Time-Series Prediction Using Python and DataRobot
-by Qi Sun

The purpose of this study is to predict Superstore Sales on the category of technology.

The dataset was downloaded from: https://community.tableau.com/s/question/0D54T00000CWeX8SAL/sample-superstore-sales-excelxls . There are 9994 records and 21 columns in this dataset.

There are several categories in the Superstore sales data, I will perform time series analysis and forecasting for Technology sales.

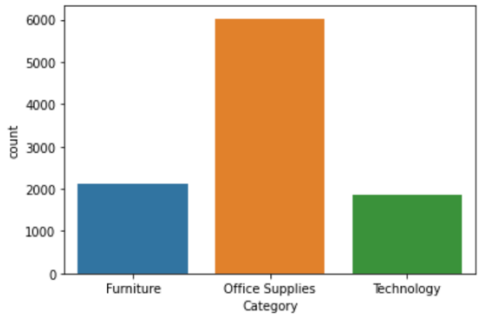
```
print(df.shape)
print(list(df.columns))
```

executed in 27ms, finished 19:45:32 2020-11-08

```
(9994, 21)
['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']
```

```
# plot the category column
sns.countplot(df['Category']);
```

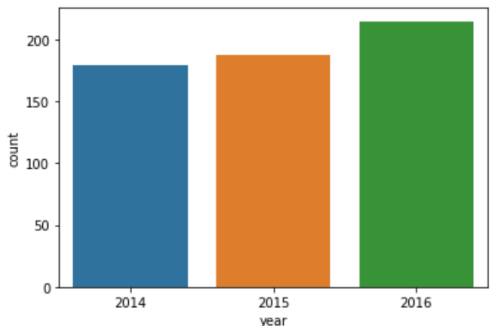
executed in 204ms, finished 19:48:08 2020-11-08



I created train and test files for modeling. The data of 2014, 2015, and 2016 are used as training set, and the data of 2017 as testing set.

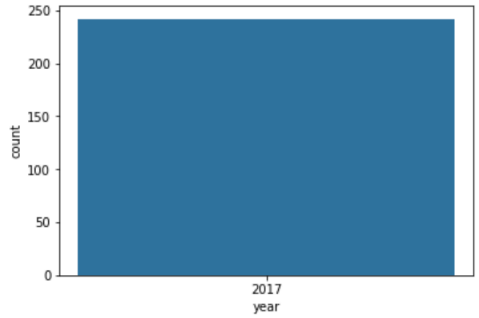
```
# plot the train year column
sns.countplot(train['year']);
```

executed in 195ms, finished 20:41:52 2020-11-08



```
# plot the test year column
sns.countplot(test['year']);
```

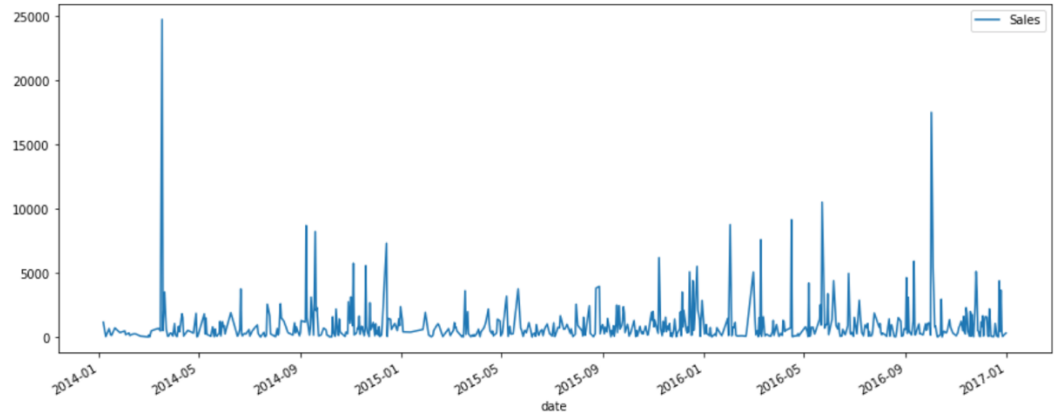
executed in 199ms, finished 20:39:07 2020-11-08



Visualize Technology Sales Time Series Data

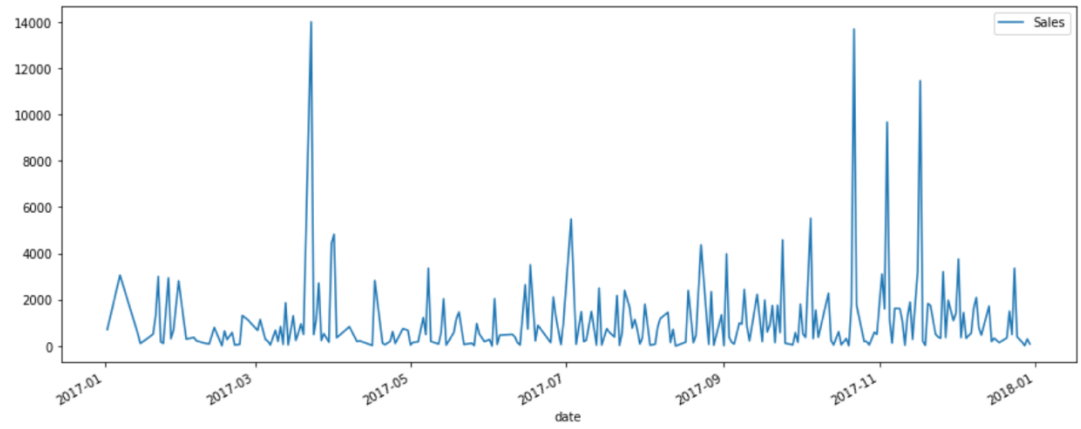
```
# train data set
train02.plot(figsize=(15, 6))
plt.show()
```

executed in 810ms, finished 00:12:54 2020-11-09




```
# test data set
test02.plot(figsize=(15, 6))
plt.show()
```

executed in 376ms, finished 20:54:06 2020-11-08



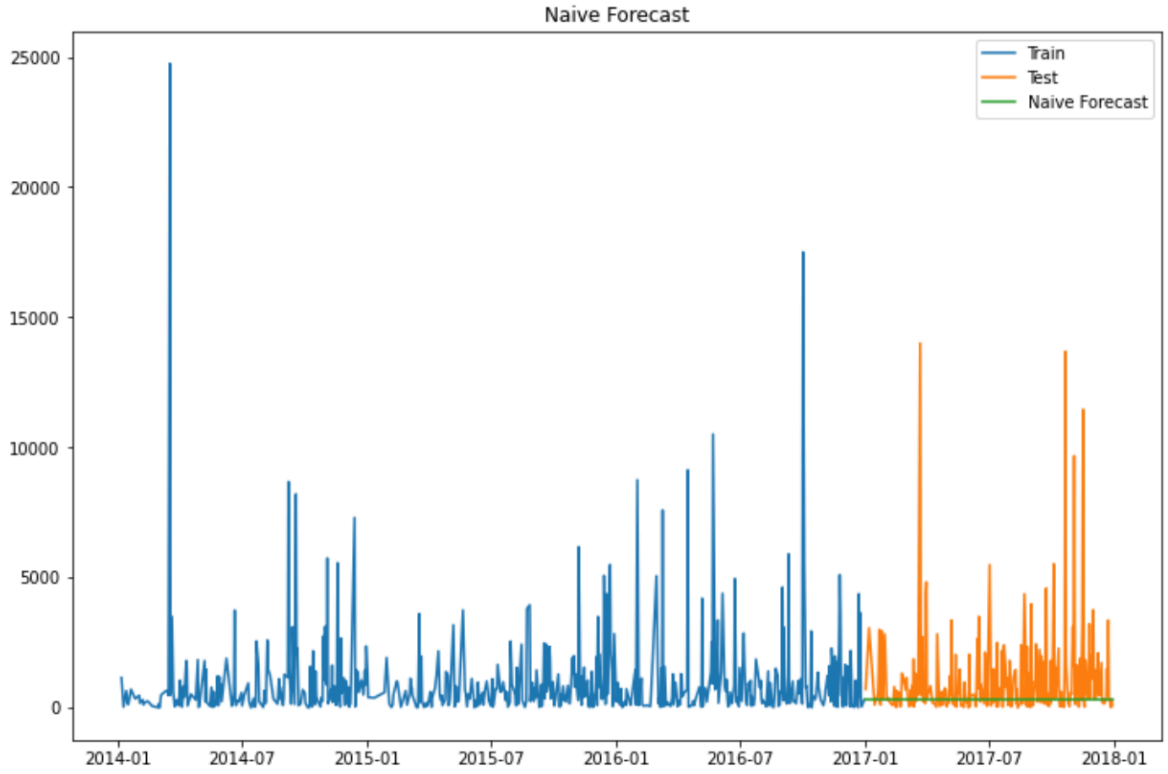
Model 1: Naive Approach

If we want to forecast the sales for the next day, we can simply take the last day sales and estimate the same sales for the next day. Such forecasting technique which assumes that the next expected point is equal to the last observed point is called Naive Method.

$$\text{Hence } \hat{y}_{t+1} = y_t.$$

https://www.analyticsvidhya.com/blog/2018/02/time-series-forecasting-methods/

Now I will implement the Naive method to forecast the sales for test data.



Model Evaluation

I will calculate RMSE to check to accuracy of our model on test data set.

```
from sklearn.metrics import mean_squared_error
from math import sqrt

# calculate RMSE
rms = sqrt(mean_squared_error(test02.Sales, y_hat.naive))
print(rms)
```

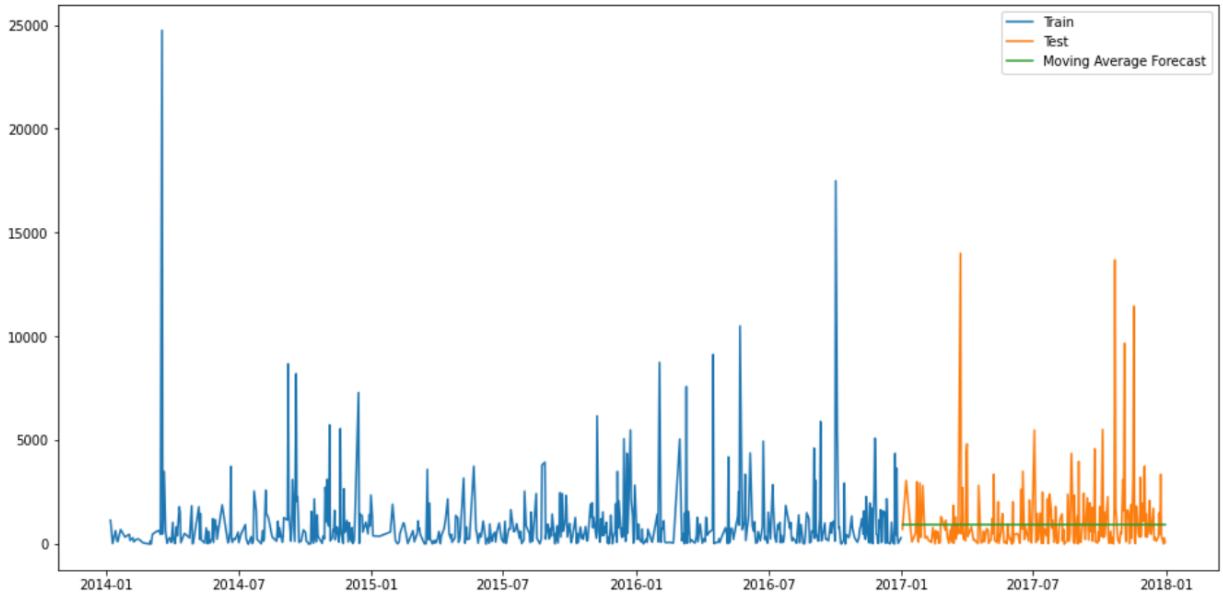
executed in 7ms, finished 21:25:25 2020-11-08

```
1984.1617246655549
```

Model 2 – Moving Average

Using the sales of the initial period would highly affect the forecast for the next period. Therefore as an improvement over simple average, we will take the average of the sales for last few time periods only. Such forecasting technique which uses window of time period for calculating the average is called Moving Average technique.

$$\hat{y}_i = \frac{1}{p}(y_{i-1} + y_{i-2} + y_{i-3} \ldots\ldots + y_{i-p})$$



Model Evaluation

```
# calculate RMSE
rms = sqrt(mean_squared_error(test02.Sales, y_hat_avg.moving_avg_forecast))
print(rms)
```

executed in 8ms, finished 21:12:58 2020-11-08

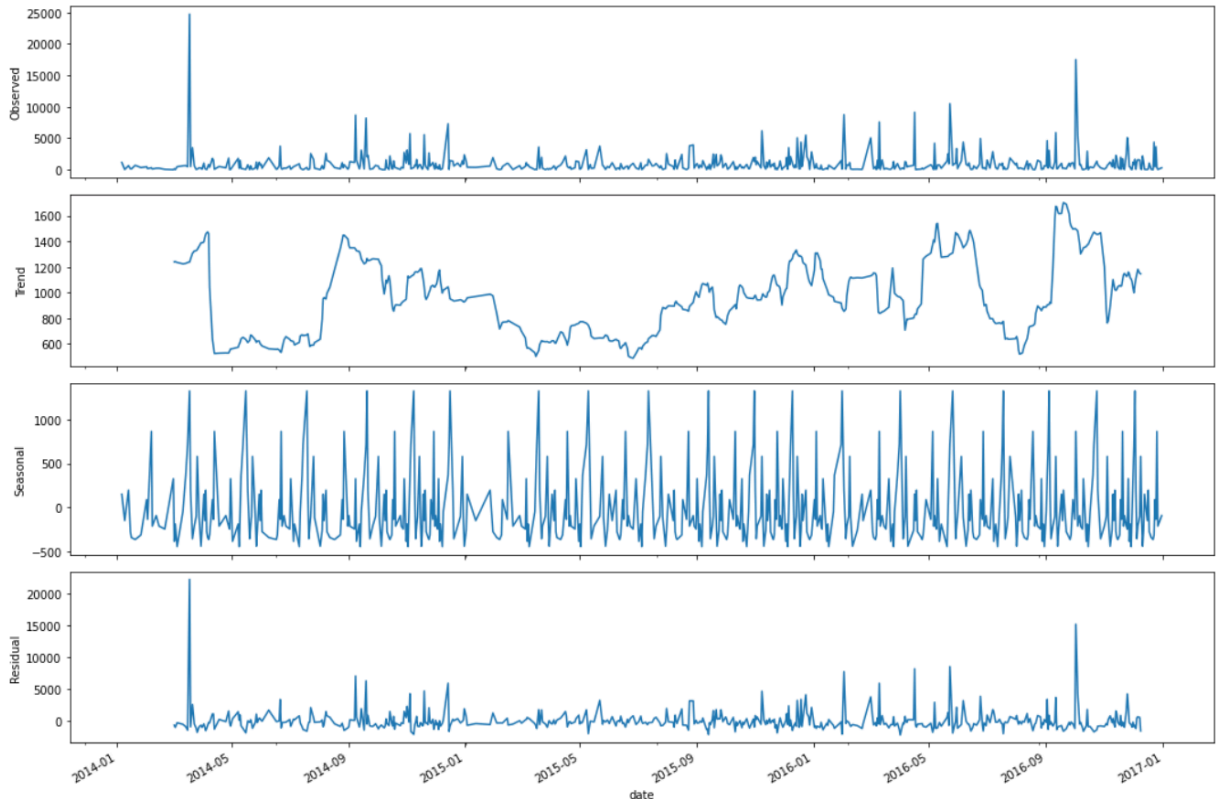
```
1814.6304244905514
```

Model 3 – `Holt's Linear Trend method` and `Holt-Winters Method`

The Naive method would assume that trend between last two points is going to stay the same. But we need a method that can map the trend accurately without any assumptions. Such a method that takes into account the trend of the dataset is called `Holt's Linear Trend method`.

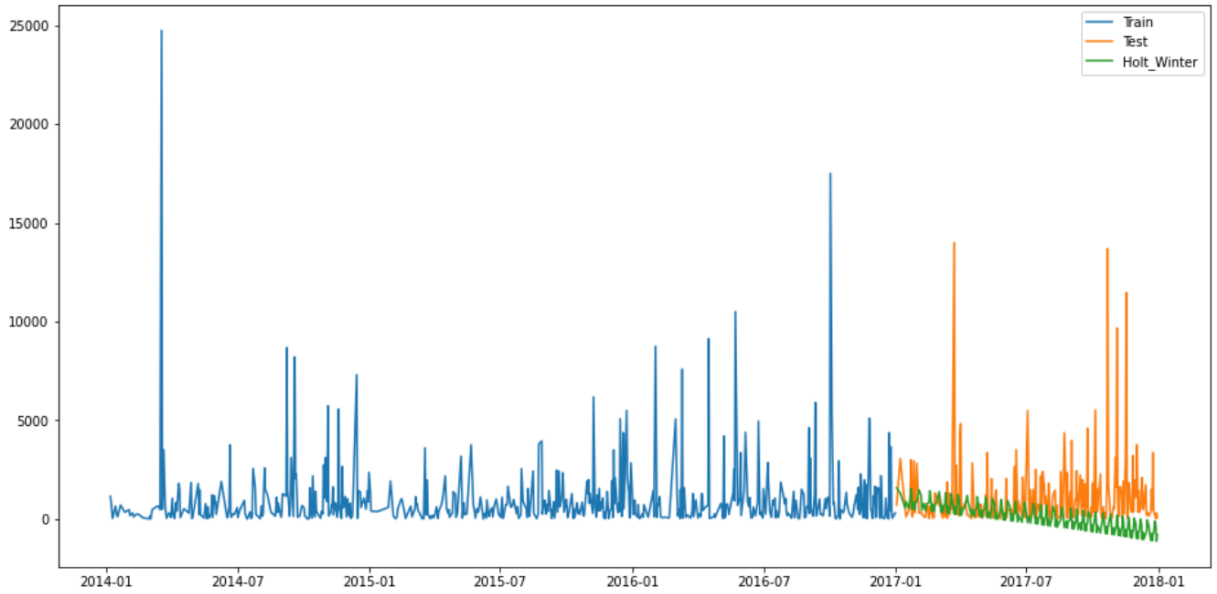
Each Time series dataset can be decomposed into it's componenets which are Trend, Seasonality and Residual. Any dataset that follows a trend can use Holt's linear trend method for forecasting.

However, the above mentioned model does not take into account the seasonality of the dataset while forecasting. Hence we need a method that takes into account both trend and seasonality to forecast future sales. One such algorithm that we can use in such a scenario is `Holt's Winter method`. The idea behind triple exponential smoothing(Holt's Winter) is to apply exponential smoothing to the seasonal components in addition to level and trend.



Findings:

1. We can see from the graphs above that this dataset does not follow an increasing trend. Hence we cannot use Holt's linear trend to forecast the future sales. Holt extended simple exponential smoothing to allow forecasting of data with a trend.
2. They also show that the sales of Technology is unstable, along with its obvious seasonality.



Model Evaluation

```
# calculate RMSE
rms = sqrt(mean_squared_error(test02.Sales, y_hat_avg.Holt_Winter))
print(rms)
```

executed in 8ms, finished 21:48:27 2020-11-08

```
2201.2225054628307
```

Results:

We can see from the graph and RMSE that seasonality did not provide a better solution.

Model 4 - ARIMA

ARIMA, which stands for Autoregressive Integrated Moving Average, is the most commonly used method for time-series forecasting.

ARIMA models are denoted with the notation ARIMA(p, d, q). These three parameters account for seasonality, trend, and noise in data.

ARIMA models are denoted with the notation ARIMA(p, d, q). These three parameters account for seasonality, trend, and noise in data:

The step below is parameter Selection for our technology's sales ARIMA Time Series Model. Our goal here is to use a "grid search" to find the optimal set of parameters that yields the best performance for our model.

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

executed in 4.98s, finished 00:03:47 2020-11-09

```
ARIMA(0, 0, 0)x(0, 0, 0, 12)12 - AIC:791.4891765009955
ARIMA(0, 0, 0)x(0, 0, 1, 12)12 - AIC:1489.3890107482791
ARIMA(0, 0, 0)x(0, 1, 0, 12)12 - AIC:549.0156237882128
ARIMA(0, 0, 0)x(1, 0, 0, 12)12 - AIC:564.1031529698099
ARIMA(0, 0, 0)x(1, 0, 1, 12)12 - AIC:542.9210079120164
ARIMA(0, 0, 0)x(1, 1, 0, 12)12 - AIC:372.6612068172691
ARIMA(0, 0, 1)x(0, 0, 0, 12)12 - AIC:750.8969484750883
```

```
ARIMA(1, 1, 1)x(0, 0, 1, 12)12 - AIC:2600.6915971786916
ARIMA(1, 1, 1)x(0, 1, 0, 12)12 - AIC:518.790371045421
ARIMA(1, 1, 1)x(1, 0, 0, 12)12 - AIC:513.1256895923938
ARIMA(1, 1, 1)x(1, 0, 1, 12)12 - AIC:2718.4094629708843
ARIMA(1, 1, 1)x(1, 1, 0, 12)12 - AIC:343.6037335973312
```

Findings:

The above output suggests that SARIMAX(1, 1, 1)x(1, 1, 0, 12) yields the lowest AIC value of 343.6. Therefore we should consider this to be optimal option.

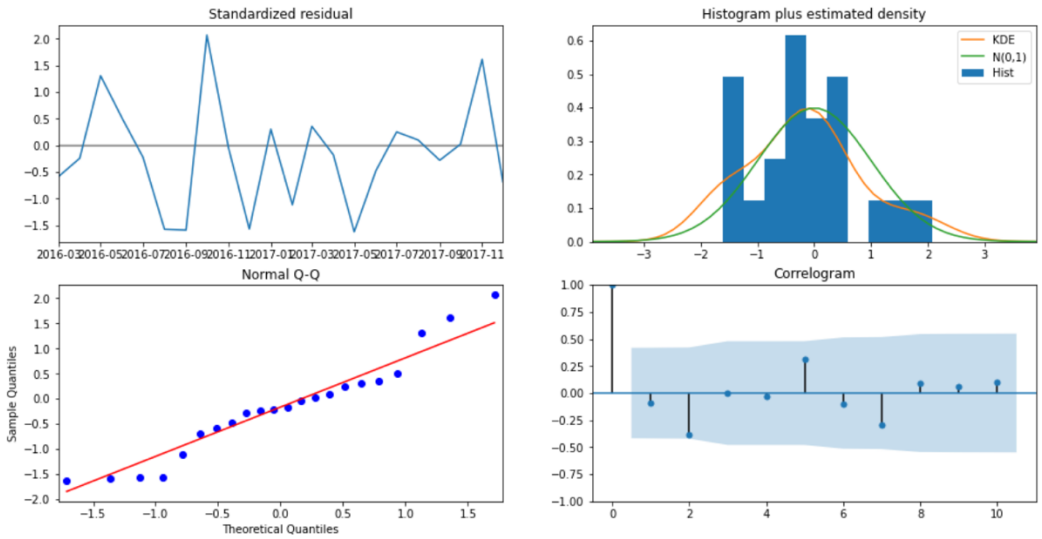
```
# Fitting the ARIMA model
mod = sm.tsa.statespace.SARIMAX(y,
                                order=(1, 1, 1),
                                seasonal_order=(1, 1, 0, 12),
                                enforce_stationarity=False,
                                enforce_invertibility=False)
results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

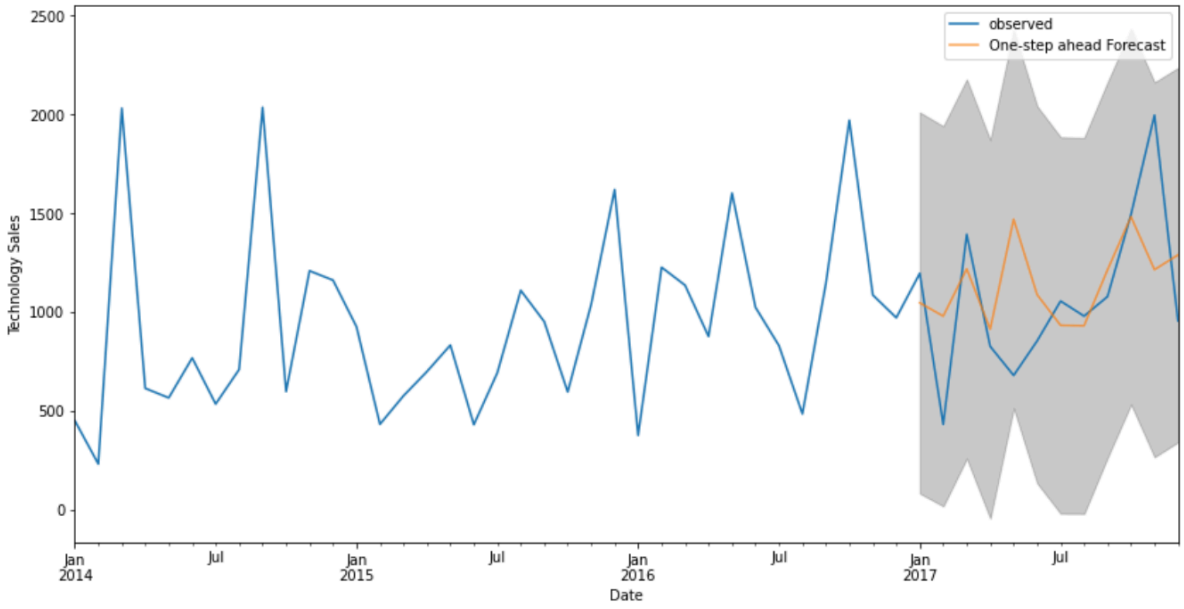
executed in 259ms, finished 00:04:19 2020-11-09

```
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.2660      0.249     -1.067      0.286      -0.755       0.223
ma.L1         -1.0001      0.348     -2.870      0.004      -1.683      -0.317
ar.S.L12      -0.5003      0.175     -2.852      0.004      -0.844      -0.157
sigma2      2.243e+05   1.55e-06   1.44e+11      0.000    2.24e+05    2.24e+05
==============================================================================
```

```
# run model diagnostics to investigate any unusual behavior
results.plot_diagnostics(figsize=(16, 8))
plt.show()
```

executed in 2.02s, finished 00:44:06 2020-11-09





Results:

The line plot above showa the observed values compared to the rolling forecast predictions. Overall, the forecasts align with the true values well, showing an upward trend starts from the beginning of the year and captured the seasonality toward the end of the year.

Model Evaluation

```
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)))
```

executed in 64ms, finished 00:19:39 2020-11-09

```
The Mean Squared Error of our forecasts is 150095.64
```

```
print('The Root Mean Squared Error of our forecasts is {}'.format(round(np.sqrt(mse), 2)))
```

executed in 8ms, finished 00:19:49 2020-11-09

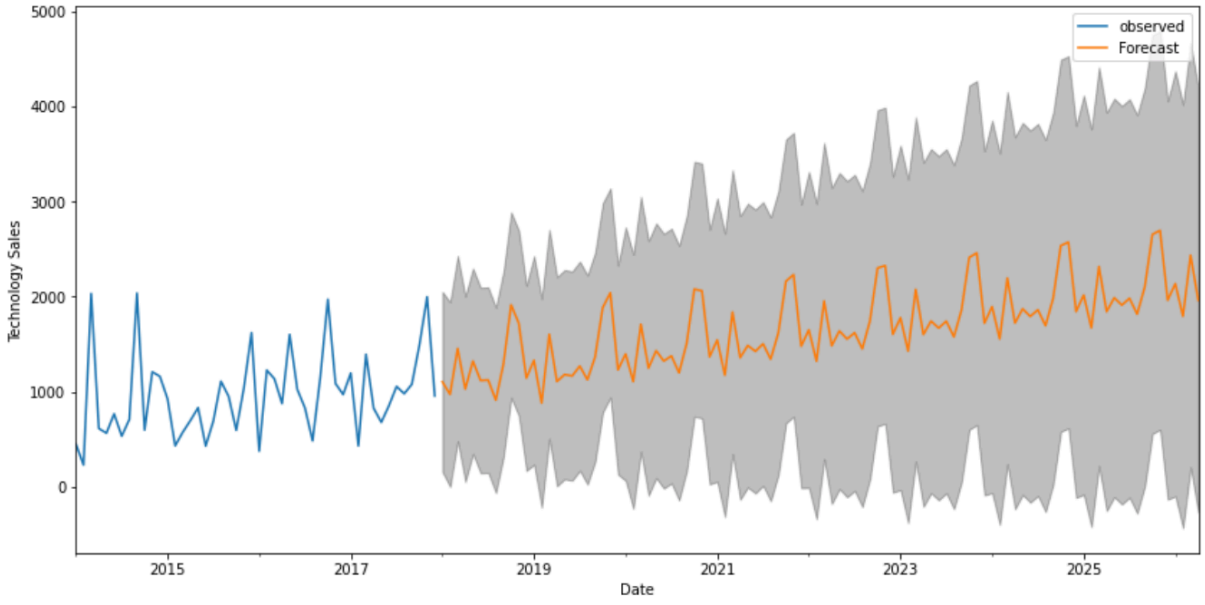
```
The Root Mean Squared Error of our forecasts is 387.42
```

Results:

From all RMSE sbove, the ARIMA model has the lowest RMSE that means the best.

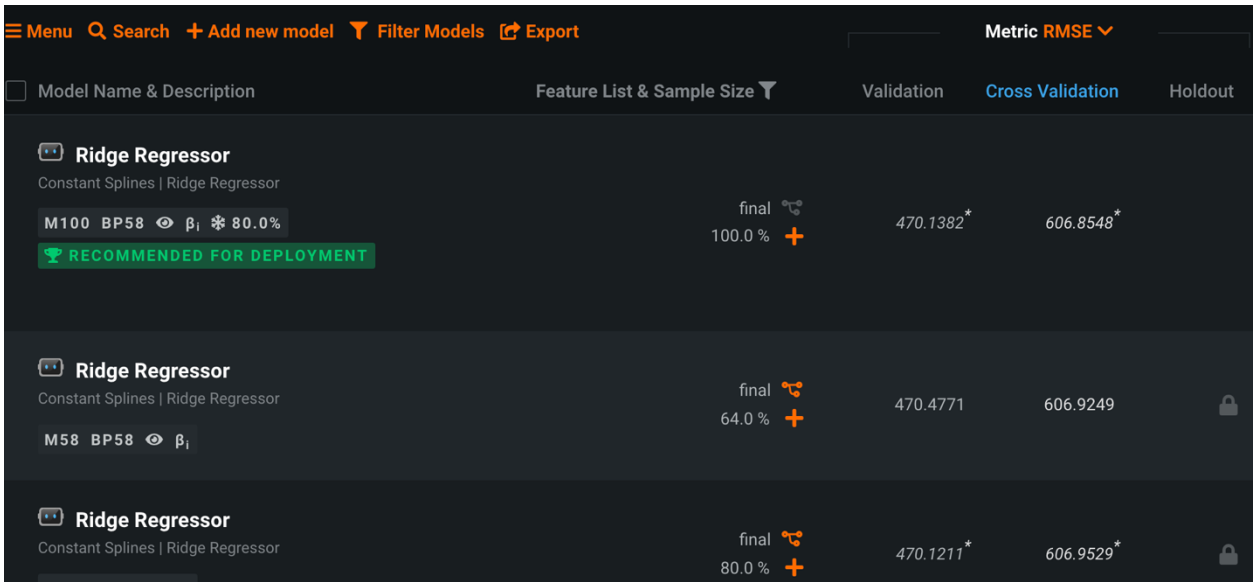
Producing and visualizing forecasts

The ARIMA model below captures technology sales seasonality

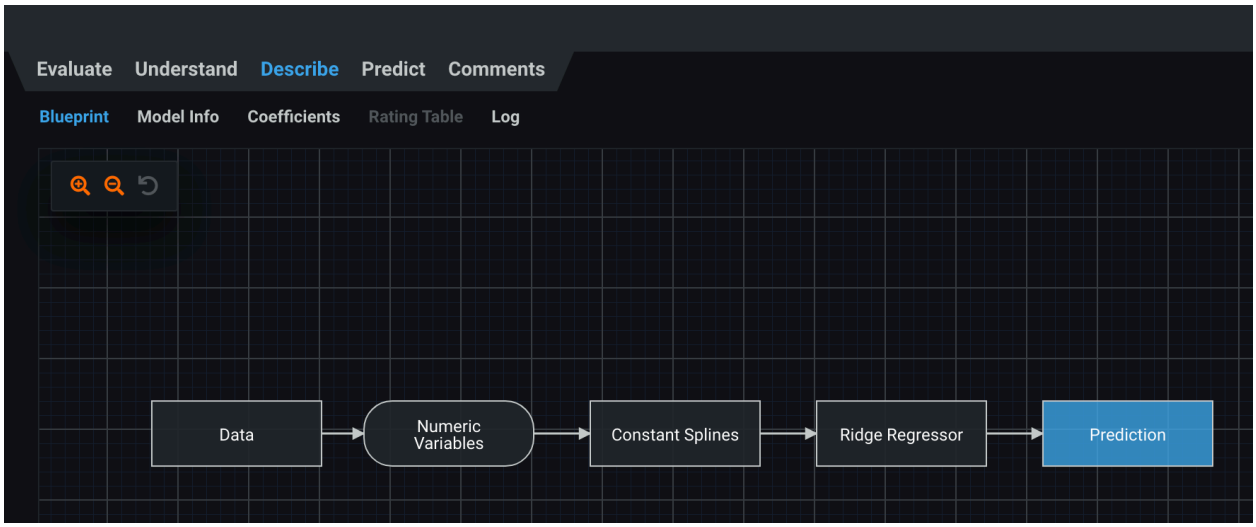


DataRobot:

Since I don't have access to Time Series Modeling in DataRobot, I conducted regular models by using 'sales' as the dependent variable. After automated modeling was complete, the models Leaderboard ranked each machine learning model. The RMSE of recommended model is 606.85.



Here is the model description of the recommended model:



References:

https://towardsdatascience.com/an-end-to-end-project-on-time-series-analysis-and-forecasting-with-python-4835e6bf050b

https://www.analyticsvidhya.com/blog/2018/02/time-series-forecasting-methods/