```
import pandas as pd
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
from statsmodels.tsa.seasonal import seasonal_decompose
```

```
!pip install pmdarima
```

```
Requirement already satisfied: pmdarima in /usr/local/lib/python3.7/dist-packages (1.8.
Requirement already satisfied: numpy>=1.19.3 in /usr/local/lib/python3.7/dist-packages
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in /usr/local/lib/python3.7/
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: statsmodels!=0.12.0,>=0.11 in /usr/local/lib/python3.7/d
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: Cython!=0.29.18,>=0.29 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.7/dist-pack
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: python-dateutil>=2.7.3 in /usr/local/lib/python3.7/dist-
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.7/dist-packages (from
Requirement already satisfied: threadpoolctl>=2.0.0 in /usr/local/lib/python3.7/dist-pa
Requirement already satisfied: patsy>=0.5.2 in /usr/local/lib/python3.7/dist-packages (
Requirement already satisfied: packaging>=21.3 in /usr/local/lib/python3.7/dist-package
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in /usr/local/lib/python3.7/dis
```

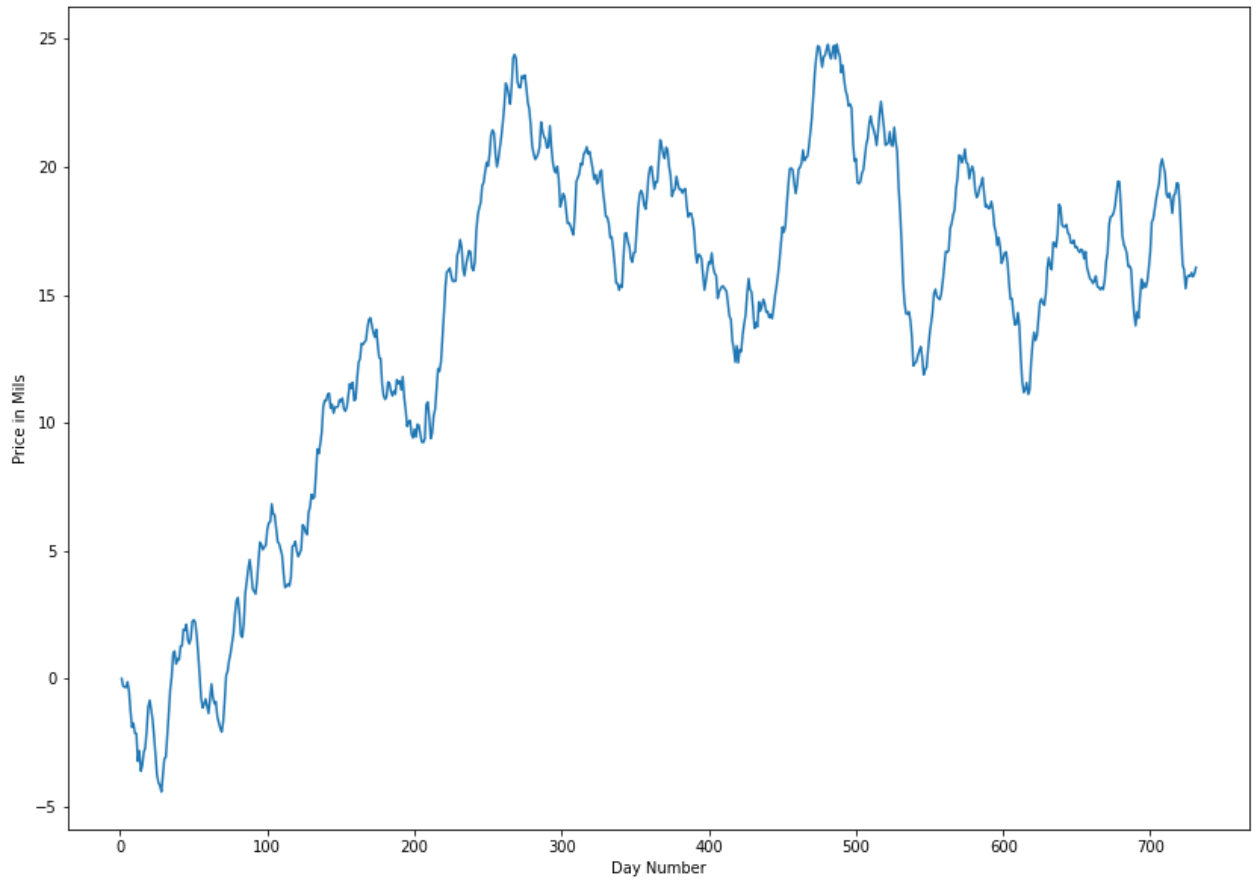
```
df = pd.read_csv('medical_time_series.csv')
print(df.head())
```

```
   Day   Revenue
0    1  0.000000
1    2 -0.292356
2    3 -0.327772
3    4 -0.339987
4    5 -0.124888
```

```
print(df.isnull().sum())
```

```
Day        0
Revenue    0
dtype: int64
```

```
plt.figure(figsize=(14,10))
plt.plot(df.Day, df.Revenue)
plt.ylabel('Price in Mils')
plt.xlabel('Day Number')
plt.show()
```




```
df.info
```

```
<bound method DataFrame.info of      Day    Revenue
0      1   0.000000
1      2  -0.292356
2      3  -0.327772
3      4  -0.339987
4      5  -0.124888
..   ...        ...
726  727  15.722056
727  728  15.865822
728  729  15.708988
729  730  15.822867
730  731  16.069429

[731 rows x 2 columns]>
```

```
print(df.duplicated().sum())
```

```
0
```

```
print(df.describe())
```

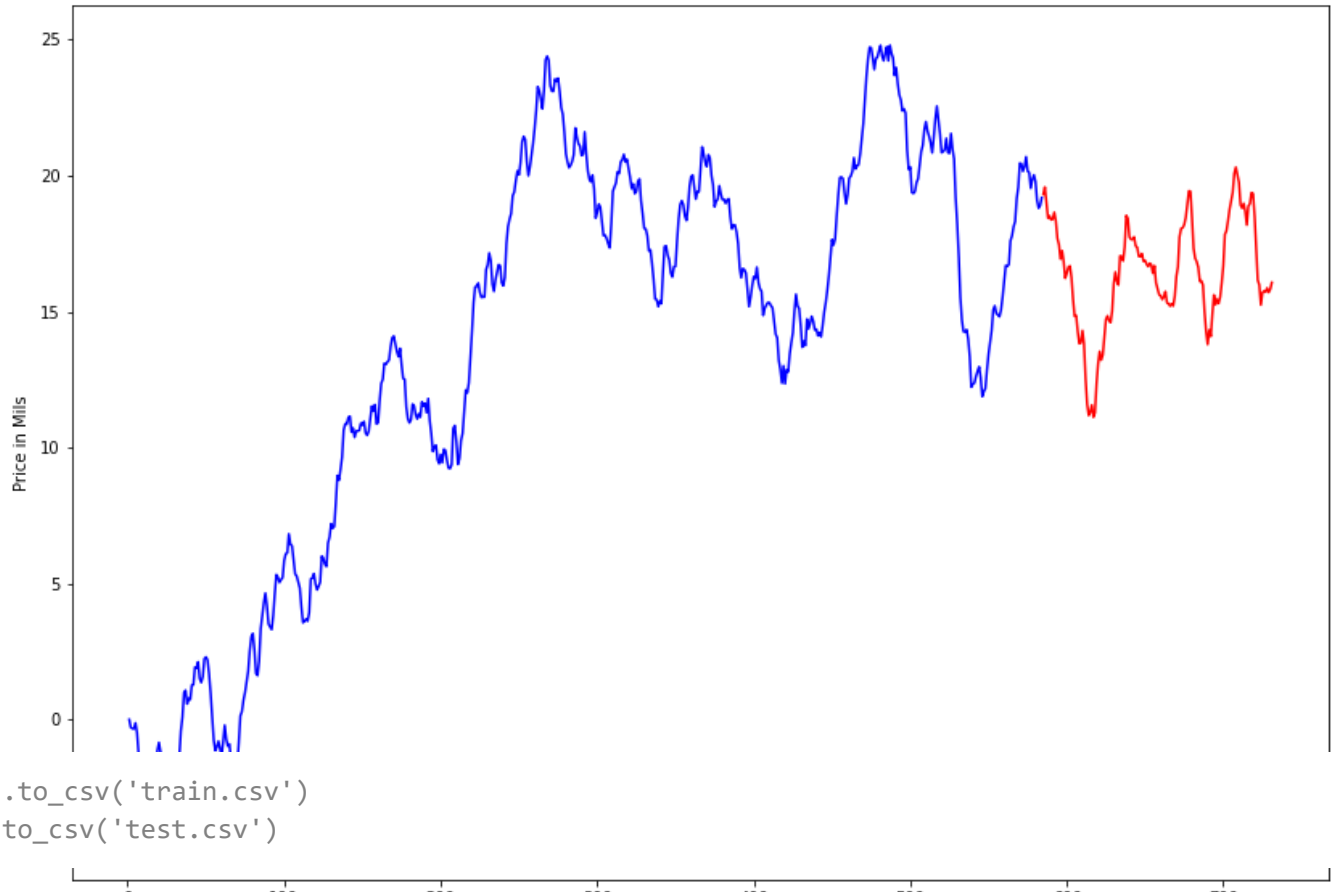
```
                Day     Revenue
count  731.000000  731.000000
mean   366.000000   14.179608
std    211.165812    6.959905
min      1.000000   -4.423299
25%    183.500000   11.121742
50%    366.000000   15.951830
75%    548.500000   19.293506
max    731.000000   24.792249
```

```
from statsmodels.tsa.stattools import adfuller
df = pd.read_csv('medical_time_series.csv')
new_result = adfuller(df.Revenue)
print('ADF Stat: %f' % new_result[0])
print('p-value: %f' % new_result[1])
print('Critical Values:')
for key, value in new_result[4].items():
  print('\t%s: %.3f' % (key, value))
```

```
ADF Stat: -2.218319
p-value: 0.199664
Critical Values:
        1%: -3.439
        5%: -2.866
        10%: -2.569
```

```
from sklearn.model_selection import train_test_split
train, test = train_test_split(df, test_size=0.2, random_state=42, shuffle=False)
```

```
plt.figure(figsize=(14,10))
plt.plot(train.Day, train.Revenue, color='blue')
plt.plot(test.Day, test.Revenue, color='red')
plt.ylabel('Price in Mils')
plt.xlabel('Day Number')
plt.show()
```




```
train.to_csv('train.csv')
test.to_csv('test.csv')
```

```
print(train.columns)
print(test.columns)
```

```
Index(['Day', 'Revenue'], dtype='object')
Index(['Day', 'Revenue'], dtype='object')
```

```
print(df)
```

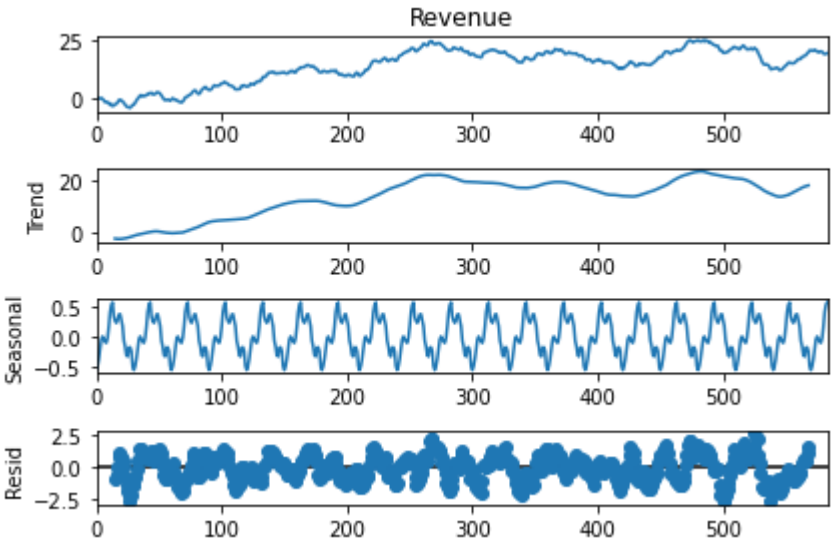
```
     Day    Revenue
0      1   0.000000
1      2  -0.292356
2      3  -0.327772
3      4  -0.339987
4      5  -0.124888
..   ...        ...
726  727  15.722056
727  728  15.865822
728  729  15.708988
729  730  15.822867
730  731  16.069429

[731 rows x 2 columns]
```

```
print(df.columns)
#train_copy = train[['Day', 'Revenue']].copy()
#test_copy = test[['Day', 'Revenue']].copy()
df.to_numpy()
```

```
print(train.head)
print(test.head)
```

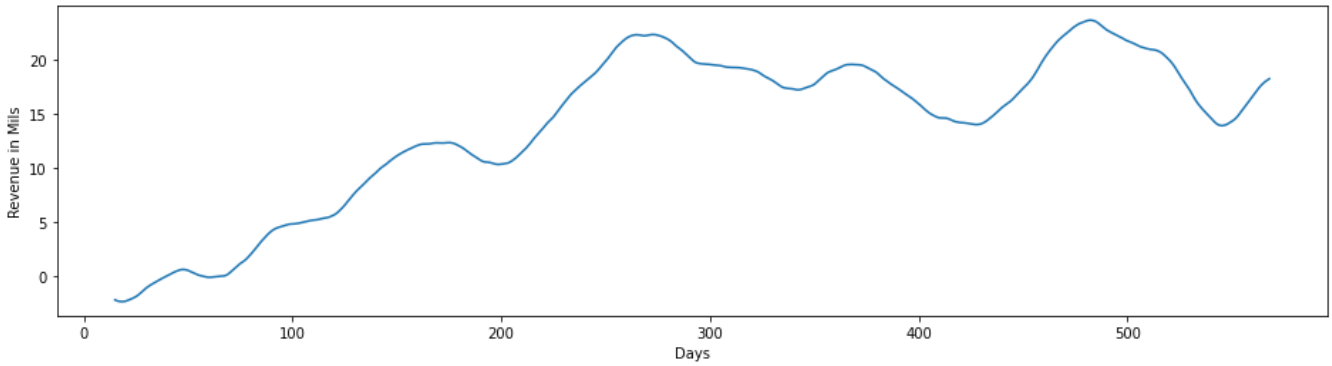
```
Index(['Day', 'Revenue'], dtype='object')
<bound method NDFrame.head of      Day    Revenue
0      1   0.000000
1      2  -0.292356
2      3  -0.327772
3      4  -0.339987
4      5  -0.124888
..   ...        ...
579  580  19.782635
580  581  19.088265
581  582  18.805501
582  583  18.910233
583  584  19.186089

[584 rows x 2 columns]>
<bound method NDFrame.head of      Day    Revenue
584  585  19.312734
585  586  19.576725
586  587  18.988035
587  588  18.437608
588  589  18.519085
..   ...        ...
726  727  15.722056
727  728  15.865822
728  729  15.708988
729  730  15.822867
730  731  16.069429

[147 rows x 2 columns]>
```

```
decomp = seasonal_decompose(train['Revenue'], model='additive', period=30) # freq or period.
decomp.plot()
plt.show()
```

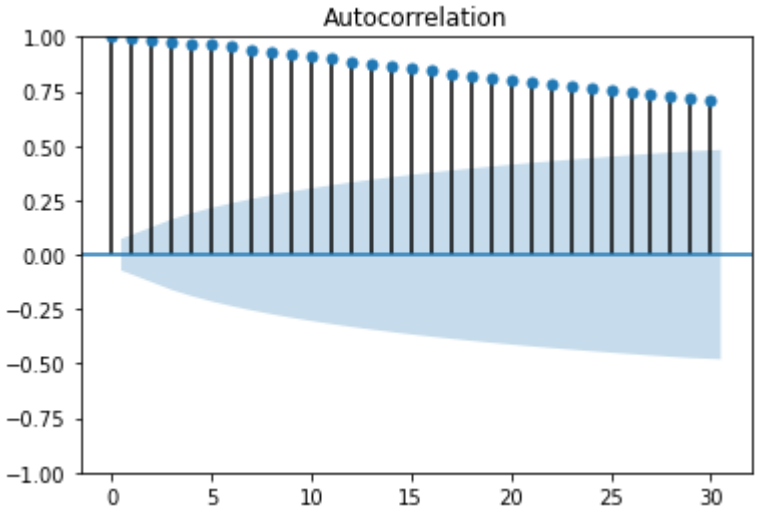


```
decomp_trend = decomp.trend
decomp_trend.plot(figsize=(16,4))
plt.xlabel('Days')
plt.ylabel('Revenue in Mils')
```

```
Text(0, 0.5, 'Revenue in Mils')
```

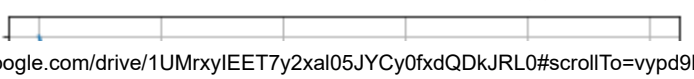


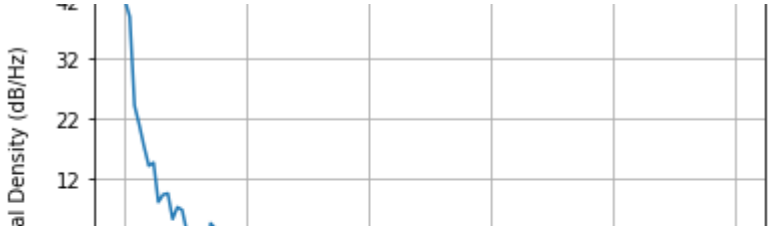

```
from statsmodels.graphics import tsaplots

fig = tsaplots.plot_acf(df['Revenue'], lags = 30)
plt.show()
```




```
plt.psd(df['Revenue'])
```

```
(array([1.52612936e+04, 8.12118786e+03, 2.62760550e+02, 1.24454986e+02,
        5.40026379e+01, 2.55196420e+01, 2.89237553e+01, 6.29449800e+00,
        8.38967529e+00, 8.79735213e+00, 3.26244345e+00, 5.16464998e+00,
        4.64370619e+00, 1.96867736e+00, 1.48167320e+00, 2.11256649e+00,
        1.08550271e+00, 5.81988645e-01, 2.76744723e+00, 2.22308983e+00,
        5.50876230e-01, 7.49752606e-01, 1.26157210e+00, 8.77859381e-01,
        4.84404330e-01, 4.39565525e-01, 2.05055025e-01, 9.50457901e-01,
        5.23499207e-01, 4.35456489e-01, 1.69884060e+00, 8.97977002e-01,
        7.28275718e-01, 6.11527179e-01, 1.49053313e+00, 5.44383887e-01,
        2.17350902e-01, 5.07814496e-01, 4.43820257e-01, 2.94184611e-01,
        2.55373626e-01, 9.44702588e-02, 1.58619217e-01, 1.88829134e-01,
        2.11804293e-01, 6.04121179e-01, 4.95073852e-01, 2.15698285e-01,
        1.04432711e-01, 1.46393379e-01, 2.07457981e-01, 1.26602668e-01,
        1.74592665e-01, 1.42674366e-01, 9.63196690e-02, 1.17768273e-01,
        1.12081971e-02, 7.05631210e-02, 8.82393791e-02, 1.79463366e-02,
        4.63319518e-02, 8.22217635e-02, 8.62002792e-02, 8.11471442e-02,
        9.76265725e-02, 7.64686033e-02, 4.69018363e-02, 2.27885092e-02,
        3.22945840e-02, 4.15665768e-02, 3.87010238e-02, 1.37096984e-02,
        2.54563144e-02, 5.96461061e-02, 8.54027434e-02, 5.14038459e-02,
        3.17470361e-02, 2.66619122e-02, 5.20528881e-02, 3.74783288e-03,
        4.60111740e-02, 4.72280258e-02, 2.64401277e-02, 2.65654525e-02,
        2.74537510e-02, 1.30164715e-02, 2.52371618e-02, 5.09056679e-02,
        2.36007364e-02, 3.60978502e-02, 1.34740313e-02, 2.47166596e-02,
        5.63282623e-02, 7.25977232e-02, 2.73963882e-02, 3.40075699e-02,
        7.28076058e-03, 2.37376075e-02, 3.35477194e-02, 1.11675245e-02,
        3.86130031e-02, 5.52632330e-02, 3.06776062e-02, 1.02314465e-02,
        1.00437589e-02, 5.88763445e-02, 6.55634547e-02, 6.48669973e-02,
        4.56754349e-02, 1.17566185e-02, 4.41995523e-03, 2.14450712e-02,
        3.82815436e-02, 3.32170073e-02, 3.01496591e-02, 1.53250228e-02,
        5.43350164e-03, 1.84455310e-02, 1.19157543e-02, 3.05548420e-02,
        8.40529002e-02, 3.87920060e-02, 1.07052164e-02, 4.84092102e-02,
        4.49317201e-02, 3.20797981e-02, 2.10764097e-02, 4.37206681e-02,
        1.63676609e-02]),
 array([0.       , 0.0078125, 0.015625 , 0.0234375, 0.03125  , 0.0390625,
        0.046875 , 0.0546875, 0.0625   , 0.0703125, 0.078125 , 0.0859375,
        0.09375  , 0.1015625, 0.109375 , 0.1171875, 0.125    , 0.1328125,
        0.140625 , 0.1484375, 0.15625  , 0.1640625, 0.171875 , 0.1796875,
        0.1875   , 0.1953125, 0.203125 , 0.2109375, 0.21875  , 0.2265625,
        0.234375 , 0.2421875, 0.25     , 0.2578125, 0.265625 , 0.2734375,
        0.28125  , 0.2890625, 0.296875 , 0.3046875, 0.3125   , 0.3203125,
        0.328125 , 0.3359375, 0.34375  , 0.3515625, 0.359375 , 0.3671875,
        0.375    , 0.3828125, 0.390625 , 0.3984375, 0.40625  , 0.4140625,
        0.421875 , 0.4296875, 0.4375   , 0.4453125, 0.453125 , 0.4609375,
        0.46875  , 0.4765625, 0.484375 , 0.4921875, 0.5      , 0.5078125,
        0.515625 , 0.5234375, 0.53125  , 0.5390625, 0.546875 , 0.5546875,
        0.5625   , 0.5703125, 0.578125 , 0.5859375, 0.59375  , 0.6015625,
        0.609375 , 0.6171875, 0.625    , 0.6328125, 0.640625 , 0.6484375,
        0.65625  , 0.6640625, 0.671875 , 0.6796875, 0.6875   , 0.6953125,
        0.703125 , 0.7109375, 0.71875  , 0.7265625, 0.734375 , 0.7421875,
        0.75     , 0.7578125, 0.765625 , 0.7734375, 0.78125  , 0.7890625,
        0.796875 , 0.8046875, 0.8125   , 0.8203125, 0.828125 , 0.8359375,
        0.84375  , 0.8515625, 0.859375 , 0.8671875, 0.875    , 0.8828125,
        0.890625 , 0.8984375, 0.90625  , 0.9140625, 0.921875 , 0.9296875,
        0.9375   , 0.9453125, 0.953125 , 0.9609375, 0.96875  , 0.9765625,
        0.984375 , 0.9921875, 1.       ]))
```






```
from pmdarima import auto_arima
auto_arima_model = auto_arima(df['Revenue'], start_p=1, start_q=1,
                          max_p=3, max_q=3, m=12,
                          start_P=0, seasonal=True,
                          d=1, D=1, trace=True,
                          error_action='ignore',
                          suppress_warnings=True,
                          stepwise=True)
print(auto_arima_model.aic())
print(auto_arima_model.summary())
```

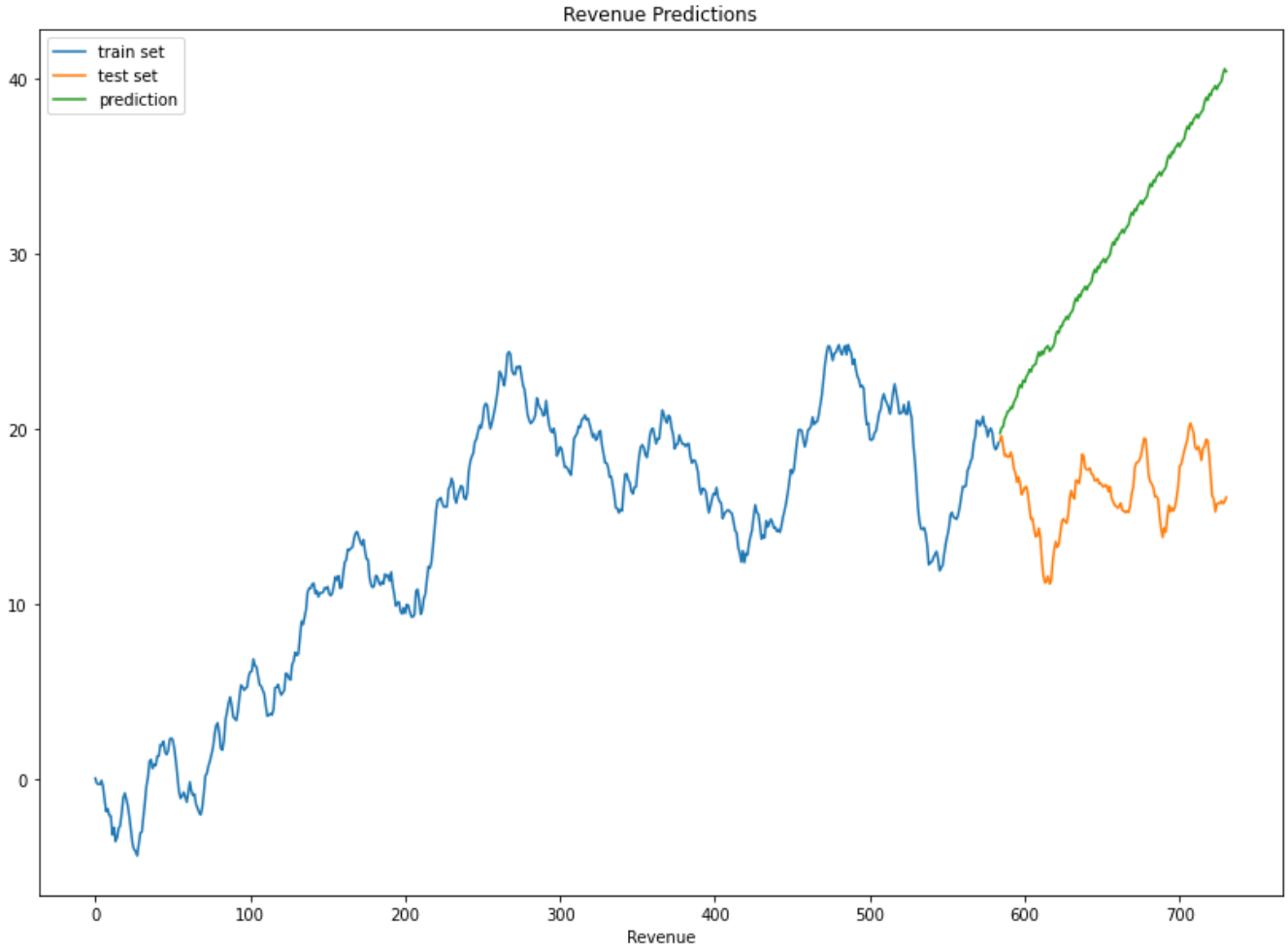
```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,1,1)[12]             : AIC=inf, Time=4.53 sec
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=1548.267, Time=0.08 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=1161.213, Time=0.54 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=7.12 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=1413.401, Time=0.14 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=1076.065, Time=1.72 sec
 ARIMA(1,1,0)(2,1,1)[12]             : AIC=inf, Time=19.52 sec
 ARIMA(1,1,0)(1,1,1)[12]             : AIC=inf, Time=9.33 sec
 ARIMA(0,1,0)(2,1,0)[12]             : AIC=1229.919, Time=1.09 sec
 ARIMA(2,1,0)(2,1,0)[12]             : AIC=1078.060, Time=2.29 sec
 ARIMA(1,1,1)(2,1,0)[12]             : AIC=1078.060, Time=2.33 sec
 ARIMA(0,1,1)(2,1,0)[12]             : AIC=1103.447, Time=0.97 sec
 ARIMA(2,1,1)(2,1,0)[12]             : AIC=1078.936, Time=4.18 sec
 ARIMA(1,1,0)(2,1,0)[12] intercept   : AIC=1078.064, Time=3.69 sec

Best model:  ARIMA(1,1,0)(2,1,0)[12]
Total fit time: 57.557 seconds
1076.0645264844293
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                   y   No. Observations:
Model:             SARIMAX(1, 1, 0)x(2, 1, 0, 12)   Log Likelihood                -534.
Date:                            Tue, 08 Mar 2022   AIC                           1076.
Time:                                    00:02:36   BIC                           1094.
Sample:                                         0   HQIC                          1083.
                                              - 731
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4420      0.034     12.987      0.000       0.375       0.509
ar.S.L12      -0.7433      0.038    -19.718      0.000      -0.817      -0.669
ar.S.L24      -0.3460      0.038     -9.082      0.000      -0.421      -0.271
sigma2         0.2564      0.015     17.415      0.000       0.228       0.285
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):                 4.02
```

```
Prob(Q):                              0.94   Prob(JB):                         0.13
Heteroskedasticity (H):               0.96   Skew:                             0.11
Prob(H) (two-sided):                  0.75   Kurtosis:                         2.70
===================================================================================

Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).
```

```
#arima forcast
# SARIMAX(1, 1, 0)x(2, 1, 0, 12)
import statsmodels.api as sm
model = sm.tsa.statespace.SARIMAX(train['Revenue'], order=(1,1,0), seasonal_order = (2,1,0,12
SARIMAXresults = model.fit()


#df['forcast'] = results.predict(start = 0, end = 730, dynamic = True)
#df['Revenue'].plot(figsize = 12,8)
result = SARIMAXresults.get_forecast()
forecast_test = test['Revenue'].values # using test set to see what predicted would be
forecast = result.predicted_mean
print('expected:', forecast)
print('forecast:', forecast_test[0])
print('std error:', result.se_mean)
```

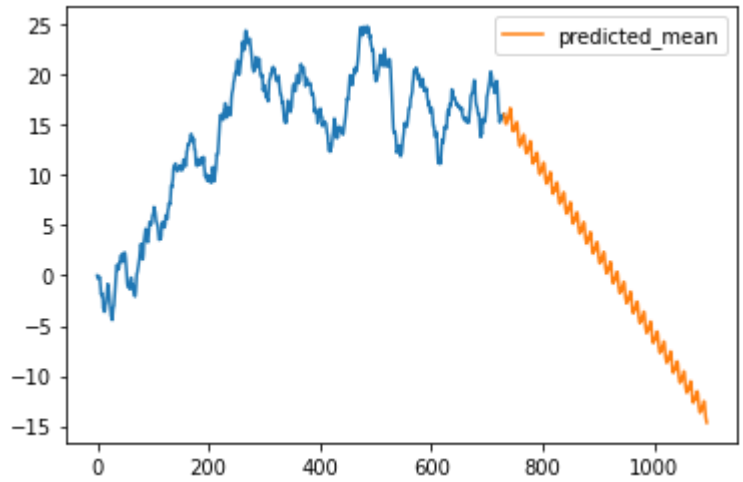
```
expected: 584    19.740296
dtype: float64
forecast: 19.31273398
std error: 584    0.511004
dtype: float64
```

```
start = len(train.Revenue)
end = len(train.Revenue) + len(test.Revenue) - 1

plt.figure(figsize=(14,10))
predict = SARIMAXresults.predict(start, end, typ='levels')
plt.plot(train.Revenue, label = 'train set')
plt.plot(test.Revenue, label = 'test set')
plt.plot(predict, label = 'prediction')
plt.title('Revenue Predictions')
plt.xlabel('Day')
plt.xlabel('Revenue')
plt.legend(loc='upper left')
plt.show()
```




```
model = sm.tsa.statespace.SARIMAX(df['Revenue'], order=(1,1,0), seasonal_order=(2,1,0,12))
results = model.fit()
forecast = results.predict(start = len(df['Revenue']), end = (len(df['Revenue']) - 1) + 365)

df['Revenue'].plot()
forecast.plot(legend = True)
```

```
<matplotlib.axes._subplots.AxesSubplot at 0x7f124d8cba50>
```




```
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse
```

```
MSE = mean_squared_error(test['Revenue'], predict)
RMSE = rmse(test['Revenue'], predict)
print('MSE: ', MSE)
print('RMSE: ', RMSE)
```

```
MSE:  228.1822083849405
RMSE:  15.105701188125645
```

```
interval = [0.25, 0.1, 0.05, 0.01]
for i in interval:
  conf = result.conf_int(alpha=i)
  print("confidence intervals: ", ((1-i)*100), conf)
```

```
confidence intervals:  75.0      lower Revenue  upper Revenue
584      19.152463       20.32813
confidence intervals:  90.0      lower Revenue  upper Revenue
584      18.899769      20.580824
confidence intervals:  95.0      lower Revenue  upper Revenue
584      18.738746      20.741847
confidence intervals:  99.0      lower Revenue  upper Revenue
584      18.424036      21.056557
```

✓ 0s completed at 7:02 PM