8.2 Exercise. DSC630 - Jennifer Barrera Conde

July 26, 2024

1 Exercise 8.2

2 DSC630

3 Jennifer Barrera Conde

4 Time Series Modeling

4.1 Getting to know the data and cleaning:

[1]:
```
import pandas as pd
import matplotlib.pyplot as plt
from prophet import Prophet
from sklearn.metrics import mean_squared_error
import numpy as np

# Load the dataset
file = 'us_retail_sales.csv'
data = pd.read_csv(file)

# Display the first few rows of the dataset
data.head()
```

[1]:
```
   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
```

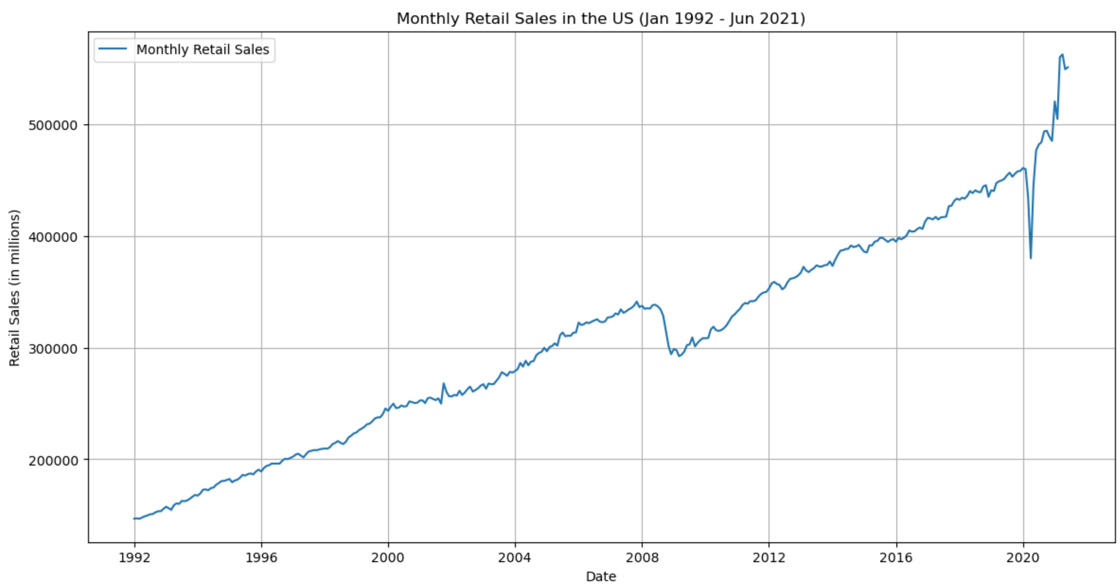
[ ]:

```

```

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

[2]:

```
# Reshape the dataset to long format
data_long = data.melt(id_vars=['YEAR'], var_name='MONTH', value_name='SALES')

# Create a datetime column
data_long['DATE'] = pd.to_datetime(data_long['YEAR'].astype(str) +
 ↪data_long['MONTH'], format='%Y%b')

# Sort by date
data_long = data_long.sort_values('DATE')

# Plot the data
plt.figure(figsize=(14, 7))
plt.plot(data_long['DATE'], data_long['SALES'], label='Monthly Retail Sales')
plt.xlabel('Date')
plt.ylabel('Retail Sales (in millions)')
plt.title('Monthly Retail Sales in the US (Jan 1992 - Jun 2021)')
plt.legend()
plt.grid(True)
plt.show()
```



Here are some observations from the plot of the monthly retail sales data:

Trend: There is a clear upward trend in retail sales over the years, indicating growth in consumer spending.

Seasonality: There is a noticeable seasonal pattern with peaks typically occurring around the end of each year, likely due to holiday shopping.

Anomalies: Some anomalies are visible, such as the dip around early 2020, which could be related to the impact of the COVID-19 pandemic on retail sales.

[ ]:

```

```

4.3 Step 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.

[3]:

```
# Reshape the dataset to long format
data_long = data.melt(id_vars=['YEAR'], var_name='MONTH', value_name='SALES')
data_long['DATE'] = pd.to_datetime(data_long['YEAR'].astype(str) +␣
 ↪data_long['MONTH'], format='%Y%b')
data_long = data_long.sort_values('DATE')

# Fill missing values using forward fill method
data_long['SALES'].fillna(method='ffill', inplace=True)

# Prophet requires the columns to be named 'ds' and 'y'
data_long = data_long.rename(columns={'DATE': 'ds', 'SALES': 'y'})

# Verify no more missing values
print(data_long.isnull().sum())
```

```
YEAR     0
MONTH    0
y        0
ds       0
dtype: int64
```

[4]:

```
# Split the data into training and test sets
train = data_long[data_long['ds'] < '2020-07-01']
test = data_long[data_long['ds'] >= '2020-07-01']
```

[ ]:

```

```

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

[5]:

```
# Initialize and fit the model
model = Prophet(yearly_seasonality=True)
model.fit(train)
```

```
19:50:24 - cmdstanpy - INFO - Chain [1] start processing
19:50:24 - cmdstanpy - INFO - Chain [1] done processing
```

[5]: <prophet.forecaster.Prophet at 0x19b48895090>

[ ]:

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

I came to several issues so I decided to run some troubleshooting steps

[8]:
```
# Print the test and forecasted_sales dataframes before merging to ensure they␣
↪have matching dates.
print(test.head())
print(forecasted_sales.head())
```

```
     YEAR MONTH    actual         ds
208  2020   JUL  481627.0 2020-07-01
238  2020   AUG  483716.0 2020-08-01
268  2020   SEP  493327.0 2020-09-01
298  2020   OCT  493991.0 2020-10-01
328  2020   NOV  488652.0 2020-11-01
            ds           yhat
342 2020-06-01  459567.385895
343 2020-07-01  461807.759097
344 2020-08-01  460158.509224
345 2020-09-01  461691.687896
346 2020-10-01  465247.356890
```

The dates in the test dataframe and the forecasted_sales dataframe are not perfectly aligned. The test dates are on the first of each month, while the forecasted_sales dates are at the end of each month.

The dates needed to be align correctly for the test and forecasted values to work.

[9]:
```
# Ensure that the dates in test and forecasted_sales align properly.
print(test['ds'].tail())
print(forecasted_sales['ds'].tail())
```

```
239   2021-08-01
269   2021-09-01
299   2021-10-01
329   2021-11-01
359   2021-12-01
Name: ds, dtype: datetime64[ns]
355   2021-07-01
356   2021-08-01
357   2021-09-01
358   2021-10-01
359   2021-11-01
Name: ds, dtype: datetime64[ns]
```

Now I am able to continue.

[6]:
```
# Make predictions for the test set
future = model.make_future_dataframe(periods=len(test), freq='M')
forecast = model.predict(future)

# Extract the forecasted values for the test period
forecasted_sales = forecast[['ds', 'yhat']].tail(len(test))

# Adjust the dates in forecasted_sales to match the start of the month
forecasted_sales['ds'] = forecasted_sales['ds'] + pd.offsets.MonthBegin(-1)

# Merge the actual test values with the forecasted values
test = test.rename(columns={'y': 'actual'})
results = test.merge(forecasted_sales, on='ds')

# Verify the merge
print(results)

# Display the forecasted results
print(results[['ds', 'actual', 'yhat']])
```

```
   YEAR MONTH    actual         ds           yhat
0  2020   JUL  481627.0 2020-07-01  461807.759097
1  2020   AUG  483716.0 2020-08-01  460158.509224
2  2020   SEP  493327.0 2020-09-01  461691.687896
3  2020   OCT  493991.0 2020-10-01  465247.356890
4  2020   NOV  488652.0 2020-11-01  464253.098346
5  2020   DEC  484782.0 2020-12-01  464453.169581
6  2021   JAN  520162.0 2021-01-01  465134.572037
7  2021   FEB  504458.0 2021-02-01  463612.611809
8  2021   MAR  559871.0 2021-03-01  473457.750798
9  2021   APR  562269.0 2021-04-01  468236.502383
10 2021   MAY  548987.0 2021-05-01  467634.899299
11 2021   JUN  550782.0 2021-06-01  472197.232099
12 2021   JUL  550782.0 2021-07-01  474818.123209
13 2021   AUG  550782.0 2021-08-01  472320.153063
14 2021   SEP  550782.0 2021-09-01  473948.687529
15 2021   OCT  550782.0 2021-10-01  478570.669202
16 2021   NOV  550782.0 2021-11-01  476895.046995
          ds    actual           yhat
0 2020-07-01  481627.0  461807.759097
1 2020-08-01  483716.0  460158.509224
2 2020-09-01  493327.0  461691.687896
3 2020-10-01  493991.0  465247.356890
4 2020-11-01  488652.0  464253.098346
5 2020-12-01  484782.0  464453.169581
6 2021-01-01  520162.0  465134.572037
```

```
7   2021-02-01  504458.0  463612.611809
8   2021-03-01  559871.0  473457.750798
9   2021-04-01  562269.0  468236.502383
10  2021-05-01  548987.0  467634.899299
11  2021-06-01  550782.0  472197.232099
12  2021-07-01  550782.0  474818.123209
13  2021-08-01  550782.0  472320.153063
14  2021-09-01  550782.0  473948.687529
15  2021-10-01  550782.0  478570.669202
16  2021-11-01  550782.0  476895.046995
```

[ ]:

4.6 Step 5. Report the RMSE of the model predictions on the test set.

[7]:
```
# Calculate the RMSE
rmse = np.sqrt(mean_squared_error(results['actual'], results['yhat']))
print(f'RMSE: {rmse}')
```

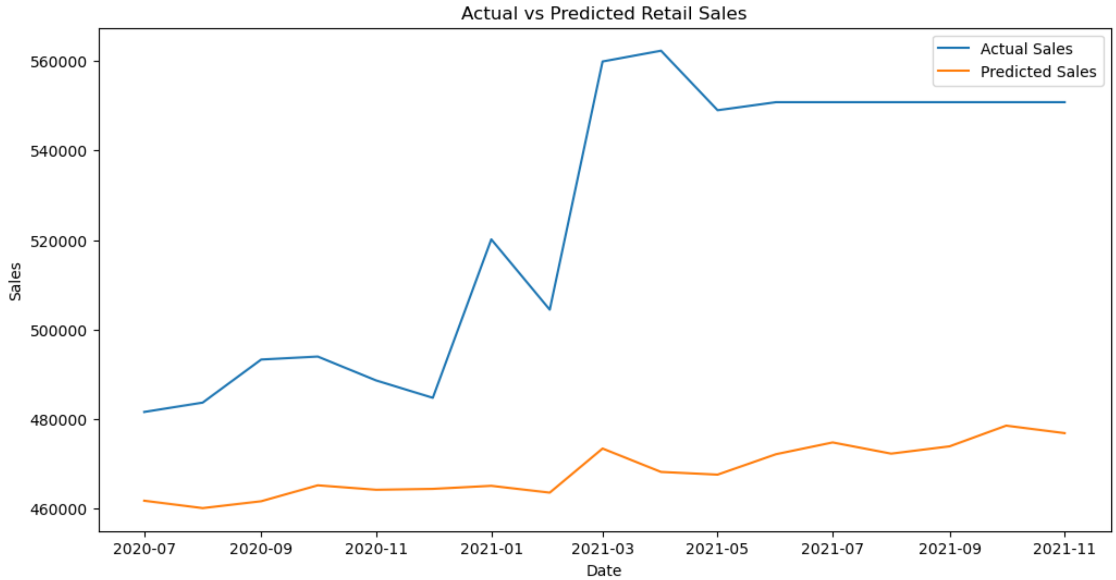
```
RMSE: 62349.42204232405
```

[ ]:

4.7 Step 6. More analysis and interpreting results.

After obtaining the "results" dataframe, we can plot the actual sales against the predicted sales to visually compare the model's performance.

[10]:
```
# Plot actual vs predicted sales
plt.figure(figsize=(12, 6))
plt.plot(results['ds'], results['actual'], label='Actual Sales')
plt.plot(results['ds'], results['yhat'], label='Predicted Sales')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Actual vs Predicted Retail Sales')
plt.legend()
plt.show()
```



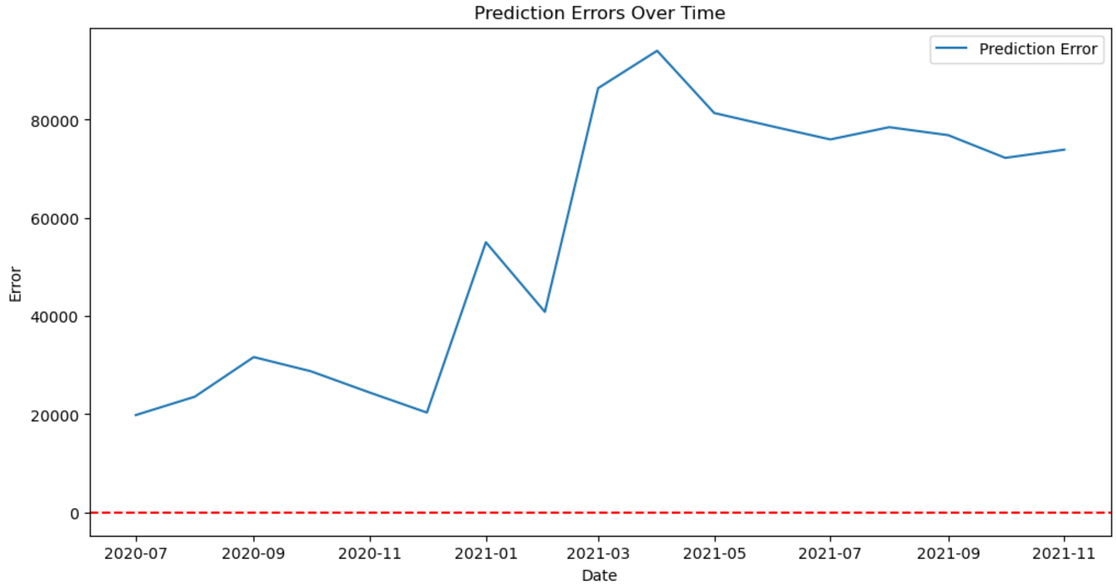
[ ]:

```

```

The differences between actual and predicted sales can be inspected to understand where the model performs well and where it may fall short.

[11]:

```python
# Calculate prediction errors
results['error'] = results['actual'] - results['yhat']

# Plot prediction errors
plt.figure(figsize=(12, 6))
plt.plot(results['ds'], results['error'], label='Prediction Error')
plt.xlabel('Date')
plt.ylabel('Error')
plt.title('Prediction Errors Over Time')
plt.axhline(y=0, color='r', linestyle='--')
plt.legend()
plt.show()
```



[ ]:

Next, we summarize the error metrics to get a quantitative sense of the model's performance.

[12]:

```
# Calculate summary statistics for prediction errors
error_mean = results['error'].mean()
error_std = results['error'].std()
error_median = results['error'].median()

print(f'Mean Error: {error_mean}')
print(f'Standard Deviation of Error: {error_std}')
print(f'Median Error: {error_median}')
```

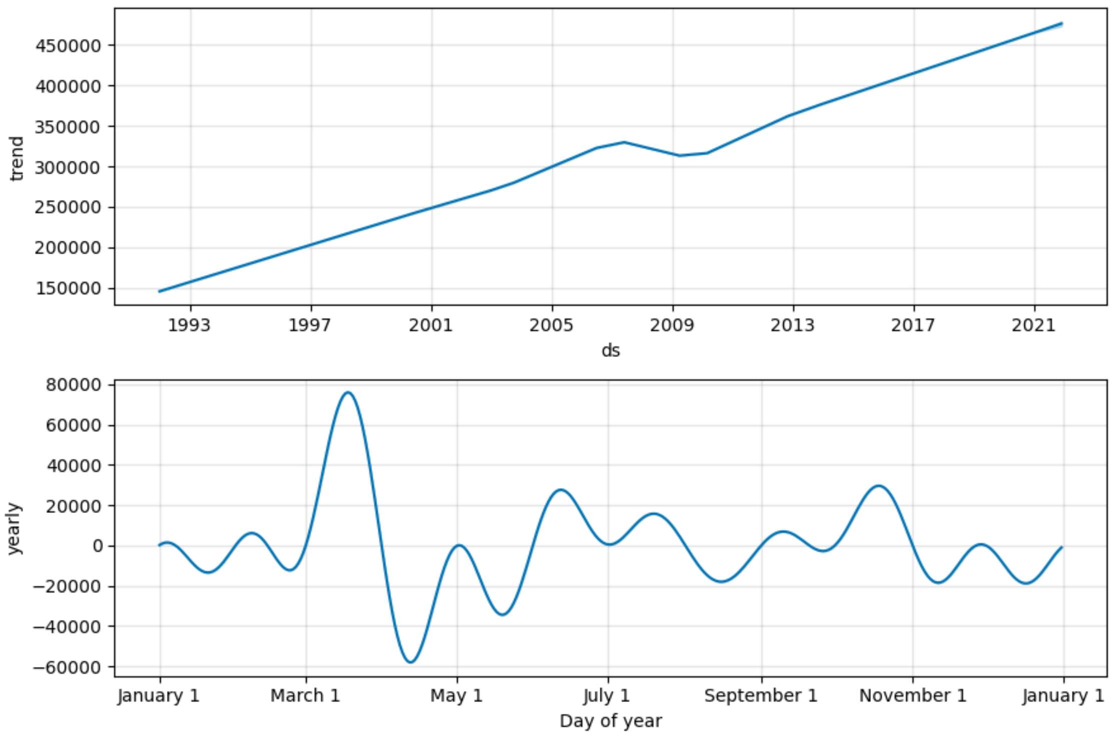
```
Mean Error: 56593.89238495414
Standard Deviation of Error: 26969.763339531328
Median Error: 72211.33079849248
```

[ ]:

One of the resons I chose Prophet as my model is because it allows you to visualize the seasonal components of the time series, such as yearly seasonality. This can provide insights into underlying patterns in the data.

[13]:

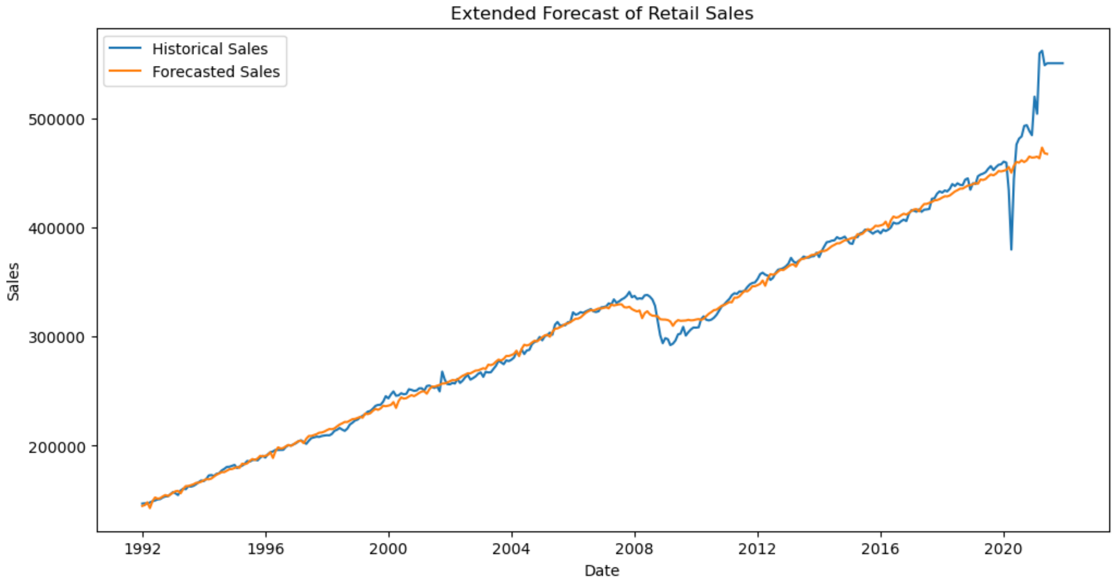
```
# Plot the components of the forecast
model.plot_components(forecast)
plt.show()
```



[ ]:

Lastly, we can use the model to predict future sales beyond the test set period. This helps in planning and decision-making.

[14]:
```
# Extend the forecast period by an additional 12 months
future_extended = model.make_future_dataframe(periods=12, freq='M')
forecast_extended = model.predict(future_extended)

# Plot the extended forecast
plt.figure(figsize=(12, 6))
plt.plot(data_long['ds'], data_long['y'], label='Historical Sales')
plt.plot(forecast_extended['ds'], forecast_extended['yhat'], label='Forecasted␣
 ↪Sales')
plt.xlabel('Date')
plt.ylabel('Sales')
plt.title('Extended Forecast of Retail Sales')
plt.legend()
plt.show()
```



Based on all the previous analysis the extended forecast follows a similar path where sales minimally fluctuate from the current year.

```
[ ]:
```