Machine Learning Engineer Nanodegree

Capstone Project Report

Lee, Seang Mei

June 20, 2020

Title: Supply Chain Order Demand Forecasting

I. Definition

Project Overview

Enterprises are attaining double-digit improvements in forecast error rates, demand planning productivity, cost reductions and on-time shipments using machine learning today, revolutionizing supply chain management in the process.

Examples where data analytics and machine learning can be beneficial for supply chain management is within demand forecasting and warehouse optimization.

Accurate demand forecasting enables increased profitability, increased customer satisfaction, reduced inventory stockouts, reduced safety stock requirements and reduced product obsolescence costs.

Problem Statement

For this project, a company has gathered 7 years of their order demand data from 2011 to 2017. Like many other companies, this company wish to create more revenue by accurately master the order demand for coming months and thus make the right decision on their supply chain to fulfill the demand.

With 1.05 million data volume for that 7 years, the goal of this capstone project is to study the trend of order demand by applying machine learning time series model, SARIMA and to produce with a prediction of order demand for the next few years.

Metrics

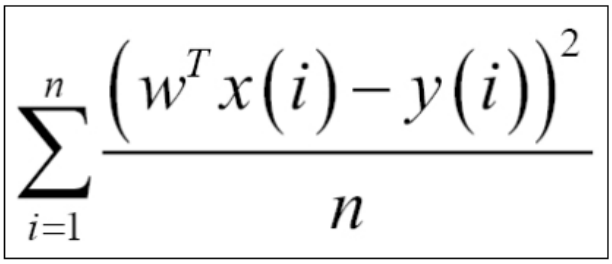
Four commonly used metrics to evaluate a time series model are:

MAE (Mean Average Error)

RSS (Residual Sum Squares)

MSE (Mean Squared Error)

RMSE (Root Mean Squared Error



In this project, MSE is calculated as:

```
mse = ((y_forecasted - y_truth) ** 2).mean()
```

It is the sum, over all the data points, of the square of the difference between the predicted and actual target variables, divided by the number of data points.

RMSE is the square root of MSE.

II. Analysis

Data Exploration

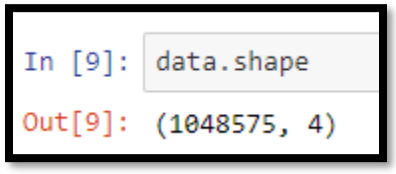
Data is getting from Kaggle. Historical Product Demand.csv - CSV data file containing product demand.

Data is uploaded to AWS S3 bucket. During the research of finding best model, output data in json file format can be stored in S3 bucket.

First, data preprocessing is kick off with checking the number of columns, data types, data size and then convert the date field to datetime as it is found that the date is stored as object. It will not be able to fit into the model later. After the date datatype conversion, proceed to find any null value and drop them. Only 1% of data contains null value and dropped from the data frame.

	Before	After
Warehouse	object	Object
Product_Category	object	object
Date	**object**	**datetime64**
Order_Demand	int64	int64

Fig 1: data types



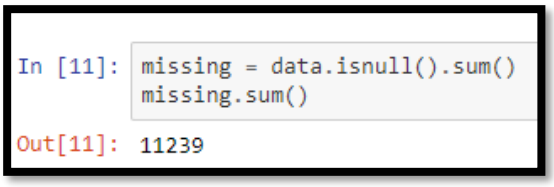


Fig 2.1 and Fig 2.2: total data volume and missing value volume

Algorithms and Techniques

To train the data, there are three trend elements that require configuration.

They are the same as the ARIMA model; specifically:

p: Trend autoregression order.

d: Trend difference order.

q: Trend moving average order.

On top of that, for seasonal trend, I added seasonal_order parameter.

```
import itertools
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)
```

I get the best result with the following configuration which will be discussed further later:

Order: (0, 1, 2)

Seasonal Order: (1, 1, 0, 12)

Benchmark

A widely used model for time series prediction is ARIMA - Autoregressive Integrated Moving Average. ARIMA does not support time series with a seasonal component. The ARIMA model is then extended to SARIMA to support the seasonal component.

To determine whether I should use ARIMA or SARIMA, a decomposition of data can help to check whether the data show a seasonal trend. From the third chart, I can basically decide that there is an obvious repetitive trend over the time. Therefore, a SARIMA model should work with this dataset. To benchmark, I tried with different sets of seasonal_order parameters and get the best model from lowest MSE value. And when I applied (0,0,0,0) as seasonal_order, that is basically ARIMA model.

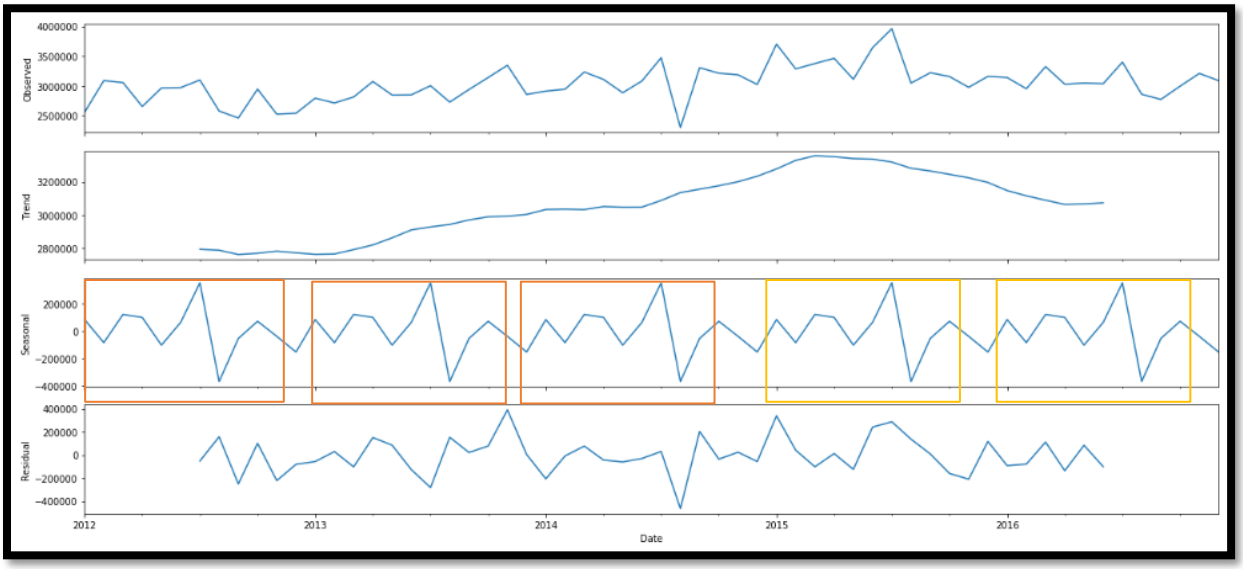


Fig 3: plot the data to check decomposition

III. Methodology

Data preprocessing

1. Explore the datasets, understand the data volume, data types

Product_Code	Warehouse	Product_Category	Date	Order_Demand
Product_0993	Whse_J	Category_028	7/27/2012	100
Product_0979	Whse_J	Category_028	1/19/2012	500
Product_0979	Whse_J	Category_028	2/3/2012	500
Product_0979	Whse_J	Category_028	2/9/2012	500
Product_0979	Whse_J	Category_028	3/2/2012	500

Fig 4: sample data

Checking the data volume and columns in the file by using data.shape (fig2.1).

2. Convert date to datetime format for time series model purpose
   For checking of data type, we found that the date column is 'object' which will caused some error when training the data. Therefore, I converted the field to 'datetime'.

```
data["Date"]= pd.to_datetime(data["Date"])
```

3. Drop any null value found
   See the sample data (fig4) by displaying a few columns then check the missing value in the file. (fig2.2).

4. Check data skew, if it is low skew, it won't impact the result of prediction

```
In [141]: df['Order_Demand'].skew()
Out[141]: 0.0199098134975783
```

   Data skew is pretty low for this dataset.

5. Get the meaningful data range especially oldest data may not be complete

Product_Code	Warehouse	Product_Category	Date	Order_Demand
Product_0642	Whse_C	Category_019	2011-10-31	3
Product_0202	Whse_A	Category_007	2011-11-04	-100
Product_0202	Whse_A	Category_007	2011-11-04	-400
Product_2143	Whse_S	Category_009	2011-11-18	-25
Product_0131	Whse_S	Category_021	2011-11-18	-12
Product_0288	Whse_S	Category_021	2011-11-18	-50
Product_0980	Whse_A	Category_028	2011-11-18	4000
Product_2138	Whse_S	Category_009	2011-11-18	-49
Product_2137	Whse_S	Category_009	2011-11-18	-25
Product_0965	Whse_A	Category_006	2011-11-18	1

Fig 5: data of year 2011 is very less and not complete for a full year, thus considering removing 2011 from analysis.

```
data = data[(data['Date']>='2012-01-01') & (data['Date']<='2016-12-31')].s
ort_values('Date', ascending=True)

df = data.groupby('Date')['Order_Demand'].sum().reset_index()
```

```
In [71]: #Index the date
         df = df.set_index('Date')
         df.index #Lets check the index

Out[71]: DatetimeIndex(['2012-01-01', '2012-01-02', '2012-01-03', '2012-01-04',
                        '2012-01-05', '2012-01-06', '2012-01-08', '2012-01-09',
                        '2012-01-10', '2012-01-11',
                        ...
                        '2016-12-20', '2016-12-21', '2016-12-22', '2016-12-23',
                        '2016-12-25', '2016-12-26', '2016-12-27', '2016-12-28',
                        '2016-12-29', '2016-12-30'],
                       dtype='datetime64[ns]', name='Date', length=1681, freq=None)
```

Fig 6: data is from 2012, starting from January and end with 2016 December.

Implementation

To produce a high accuracy demand forecast, we need to study it along a time series. As such, the machine learning solution for this problem will be using SARIMA - Seasonal Autoregressive Integrated Moving Average method for time series forecasting with univariate data containing trends and seasonality.

First, I used a function to create a time series which will consider leap year and non-leap year. Then charts are plotted to study the trend of order demand for a particular year.

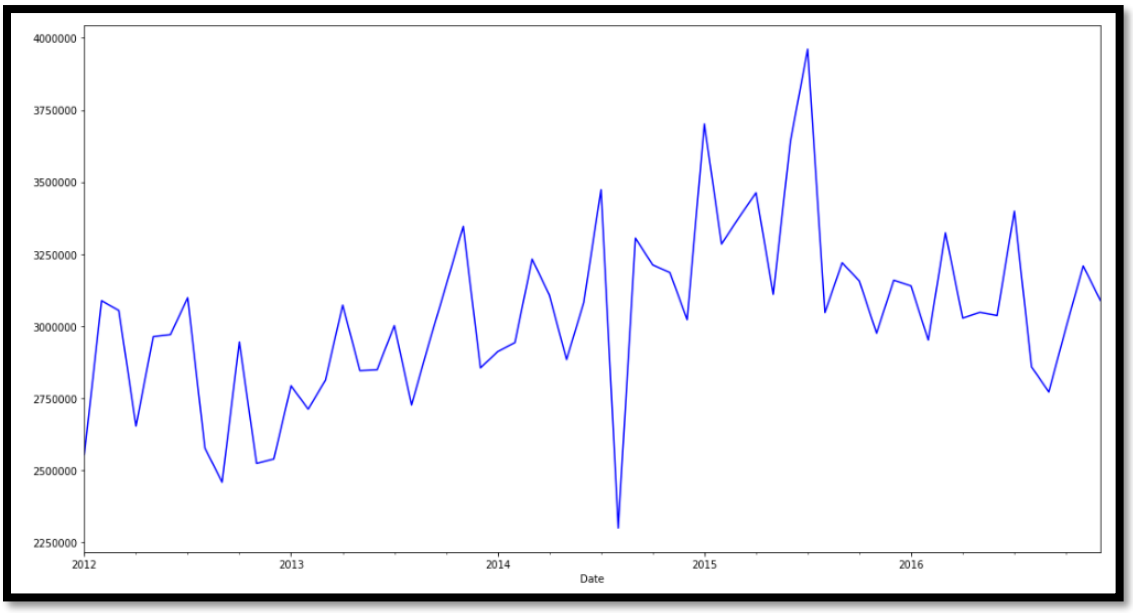


Fig 7: plot the order demand data by year

Train

Data is trained by applying the parameter p, d, q and seasonal_order.

```
In [144]: for param in pdq:
              for param_seasonal in seasonal_pdq:
                  try:
                      mod = sm.tsa.statespace.SARIMAX(y,
                                                      order=param,
                                                      seasonal_order=param_seasonal,
                                                      enforce_stationarity=False,
                                                      enforce_invertibility=False)

                      results = mod.fit()

                      print('SARIMA{}x{}12 - AIC:{}'.format(param, param_seasonal, results.aic))
                  except:
                      continue

SARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1931.5634282849132
SARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1512.4361148419835
SARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:1339.6881722069236
SARIMA(0, 0, 0)x(0, 1, 1, 12)12 - AIC:3234.861516311762
SARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:1367.342362003335
SARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:1341.6788736718381
SARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:1023.8874178559138
```

Fig 8: Data training

Fit

Data is fit to the model and study its coefficient and standard error metrics.

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(0, 1, 2),
                                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]
ma.L1	-0.6170	0.192	-3.212	0.001	-0.994	-0.241
ma.L2	0.0995	0.197	0.504	0.614	-0.287	0.486
ar.S.L12	-0.2518	0.130	-1.936	0.053	-0.507	0.003
sigma2	1.088e+11	1.07e-13	1.01e+24	0.000	1.09e+11	1.09e+11

Fig 9: Data fit to the model

Test

```
pred = results.get_prediction(start=50, dynamic=False) #false is when using the entire history.
#Confidence interval.
pred_ci = pred.conf_int()

#Plotting real and forecasted values.
ax = y['2012':].plot(label='observed')
pred.predicted_mean.plot(ax=ax, label='One-step ahead Forecast', alpha=.8, figsize=(14, 7))
ax.fill_between(pred_ci.index,
                pred_ci.iloc[:, 0],
                pred_ci.iloc[:, 1], color='blue', alpha=.2)
ax.set_xlabel('Date')
ax.set_ylabel('Order_Demand')
plt.legend()
plt.show()
```

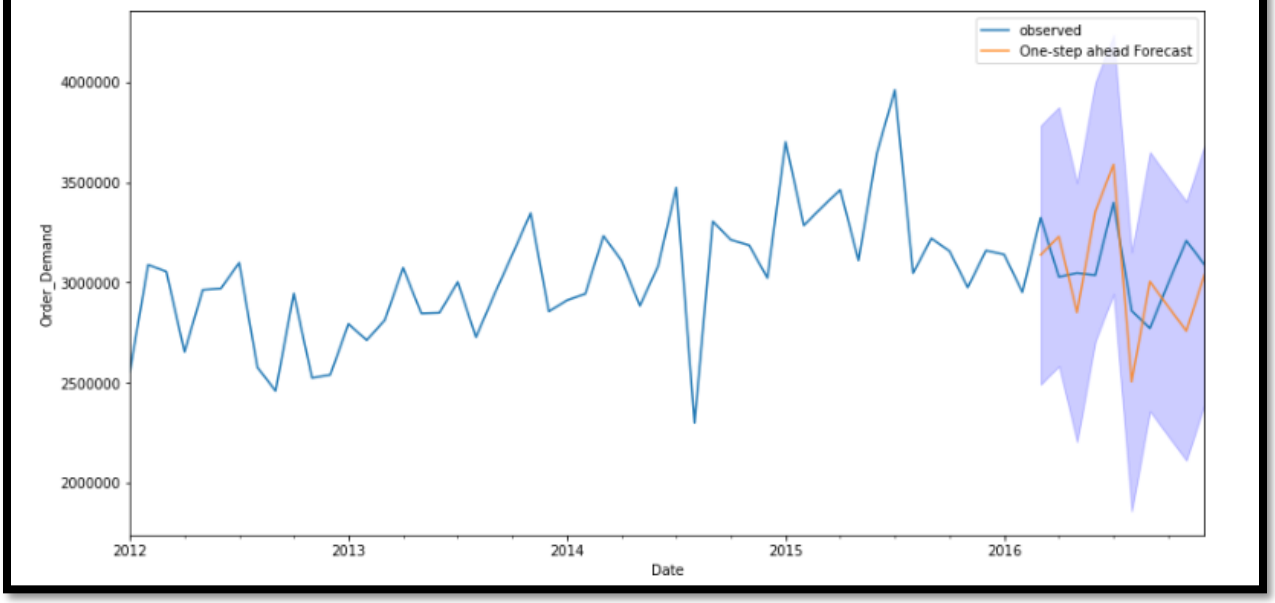


Fig 10: Test the data over 2016 data. The forecast line is basically showing the similar trend with observed line.

Predict

Apply the model to generate a forecasted order demand from 2016 onwards.

```
pred_uc = results.get_forecast(steps=40)
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('Order_Demand')
plt.legend()
plt.show()
```

```
/home/ec2-user/anaconda3/envs/amazonei_mxnet_p36/lib/python3.6/site-packages/statsmodels/tsa/base/tsa_model.py:32
ng: Creating a DatetimeIndex by passing range endpoints is deprecated.  Use `pandas.date_range` instead.
  freq=base_index.freq)
```

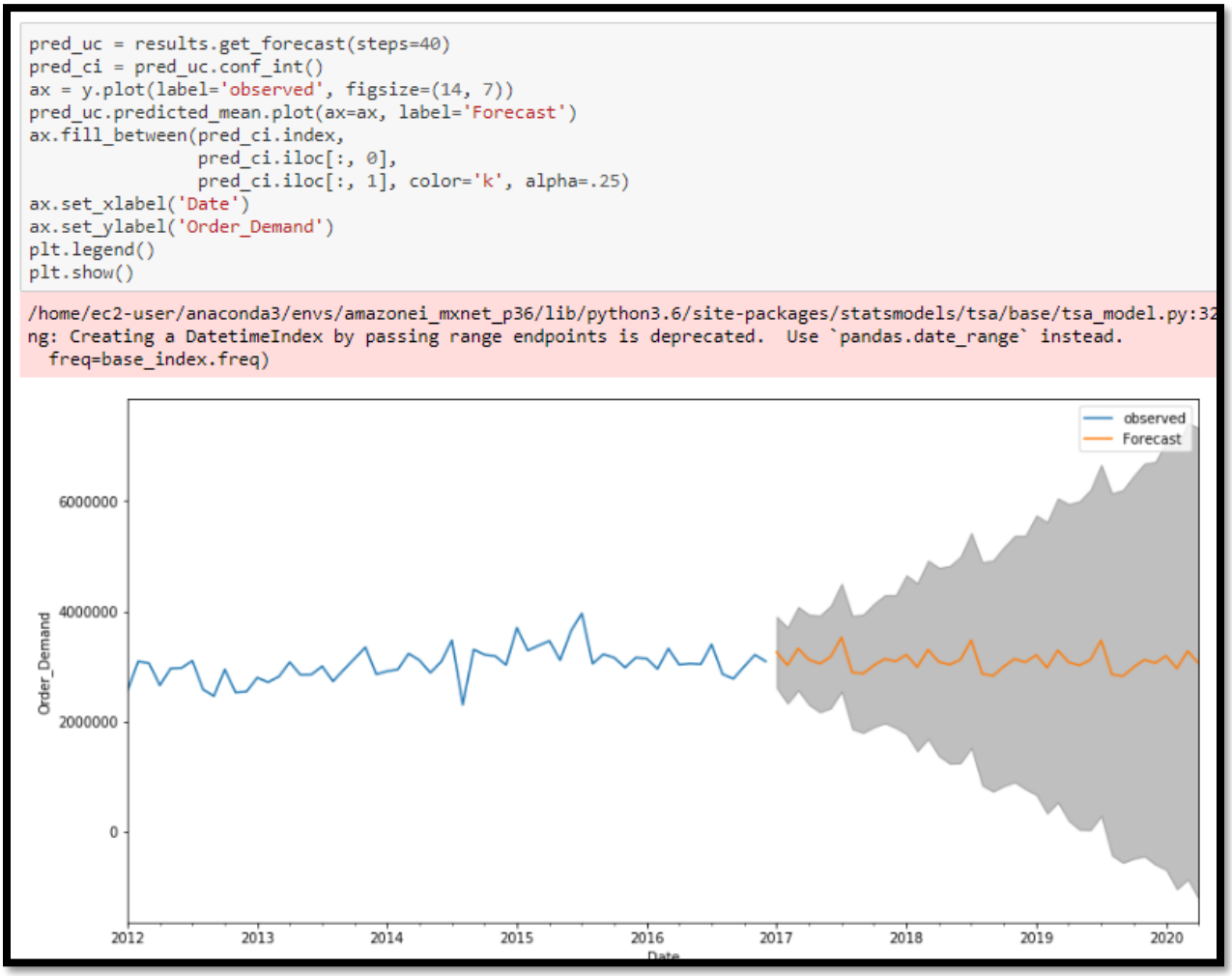


Fig 11: The forecast line is generated for 2017 to 2020.

Refinement

I have applied different set of parameters to the model and run the MSE / RMSE result. I found that the combination of seasonal_order(1,1,0,12) has given the lowest MSE among other combinations and thus I proceed to use that as prediction.

Order: (0, 1, 2)

Seasonal Order: (1, 1, 0, 12)

Example of different combination of parameters:

```
SARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:1931.5634282849132
SARIMA(0, 1, 1)x(1, 1, 0, 12)12 - AIC:989.5378960548436
SARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:960.4017832654685
SARIMA(1, 1, 1)x(1, 1, 1, 12)12 - AIC:3018.72141575552
```

IV. Results

Model Evaluation and Validation

```
In [173]: #Getting the mean squared error (average error of forecasts).
          y_forecasted = pred.predicted_mean
          y_truth = y['2016-12-01':]
          mse = ((y_forecasted - y_truth) ** 2).mean()
          print('MSE {}'.format(round(mse, 2)))

          #Smaller the better.

          MSE 2469360609.11

In [174]: print('RMSE: {}'.format(round(np.sqrt(mse), 2)))

          RMSE: 49692.66
```

Fig 12: result in MSE and RMSE

MSE and RMSE are calculated, though the number is pretty large, this is the lowest MSE and RMSE returned from various combination of parameters.

Justification

The final model achieved better result from benchmark models.

The rest of the combination of parameters applied to the model has produced even higher MSE and RMSE. Therefore, I will not select them as the parameters for final model.

V. Conclusion

Free-Form Visualization

The data show an even distribution on the graphs especially the normal Q-Q, we don't see outliers and data points are scattered along the line.

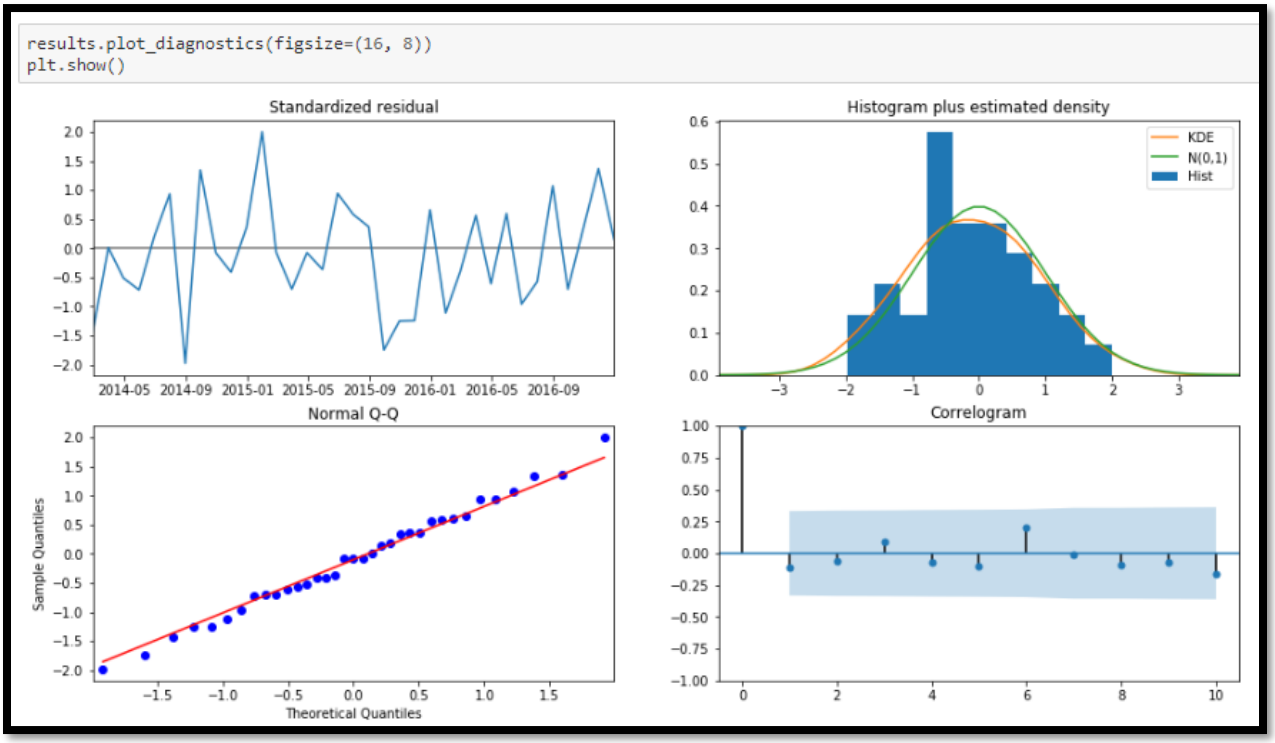


Fig 13: data distributions

Reflection

In this project, I have created time series forecast based on 7 years of data from 2011. We have performed data clean up, study the trend and came out with the prediction for the next few years. The data volume is good enough to feed into the training which will be essential to apply machine learning algorithm as prediction.

Improvement

In this dataset, there are other features like warehouse and products. I have not explored into other options by taking other features into the forecast, it could be a better model if those features are added as part of the training and testing which eventually generate better prediction from different perspectives and help the company in decision making.

References:

Using Machine Learning for Supply Chain

1. https://www.forbes.com/sites/louiscolumbus/2019/04/28/how-to-improve-supply-chains-with-machine-learning-10-proven-ways/#645a66ed3f3c
2. https://www.forbes.com/sites/louiscolumbus/2018/06/11/10-ways-machine-learning-is-revolutionizing-supply-chain-management/#1e3b03ba3e37
3. https://www.blumeglobal.com/learning/machine-learning/
4. https://towardsdatascience.com/artificial-intelligence-in-supply-chain-management-predictive-analytics-for-demand-forecasting-80d2d512f155
5. https://towardsdatascience.com/https-medium-com-h-javedani-how-smart-are-your-supply-chain-predictions-daf5a154ac6d
6. https://en.wikipedia.org/wiki/Inventory_optimization

Data source

7. https://www.kaggle.com/felixzhao/productdemandforecasting

Benchmark model

8. https://www.kaggle.com/imsanjoykb/forecast-order-demand-and-visualization

MSE explanation

9. https://www.freecodecamp.org/news/machine-learning-mean-squared-error-regression-line-c7dde9a26b93/

Techniques for forecasting

10. http://rstudio-pubs-static.s3.amazonaws.com/15285_1260a766696d4960b9c63442a399831e.html