Product_620

October 12, 2020

1 Product 620

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]:
```
_620 = data.loc[(data['prod_code'] == 620)]
```

[4]:
```
_620.head(5)
```

[4]:
```
      Unnamed: 0        Date  Order_Demand    year  month   day  prod_code  \
1858        1858  2012-02-01             1  2012.0    2.0   1.0        620
1861        1861  2012-01-20             3  2012.0    1.0  20.0        620
1864        1864  2012-01-26             2  2012.0    1.0  26.0        620
1865        1865  2012-02-01             4  2012.0    2.0   1.0        620
1871        1871  2012-01-20             2  2012.0    1.0  20.0        620

     warehouse  category
1858         J         1
1861         J         1
1864         J         1
1865         J         1
1871         J         1
```

[5]:
```
_620['category'].value_counts()
```

[5]:
```
1    9428
Name: category, dtype: int64
```

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

```
[6]: _620 = _620.drop(columns =['Unnamed: 0', 'warehouse', 'category', 'prod_code'])
```

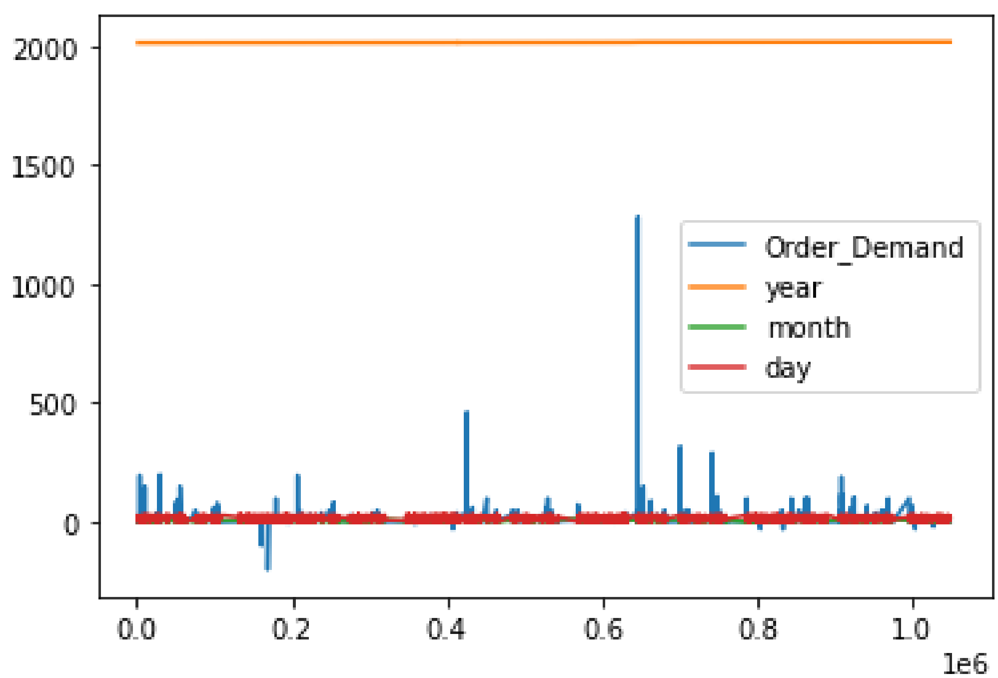
```
[7]: _620.set_index('Date')
```

```
[7]:             Order_Demand    year  month   day
     Date
     2012-02-01             1  2012.0    2.0   1.0
     2012-01-20             3  2012.0    1.0  20.0
     2012-01-26             2  2012.0    1.0  26.0
     2012-02-01             4  2012.0    2.0   1.0
     2012-01-20             2  2012.0    1.0  20.0
     …                    …      …     …    …
     2016-10-17             1  2016.0   10.0  17.0
     2016-12-06             1  2016.0   12.0   6.0
     2016-12-08             1  2016.0   12.0   8.0
     2016-04-12             1  2016.0    4.0  12.0
     2016-10-17             2  2016.0   10.0  17.0

     [9428 rows x 4 columns]
```

```
[8]: _620.plot()
```

```
[8]: <matplotlib.axes._subplots.AxesSubplot at 0x1249dabe0>
```

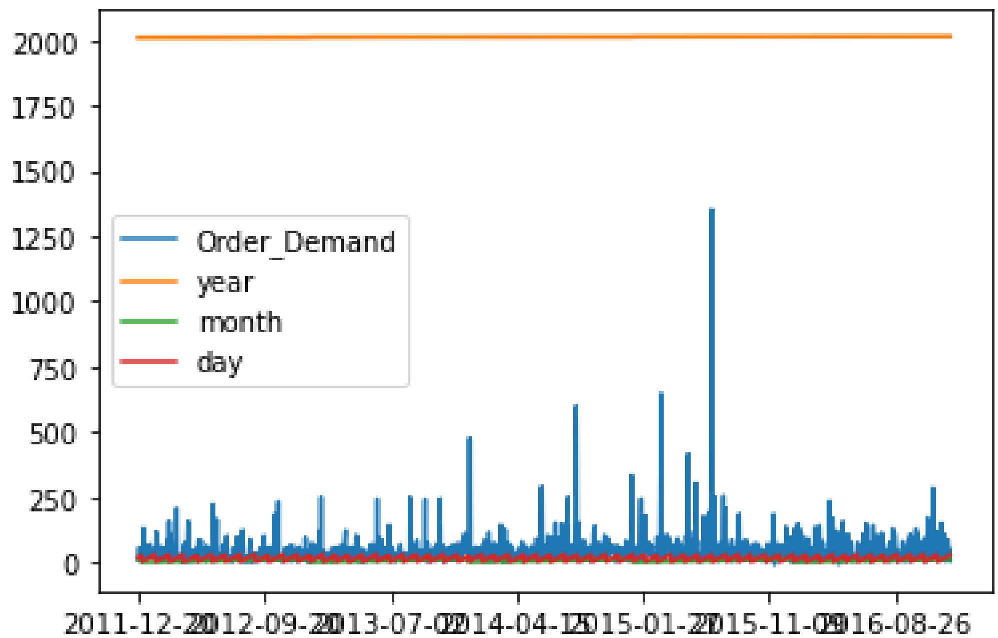


There are some negative values. I realized there could be multiple orders on a specific day. So, I needed to take the sum of all the orders on a particular day.

[9]:
```
aggregation_functions = {'Order_Demand':'sum', 'year':'first', 'month':'first',␣
 ↪'day':'first'}
prod620 = _620.groupby(_620['Date']).aggregate(aggregation_functions)
prod620.index.name = None
withdates_620 = prod620
```

[10]:
```
prod620.plot()
```

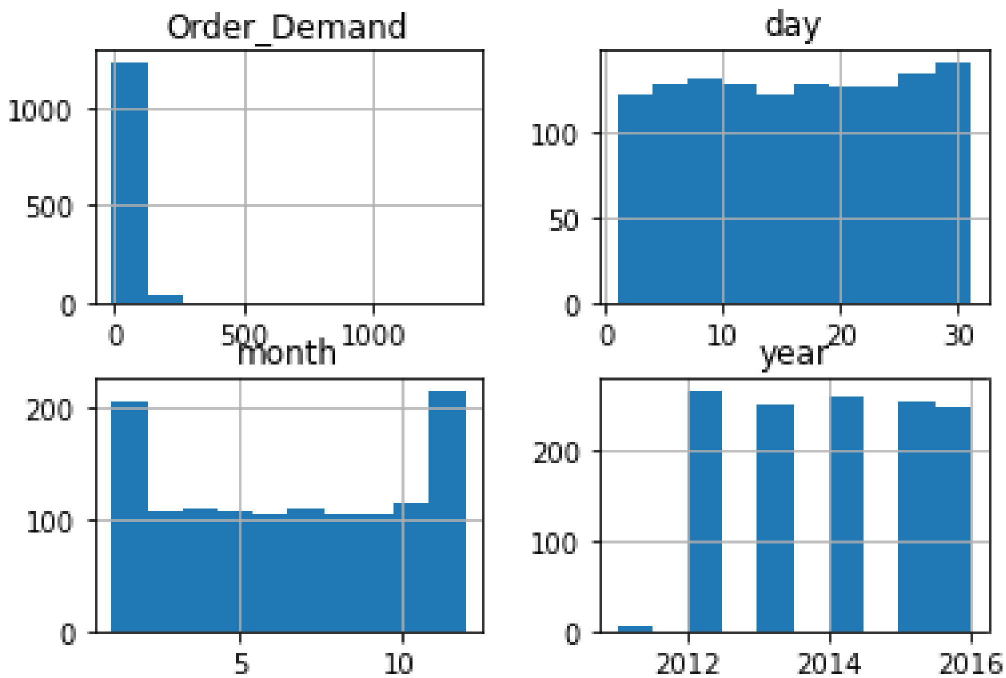
[10]:
```
<matplotlib.axes._subplots.AxesSubplot at 0x125ca50a0>
```



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.

[11]:
```
prod620.hist()
```

[11]:
```
array([[<matplotlib.axes._subplots.AxesSubplot object at 0x1232912e0>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1232bea60>],
       [<matplotlib.axes._subplots.AxesSubplot object at 0x123231f10>,
        <matplotlib.axes._subplots.AxesSubplot object at 0x1233803d0>]],
      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]:
```
prod620 = prod620.drop(columns =['day', 'month', 'year'])
```

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(prod620, 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]:
```python
model_fit = model.fit(disp=0)
```

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

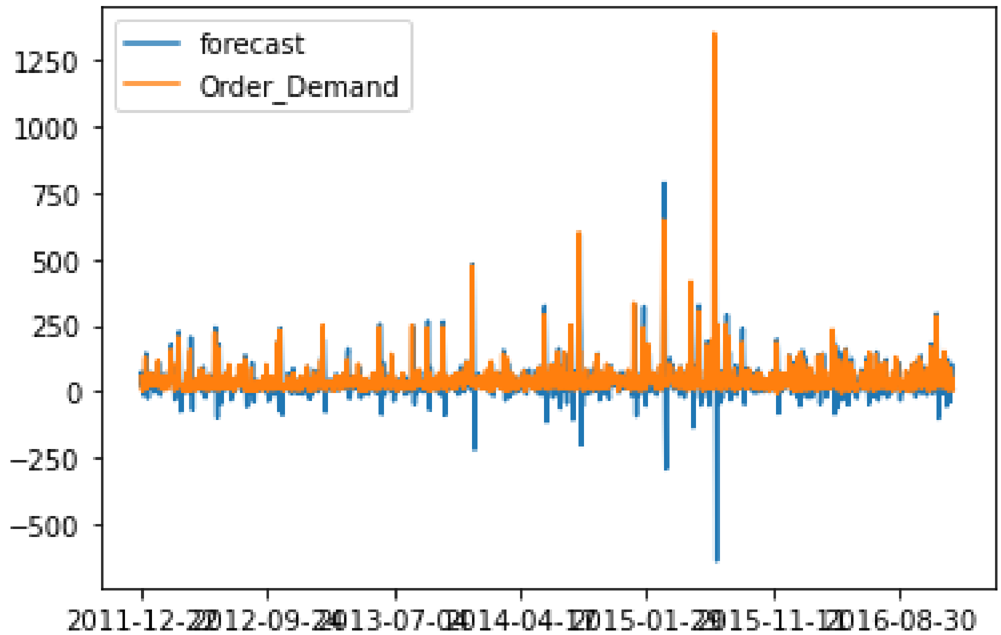
```
                             ARIMA Model Results
==============================================================================
Dep. Variable:        D2.Order_Demand   No. Observations:                 1286
Model:                 ARIMA(2, 2, 0)   Log Likelihood               -7683.597
Method:                       css-mle   S.D. of innovations             95.144
Date:                Mon, 12 Oct 2020   AIC                          15375.195
Time:                        10:28:51   BIC                          15395.832
Sample:                             2   HQIC                         15382.942

================================================================================
=========
                            coef    std err          z      P>|z|      [0.025
0.975]
--------------------------------------------------------------------------------
---------
const                    -0.0180      1.052     -0.017      0.986      -2.080
2.044
ar.L1.D2.Order_Demand    -0.9917      0.024    -42.049      0.000      -1.038
-0.945
ar.L2.D2.Order_Demand    -0.5321      0.024    -22.571      0.000      -0.578
-0.486
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1           -0.9319           -1.0055j            1.3709           -0.3690
AR.2           -0.9319           +1.0055j            1.3709            0.3690
-----------------------------------------------------------------------------
```

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()
```



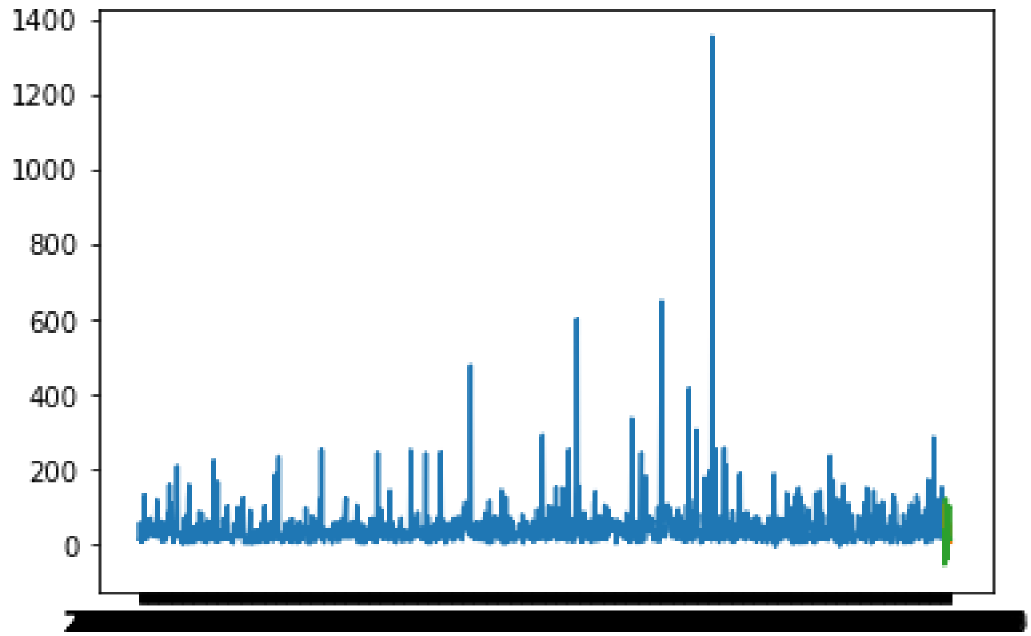
Now we forecast.

[18]:
```
#X = x_620.values
X = prod620
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 0x127659dc0>]



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

[20]: <matplotlib.legend.Legend at 0x127e5a040>

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.

3 Trying Again

This time, used 1 as the difference order.

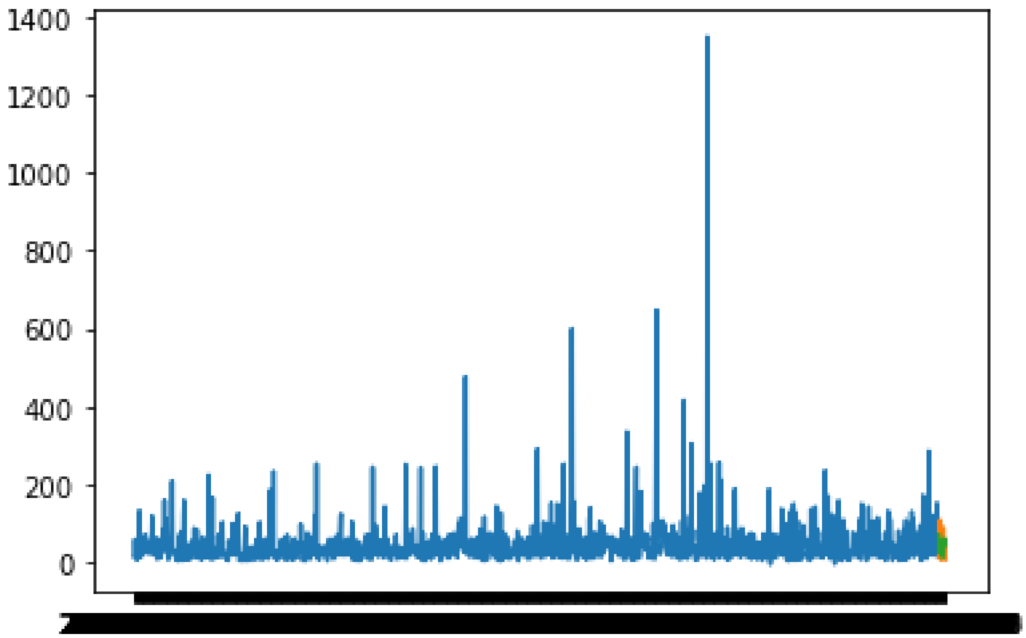
[21]:
```
model2 = ARIMA(prod620, order=(2,1,0))
model_fit = model2.fit(disp=0)

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')
```

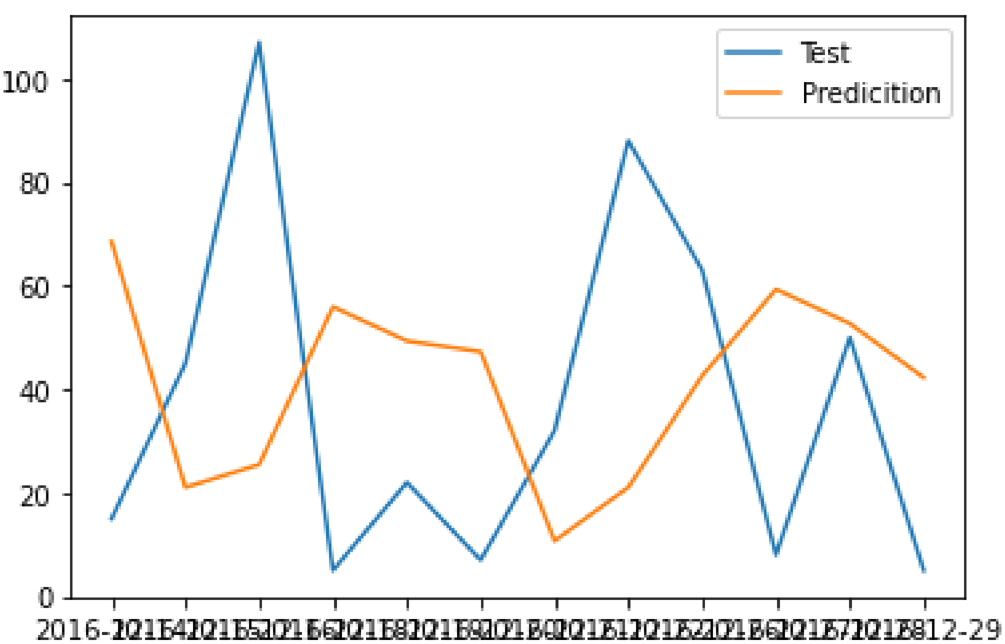
```
/Users/annarosefritz/opt/anaconda3/lib/python3.8/site-
packages/statsmodels/tsa/base/tsa_model.py:216: ValueWarning: A date index has
been provided, but it has no associated frequency information and so will be
ignored when e.g. forecasting.
  warnings.warn('A date index has been provided, but it 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 0x12825e5e0>]
```



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

[22]: <matplotlib.legend.Legend at 0x128e11e20>



The model still doesn't fit great. Next time I would try by changing the lag value or use a model that incorperates seasonality. I could also use differencing to better fit the data.

4 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_620 = withdates_620.drop(columns=['Order_Demand'])
X = x_620.values
size = int(len(X) * 0.66)
train, test = X[0:size], X[size:len(X)]
```

[25]:
```
y = withdates_620.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: -0.027614685439295572
```

Terrible coefficient of determination.

4.1 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)
```

```
[64.02393193]
```

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

```
[79.44941998]
```