggest difference in

values happens in the year 2020 which could be the result of incomplete data or some significant event (i.e. Covid).

In [412…
```
# Plot 6 months of data simultaneously.
plt.scatter(x = df_sales["JUL"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["AUG"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["SEP"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["OCT"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["NOV"], y = df_sales["YEAR"])
plt.scatter(x = df_sales["DEC"], y = df_sales["YEAR"])
plt.show()
```



Similar to the previous observation, we see an almost identical dataset between each scatterplot. They almost appear to overlap one another. Based on the trends, it appears that there is a massive dip downwards from 2005 to 2010 like the previous graph as well. This can be rationalized with understanding about the 2008 economic crisis that took place. Since the data appear to be similiar with an upward trend, we can assume that our predictions should be higher than the year prior to the most recent one. Alternatively, if economic factors are at play, such as the impact of Covid, we can likewise expect a downtrend.

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.

In [413…
```
# Use the melt method to convert the dataframe.
df_melt = pd.melt(df_sales, id_vars  ="YEAR", value_vars=df_sales.columns[1:13] )

# Convert years to date object.
df_melt['date'] = pd.to_datetime(df_melt['YEAR'].astype(str) + df_melt['variable'], form
df_melt['date'] = df_melt['date'].dt.strftime('%m/%d/%Y')
```

In [414…
```
# Sort the values by year.
df_melt = df_melt.sort_values(by="YEAR")

# Drop the year and variable columns.
df_melt = df_melt.drop(["YEAR", "variable"], axis = 1)
```

In [415…
```
# Create a new dataframe for the test and train set.
from datetime import datetime

# Create new dataframes.
df_test = pd.DataFrame(columns = ['date', 'value'])
```

```
df_train = pd.DataFrame(columns = ['date', 'value'])
counter = 0

# Set sentinel datetime.
maxDate = datetime(2020, 7, 1)

for index, row in df_melt.iterrows():
    if(datetime.strptime(row.date, '%m/%d/%Y') >= maxDate):
        df_test.loc[len(df_test.index)] = [datetime.strptime(row.date, '%m/%d/%Y'),  row
        counter = counter + 1
    else:
        df_train.loc[len(df_train.index)] = [datetime.strptime(row.date, '%m/%d/%Y'),  r
```

In [416…

```
# Sort the new dataframes.
df_train = df_train.sort_values(by="date")
df_test = df_test.sort_values(by="date")

# Set the mean values for the test data.
df_test = df_test.fillna(df_test['value'].mean())
```

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

In [417…

```
import statsmodels.api as sm

# set the date as the index.
df_train = df_train.set_index('date')
df_train.index = pd.to_datetime(df_train.index)

# Use the sarima model for seasonal time forecasting.
model = sm.tsa.SARIMAX(df_train, trend='n', order=(0,1,0), seasonal_order=(1,1,1,12))
model_fit = model.fit()
```

```
C:\Users\Josh\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWa
rning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
C:\Users\Josh\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: ValueWa
rning: No frequency information was provided, so inferred frequency MS will be used.
  self._init_dates(dates, freq)
```

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

In [418…

```
# Predict the future forecast using the new model.
future_forecast = model_fit.predict(start=pd.to_datetime('2020-07-01'), end=pd.to_dateti
```

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

In [419…

```
from sklearn.metrics import mean_squared_error
import numpy as np

# set the date field as the index for the test set.
df_test = df_test.set_index('date')

# Create the predictions for the test set.
test_future_forecast = model_fit.predict(start=df_test.index[0], end=df_test.index[-1],

# Calculate the RMSE
```

```
rmse = np.sqrt(mean_squared_error(df_test['value'], future_forecast))

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

```
RMSE: 44787.19374295509
```