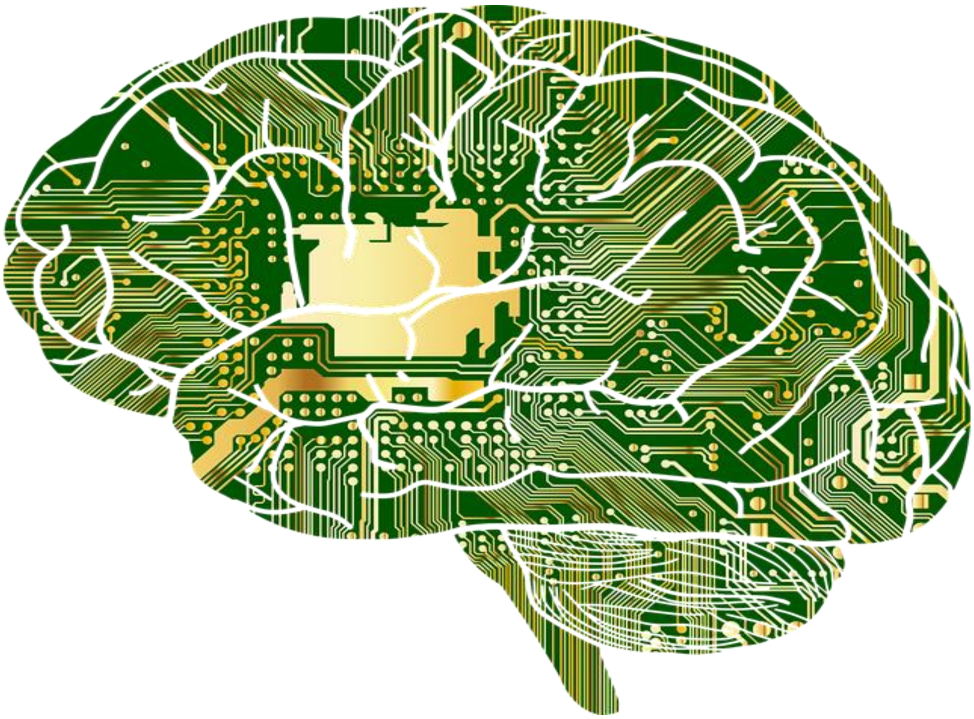
TIME _SERIES FORECASTING

USING FACEBOOK PROPHET



Data Mining & Analysis

 Priyanshi Chakraborty

Table of Contents

Abstract:-

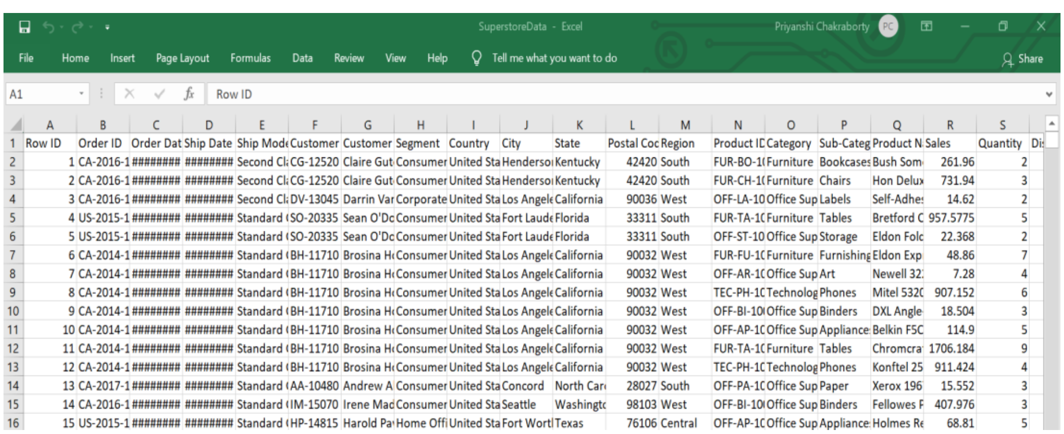
Apply ARIMA Model to the time-series data, which stands for Autoregressive Integrated Moving Average to the data to get the time series forecasting. It will help us understand our forecasts' accuracy and then compare predicted sales to real sales of the time series. We have also produced and visualize predictions. Will also perform Time Series Modelling with Prophet.

Data Source:-

https://community.tableau.com/s/question/0D54T00000CWeX8SAL/sample-superstore-sales-excelxls

Attributes:-

Row ID
Order ID
Order Date
Ship Date
Ship Mode
Customer ID
Customer Name
Segment
Country
City
State
Postal Code
Region
Product ID
Category
Sub-Category
Product Name
Sales
Quantity
Discount
Profit



Row ID	Order ID	Order Dat	Ship Date	Ship Mode	Customer	Customer	Segment	Country	City	State	Postal Coc	Region	Product ID	Category	Sub-Categ	Product N	Sales	Quantity	Di
1	CA-2016-1	########	########	Second Cl	CG-12520	Claire Gut	Consumer	United Sta	Henderso	Kentucky	42420	South	FUR-BO-1	Furniture	Bookcases	Bush Som	261.96	2	
2	CA-2016-1	########	########	Second Cl	CG-12520	Claire Gut	Consumer	United Sta	Henderso	Kentucky	42420	South	FUR-CH-1	Furniture	Chairs	Hon Delux	731.94	3	
3	CA-2016-1	########	########	Second Cl	DV-13045	Darrin Var	Corporate	United Sta	Los Angele	California	90036	West	OFF-LA-10	Office Sup	Labels	Self-Adhe	14.62	2	
4	US-2015-1	########	########	Standard	SO-20335	Sean O'Dc	Consumer	United Sta	Fort Laude	Florida	33311	South	FUR-TA-1C	Furniture	Tables	Bretford C	957.5775	5	
5	US-2015-1	########	########	Standard	SO-20335	Sean O'Dc	Consumer	United Sta	Fort Laude	Florida	33311	South	OFF-ST-10	Office Sup	Storage	Eldon Folc	22.368	2	
6	CA-2014-1	########	########	Standard	BH-11710	Brosina H	Consumer	United Sta	Los Angele	California	90032	West	FUR-FU-1C	Furniture	Furnishing	Eldon Exp	48.86	7	
7	CA-2014-1	########	########	Standard	BH-11710	Brosina H	Consumer	United Sta	Los Angele	California	90032	West	OFF-AR-1C	Office Sup	Art	Newell 32	7.28	4	
8	CA-2014-1	########	########	Standard	BH-11710	Brosina H	Consumer	United Sta	Los Angele	California	90032	West	TEC-PH-1C	Technolog	Phones	Mitel 532C	907.152	6	
9	CA-2014-1	########	########	Standard	BH-11710	Brosina H	Consumer	United Sta	Los Angele	California	90032	West	OFF-BI-10	Office Sup	Binders	DXL Angle	18.504	3	
10	CA-2014-1	########	########	Standard	BH-11710	Brosina H	Consumer	United Sta	Los Angele	California	90032	West	OFF-AP-1C	Office Sup	Appliance	Belkin F5C	114.9	5	
11	CA-2014-1	########	########	Standard	BH-11710	Brosina H	Consumer	United Sta	Los Angele	California	90032	West	FUR-TA-1C	Furniture	Tables	Chromcra	1706.184	9	
12	CA-2014-1	########	########	Standard	BH-11710	Brosina H	Consumer	United Sta	Los Angele	California	90032	West	TEC-PH-1C	Technolog	Phones	Konftel 25	911.424	4	
13	CA-2017-1	########	########	Standard	AA-10480	Andrew A	Consumer	United Sta	Concord	North Car	28027	South	OFF-PA-1C	Office Sup	Paper	Xerox 196	15.552	3	
14	CA-2016-1	########	########	Standard	IM-15070	Irene Mad	Consumer	United Sta	Seattle	Washingtc	98103	West	OFF-BI-10	Office Sup	Binders	Fellowes F	407.976	3	
15	US-2015-1	########	########	Standard	HP-14815	Harold Pa	Home Offi	United Sta	Fort Wort	Texas	76106	Central	OFF-AP-1C	Office Sup	Appliance	Holmes Re	68.81	5	

Figure 1

Using the Superstore Sales data, we start from time series analysis and forecasting for furniture sales.

We have a 4-year furniture sales data.

Pre-Processing of Data:-

This progression incorporates evacuating segments we needn't bother with, check missing qualities, total deals by date, etc.

```
In [4]: #Data Preprocessing
        cols = ['Row ID', 'Order ID', 'Ship Date', 'Ship Mode', 'Customer ID', 'Customer Name', 'Segment', 'Country', 'City', 'State', 'P
        furniture.drop(cols, axis=1, inplace=True)
        furniture = furniture.sort_values('Order Date')
        furniture.isnull().sum()

Out[4]: Order Date    0
        Sales         0
        dtype: int64
```

Figure 2

Indexing Time Series Data:-

Our current DateTime data is tricky to work. Therefore, we are using the averages daily sales value for that month; what's more, we utilize the beginning of every month as the timestamp.

Looking at the 2017 furniture sales data

```
In [7]: y = furniture['Sales'].resample('MS').mean()

In [8]: y['2017':]

Out[8]: Order Date
        2017-01-01     397.602133
        2017-02-01     528.179800
        2017-03-01     544.672240
        2017-04-01     453.297905
        2017-05-01     678.302328
        2017-06-01     826.460291
        2017-07-01     562.524857
        2017-08-01     857.881889
        2017-09-01    1209.508583
        2017-10-01     875.362728
        2017-11-01    1277.817759
        2017-12-01    1256.298672
        Freq: MS, Name: Sales, dtype: float64
```

Figure 3

When we work on the data, the time-series has a regular and predictable pattern as we can see that sales are always low at the beginning of the year and high at the end of the year. There was an upward trend within any single year with a couple of quiet months in the mid of the year.

```
In [9]: #Visualizing Furniture Sales Data
        y.plot(figsize=(15, 6))
        plt.show()
```

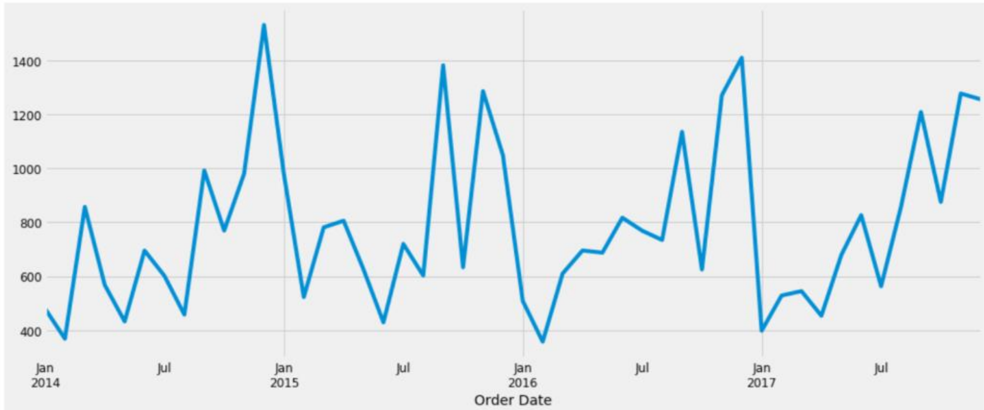


Figure 4

We can likewise picture our information utilizing "time-arrangement deterioration" that permits us to break down our time arrangement into three particular segments: pattern, irregularity, and commotion.

Earlier, when we plotted "Time – Series Data," we saw that the magnitude of seasonal fluctuations as a function of time was trending towards zero; in other words, time-series has a regular pattern.

In the following scenario, it is helpful to use "Additive Decomposition" to check for the underlying pattern.

Assuming,

Y(t) =Data as a function of time

S(t)= Seasonal Component as a function of time

T(t)= Trend Cycle Component as a function of time

R(t)= Remainder/Residual Component as a function of time

The Formula for Additive Decomposition is:

$Y(t) = S(t) + T(t) + R(t)$

In the below graph, we can see that adding S(t)+T(t)+ R(t) we get Y(t)(Sales) is very unstable when compared to S(t).

Figure 5

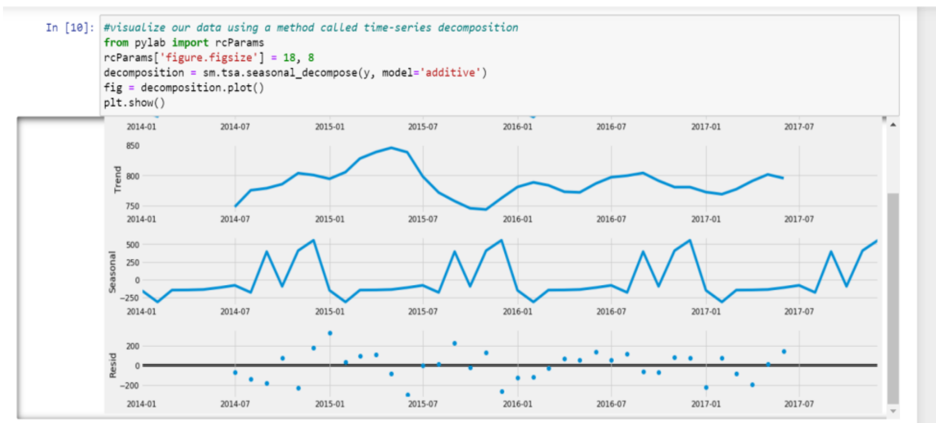


Figure 6

Time Series Forecasting with ARIMA:-

There are two most useful approaches to "Time – Series Forecasting" one of them is "Exponential Smoothing," and the other is the "ARIMA Model." However, Exponential Smoothing is based on the description of trend and seasonality in data whereas, ARIMA Model aims to describe the autocorrelations in the data.

We will apply one of the most commonly used methods for time-series forecasting, known as ARIMA, which stands for Autoregressive Integrated Moving Average.

THE seasonal ARIMA model was built using additional seasonal terms (p, d, q). These three represents:

(p) is the autoregressive part of the model. It permits us to fuse the impact of past qualities into our model.

(d) is the incorporated piece of the model. It remembers terms for the model that consolidates the measure of differencing (i.e., the quantity of past time focuses on deducting from the current worth) to apply to the time arrangement.

(q) is the standard piece of the model. It permits us to set our model's mistake as a straight blend of the blunder saw at a past time focuses previously.

```
In [11]: #Time series forecasting with ARIMA
p = d = q = range(0, 2)
pdq = list(itertools.product(p, d, q))
seasonal_pdq = [(x[0], x[1], x[2], 12) for x in list(itertools.product(p, d, q))]
print('Examples of parameter combinations for Seasonal ARIMA...')
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[1]))
print('SARIMAX: {} x {}'.format(pdq[1], seasonal_pdq[2]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[3]))
print('SARIMAX: {} x {}'.format(pdq[2], seasonal_pdq[4]))

Examples of parameter combinations for Seasonal ARIMA...
SARIMAX: (0, 0, 1) x (0, 0, 1, 12)
SARIMAX: (0, 0, 1) x (0, 1, 0, 12)
SARIMAX: (0, 1, 0) x (0, 1, 1, 12)
SARIMAX: (0, 1, 0) x (1, 0, 0, 12)
```

Figure 7

In the above code, we try to generate all possible combinations for seasonal(p,d,q) triplets and the parameter selection for our furniture data sales using ARIMA Time Series Model. We want to use a "grid search" to find the optimal set of parameters that will give our model's best performance.

Grid Search is nothing but finding the specific parameters for which the above model fits the best in correspondence to Seasonal ARIMA using nested loops on a combination of parameters that we saw in the above code.

The below operation results in its respective AIC score.

```
In [12]: #Grid Search
for param in pdq:
    for param_seasonal in seasonal_pdq:
        try:
            mod = sm.tsa.statespace.SARIMAX(y,seasonal_order=param_seasonal,enforce_stationarity=False,enforce_invertibility=Fal
            results = mod.fit()
            print('ARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
        except:
            continue
```

Figure 8

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:692.1645522067712

C:\Users\Priyanshi Chakrabort\Anaconda 3\lib\site-packages\statsmodels\base\model.py:568: ConvergenceWarning: Maximum Likelih
ood optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)

ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1343.1777877543473
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:479.46321478521355
ARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:304.2077675160913
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:480.92593679352177

C:\Users\Priyanshi Chakrabort\Anaconda 3\lib\site-packages\statsmodels\base\model.py:568: ConvergenceWarning: Maximum Likelih
ood optimization failed to converge. Check mle_retvals
  "Check mle_retvals", ConvergenceWarning)

ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1243.8088413604426
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:304.4664675084554
ARIMA(0, 0, 0)x(1, 1, 1, 12)12 - AIC:304.5842692143882
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:692.1645522067712

C:\Users\Priyanshi Chakrabort\Anaconda 3\lib\site-packages\statsmodels\base\model.py:568: ConvergenceWarning: Maximum Likelih
```

Figure 9

The result from the above suggests that SARIMAX (1, 1, 1) x (1, 1, 0, 12) gives the lowest AIC value of 297.78 i.e., at this point, the relevant information the model is going to lose is minimum. Therefore, we should consider this to be the optimal option.

Fitting the ARIMA model:-

Now, based on the AIC value, we fit the model to our data.

```
In [13]: #Fitting the ARIMA Model
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(1, 1, 1),
                                seasonal_order=(1, 1, 0, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results = mod.fit()
print(results.summary().tables[1])
```

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.0146      0.342      0.043      0.966      -0.655       0.684
ma.L1         -1.0000      0.360     -2.781      0.005      -1.705      -0.295
ar.S.L12      -0.0253      0.042     -0.609      0.543      -0.107       0.056
sigma2      2.958e+04   1.22e-05   2.43e+09      0.000    2.96e+04    2.96e+04
==============================================================================
```

Figure 10

The results from the output of SARIMAX returns a great deal of information, but we focus our attention on coefficients. The coefficient column shows the weight importance of each feature and how each one impacts the time series.

Performed the diagnostics on the above model to investigate any unusual behavior:-

When fitting seasonal ARIMA models, it is essential to run model diagnostics to make sure that there are no violations assured by the model. The plot diagnostics allows us to generate model diagnostics and investigate any unusual behavior quickly.

```
In [14]: #Plotting the result
         results.plot_diagnostics(figsize=(16, 8))
         plt.show()
```

Figure 11

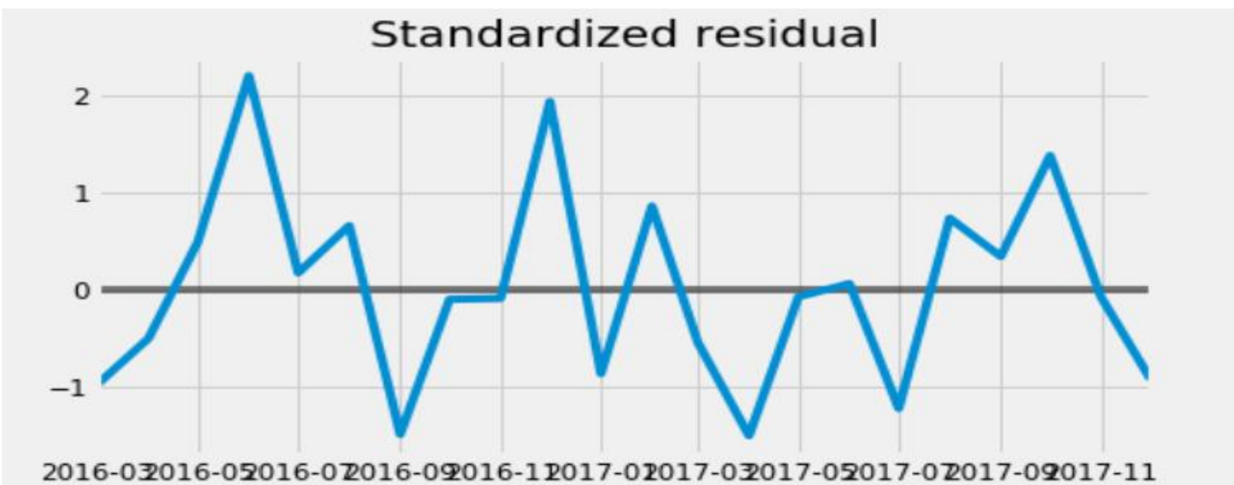


Figure 12

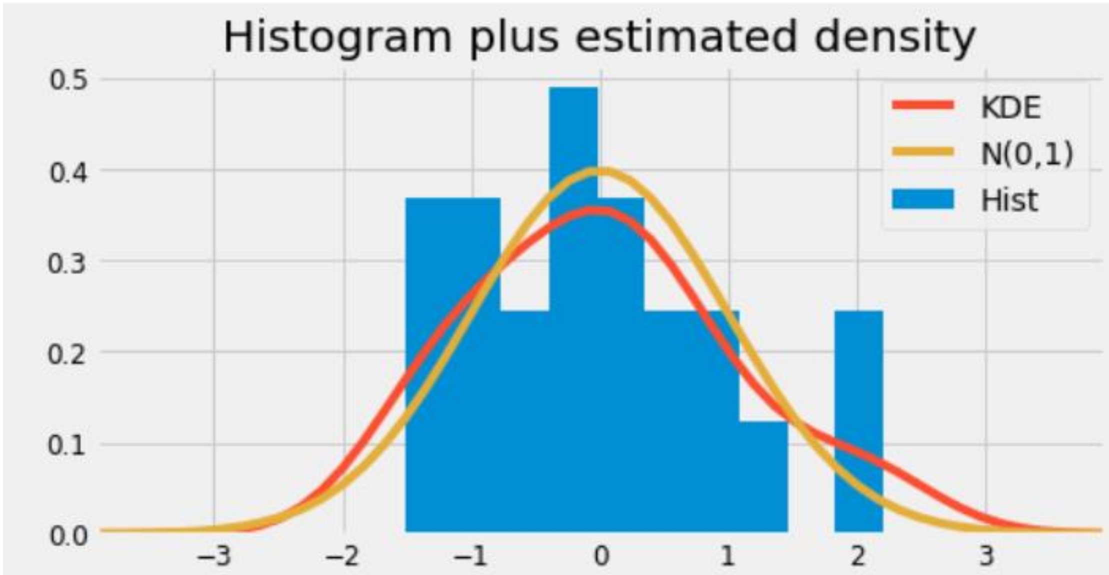


Figure 13

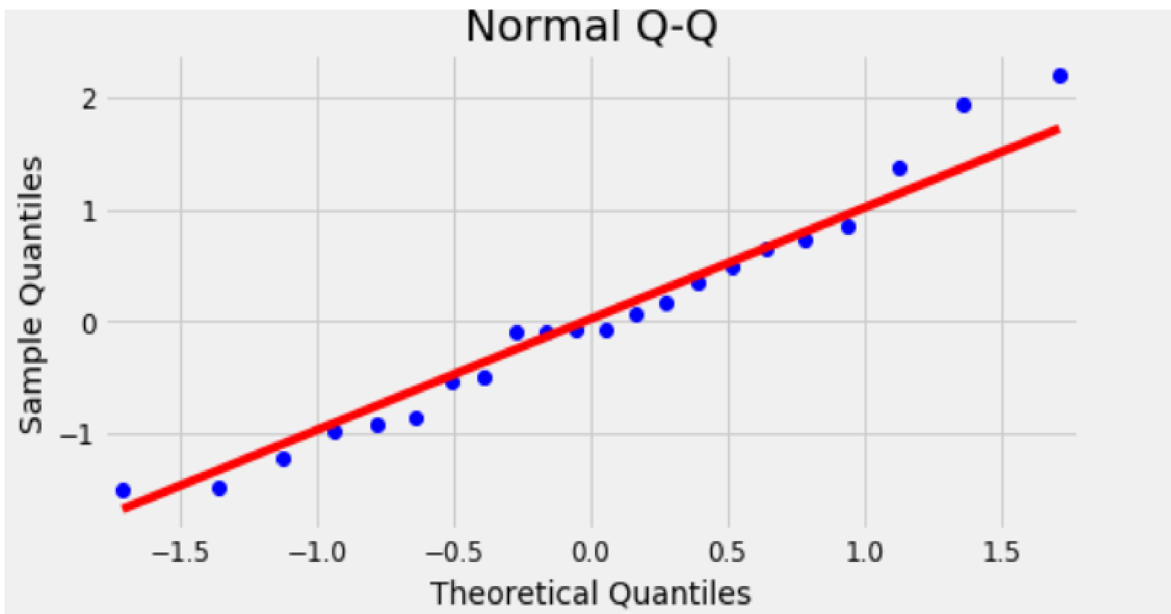


Figure 14

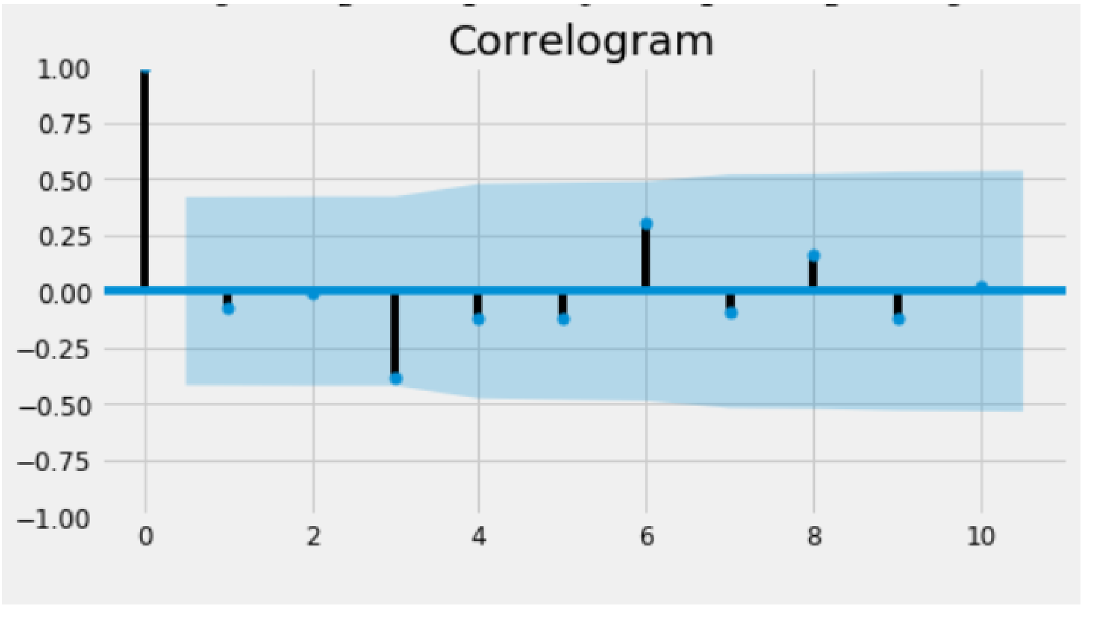


Figure 15

It is not perfect. Nonetheless, our model diagnostics recommend that the model residuals are close and regularly conveyed.

Validating Forecasts:-

To understand our model's correctness, we compare predicted sales to the actual purchase of the time series, and we set projections to start at 2017–01–01 to the end of the data.

```
In [15]: #Validating the Forecasts
pred = results.get_prediction(start=pd.to_datetime('2017-01-01'), dynamic=False)
pred_ci = pred.conf_int()
ax = y['2014':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.7, figsize=(14, 7))
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='k', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Furniture Sales')
plt.legend()
plt.show()
```

Figure 16

Figure 17

The line plot showed the observed values in comparison to the rolling forecast predictions. Overall, the forecasts align within the actual costs, indicates that an upward trend that starts from the earlier of the year and store the seasonality toward the end of the year.

Error:-

In statistics, the mean square error, also known as MSE of an estimator calculates the average squares of the mistakes — that is, the average squared difference between the estimated values and expected. The MSE is a measure of an estimator's quality; it never falls non-negative amount, and the lower MSE, the closer we can find a decent fit.

Root Mean Square Error, well known as RMSE, tells us that our model could predict the average day to day furniture sales within 151.64 of the real deals. The regular furniture sale value falls between 400 to 1200. In our opinion, this is a pretty good model so far.

```
In [16]: #Calculate MSE of Forecasts
         y_forecasted = pred.predicted_mean
         y_truth = y['2017-01-01':]
         mse = ((y_forecasted - y_truth) ** 2).mean()
         print('The Mean Squared Error of our forecasts is {}'.format(round(mse, 2)))

         The Mean Squared Error of our forecasts is 22993.58

In [17]: #Calculating the RMSE
         print('The Root Mean Squared Error of our forecasts is {}'.format(round(np.sqrt(mse), 2)))

         The Root Mean Squared Error of our forecasts is 151.64
```

Figure 18

Producing and Visualizing Forecasts:-

Our model caught furniture deals irregularity. As we gauge farther into the future, it is normal for us to turn out to be less positive about our qualities. It is reflected by the certainty spans created by our model, which develop huge as we move farther into what's to come.

The below time series analysis for Furniture makes me curious about other categories, and how they compare with each other over time. Therefore, we are going to compare the time series of furniture and office suppliers.

```
In [18]: #Producing and Visualizing Forecasts
         pred_uc = results.get_forecast(steps=100)
         pred_ci = pred_uc.conf_int()
         ax = y.plot(label='observed', figsize=(14, 7))
         pred_uc.predicted_mean.plot(ax=ax, label='Forecast')
         ax.fill_between(pred_ci.index,
                         pred_ci.iloc[:, 0],
                         pred_ci.iloc[:, 1], color='k', alpha=.25)
         ax.set_xlabel('Date')
         ax.set_ylabel('Furniture Sales')
         plt.legend()
         plt.show()
```

Figure 19

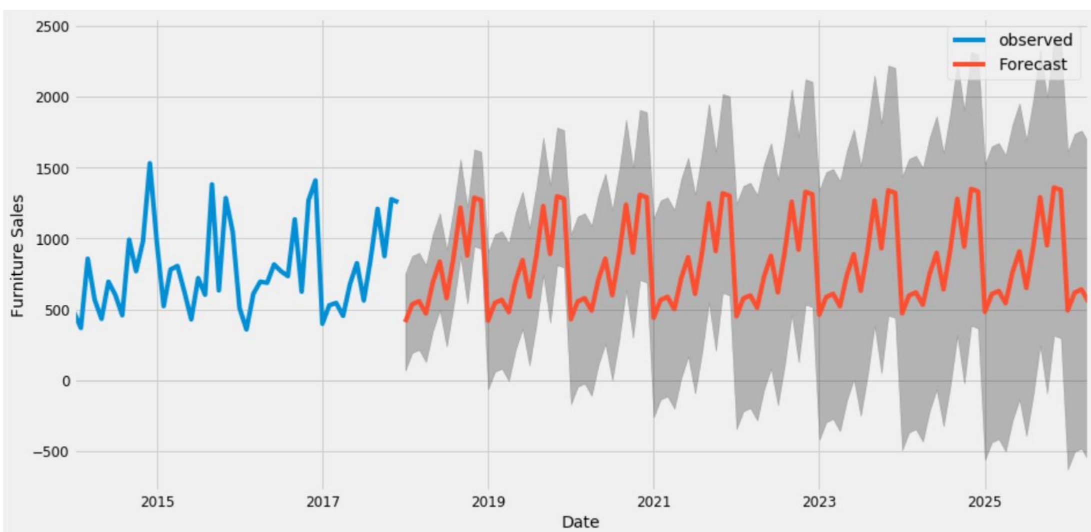


Figure 20

Time Series of Furniture vs. Office Supplies:-

According to our data, there are a way more sales from Office Supplies than from Furniture over the years.

```
In [19]: #compare time series of furniture and office supplier
         furniture = df.loc[df['Category'] == 'Furniture']
         office = df.loc[df['Category'] == 'Office Supplies']
         furniture.shape, office.shape

Out[19]: ((2121, 21), (6026, 21))
```

Figure 21

Data Exploration:-

We compare two categories' sales in the same period, we have combined two data frames into one and plotted these two categories' time series in one plot. We can notice that sales of furniture and office supplies shared a similar seasonal pattern. At the start of the year, the sales are offseason for both of the two categories. It seems summertime is quiet for office supplies too. Also, average daily sales for Furniture are higher than those of office supplies in most months. It is justifiable, as Furniture's estimation ought to be a lot higher than those of office supplies. Every so often, office supplies passed Furniture on a typical day by day deals.

```
In [20]: #compare two categories' sales in the same time period
         cols = ['Row ID', 'Order ID', 'Ship Date', 'Ship Mode', 'Customer ID', 'Customer Name', 'Segment', 'Country', 'City', 'State', 'F
         furniture.drop(cols, axis=1, inplace=True)
         office.drop(cols, axis=1, inplace=True)
         furniture = furniture.sort_values('Order Date')
         office = office.sort_values('Order Date')
         furniture = furniture.groupby('Order Date')['Sales'].sum().reset_index()
         office = office.groupby('Order Date')['Sales'].sum().reset_index()
         furniture = furniture.set_index('Order Date')
         office = office.set_index('Order Date')
         y_furniture = furniture['Sales'].resample('MS').mean()
         y_office = office['Sales'].resample('MS').mean()
         furniture = pd.DataFrame({'Order Date':y_furniture.index, 'Sales':y_furniture.values})
         office = pd.DataFrame({'Order Date': y_office.index, 'Sales': y_office.values})
         store = furniture.merge(office, how='inner', on='Order Date')
         store.rename(columns={'Sales_x': 'furniture_sales', 'Sales_y': 'office_sales'}, inplace=True)
         store.head()

Out[20]:
```

	Order Date	furniture_sales	office_sales
0	2014-01-01	480.194231	285.357647
1	2014-02-01	367.931600	63.042588
2	2014-03-01	857.291529	391.176318
3	2014-04-01	567.488357	464.794750
4	2014-05-01	432.049188	324.346545

Figure 22

```python
In [21]: plt.figure(figsize=(20, 8))
         plt.plot(store['Order Date'], store['furniture_sales'], 'b-', label = 'furniture')
         plt.plot(store['Order Date'], store['office_sales'], 'r-', label = 'office supplies')
         plt.xlabel('Date'); plt.ylabel('Sales'); plt.title('Sales of Furniture and Office Supplies')
         plt.legend();
```

Figure 23

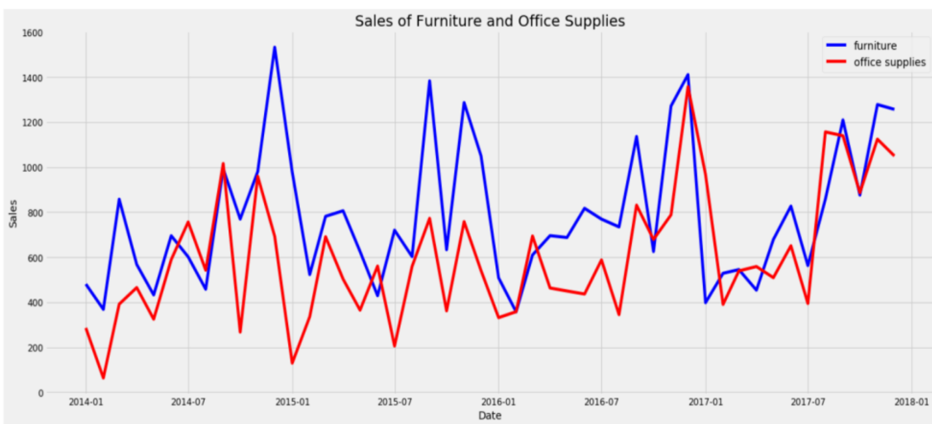


Figure 24

Time Series Modeling with Prophet:-

The Facebook launched Prophet in 2017; forecasting tool Prophet designed for analyzing time-series data that shows the designs on various time scales, for example, yearly, week by week, and every day. Likewise, it has propelled capacities for demonstrating the impacts of occasions on a period arrangement and actualizing custom changepoints. Hence, we are utilizing Prophet to show.

For using FB Prophet, we need PyStan, and PyStan requires a compiler. So to install the compiler go to the below link and follow the instructions:

https://pystan.readthedocs.io/en/latest/windows.html

```
In [30]: #Time Series Modeling using ProphET
import pystan
from fbprophet import Prophet

furniture = furniture.rename(columns={'Order Date': 'ds', 'Sales': 'y'})
furniture_model = Prophet(interval_width=0.95)
furniture_model.fit(furniture)
office = office.rename(columns={'Order Date': 'ds', 'Sales': 'y'})
office_model = Prophet(interval_width=0.95)
office_model.fit(office)
furniture_forecast = furniture_model.make_future_dataframe(periods=36, freq='MS')
furniture_forecast = furniture_model.predict(furniture_forecast)
office_forecast = office_model.make_future_dataframe(periods=36, freq='MS')
office_forecast = office_model.predict(office_forecast)
plt.figure(figsize=(18, 6))
furniture_model.plot(furniture_forecast, xlabel = 'Date', ylabel = 'Sales')
plt.title('Furniture Sales');
```

```
ERROR:fbprophet.plot:Importing plotly failed. Interactive plots will not work.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
INFO:fbprophet:Disabling weekly seasonality. Run prophet with weekly_seasonality=True to override this.
INFO:fbprophet:Disabling daily seasonality. Run prophet with daily_seasonality=True to override this.
```

```
<Figure size 1296x432 with 0 Axes>
```

Figure 25

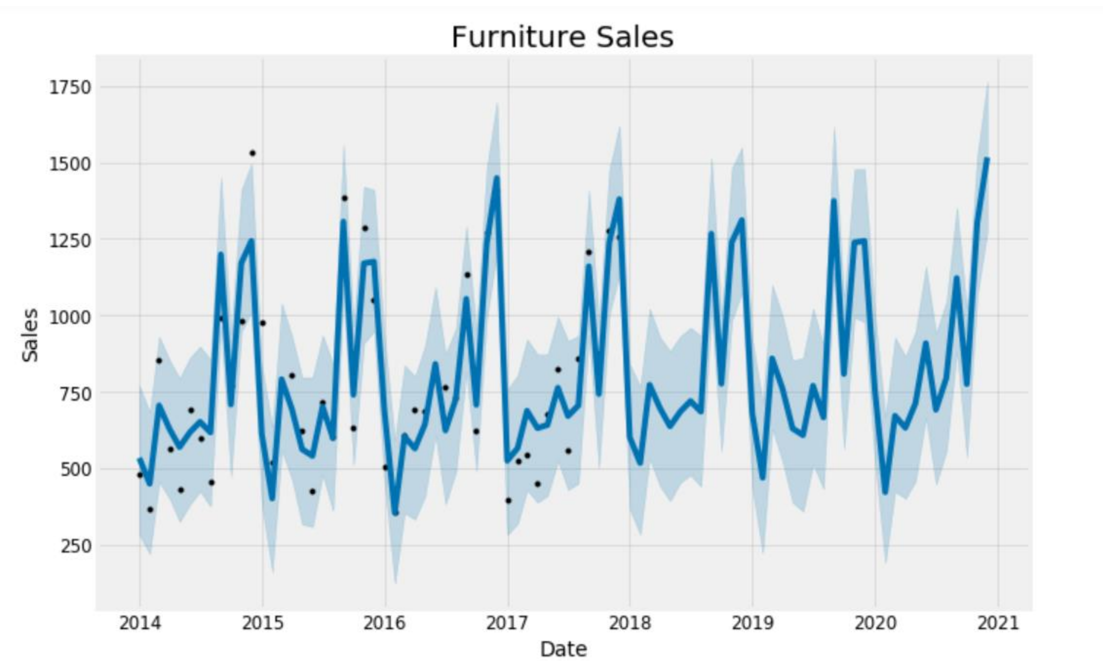


Figure 26

```
In [31]: plt.figure(figsize=(18, 6))
         office_model.plot(office_forecast, xlabel = 'Date', ylabel = 'Sales')
         plt.title('Office Supplies Sales');
```

Figure 27

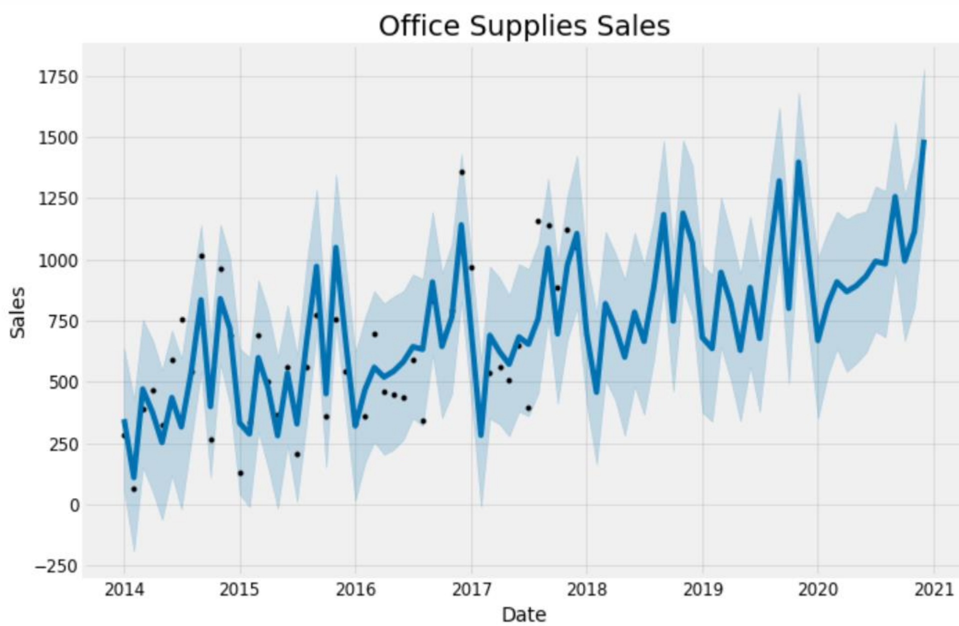


Figure 28

Compare Forecasts:-

We have performed a prediction for the next three years for the above categories. They were combined together to observe the forecast.

While studying the below graphs we have a better understanding of both “Office Supplies Sale” and “Furniture Sales”

```
In [32]: #Comparing Forecasts
furniture_names = ['furniture_%s' % column for column in furniture_forecast.columns]
office_names = ['office_%s' % column for column in office_forecast.columns]
merge_furniture_forecast = furniture_forecast.copy()
merge_office_forecast = office_forecast.copy()
merge_furniture_forecast.columns = furniture_names
merge_office_forecast.columns = office_names
forecast = pd.merge(merge_furniture_forecast, merge_office_forecast, how = 'inner', left_on = 'furniture_ds', right_on = 'office_
forecast = forecast.rename(columns={'furniture_ds': 'Date'}).drop('office_ds', axis=1)
forecast.head()
```

Out[32]:

	Date	furniture_trend	furniture_yhat_lower	furniture_yhat_upper	furniture_trend_lower	furniture_trend_upper	furniture_additive_terms	furniture_additive_terr
0	2014-01-01	726.057713	284.471532	771.151930	726.057713	726.057713	-190.685662	-19
1	2014-02-01	727.494023	222.702430	685.391033	727.494023	727.494023	-276.377703	-27
2	2014-03-01	728.791335	456.902869	931.527775	728.791335	728.791335	-22.389755	-2
3	2014-04-01	730.227645	403.032885	854.289465	730.227645	730.227645	-100.141158	-10
4	2014-05-01	731.617622	327.870406	795.874482	731.617622	731.617622	-160.815662	-16

5 rows × 31 columns

Figure 29

Trend and Forecast Visualization:-

```
In [33]: #Trend and Forecast Visualization
plt.figure(figsize=(10, 7))
plt.plot(forecast['Date'], forecast['furniture_trend'], 'b-')
plt.plot(forecast['Date'], forecast['office_trend'], 'r-')
plt.legend(); plt.xlabel('Date'); plt.ylabel('Sales')
plt.title('Furniture vs. Office Supplies Sales Trend');

WARNING:matplotlib.legend:No handles with labels found to put in legend.
```

Figure 30

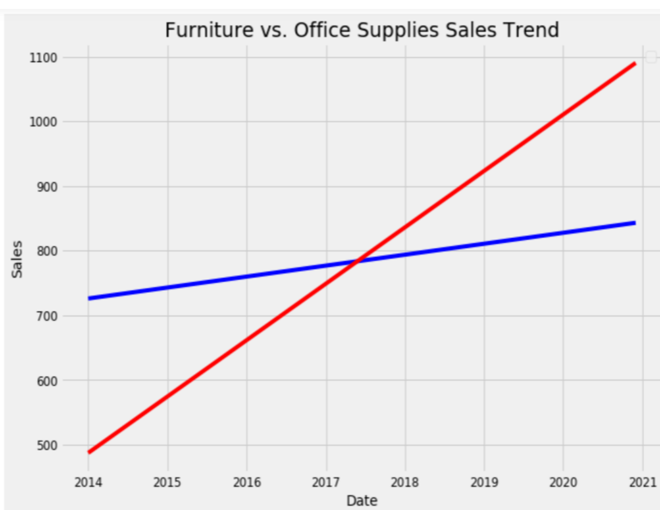


Figure 31

```
In [34]: plt.figure(figsize=(10, 7))
         plt.plot(forecast['Date'], forecast['furniture_yhat'], 'b-')
         plt.plot(forecast['Date'], forecast['office_yhat'], 'r-')
         plt.legend(); plt.xlabel('Date'); plt.ylabel('Sales')
         plt.title('Furniture vs. Office Supplies Estimate');

WARNING:matplotlib.legend:No handles with labels found to put in legend.
```

Figure 32

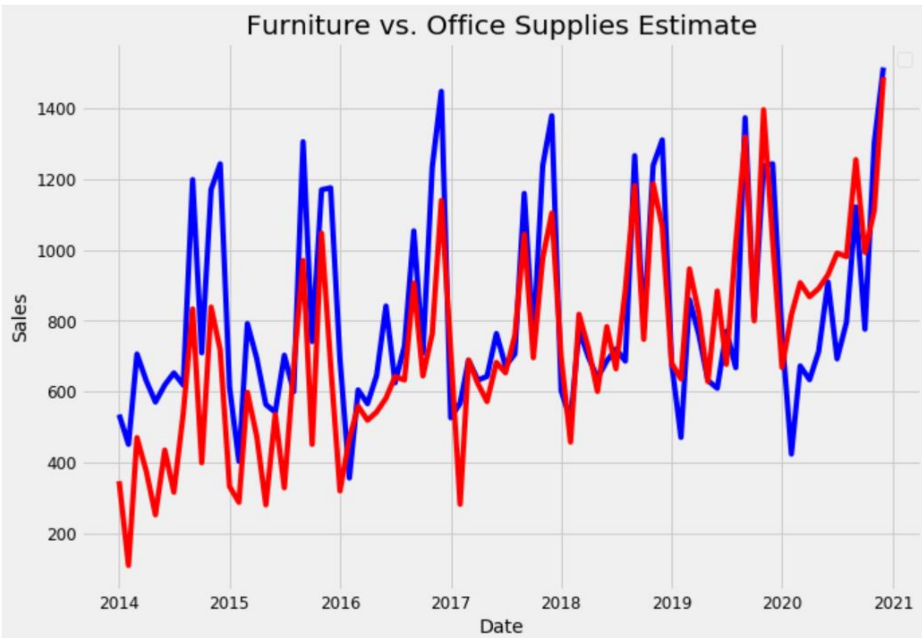


Figure 33

As we can see through Fig. 33 that at the start at 2014 the furniture sales was higher compared to the sale of office supplies.A similar kind of trend continues till end of 2015.Starting 2016 the sales of office supplies and furniture sales almost became equal.

At the start of 2020, but we see a huge drop in the furniture sales, though it gradually rises.

Trends and Patterns:-

We can now use the Prophet Models to inspect different trends of these two categories in the data.

```
In [35]: #using the Prophet Models to inspect different trends of these two categories in the data.
         furniture_model.plot_components(furniture_forecast);
```

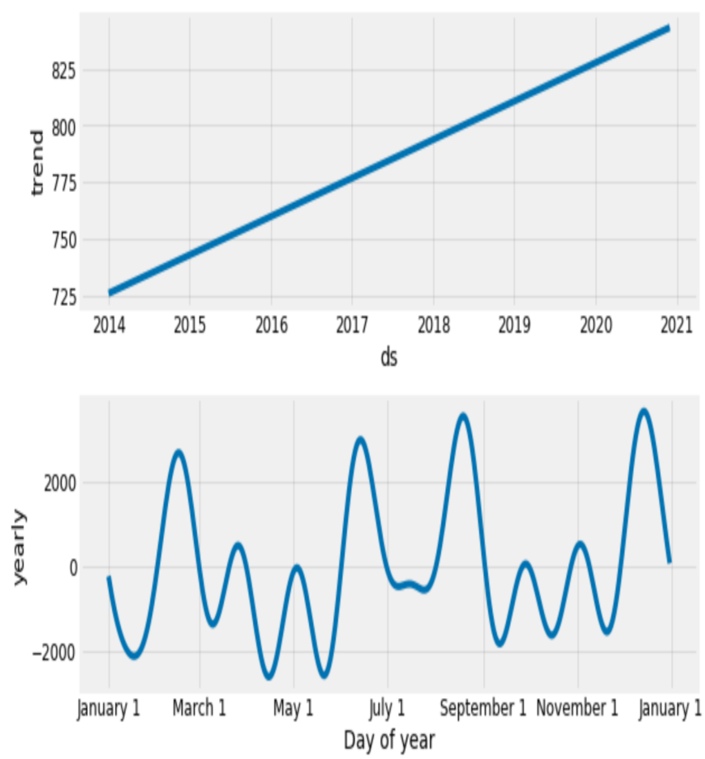


Figure 34

```
In [36]: office_model.plot_components(office_forecast);
```

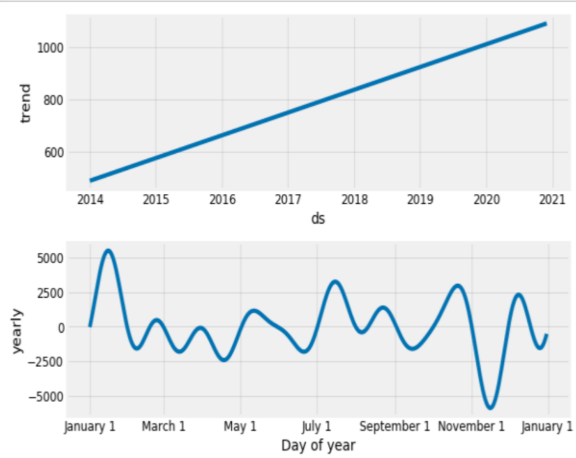


Figure 35

We see that the deals for Furniture and office supplies have been straightly expanding after some time and will continue developing, even though office supplies' development appears to be marginally more grounded.

The most noticeably awful month for Furniture is April, the most noticeably terrible month for office supplies in February. The most significant month for Furniture is December, and the highest month for office supplies in October.

References:

https://otexts.com/fpp2/seasonal-arima.html

https://www.stat.ipb.ac.id/en/uploads/KS/S2%20-%20ADW/3%20Montgomery%20-%20Introduction%20to%20Time%20Series%20Analysis%20and%20Forecasting.pdf