Product_1268_2

October 12, 2020

1 Product 1268

The products in this dataset are categorized under the `Product_Code` feature. The product for analysis in this notebook is product 620. I used ARIMA and linear regression to perform time series forecasting for product 620.

[1]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
```

[2]:
```
data = pd.read_csv('~/Documents/EECS/EECS_731/HW/EECS731_5/data/data_formatted.
↪csv')
```

[3]:
```
_1268 = data.loc[(data['prod_code'] == 1268)]
```

[4]:
```
_1268.head(5)
```

[4]:

	Unnamed: 0	Date	Order_Demand	year	month	day	prod_code	warehouse	category
16946	16946	2012-12-06	3000	2012.0	12.0	6.0	1268	J	19
24448	24448	2012-01-06	2000	2012.0	1.0	6.0	1268	J	19
24489	24489	2012-02-10	1000	2012.0	2.0	10.0	1268	J	19
24553	24553	2012-03-27	2000	2012.0	3.0	27.0	1268	J	19
24570	24570	2012-04-09	1000	2012.0	4.0	9.0	1268	J	19

[5]:
```
_1268['category'].value_counts()
```

[5]:
```
19    264
Name: category, dtype: int64
```

The `category` and `prod_code` are the same for both columns.

[6]:
```
_1268 = _1268.drop(columns =['Unnamed: 0', 'warehouse', 'category',␣
↪'prod_code'])
```

[7]:
```
_1268.set_index('Date')
```

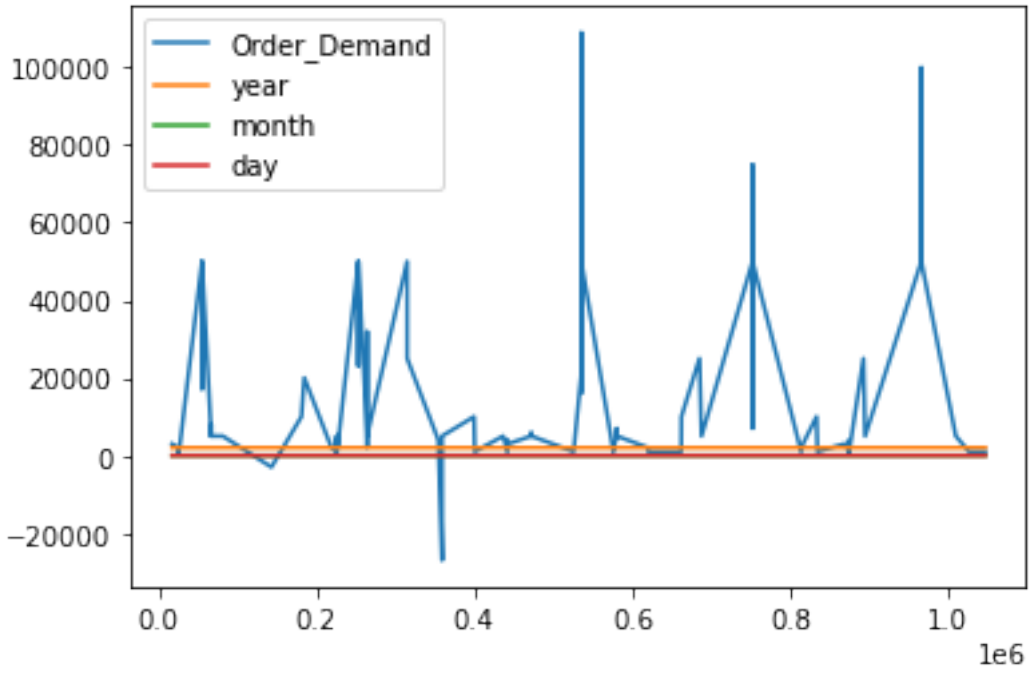
[7]:

Date	Order_Demand	year	month	day
2012-12-06	3000	2012.0	12.0	6.0
2012-01-06	2000	2012.0	1.0	6.0
2012-02-10	1000	2012.0	2.0	10.0
2012-03-27	2000	2012.0	3.0	27.0
2012-04-09	1000	2012.0	4.0	9.0
…	…	…	…	…
2016-12-27	5000	2016.0	12.0	27.0
2016-09-23	1000	2016.0	9.0	23.0
2016-09-05	1000	2016.0	9.0	5.0
2016-03-11	1000	2016.0	3.0	11.0
2016-05-11	1000	2016.0	5.0	11.0

[264 rows x 4 columns]

[8]:
```
_1268.plot()
```

[8]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x10b89dbe0>
```

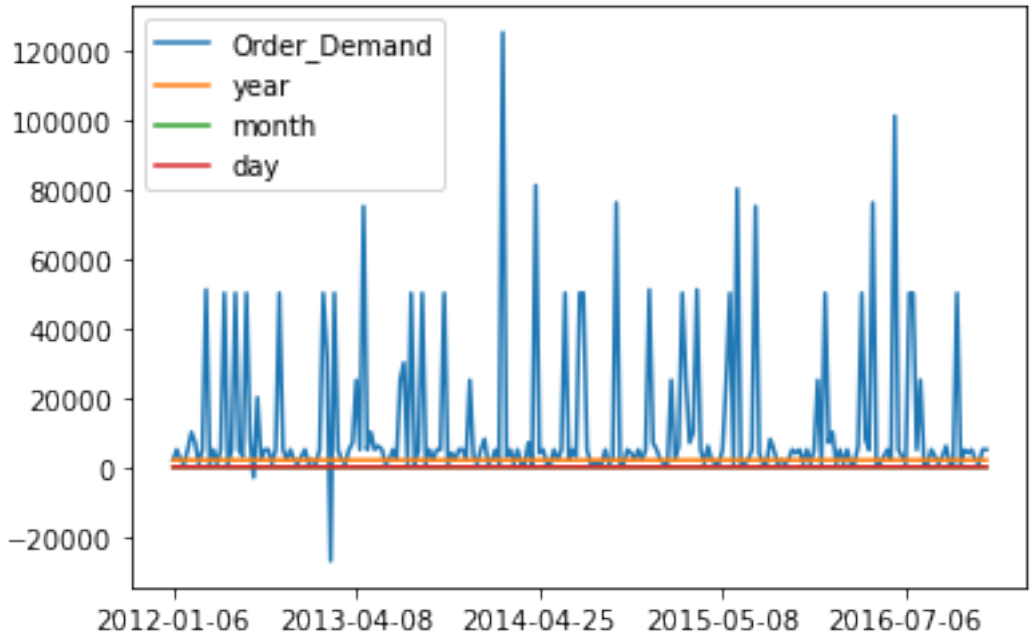


There are some negative values. I realized there could be multiple orders on a specific day. So, I needed to take the sum of all the orders on a particular day.

[9]:
```
aggregation_functions = {'Order_Demand':'sum', 'year':'first', 'month':'first',␣
↪'day':'first'}
prod1268 = _1268.groupby(_1268['Date']).aggregate(aggregation_functions)
prod1268.index.name = None
withdates_1268 = prod1268
```

[10]:
```
prod1268.plot()
```

[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x11f521eb0>
```

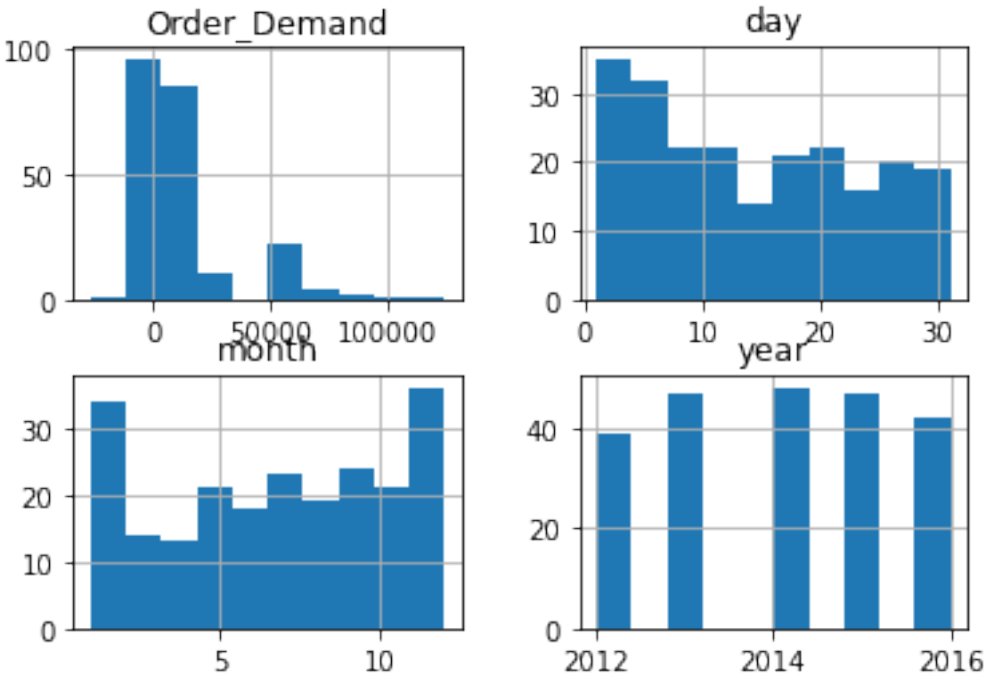


From this plot, there isn't a clear trend. It seems like there are spikes, but it is hard to see the correlation as to when those spikes occur.

I am also not sure why there are negative values. Those should have been aggregated with the sume function above.

[11]:
```
prod1268.hist()
```

[11]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x11f5a6fd0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11f5dc370>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x11f607820>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x11f632ca0>]],
      dtype=object)
```



It looks like there is no data from before 2012. It also looks like the majority of sales are in month 1 and month 12. So, it must be a seasonal item for December and January. Maybe a scarf or a coat? It also looks like the most popular amount to order is close to 1,400 items. The day appears to have no impact.

[12]:
```
from statsmodels.tsa.arima_model import ARIMA
from sklearn.metrics import mean_squared_error
from datetime import datetime
from matplotlib import pyplot
```

[13]:
```
prod1268 = prod1268.drop(columns =['day', 'month', 'year'])
```

Drop the columns for day, month, and year to feed it into the model.

2 ARIMA

I learned about the ARIMA model here https://machinelearningmastery.com/arima-for-time-series-forecasting-with-python/

In order to fit and predict the data correctly, I needed to split the data into a train and test group. I used the first 2/3 of the data for training and the last 1/3 for testing.

The parameters in the ARIMA model are set to 2,2 and 0 for the lag value, difference order, and moving average, respectively. I choose 2 for the order differencing because there isn't a trend over time. It looks like there is only a seasonal trend.

[14]:
```
model = ARIMA(prod1268, order=(2,2,0))
```

```
/Users/annarosefritz/opt/anaconda3/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:216: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
/Users/annarosefritz/opt/anaconda3/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:216: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
```

[15]:
```
model_fit = model.fit(disp=0)
```

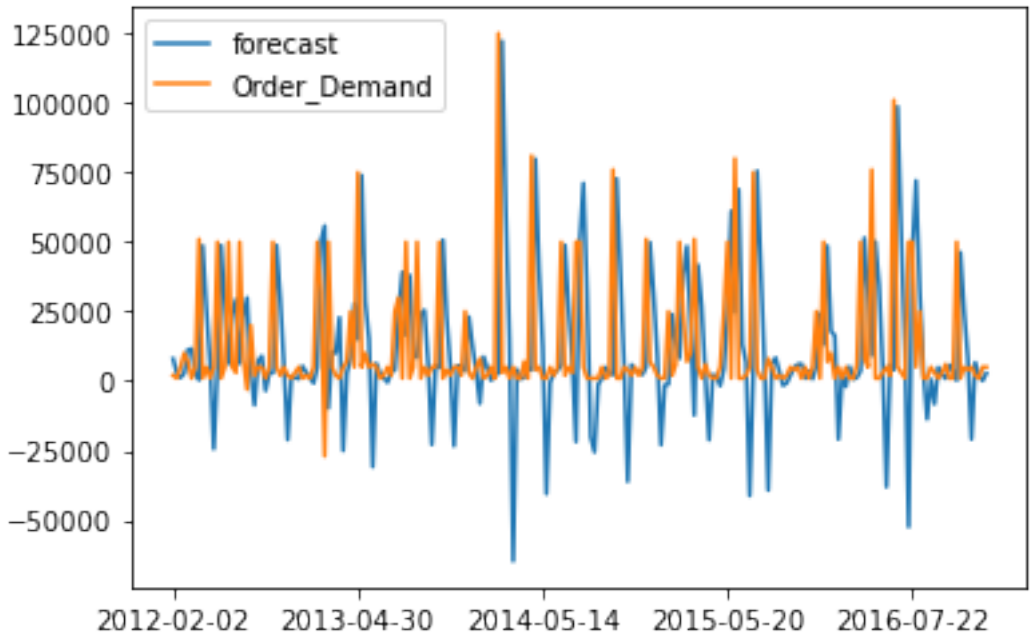
[16]:
```
print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:        D2.Order_Demand   No. Observations:                  221
Model:                 ARIMA(2, 2, 0)   Log Likelihood               -2618.878
Method:                       css-mle   S.D. of innovations          33802.552
Date:                Mon, 12 Oct 2020   AIC                           5245.757
Time:                        10:32:24   BIC                           5259.349
Sample:                             2   HQIC                          5251.245

================================================================================
=========
                               coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---------
const                    -13.5747    889.189     -0.015      0.988   -1756.353
1729.204
ar.L1.D2.Order_Demand     -1.0200      0.056    -18.250      0.000      -1.130
-0.910
ar.L2.D2.Order_Demand     -0.5468      0.056     -9.826      0.000      -0.656
-0.438
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -0.9327           -0.9793j            1.3523           -0.3711
AR.2           -0.9327           +0.9793j            1.3523            0.3711
-----------------------------------------------------------------------------
```

I passed the order values 5,1,0 which means 5 is the lag value, the difference order is 1 becuase I wanted a stationary series, and 0 is the moving average.

The `predict()` function is used to predict sales for future times. Since I was forecasting based on time, I had to be careful on how I split the training and testing group. So, I fed the training group into the model and used the testing group to generate a predicition.

[17]:
```
model_fit.plot_predict(dynamic=False)
plt.show()
```



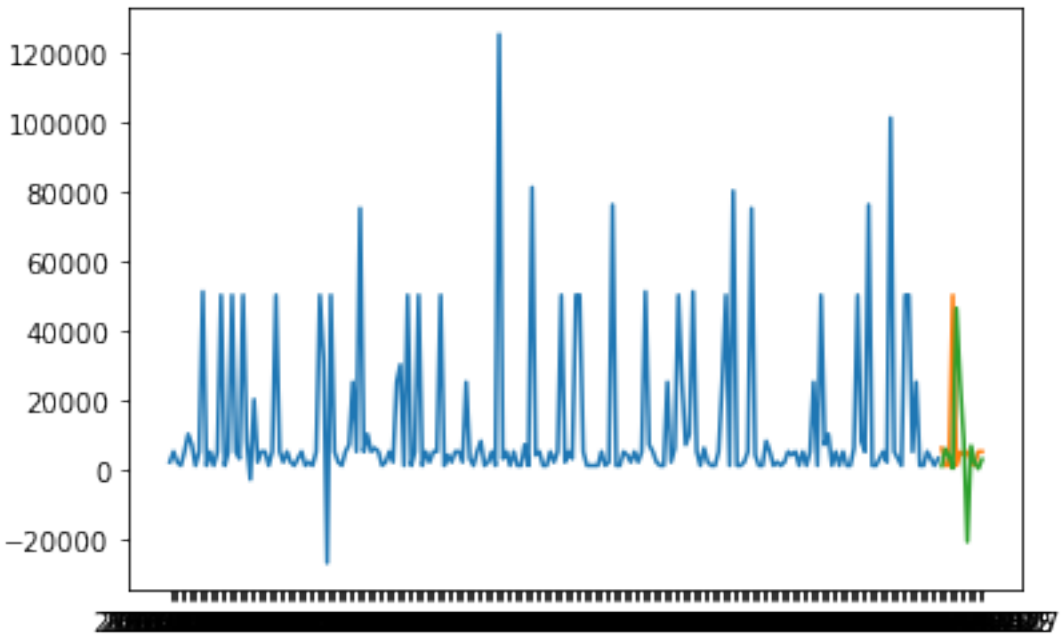
3 forecasting

Now, we forecast

[18]:
```
#X = x_620.values
X = prod1268
size = int(len(X) * 0.66)
train = X.iloc[:len(X)-12]
test = X.iloc[len(X)-12:]
```

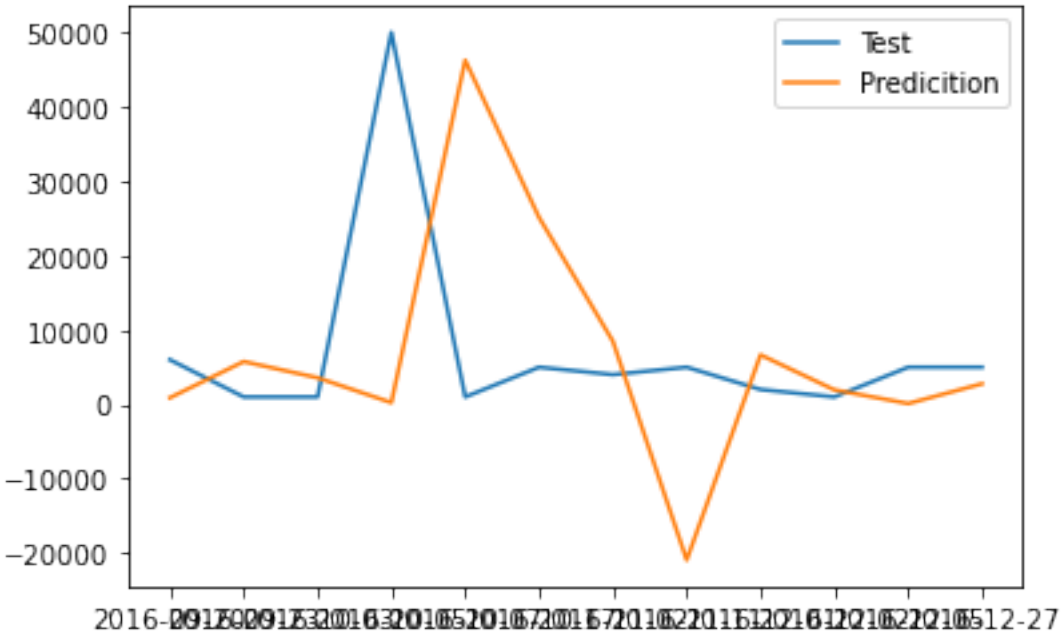
[19]:
```
forecast = model_fit.predict(start = len(train), end=len(train)+len(test)-1,␣
↪typ = 'levels').rename("Predictions")

plt.plot(train, label='Train')
plt.plot(test, label='Test')
plt.plot(forecast, label='Predicition')
```

[19]:
```
[<matplotlib.lines.Line2D at 0x12248d580>]
```



[20]:
```
plt.plot(test, label='Test')
plt.plot(forecast, label='Predicition')
plt.legend(loc='best')
#plt.show()
```

[20]: <matplotlib.legend.Legend at 0x122769eb0>



By zooming in on the predictions, we see that the test values (actual order demand) and the predictions don't match great. This means the model isn't very accurate.

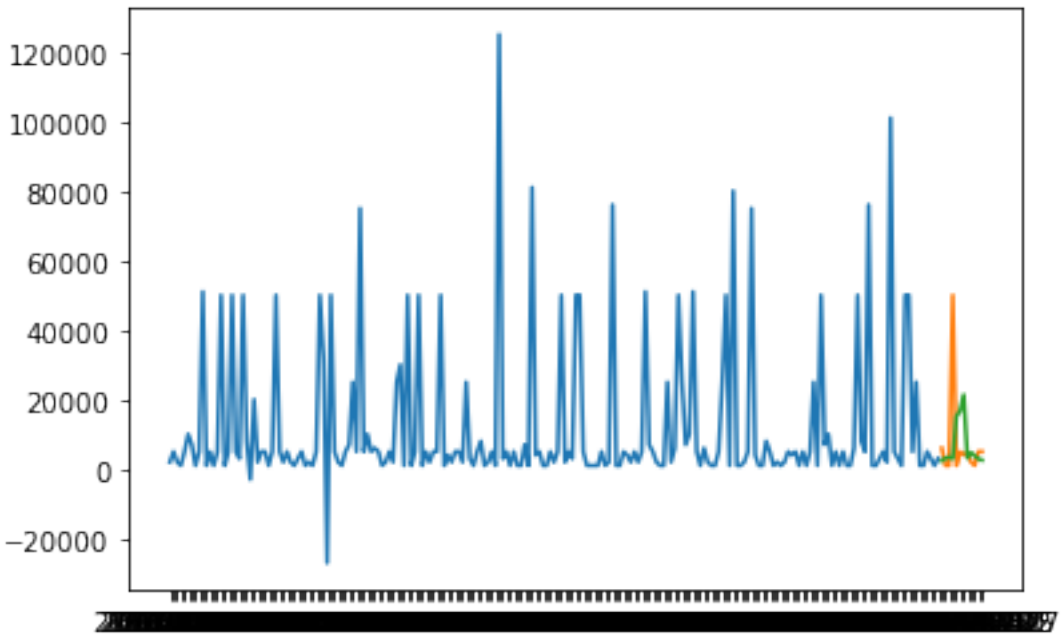
4 Trying Again

This time, used 1 as the difference order.

[21]:
```
model2 = ARIMA(prod1268, order=(2,1,0))
model_fit = model2.fit(disp=0)

forecast = model_fit.predict(start = len(train), end=len(train)+len(test)-1,␣
↪typ = 'levels').rename("Predictions")

plt.plot(train, label='Train')
plt.plot(test, label='Test')
plt.plot(forecast, label='Predicition')
```

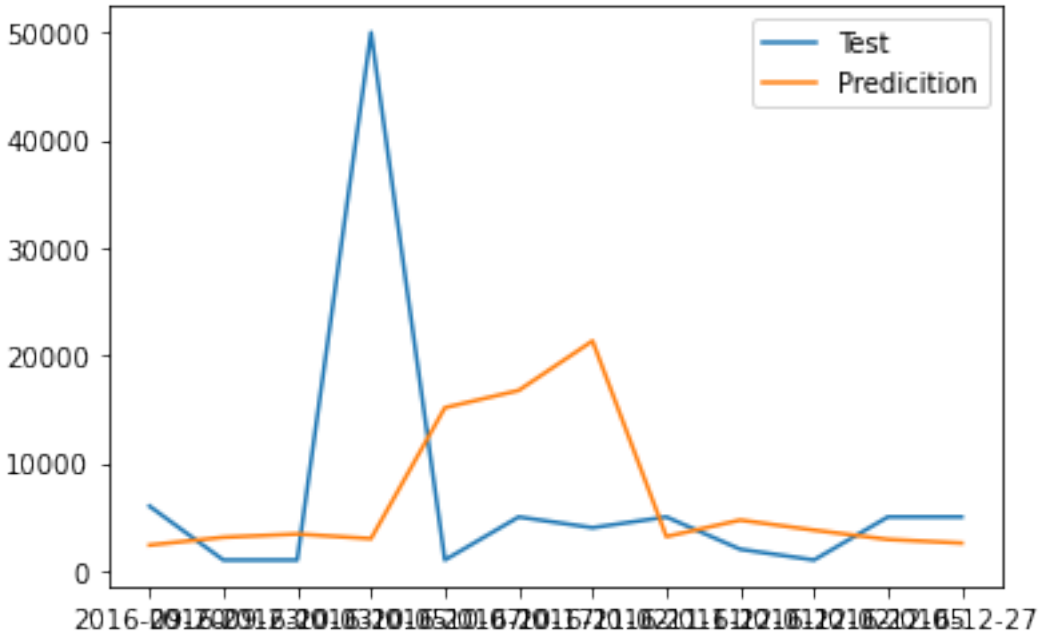
```
/Users/annarosefritz/opt/anaconda3/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:216: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
/Users/annarosefritz/opt/anaconda3/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:216: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it has no'
```

[21]:
```
[<matplotlib.lines.Line2D at 0x122404e20>]
```



[22]:
```
plt.plot(test, label='Test')
plt.plot(forecast, label='Predicition')
plt.legend(loc='best')
#plt.show()
```

[22]: <matplotlib.legend.Legend at 0x12010ecd0>



It seems to work a little better. At least it's not negative.

5 Linear Regression

The second model I tried was linear regression.

[23]:
```
from sklearn.linear_model import LinearRegression
```

Using linear regression, I wanted to predict the sales using the month.

We still must make a training set and a testing set. I split the data up into thirds and the first 2/3 were training and the last 1/3 was testing.

[24]:
```
x_1268 = withdates_1268.drop(columns=['Order_Demand'])
X = x_1268.values
size = int(len(X) * 0.66)
train, test = X[0:size], X[size:len(X)]
```

[25]:
```
y = withdates_1268.Order_Demand
size = int(len(y) * 0.66)
ytrain, ytest = y[0:size], y[size:len(X)]
```

[26]:
```
model = LinearRegression()
```

[27]:
```
model.fit(train, ytrain)
```

[27]:
```
LinearRegression()
```

[28]:
```
r_sq = model.score(test, ytest)
print('coefficient of determination:', r_sq)
```

```
coefficient of determination: -0.00802553838268194
```

Terrible coefficient of determination.

5.1 Forecasting

However, we can still use the model to make predicitions and forecast future sales. So, if you want to predict the demand for the first of april, 2016 then you can feed that into the model.

[29]:
```
pred = [[2016, 4, 1]]
```

[30]:
```
[[2016, 4, 1]]
```

[30]:
```
[[2016, 4, 1]]
```

[31]:
```
pred_demand = model.predict(pred)
print(pred_demand)
```

```
[14943.20269673]
```

[32]:
```
pred = [[2018, 1, 1]]
pred_demand = model.predict(pred)
print(pred_demand)
```

```
[18271.64114472]
```