Time Series corrected

June 26, 2023

[1]:
```
import pandas as pd
import numpy as np
from statsmodels.tsa.arima.model import ARIMA
import seaborn as sns
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import acf, adfuller, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pmdarima import auto_arima
import warnings
warnings.filterwarnings("ignore")
from statsmodels.tsa.seasonal import seasonal_decompose
```

[2]:
```
med=pd.read_csv('C:/Users/dscha/Downloads/D213/medical_time_series.csv',␣
 ↪index_col=0, parse_dates=False)
```

[3]:
```
med.shape
```

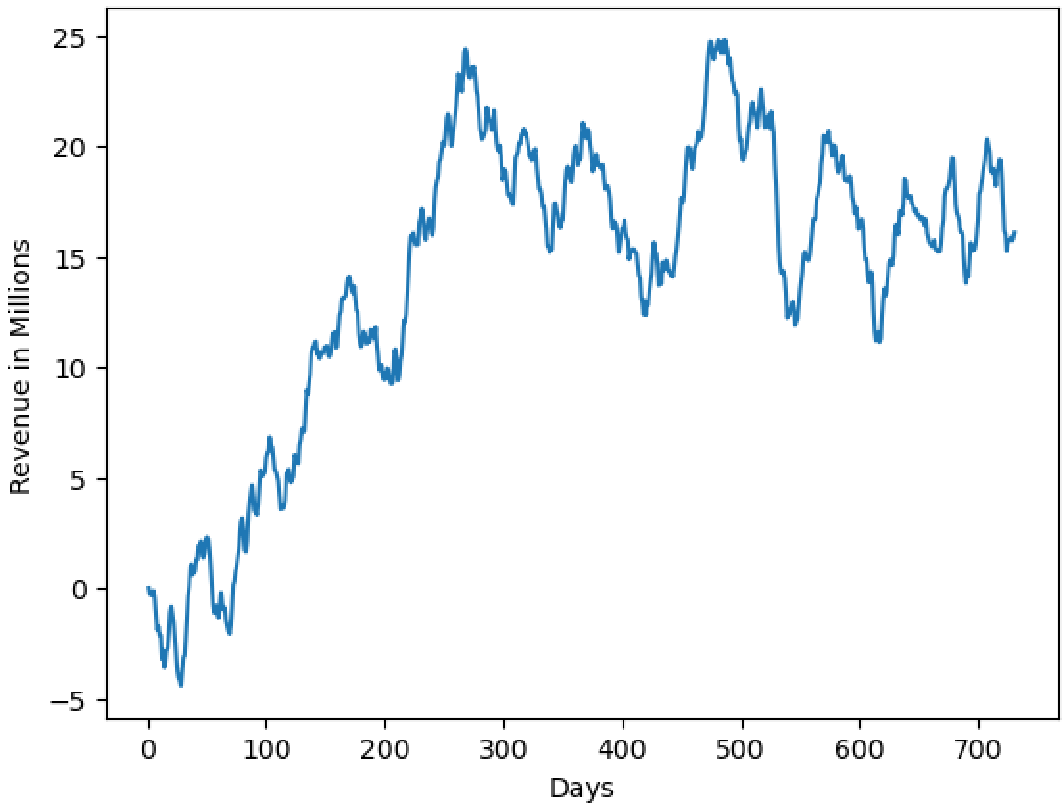
[3]:
```
(731, 1)
```

[4]:
```
med.isnull().any()
```

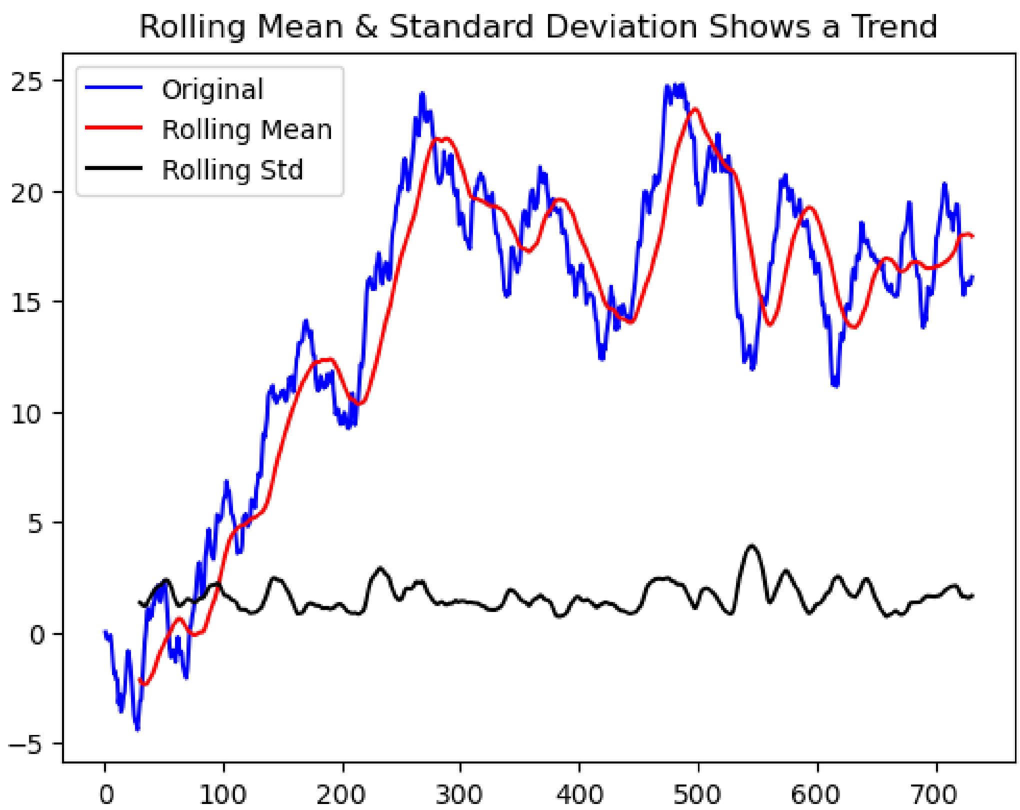
[4]:
```
Revenue    False
dtype: bool
```

[5]:
```
plt.xlabel('Days')
plt.ylabel('Revenue in Millions')
plt.plot(med)
```

[5]:
```
[<matplotlib.lines.Line2D at 0x278e34bacb0>]
```

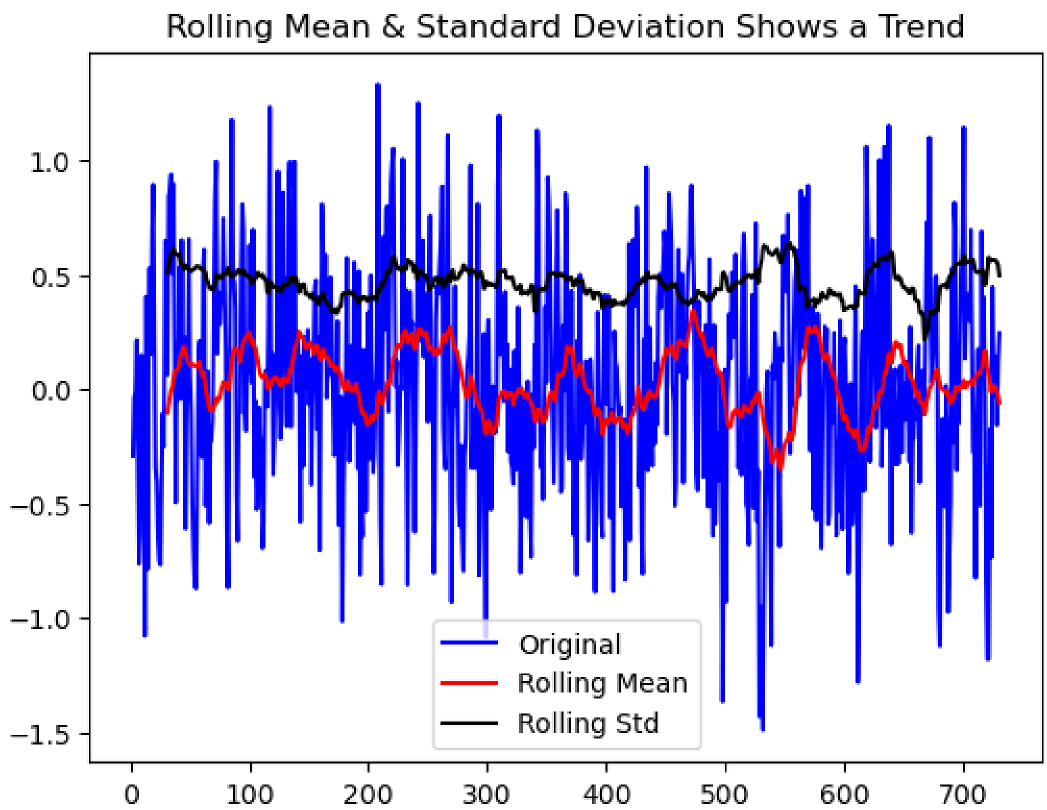



```
[6]: def test_stationarity(timeseries):
    movingAverage=timeseries.rolling(window=30).mean()
    movingSTD=timeseries.rolling(window=30).std()
    orig=plt.plot(timeseries, color='blue', label='Original')
    mean=plt.plot(movingAverage, color='red', label='Rolling Mean')
    std=plt.plot(movingSTD, color='black', label='Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean & Standard Deviation Shows a Trend')
    plt.show(block=False)
    print ('Results of Dickey-Fuller test: ')
    dftest=adfuller(timeseries['Revenue'], autolag='AIC')
    dfoutput=pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags␣
 ↪Used','No. of Observations'])
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s) '%key] = value # Critical Values should␣
 ↪always be more than the test statistic
    print(dfoutput)
test_stationarity(med)
```




```
Results of Dickey-Fuller test:
Test Statistic                 -2.218319
p-value                         0.199664
#Lags Used                      1.000000
No. of Observations           729.000000
Critical Value (1%)            -3.439352
Critical Value (5%)            -2.865513
Critical Value (10%)           -2.568886
dtype: float64
```

[7]:
```
med_shift=med-med.shift()
med_shift.dropna(inplace=True)
test_stationarity(med_shift)
```

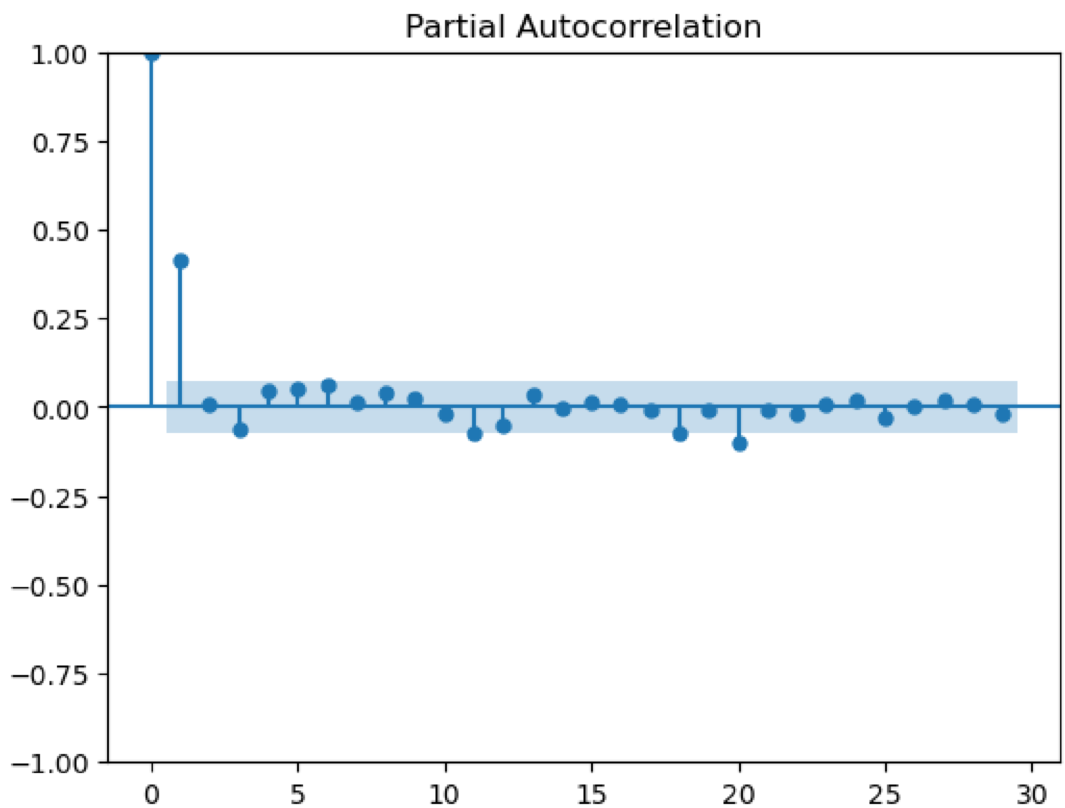


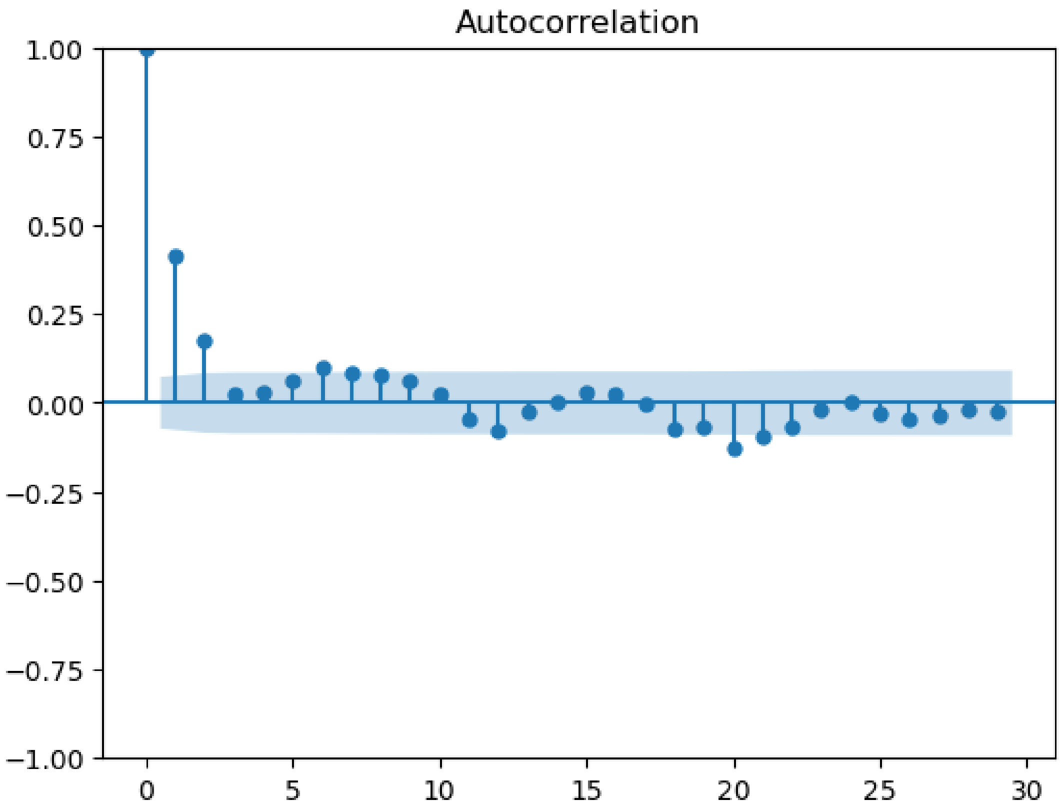

```
Results of Dickey-Fuller test:
Test Statistic                -1.737477e+01
p-value                        5.113207e-30
#Lags Used                     0.000000e+00
No. of Observations            7.290000e+02
Critical Value (1%)           -3.439352e+00
Critical Value (5%)           -2.865513e+00
Critical Value (10%)          -2.568886e+00
dtype: float64
```

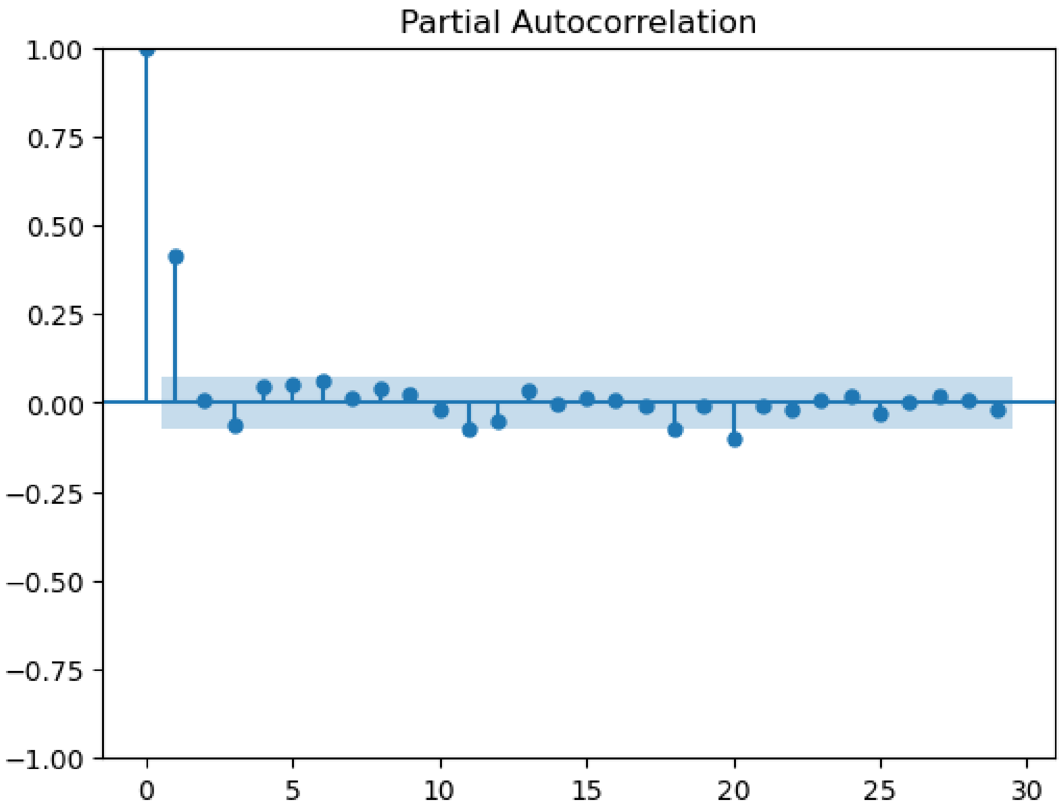
[8]:

```
plot_acf(med_shift)
plot_pacf(med_shift)
```

[8]:

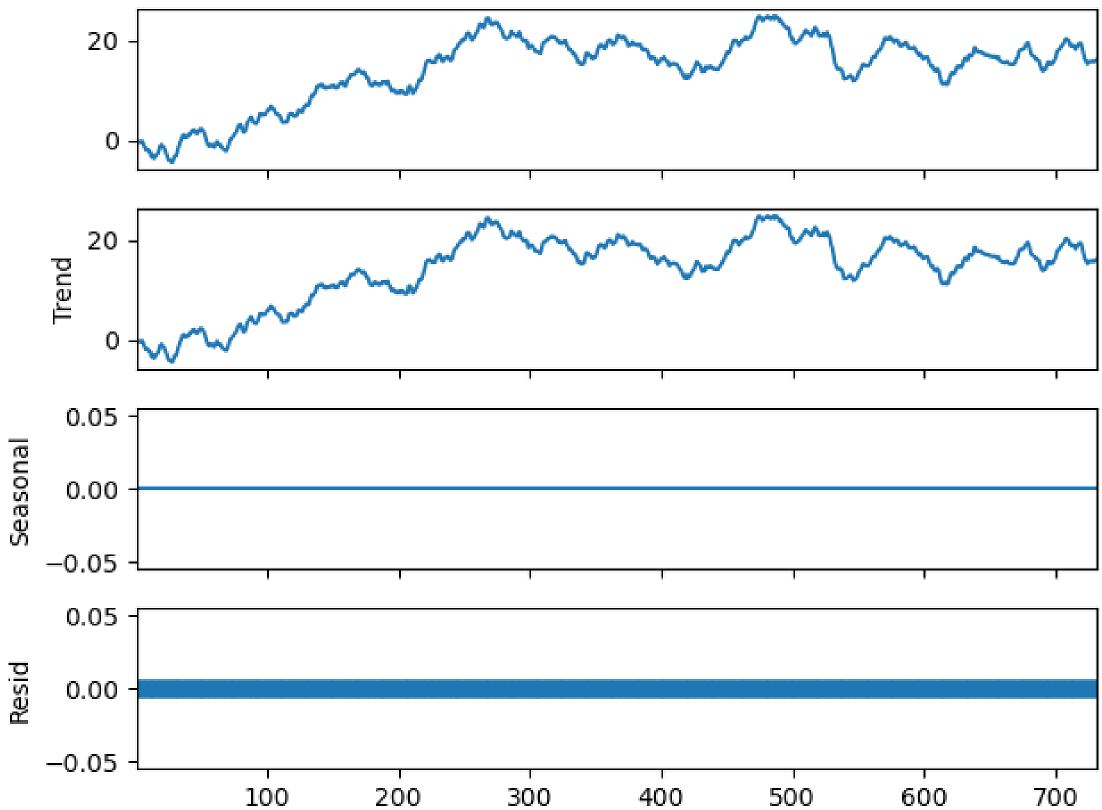

Partial Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30


Autocorrelation
1.00
0.75
0.50
0.25
0.00
−0.25
−0.50
−0.75
−1.00
0
5
10
15
20
25
30

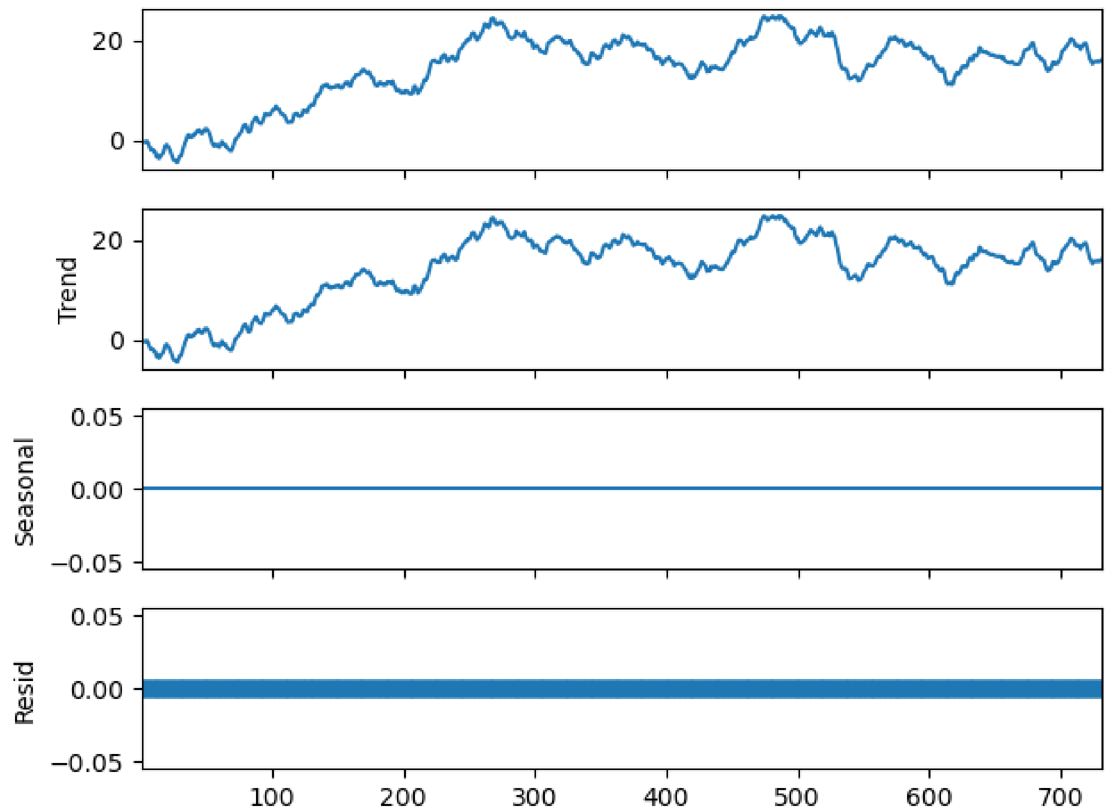


[9]:
```
decomp=seasonal_decompose(med, model='additive', period=1)
decomp.plot()
```

[9]:



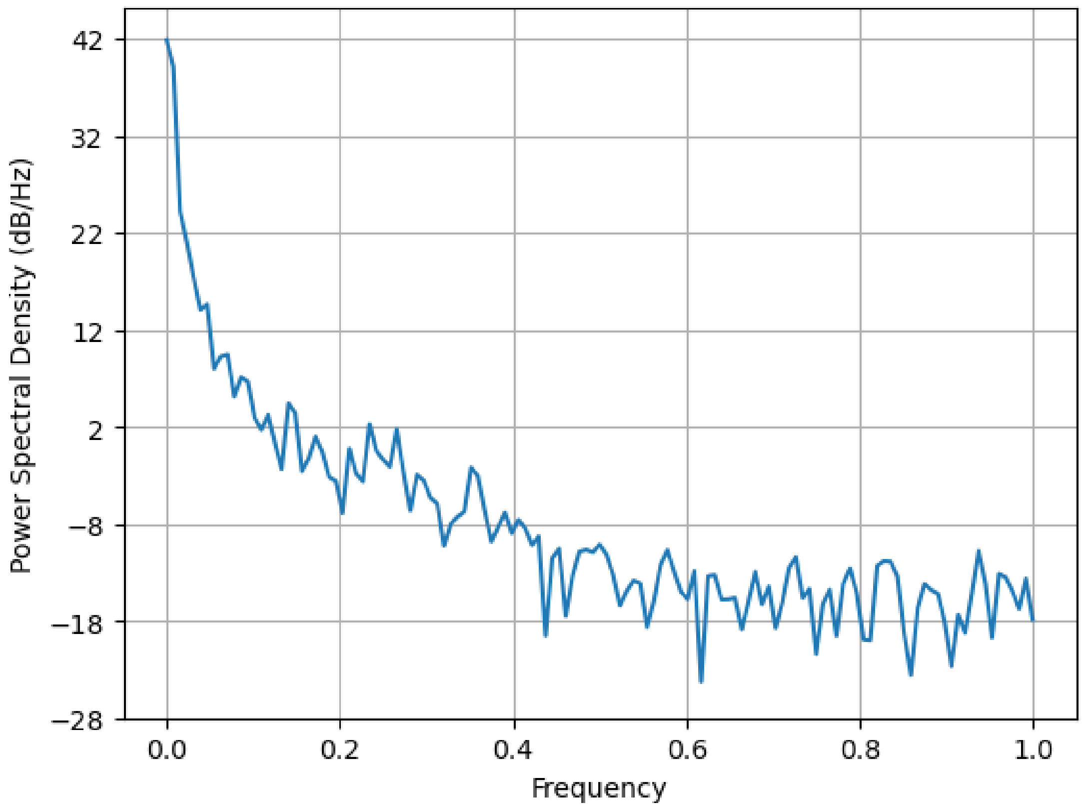
20
0
Trend
20
0
Seasonal
0.05
0.00
−0.05
Resid
0.05
0.00
−0.05
100
200
300
400
500
600
700



[10]:
```
plt.psd(med['Revenue'])
```

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



[11]:

```
stepwise_fit=auto_arima(med['Revenue'], trace=True, suppress_warnings=True)
stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=883.277, Time=0.47 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1015.972, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=881.359, Time=0.08 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=906.199, Time=0.09 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1015.481, Time=0.03 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=883.300, Time=0.08 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=883.314, Time=0.11 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=883.348, Time=0.33 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=879.982, Time=0.04 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=881.911, Time=0.06 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=881.927, Time=0.08 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=905.166, Time=0.04 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=881.947, Time=0.14 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 1.640 seconds
```

[11]:

Dep. Variable:	y	**No. Observations:**	731
Model:	SARIMAX(1, 1, 0)	**Log Likelihood**	-437.991
Date:	Mon, 26 Jun 2023	**AIC**	879.982
Time:	09:36:11	**BIC**	889.168
Sample:	0 - 731	**HQIC**	883.526
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
ar.L1	0.4142	0.034	12.258	0.000	0.348	0.480
sigma2	0.1943	0.011	17.842	0.000	0.173	0.216

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	1.92
Prob(Q):	0.90	**Prob(JB):**	0.38
Heteroskedasticity (H):	1.00	**Skew:**	-0.02
Prob(H) (two-sided):	0.97	**Kurtosis:**	2.75

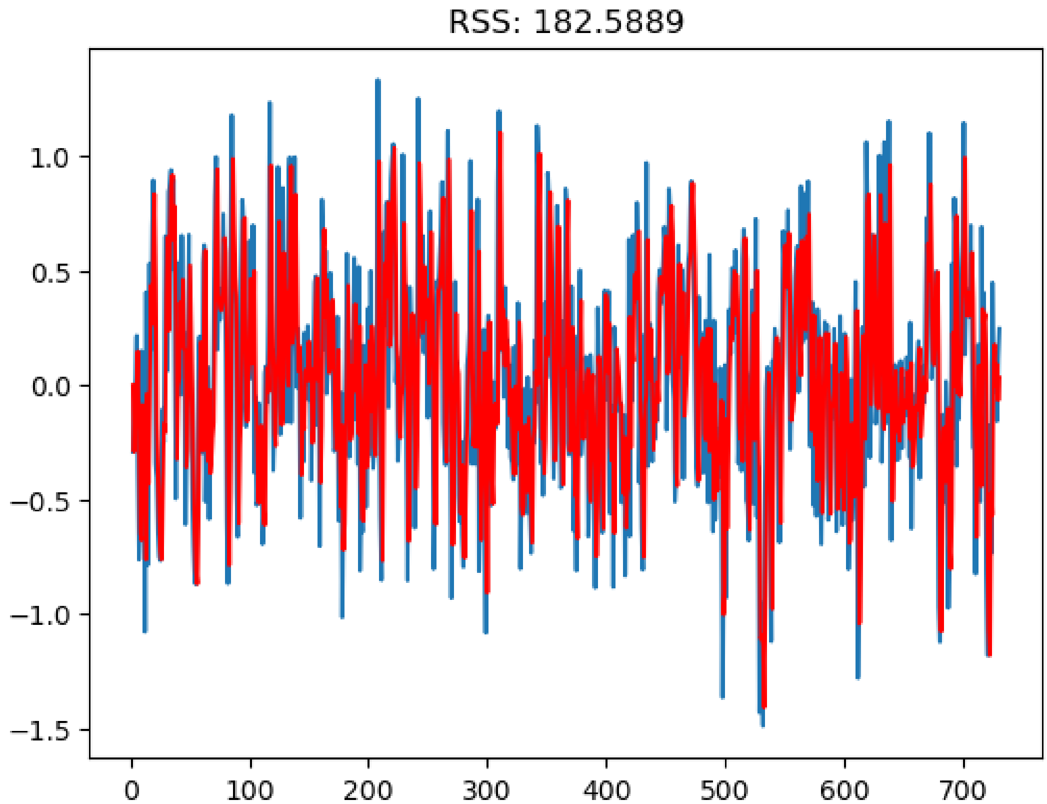
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

[12]:
```
print(med.shape)
train=med.iloc[:-30]
test=med.iloc[-30:]
print(train.shape, test.shape)
```

```
(731, 1)
(701, 1) (30, 1)
```

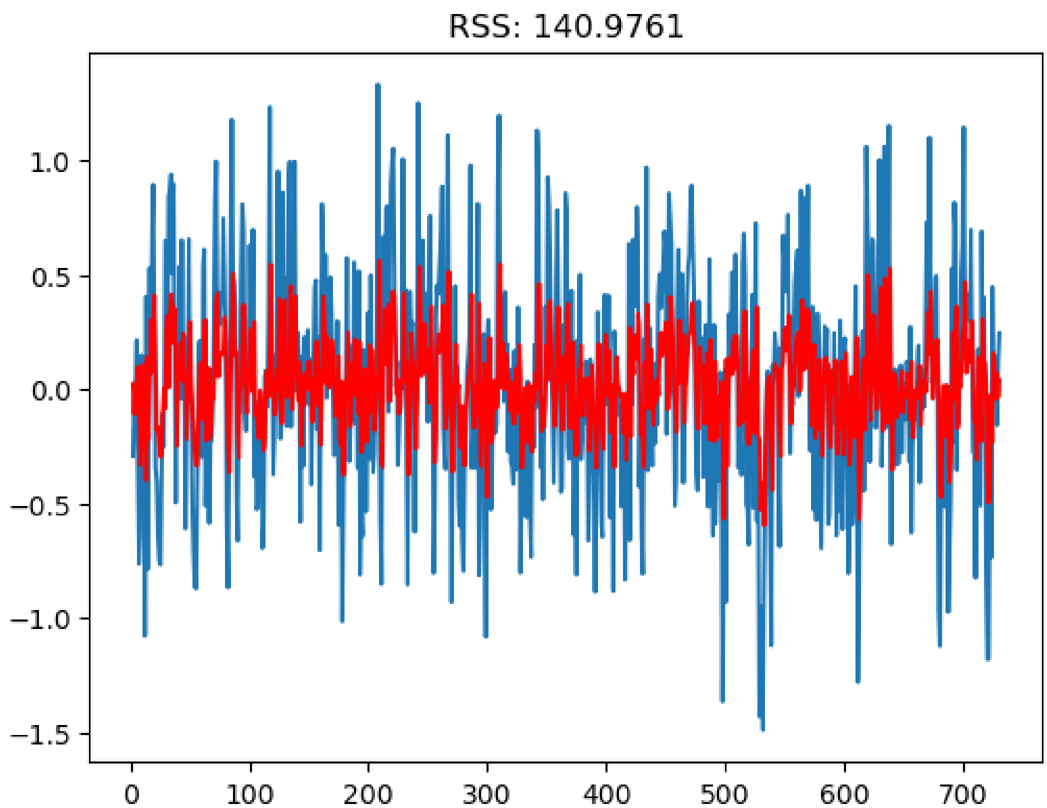
[13]:
```
#AR MODEL (Predicted best from auto-ARIMA)
model = ARIMA(med_shift, order=(1,1,0))
results_ARIMA = model.fit()
plt.plot(med_shift)
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_ARIMA.
 ↪fittedvalues-med_shift["Revenue"])**2))
print('Plotting AR model')
```

```
Plotting AR model
```



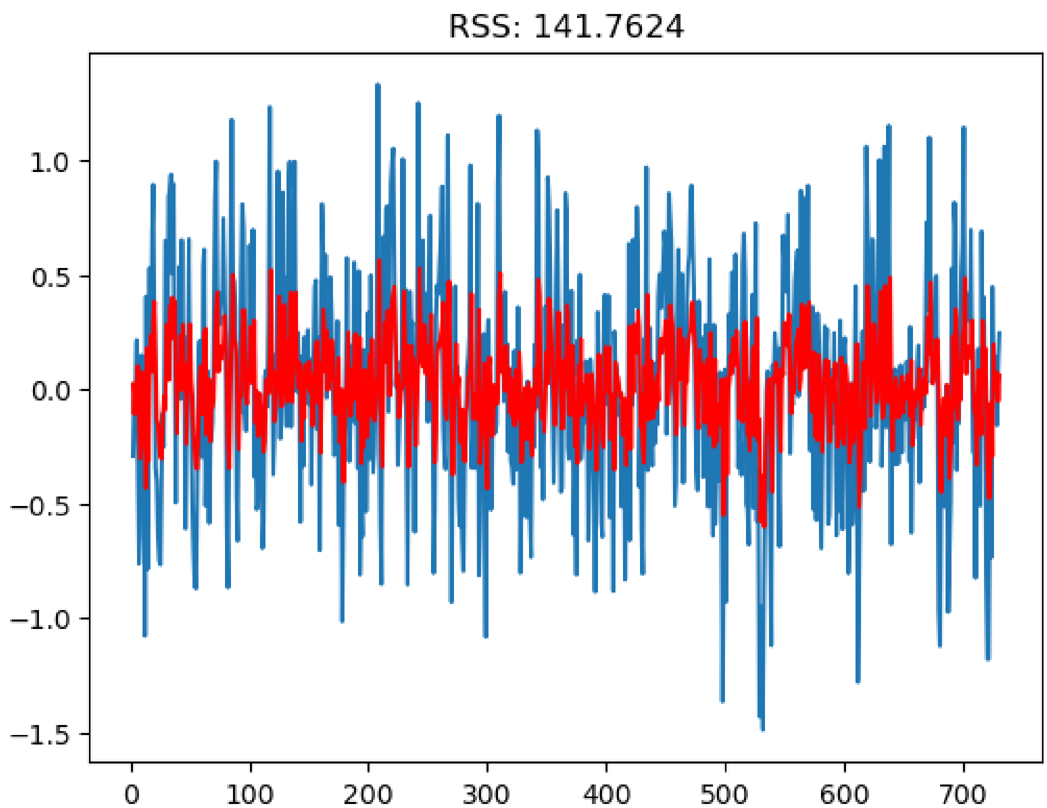
[14]:
```
#AR MODEL using P=1 from PACF, Q=2 from ACF, and D=0)
model = ARIMA(med_shift, order=(1,0,2))
results_ARIMA = model.fit()
plt.plot(med_shift)
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_ARIMA.
 ↪fittedvalues-med_shift["Revenue"])**2))
print('Plotting AR model')
```

```
Plotting AR model
```



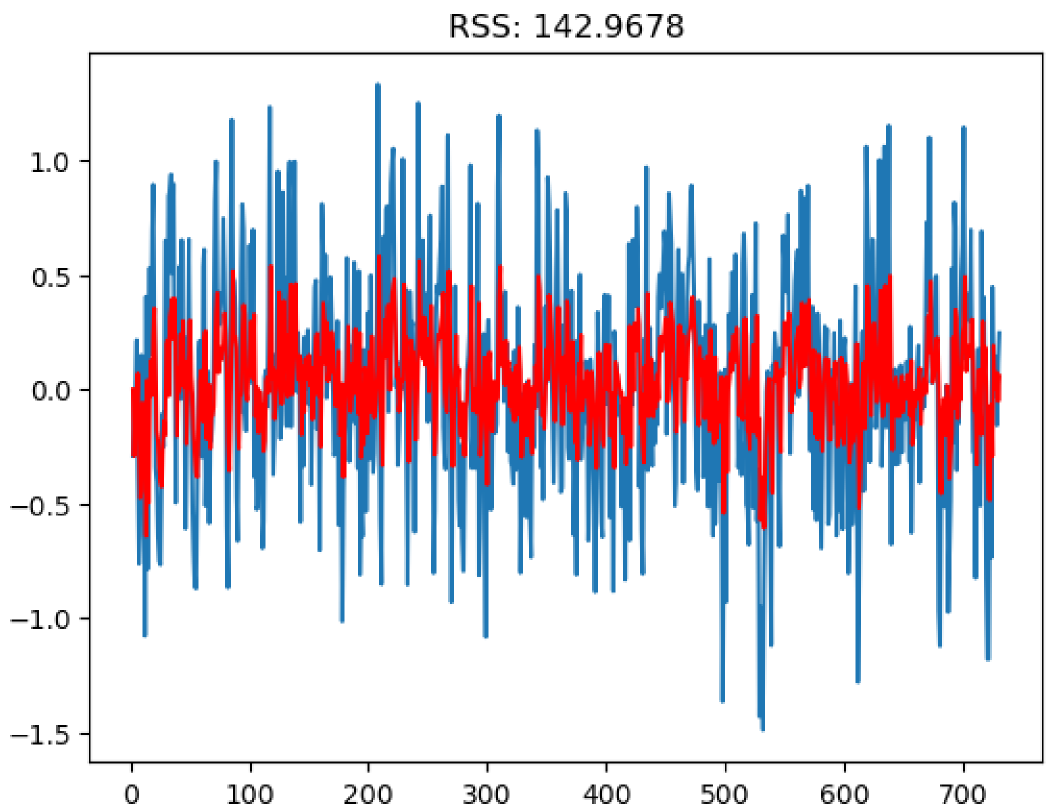
[15]:
```
#AR MODEL (Predicted best from auto-ARIMA, with the d value set to 0, since we␣
↪already shifted the data)
model = ARIMA(med_shift, order=(1,0,0))
results_ARIMA = model.fit()
plt.plot(med_shift)
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_ARIMA.
↪fittedvalues-med_shift["Revenue"])**2))
print('Plotting AR model')
```

```
Plotting AR model
```



[16]:
```
#AR MODEL using P=1 from PACF, Q=2 from ACF, and D=1)
model = ARIMA(med_shift, order=(1,1,2))
results_ARIMA = model.fit()
plt.plot(med_shift)
plt.plot(results_ARIMA.fittedvalues, color='red')
plt.title('RSS: %.4f'% sum((results_ARIMA.
 ↪fittedvalues-med_shift["Revenue"])**2))
print('Plotting AR model')
```

```
Plotting AR model
```



[17]:
```
#AR MODEL (Best fit based on RSS)
model = ARIMA(train, order=(1,0,2))
results_ARIMA = model.fit()
results_ARIMA.summary()
```

[17]:

Dep. Variable:	Revenue	**No. Observations:**	701
Model:	ARIMA(1, 0, 2)	**Log Likelihood**	-421.015
Date:	Mon, 26 Jun 2023	**AIC**	852.029
Time:	09:36:19	**BIC**	874.792
Sample:	0 - 701	**HQIC**	860.828
Covariance Type:	opg		

	coef	**std err**	**z**	**P>|z|**	**[0.025**	**0.975]**
const	13.2665	4.631	2.865	0.004	4.191	22.342
ar.L1	0.9962	0.003	301.475	0.000	0.990	1.003
ma.L1	0.4136	0.037	11.217	0.000	0.341	0.486
ma.L2	0.1985	0.037	5.305	0.000	0.125	0.272
sigma2	0.1930	0.011	17.470	0.000	0.171	0.215

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	1.89
Prob(Q):	0.97	**Prob(JB):**	0.39
Heteroskedasticity (H):	1.00	**Skew:**	-0.04
Prob(H) (two-sided):	0.99	**Kurtosis:**	2.76

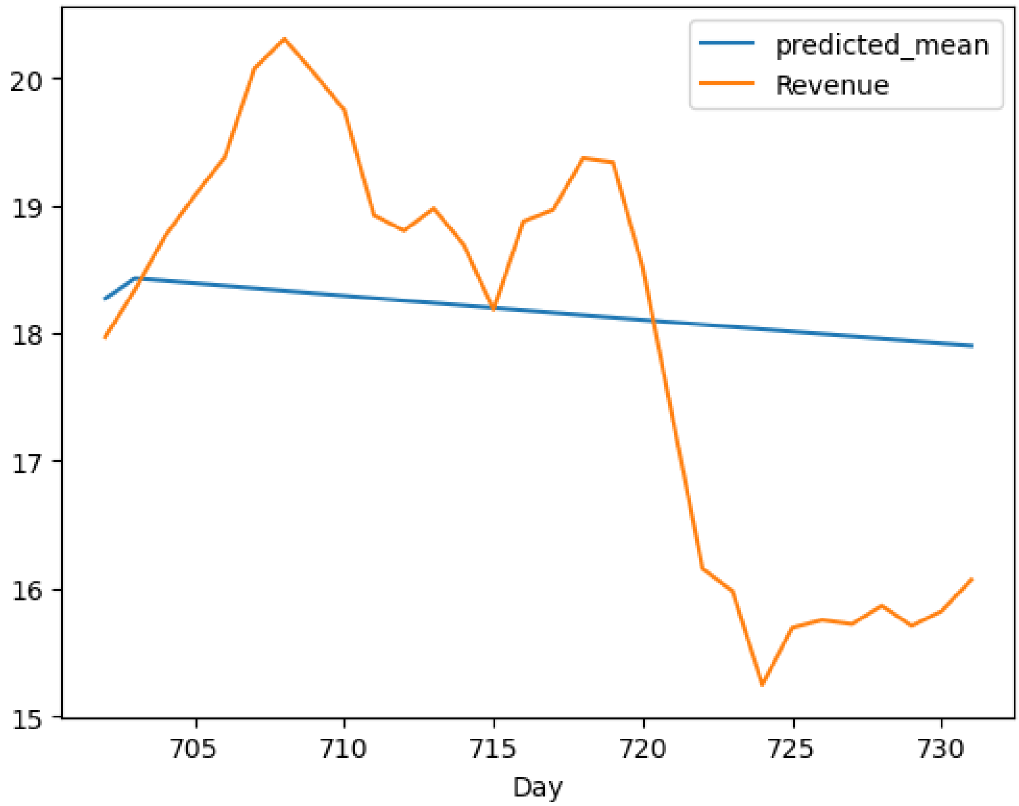
Warnings:
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

[18]:
```
start=len(train)
end=len(train)+len(test)-1
pred=results_ARIMA.predict(start=start, end=end, typ='levels')
print(pred)
pred.index=med.index[start:end+1]
```

```
701    18.271798
702    18.430822
703    18.411059
704    18.391371
705    18.371758
706    18.352221
707    18.332758
708    18.313369
709    18.294055
710    18.274815
711    18.255648
712    18.236555
713    18.217535
714    18.198587
715    18.179712
716    18.160910
717    18.142179
718    18.123520
719    18.104932
720    18.086416
721    18.067970
722    18.049595
723    18.031291
724    18.013056
725    17.994891
726    17.976796
727    17.958770
728    17.940812
729    17.922924
730    17.905104
Name: predicted_mean, dtype: float64
```

[19]:
```
pred.plot(legend=True)
test['Revenue'].plot(legend=True)
```

[19]: <Axes: xlabel='Day'>



[20]:
```
index_future_days = pd.interval_range(start=731, end=821, freq=1, closed='both')
print(index_future_days)
```

```
IntervalIndex([[731, 732], [732, 733], [733, 734], [734, 735], [735, 736] …
[816, 817], [817, 818], [818, 819], [819, 820], [820, 821]],
dtype='interval[int64, both]')
```

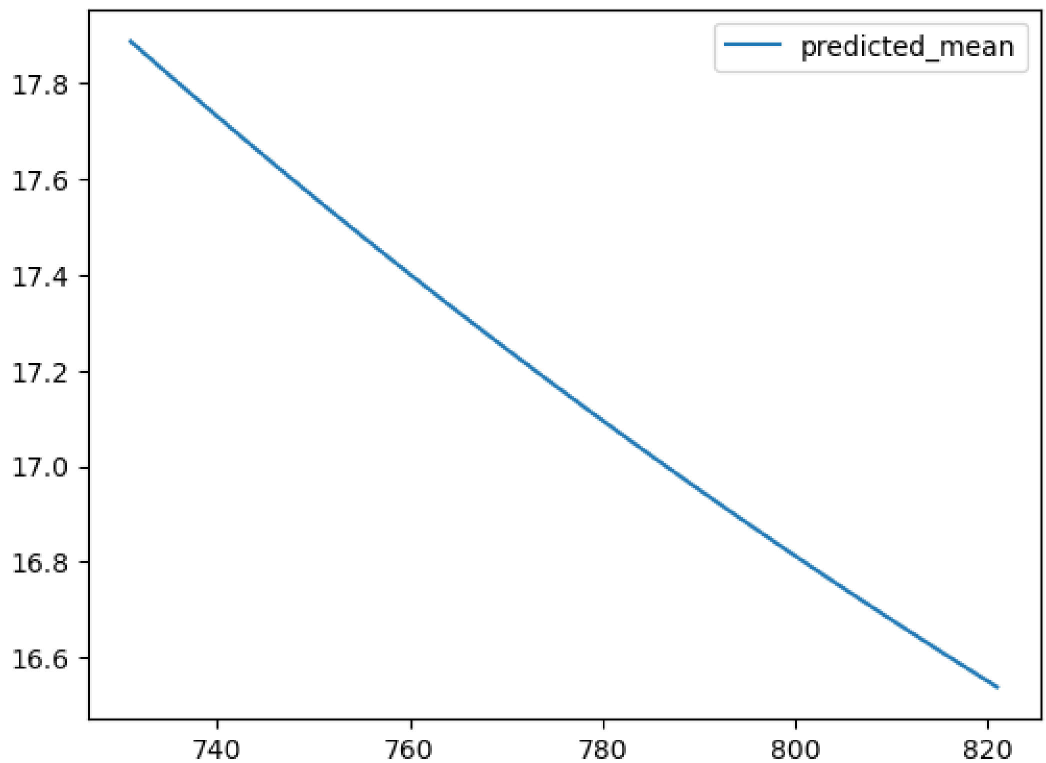
[21]:
```
pred=results_ARIMA.predict(start=len(med), end=len(med)+90, typ='levels')
print(pred)
```

```
731    17.887352
732    17.869668
733    17.852052
734    17.834504
735    17.817022
         …
817    16.589370
818    16.576654
819    16.563986
820    16.551367
```

```
821    16.538796
Name: predicted_mean, Length: 91, dtype: float64
```

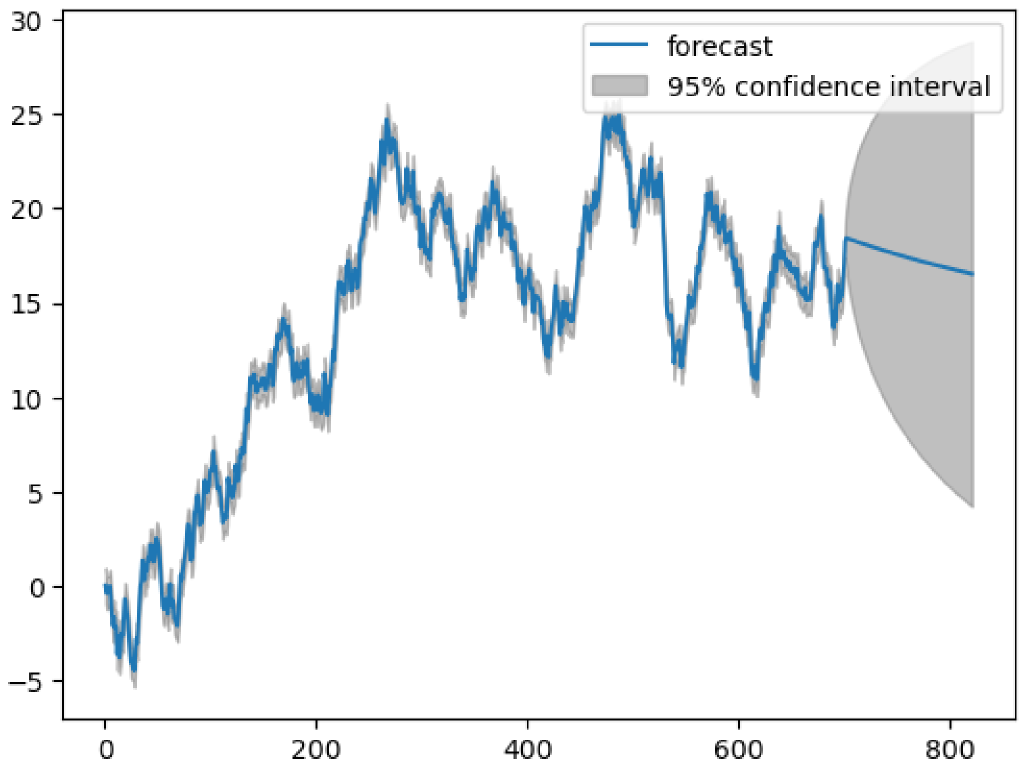
[22]:
```
pred.plot(legend=True)
```

[22]:
```
<Axes: >
```

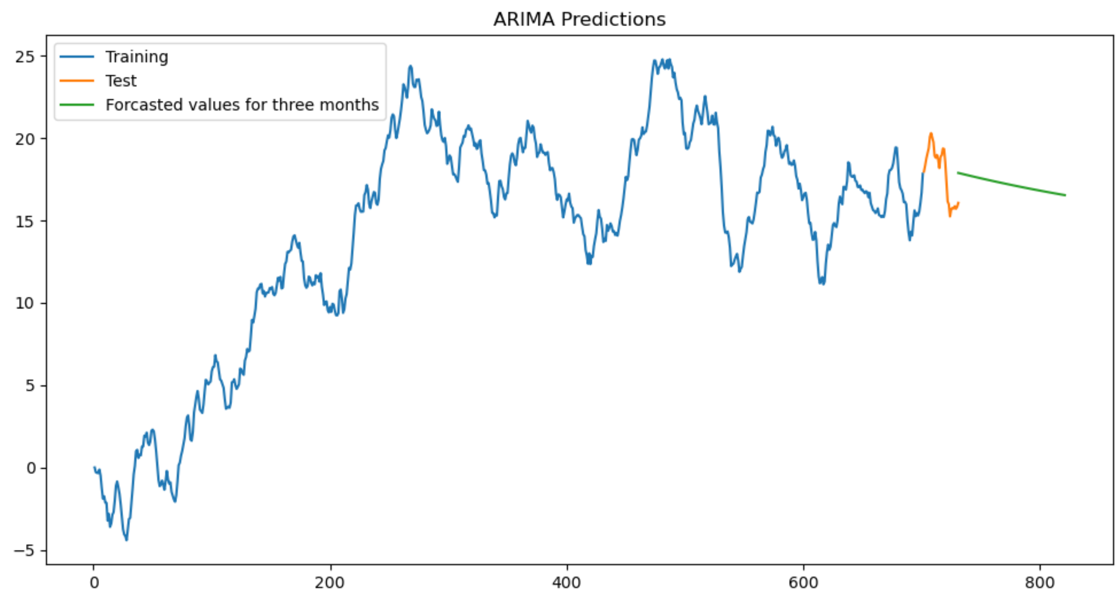


[23]:
```
med_shift.to_csv('C:/Users/dscha/Downloads/D213/med_ts_shifted.csv')
```

[24]:
```
from statsmodels.graphics.tsaplots import plot_predict
plot_predict(results_ARIMA, start=1, end=821)
plt.show()
```



[25]:
```
plt.figure(figsize=(12,6))
plt.plot(train['Revenue'], label='Training')
plt.plot(test['Revenue'], label='Test')
plt.plot(pred, label="Forcasted values for three months")
plt.legend(loc='upper left')
plt.title('ARIMA Predictions')
plt.show()
```



[26]:
```
train.to_csv('C:/Users/dscha/Downloads/D213/med_train.csv')
```

[27]:
```
test.to_csv('C:/Users/dscha/Downloads/D213/med_test.csv')
```

[28]:
```
pred.to_csv('C:/Users/dscha/Downloads/D213/med_pred.csv')
```

[ ]: