D213 - Advanced Data Analytics PA1

July 22, 2022

0.0.1 D213 - Advanced Data Analytics - PA1

0.0.2 Background Info:

As part of the "readmission" project, executives would like to see consider a time series on revenue from the first years of operation. Once they understand any patterns in that data, they feel confident in understanding the impact of readmission in current times. The given time series data records the daily revenue, in million dollars, during the first two years of operation.

A1 *Question:* Using the previous two years of data, are there any patterns present that can predict the revenue produced by the hospital for the next quarter?

0.0.3 Import Libraries

[1]:
```
import pandas as pd
from pandas.plotting import autocorrelation_plot
import seaborn as sns
import numpy as np
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima_model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
import pmdarima as pm
import matplotlib.pyplot as plt
from scipy import signal
from datetime import datetime
from sklearn.model_selection import train_test_split
from tqdm import tqdm
import warnings
warnings.filterwarnings('ignore')
#!pip install joblib
import joblib
%matplotlib inline
%time
%timeit
```

```
CPU times: user 1 µs, sys: 0 ns, total: 1 µs
Wall time: 3.1 µs
```

```
[2]: #%lsmagic
```

0.0.4 Load Data From medical_time_series.csv

```
[3]: # load data file
initial_df = pd.read_csv('medical_time_series.csv', index_col='Day',
 ↪parse_dates=True)
# quick test the data is present and see the shape
print("df shape: ", initial_df.shape)
initial_df.head()
```

```
df shape:  (731, 1)
```

[3]:

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

```
[4]: initial_df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 731 entries, 1 to 731
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Revenue  731 non-null    float64
dtypes: float64(1)
memory usage: 11.4 KB
```

```
[5]: initial_df.describe()
```

[5]:

	Revenue
count	731.000000
mean	14.179608
std	6.959905
min	-4.423299
25%	11.121742
50%	15.951830
75%	19.293506
max	24.792249

```
[6]: # Any Null Values?
initial_df.isnull().any()
```

[6]: Revenue   False
dtype: bool

0.0.5 Check for Missing Values

[7]:
```
# Mapping to view missing data...none present.
sns.heatmap(initial_df.isnull(), yticklabels=False, cbar=False, cmap='viridis');
```

[8]:
```
initial_df.columns
```

[8]: Index(['Revenue'], dtype='object')

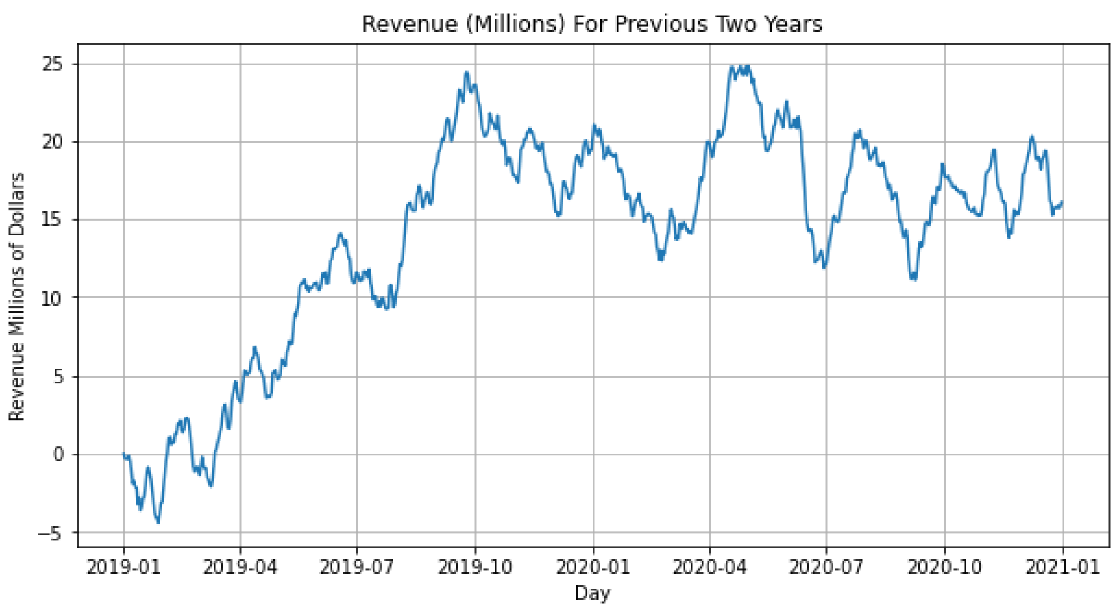
[9]:
```
# Convert Day to a Date
initial_df['Date'] = (pd.date_range(start=datetime(2019,1,1),
                      periods=initial_df.shape[0], freq='24H'))
# Set the Date as an index
initial_df.set_index('Date',inplace=True)
initial_df
```

[9]:
```
             Revenue
Date
2019-01-01  0.000000
2019-01-02 -0.292356
2019-01-03 -0.327772
2019-01-04 -0.339987
```

```
2019-01-05  -0.124888
…                 …
2020-12-27  15.722056
2020-12-28  15.865822
2020-12-29  15.708988
2020-12-30  15.822867
2020-12-31  16.069429

[731 rows x 1 columns]
```

0.1 C1 - Provide a line graph visualizing the realization of the time series

[10]:
```
#https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?
↪id=efceba6c-e8ef-47a2-b859-aec400fe18e7
plt.figure(figsize=(10,5))
plt.plot(initial_df.Revenue)
plt.title('Revenue (Millions) For Previous Two Years')
plt.xlabel('Day')
plt.ylabel('Revenue Millions of Dollars')
plt.grid(True)
plt.show()
```



[19]:
```
# Drop any null columns
df = initial_df.dropna()
df
```

[19]:

```
                Revenue
Date
2019-01-01   0.000000
2019-01-02  -0.292356
2019-01-03  -0.327772
2019-01-04  -0.339987
2019-01-05  -0.124888
…                 …
2020-12-27  15.722056
2020-12-28  15.865822
2020-12-29  15.708988
2020-12-30  15.822867
2020-12-31  16.069429

[731 rows x 1 columns]
```

[20]:

```
# Export cleaned data
pd.DataFrame(df).to_csv("df_cleaned.csv")
```

0.2 C3 - Make Time Series Stationary

[21]:

```
# Verify if data is stationary

result = adfuller(df['Revenue'])

print("Test Statistics: ", result[0])
print("p-value: ", result[1])
print("Critical Values: ",result[4])
```

```
Test Statistics:  -2.2183190476089463
p-value:  0.19966400615064323
Critical Values:  {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%':
-2.5688855736949163}
```

[22]:

```
# Accept or reject null hypothesis
if result[1] <= 0.05: #Compare result against threshold
    print("Reject null hypothesis, this time series data is stationary.")
else:
    print("Accept null hypothesis, this time series data is non-stationary!")
```

```
Accept null hypothesis, this time series data is non-stationary!
```

[23]:

```
# Make time series stationary
df_stationary = df.diff().dropna()

# View
df_stationary.head()
```

[23]:
```
                 Revenue
Date
2019-01-02 -0.292356
2019-01-03 -0.035416
2019-01-04 -0.012215
2019-01-05  0.215100
2019-01-06 -0.366702
```

[24]:
```python
# Test if data is stationary again

result = adfuller(df_stationary['Revenue'])

print("Test Statistics: ", result[0])
print("p-value: ", result[1])
print("Critical Values: ",result[4])

if result[1] <= 0.05: #Compare result against threshold
    print("Reject null hypothesis, this time series data is stationary.")
else:
    print("Accept null hypothesis, this time series data is non-stationary!")
```

```
Test Statistics:  -17.37477230355706
p-value:  5.1132069788403175e-30
Critical Values:  {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%':
-2.5688855736949163}
Reject null hypothesis, this time series data is stationary.
```

0.3 Train, Test, and Split

[25]:
```python
# Split for Training and Testing

X_train = df_stationary.loc[:'2020-09-30']
X_test = df_stationary['2020-10-01':]

print('X_train Shape: ', X_train.shape)
print('X_test Shape: ', X_test.shape)
```

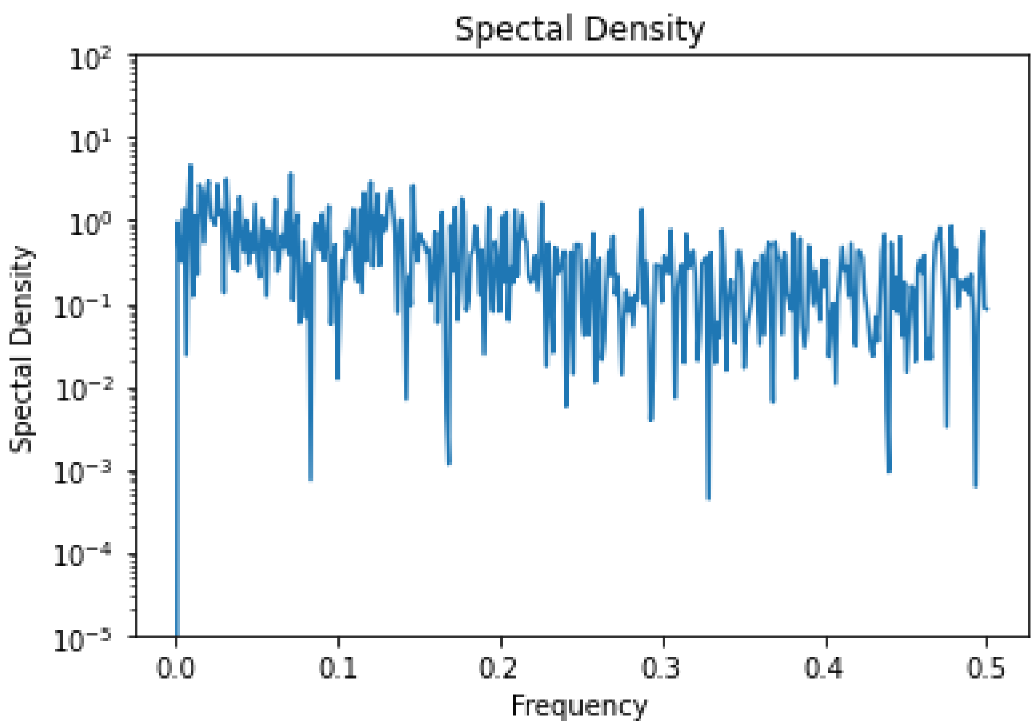
```
X_train Shape:  (638, 1)
X_test Shape:  (92, 1)
```

0.4 C5 - Prepared Dataset

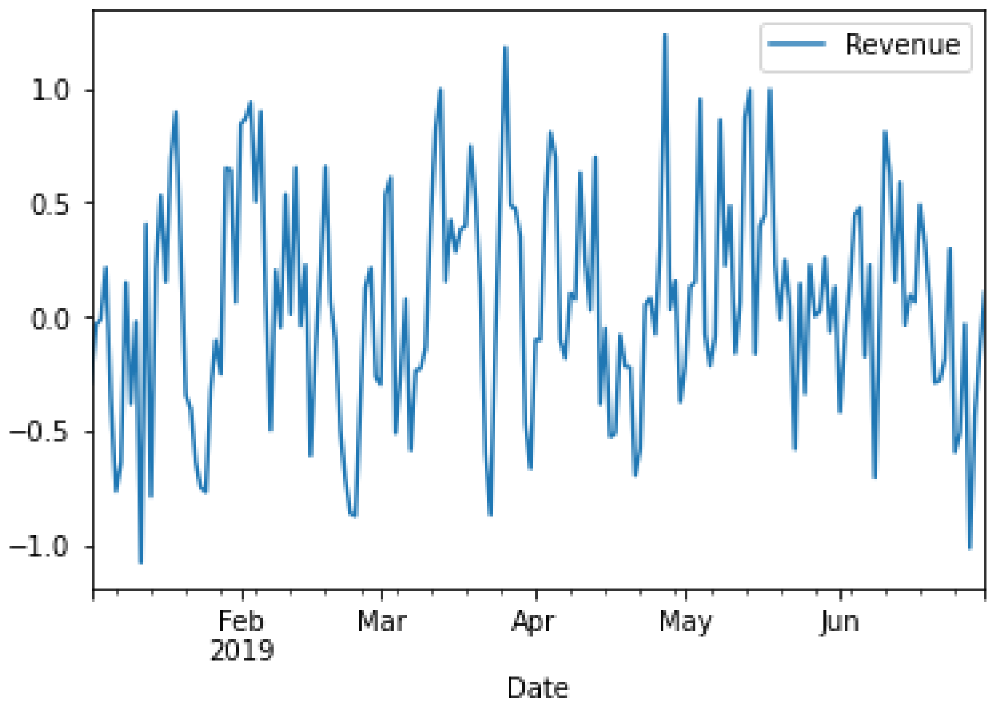
[26]:
```python
# Export stationary data
pd.DataFrame(df_stationary).to_csv("df_stationary.csv")
```

[27]:
```python
# Spectal Density
```

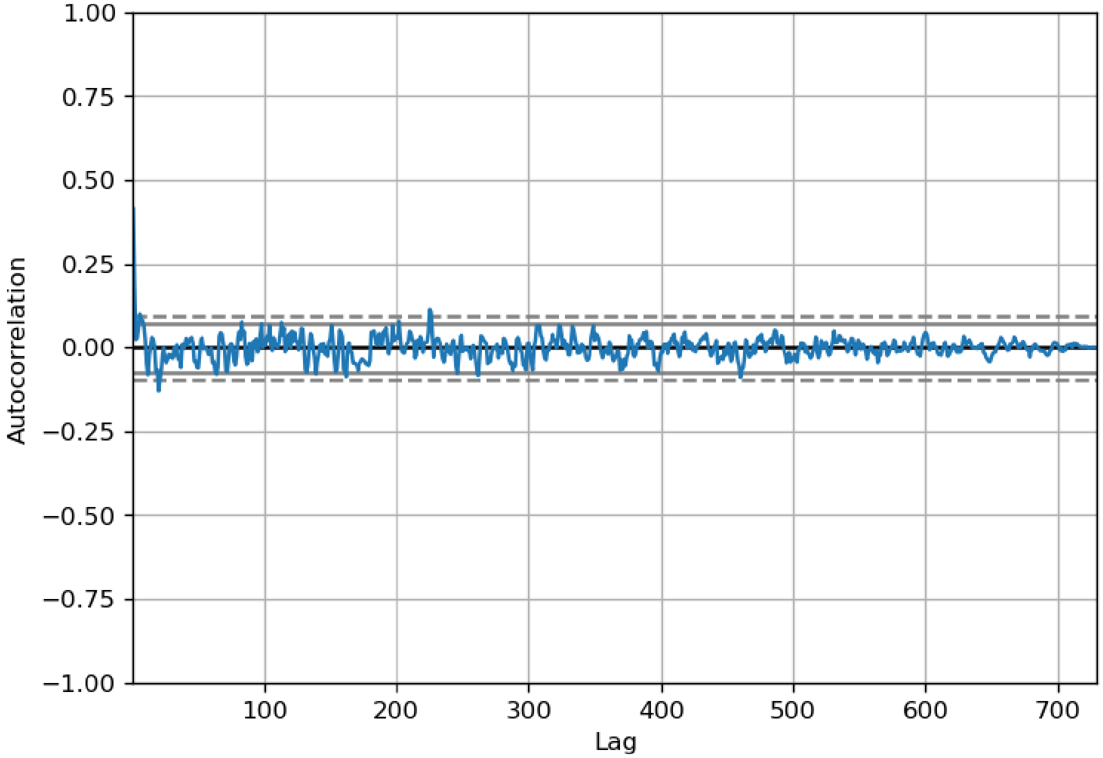
```
f, Pxx_den=signal.periodogram(df_stationary['Revenue'])
plt.semilogy(f,Pxx_den)
plt.ylim([1e-5,1e2])
plt.title('Spectal Density')
plt.xlabel('Frequency')
plt.ylabel('Spectal Density')
plt.show()
```



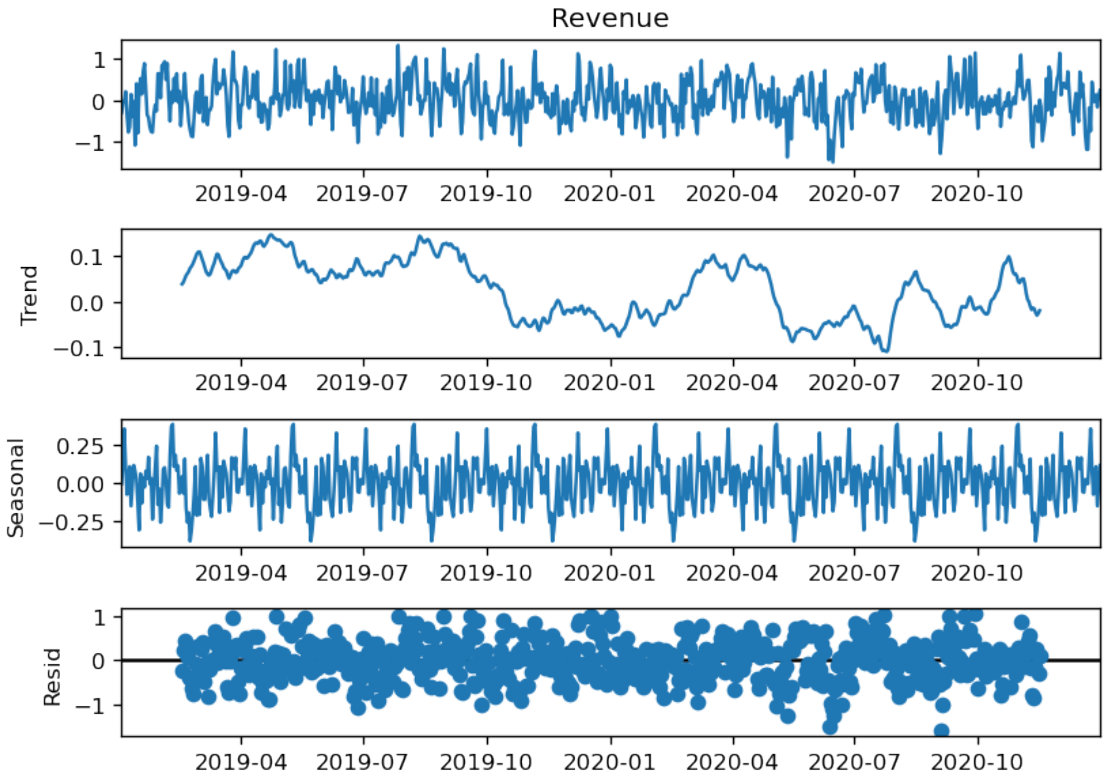
[28]:
```
# Look for seasonality in data
df_stationary.loc[:'2019-06-30'].plot();
```



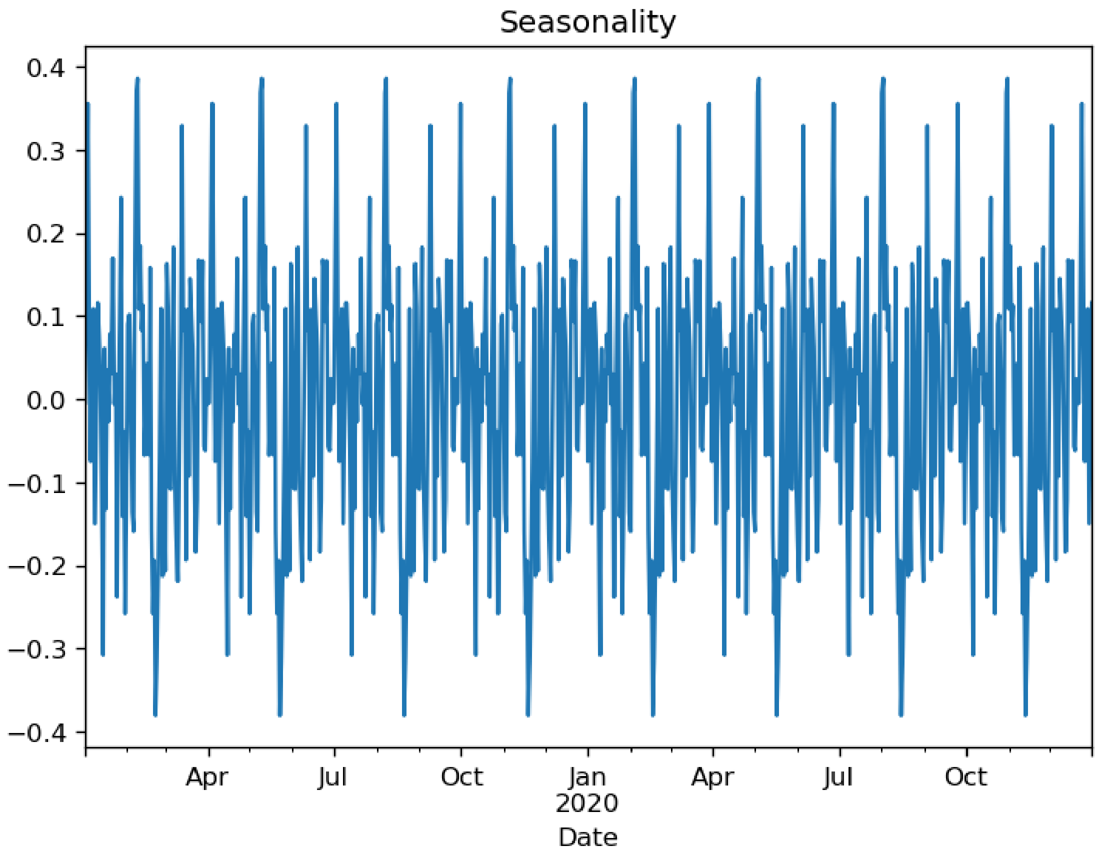
[29]:
```
# Continue looking for seasonality
plt.rcParams.update({'figure.figsize':(7,5), 'figure.dpi':120})
autocorrelation_plot(df_stationary.Revenue.tolist());
```



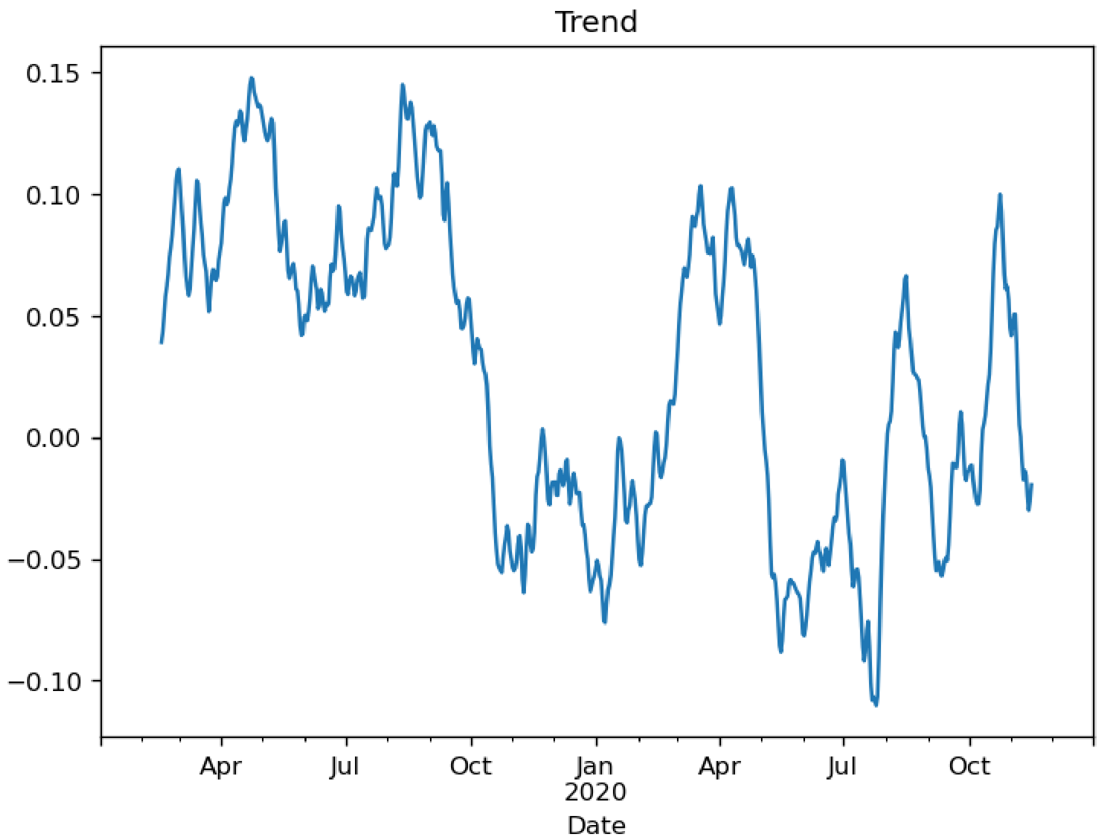
[30]:
```
# Decomposition
decomp = seasonal_decompose(df_stationary['Revenue'],period=90)

# Plot decomposition
decomp.plot()
plt.show()
```



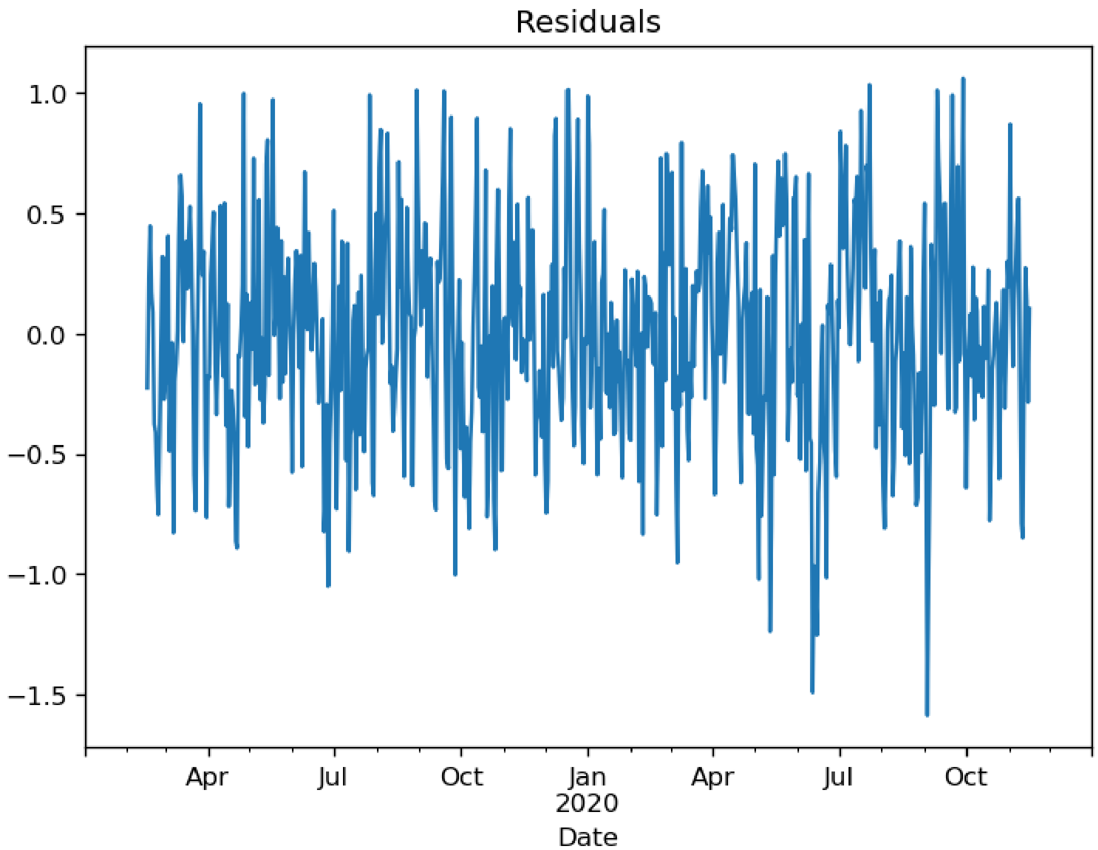
[31]:
```
# Plot Seasonality

plt.title('Seasonality')
decomp.seasonal.plot();
```



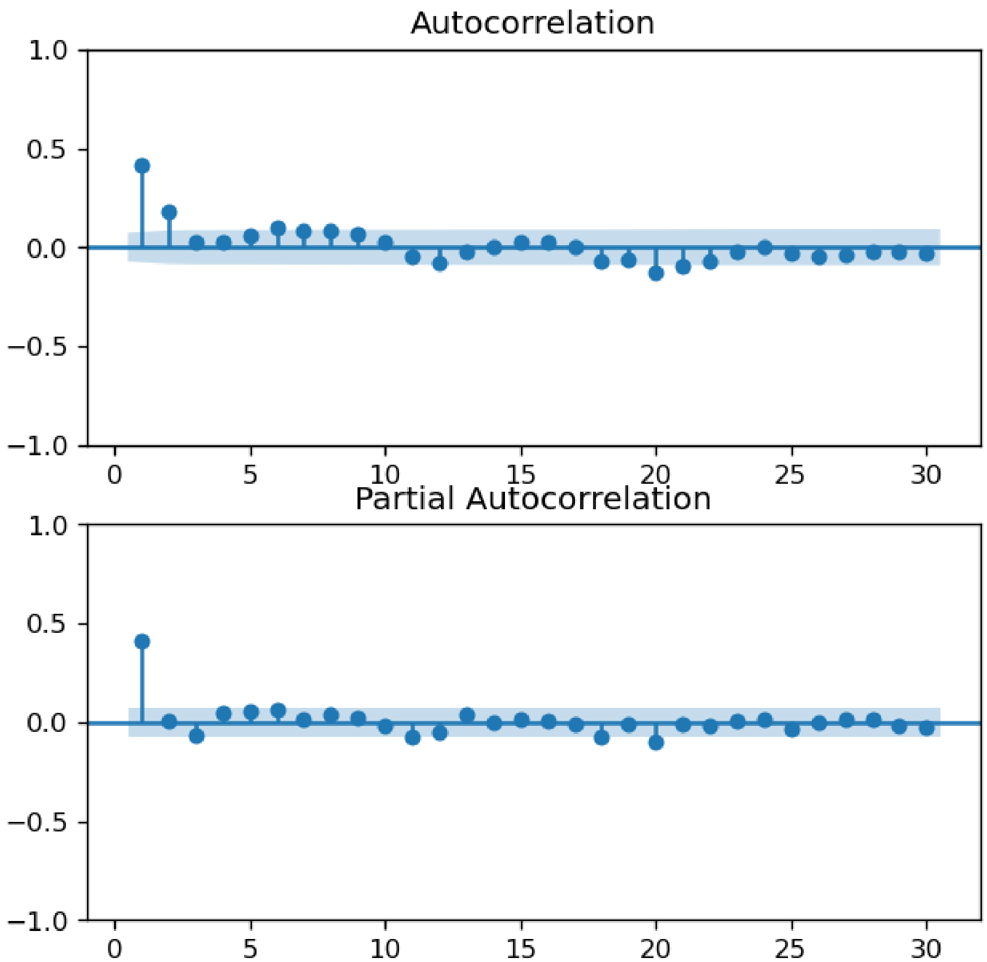
[32]:
```
# View Trend
plt.title('Trend')
decomp.trend.plot();
```



[33]:
```
# Plot Residual
plt.title('Residuals')
decomp.resid.plot();
```



[34]:
```
# ACF and PACF Autocorrelation Plots

# fig
fig, (ax1, ax2) = plt.subplots(2,1, figsize=(6,6))

# Plot ACF of df
plot_acf(df_stationary, lags=30, zero=False, ax=ax1)

# Plot PACF of df
plot_pacf(df_stationary, lags=30, zero=False, ax=ax2)
plt.show();
```



[35]:

```
# Pick best order by aic

best_aic = np.inf
best_order = None
best_mdl = None
rng = range(3)
for p in rng: # loop over p-values
    for q in rng: #loop over q values
        try:

            #create and fit ARIMA(p,q) model
            model = SARIMAX(df_stationary, order=(p,1,q), trend = 'c')
            results = model.fit()
            tmp_aic = results.aic
            print(p, q, results.aic, results.bic)
            if tmp_aic < best_aic:
```

```
                best_aic = tmp_aic
                best_order = (p, q)
                best_mdl = tmp_mdl

            # Print order and results
        except:
            print(p,q, None, None)

print('\nBest AIC: {:6.5f} | order: {}'.format(best_aic, best_order))
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  7.72420D-01    |proj g|=  3.63370D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      1     21      1     0     0   3.634D-05   7.724D-01
  F =  0.77242001314190578

ABNORMAL_TERMINATION_IN_LNSRCH
0 0 1131.7332191871824 1140.9165666511997
0 0 None None
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            3     M =           10
```

```
At X0         0 variables are exactly at the bounds

At iterate    0    f=  6.96400D-01    |proj g|=  8.69365D-02

At iterate    5    f=  6.73133D-01    |proj g|=  2.18277D-01

At iterate   10    f=  6.72711D-01    |proj g|=  4.35275D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    3     11     13      1     0     0   7.492D-08   6.727D-01
  F =  0.67271143814447110

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
0 1 988.1586996909278 1001.9337208869538
0 1 None None
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            4     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  6.48526D-01    |proj g|=  6.45857D-02
 This problem is unconstrained.

 Line search cannot locate an adequate point after MAXLS
  function and gradient evaluations.
  Previous x, f and g restored.
 Possible causes: 1 error in function or gradient evaluation;
                  2 rounding error dominate computation.
 This problem is unconstrained.
 This problem is unconstrained.
```

```
At iterate    5    f=  6.37907D-01    |proj g|=  1.07031D-01

At iterate   10    f=  6.25222D-01    |proj g|=  2.62385D-01

At iterate   15    f=  6.23246D-01    |proj g|=  7.96411D-01

At iterate   20    f=  6.20389D-01    |proj g|=  1.99214D+00

At iterate   25    f=  6.19495D-01    |proj g|=  2.21354D-01

At iterate   30    f=  6.19398D-01    |proj g|=  6.77096D-02

 Bad direction in the line search;
   refresh the lbfgs memory and restart the iteration.

 Warning:  more than 10 function and gradient
   evaluations in the last line search.  Termination
   may possibly be caused by a bad search direction.
 This problem is unconstrained.
 This problem is unconstrained.


           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    4     31     74      2     0     0   6.771D-02   6.194D-01
  F =  0.61939751825460820

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
0 2 912.320376651728 930.6870715797627
0 2 None None
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            3     M =           10
```

```
At X0         0 variables are exactly at the bounds

At iterate    0    f=  7.25538D-01    |proj g|=  2.12223D-03

At iterate    5    f=  7.25538D-01    |proj g|=  2.49879D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    3      6      8      1     0     0   1.433D-05   7.255D-01
  F =  0.72553774292633710

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
1 0 1065.2851046724522 1079.0601258684783
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            4     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  6.50193D-01    |proj g|=  1.64012D-01

At iterate    5    f=  6.36116D-01    |proj g|=  1.76948D-01

At iterate   10    f=  6.33139D-01    |proj g|=  1.06543D-01

At iterate   15    f=  6.14880D-01    |proj g|=  2.92431D+00

At iterate   20    f=  6.10487D-01    |proj g|=  6.16222D-01

At iterate   25    f=  6.03877D-01    |proj g|=  4.98576D+00

At iterate   30    f=  6.02712D-01    |proj g|=  5.53957D-01

At iterate   35    f=  6.02587D-01    |proj g|=  4.27588D-02
```

```
At iterate   40    f=  6.02509D-01    |proj g|=  4.43931D-01

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    4     43     66      1     0     0   2.585D-01   6.025D-01
  F =  0.60250925503394537

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
1 1 887.6635123495602 906.0302072775949
1 1 None None
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  6.28707D-01    |proj g|=  5.90938D-01

At iterate    5    f=  6.26311D-01    |proj g|=  1.66440D-01

At iterate   10    f=  6.20894D-01    |proj g|=  4.35677D-01

At iterate   15    f=  6.18420D-01    |proj g|=  3.31768D-02

 Warning:  more than 10 function and gradient
   evaluations in the last line search.  Termination
   may possibly be caused by a bad search direction.
 This problem is unconstrained.

At iterate   20    f=  6.18327D-01    |proj g|=  1.43175D-01

At iterate   25    f=  6.13978D-01    |proj g|=  5.64560D-01
```

```
At iterate   30    f=  6.11709D-01    |proj g|=  7.09927D-01

At iterate   35    f=  6.07473D-01    |proj g|=  3.60963D-01

At iterate   40    f=  6.03987D-01    |proj g|=  3.25555D+00

At iterate   45    f=  6.02676D-01    |proj g|=  1.41252D+00

At iterate   50    f=  6.02511D-01    |proj g|=  8.22343D-01

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     50     56      1     0     0   8.223D-01   6.025D-01
  F =  0.60251149149345451

STOP: TOTAL NO. of ITERATIONS REACHED LIMIT
1 2 889.6667775804435 912.6251462404869
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            4     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  7.10268D-01    |proj g|=  4.82104D-03

At iterate    5    f=  7.10266D-01    |proj g|=  4.30004D-06

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
```

```
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    4      5      8      1     0     0   4.300D-06   7.103D-01
  F =  0.71026607797511199

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
2 0 1044.9884738436635 1063.3551687716981
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            5     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  7.81060D-01    |proj g|=  4.12026D-01

At iterate    5    f=  7.00036D-01    |proj g|=  5.77628D-02

 This problem is unconstrained.
 This problem is unconstrained.

At iterate   10    f=  6.80209D-01    |proj g|=  1.58045D-01

At iterate   15    f=  6.26507D-01    |proj g|=  4.53023D-02

At iterate   20    f=  6.10201D-01    |proj g|=  3.80628D-01

At iterate   25    f=  6.03752D-01    |proj g|=  8.88891D+00

At iterate   30    f=  6.02826D-01    |proj g|=  9.65316D-01

At iterate   35    f=  6.02496D-01    |proj g|=  7.85718D-02

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
```

```
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    5     38     68      1     0     0   6.071D-02   6.025D-01
  F =  0.60248035125047006

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
2 1 889.6213128256863 912.5796814857297
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            6     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  9.07847D-01    |proj g|=  1.15640D+00


 Warning:  more than 10 function and gradient
   evaluations in the last line search.  Termination
   may possibly be caused by a bad search direction.
 This problem is unconstrained.


At iterate    5    f=  6.61875D-01    |proj g|=  1.11694D-01

At iterate   10    f=  6.36518D-01    |proj g|=  4.38022D-01

At iterate   15    f=  6.19589D-01    |proj g|=  3.50045D-01

At iterate   20    f=  6.03511D-01    |proj g|=  7.63593D-01

At iterate   25    f=  6.02224D-01    |proj g|=  3.36742D-01

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *
```

```
   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    6     29     48      1     0     0   2.696D-03   6.022D-01
  F =  0.60220556374000445

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
2 2 891.2201230604064 918.7701654524584

Best AIC: 887.66351 | order: (1, 1)


 Warning:  more than 10 function and gradient
   evaluations in the last line search.  Termination
   may possibly be caused by a bad search direction.
```

1 Uncomment Code to run Auto ARIMA; Takes > 120 min

[36]:
```
# Use Auto ARIMA to Find best model
# https://www.machinelearningplus.com/time-series/
 ↪arima-model-time-series-forecasting-python/

#%time
#tqdm.pandas()

#model = pm.auto_arima(df_stationary,
#                      seasonal=True, m=90,
#                      d=1, D=1,
#                      start_p=1, start_q=1,
#                      max_p=2, maxq=2,
#                      maxP=2, maxQ=2,
#                      trace=True,
#                      error_action='ignore',
#                      suppress_warnings=True)
```

[37]:
```
#print(model.summary())
```

2 Terminal: nbconvert –to pdf D213_PA1.ipynb

[38]:
```
# Create Time Series Model

model = SARIMAX(df_stationary, order=(1,1,0),seasonal_order=(1,1,0,90))
results = model.fit()
results.summary()
```

```
RUNNING THE L-BFGS-B CODE
```

```
          * * *

Machine precision = 2.220D-16
 N =            3     M =           10

At X0         0 variables are exactly at the bounds

At iterate    0    f=  8.65762D-01    |proj g|=  2.80477D-01

 This problem is unconstrained.


At iterate    5    f=  8.52032D-01    |proj g|=  2.52986D-03

          * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

          * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg       F
    3      9     12      1     0     0   2.269D-06   8.520D-01
  F =  0.85202993770228830

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```

[38]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:                            Revenue   No. Observations:
730
Model:             SARIMAX(1, 1, 0)x(1, 1, 0, 90)   Log Likelihood
-621.982
Date:                            Fri, 22 Jul 2022   AIC
1249.964
Time:                                    19:05:20   BIC
1263.343
Sample:                                01-02-2019   HQIC
1255.157
                                     - 12-31-2020
```

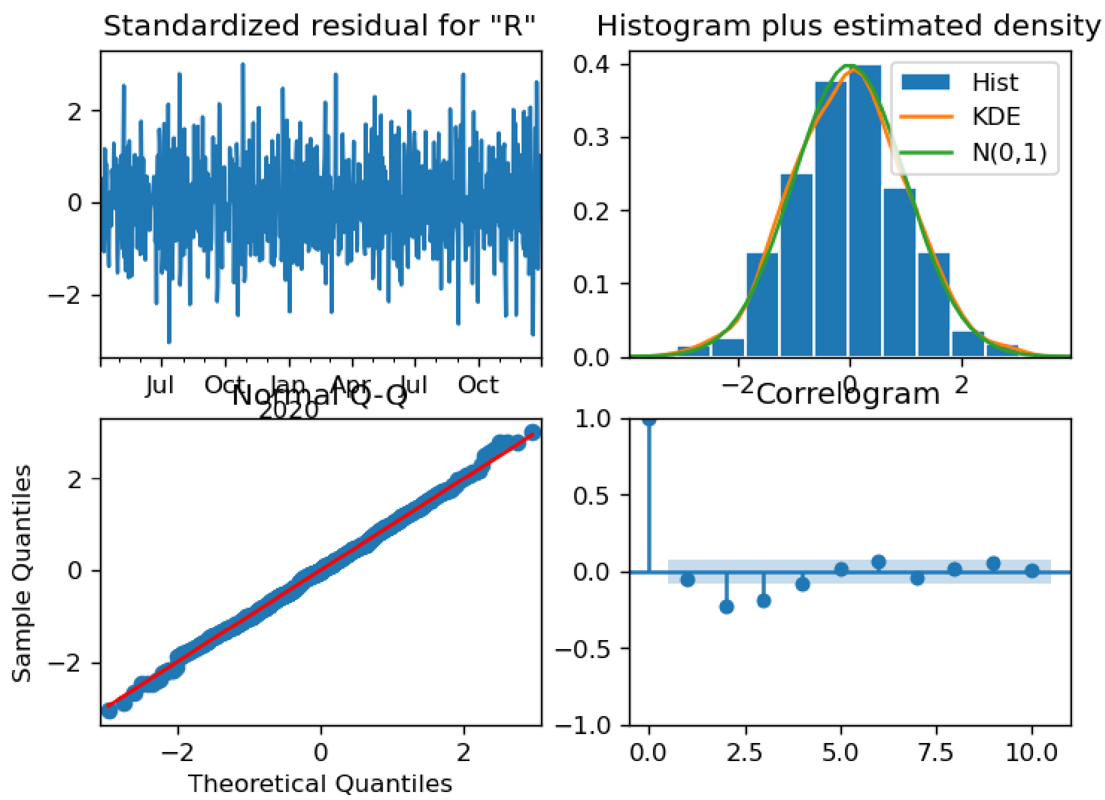
```
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3084      0.037     -8.398      0.000      -0.380      -0.236
ar.S.L90      -0.4726      0.039    -12.259      0.000      -0.548      -0.397
sigma2         0.3958      0.022     17.749      0.000       0.352       0.439
================================================================================
===
Ljung-Box (L1) (Q):                   2.01   Jarque-Bera (JB):
0.19
Prob(Q):                              0.16   Prob(JB):
0.91
Heteroskedasticity (H):               1.10   Skew:
0.04
Prob(H) (two-sided):                  0.48   Kurtosis:
2.98
================================================================================
===

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

[39]:
```python
# Warnings:
#[1] Covariance matrix calculated using the outer product of gradients (complex-step)
# Prob(Q): value indicates residuals are not correlated.
# Prob(JB): value indicates residuals are normally distributed.
# Model evaluation
```

[40]:
```python
# Print mean absolute error
mae = np.mean(np.abs(results.resid))
print("Mean Absolute Error: ", mae)
```

```
Mean Absolute Error:  0.49873094599791684
```

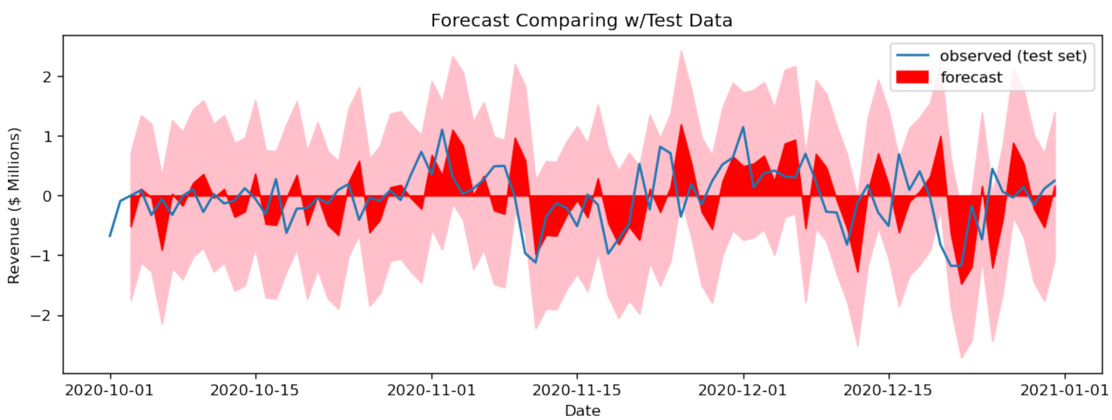
[41]:
```python
# Create the 4 diagnostics plots
results.plot_diagnostics().show()
```



[42]:
```
# Validate w/Test Set

# Predictions
prediction = results.get_prediction(start=-90)

# Prediction Mean
mean_prediction = prediction.predicted_mean

# Confidence Intervals of Predictions
confidence_intervals = prediction.conf_int()

# Upper & lower conf limits
lower_limits = confidence_intervals.loc[:,'lower Revenue']
upper_limits = confidence_intervals.loc[:,'upper Revenue']

# Print predictions (best estimate)
# print(mean_forcast)
```

[43]:
```
# Plot Data
plt.figure(figsize=(12,4))
```

```
plt.plot(np.array(X_test.index), np.array(X_test[['Revenue']]), label='observed␣
 ↪(test set)')

# shade upper conf. limit area
plt.fill_between(upper_limits.index, upper_limits, lower_limits, color='pink')

# plot mean predictions
plt.fill_between(mean_prediction.index, mean_prediction, color='red',␣
 ↪label='forecast')

# Labels and Legends
plt.title('Forecast Comparing w/Test Data')
plt.xlabel('Date')
plt.ylabel('Revenue ($ Millions)')
plt.legend()
plt.show()
```



[44]:
```
# Validate w/Test Set

# Predictions
prediction = results.get_prediction(start=0)

# Prediction Mean
mean_prediction = prediction.predicted_mean

# Confidence Intervals of Predictions
confidence_intervals = prediction.conf_int()

# Upper & lower conf limits
lower_limits = confidence_intervals.loc[:,'lower Revenue']
upper_limits = confidence_intervals.loc[:,'upper Revenue']
```

```
# Print predictions (best estimate)
# print(mean_forcast)
```

[45]:
```
# Perform forecast
diff_forecast = results.get_forecast(steps=180)
mean_forecast = diff_forecast.predicted_mean

# Conf intervals of predictions
confidence_intervals = diff_forecast.conf_int()

# Upper & Lower conf limits
lower_limits = confidence_intervals.loc[:,'lower Revenue']
upper_limits = confidence_intervals.loc[:,'upper Revenue']
```

[46]:
```
df_stationary[['Revenue']].reset_index(drop=True)
```

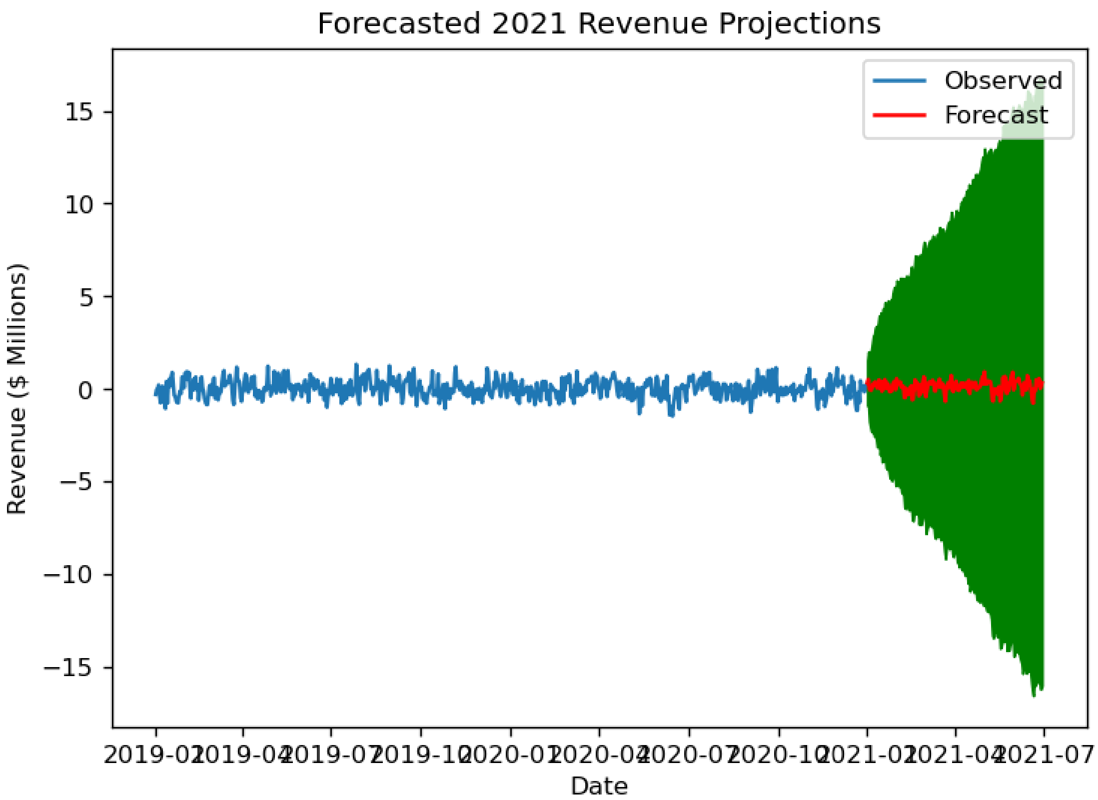
[46]:
```
      Revenue
0   -0.292356
1   -0.035416
2   -0.012215
3    0.215100
4   -0.366702
..        …
725 -0.032693
726  0.143766
727 -0.156834
728  0.113880
729  0.246562

[730 rows x 1 columns]
```

[47]:
```
lower_limits.index
```

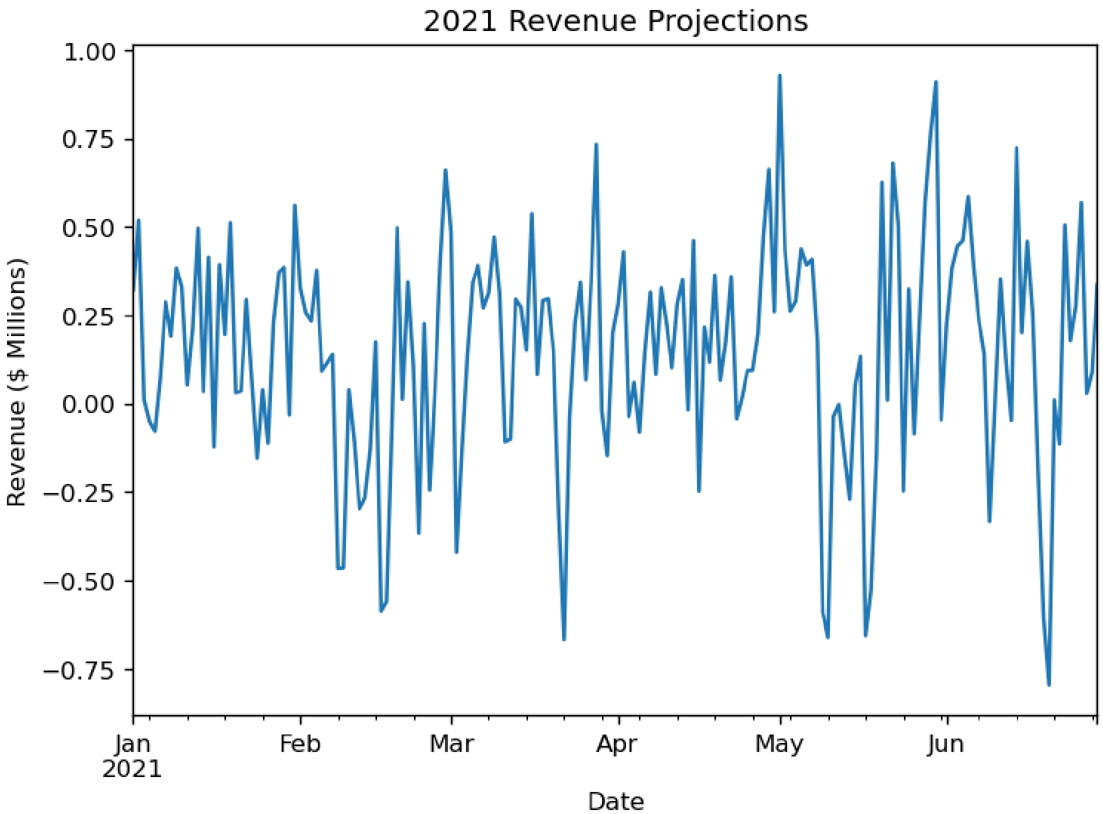
[47]:
```
DatetimeIndex(['2021-01-01', '2021-01-02', '2021-01-03', '2021-01-04',
               '2021-01-05', '2021-01-06', '2021-01-07', '2021-01-08',
               '2021-01-09', '2021-01-10',
               …
               '2021-06-20', '2021-06-21', '2021-06-22', '2021-06-23',
               '2021-06-24', '2021-06-25', '2021-06-26', '2021-06-27',
               '2021-06-28', '2021-06-29'],
              dtype='datetime64[ns]', length=180, freq='D')
```

[48]:
```
# Plot forecast

plt.plot(np.array(df_stationary.index), np.array(df_stationary[['Revenue']].
 ↪reset_index(drop=True)), label='Observed')
```

```
# shade conf. limit area
plt.fill_between(lower_limits.index, lower_limits, upper_limits, color='green')

# Plot mean predictions
plt.plot(mean_forecast.index, mean_forecast, color='red', label='Forecast')

# Labels and Legends
plt.title('Forecasted 2021 Revenue Projections')
plt.xlabel('Date')
plt.ylabel('Revenue ($ Millions)')
plt.legend()
plt.show()
```



[49]:
```
# Mean Forecast Plot
plt.title('2021 Revenue Projections')
plt.xlabel('Date')
plt.ylabel('Revenue ($ Millions)')
mean_forecast.plot();
```



[50]:
```
# Save model
joblib.dump(model, "time_series_model.pkl")
```

[50]: ['time_series_model.pkl']

3 Terminal: nbconvert –to pdf D213_PA1.ipynb

[ ]:
```
# Export cleaned_df
#pd.DataFrame(clean_df).to_csv("cleaned_df.csv")
```

3.1 End