D213 - Advanced Data Analytics - PA1

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?

Import Libraries

In [2]:
```
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: 4.05 µs
```

In [3]:
```
#%lsmagic
```

Load Data From medical_time_series.csv

In [4]:
```python
# load data file
initial_df = pd.read_csv('medical_time_series.csv', index_col='Day', parse_
# quick test the data is present and see the shape
print("df shape: ", initial_df.shape)
initial_df.head()
```

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

Out[4]:

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

Exploratory Data Analysis

In [5]:
```python
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
```

In [6]:
```
initial_df.describe()
```

Out[6]:

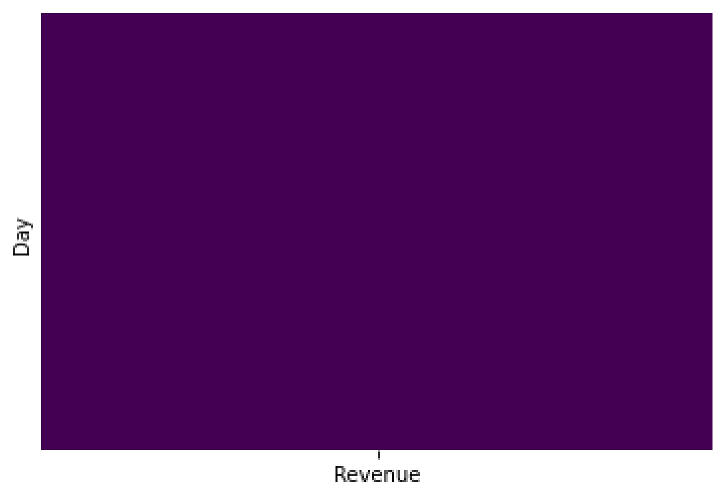
	Revenue
count	731.000000
mean	14.179608
std	6.959905
min	-4.423299
25%	11.121742
50%	15.951830
75%	19.293506
max	24.792249

In [7]:
```
# Any Null Values?
initial_df.isnull().any()
```

Out[7]:
```
Revenue    False
dtype: bool
```

Check for Missing Values

In [8]:
```
# Mapping to view missing data...none present.
sns.heatmap(initial_df.isnull(), yticklabels=False, cbar=False, cmap='virid
```



In [9]:
```
initial_df.columns
```

Out[9]:
```
Index(['Revenue'], dtype='object')
```

In [10]:
```
# 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
```

Out[10]:

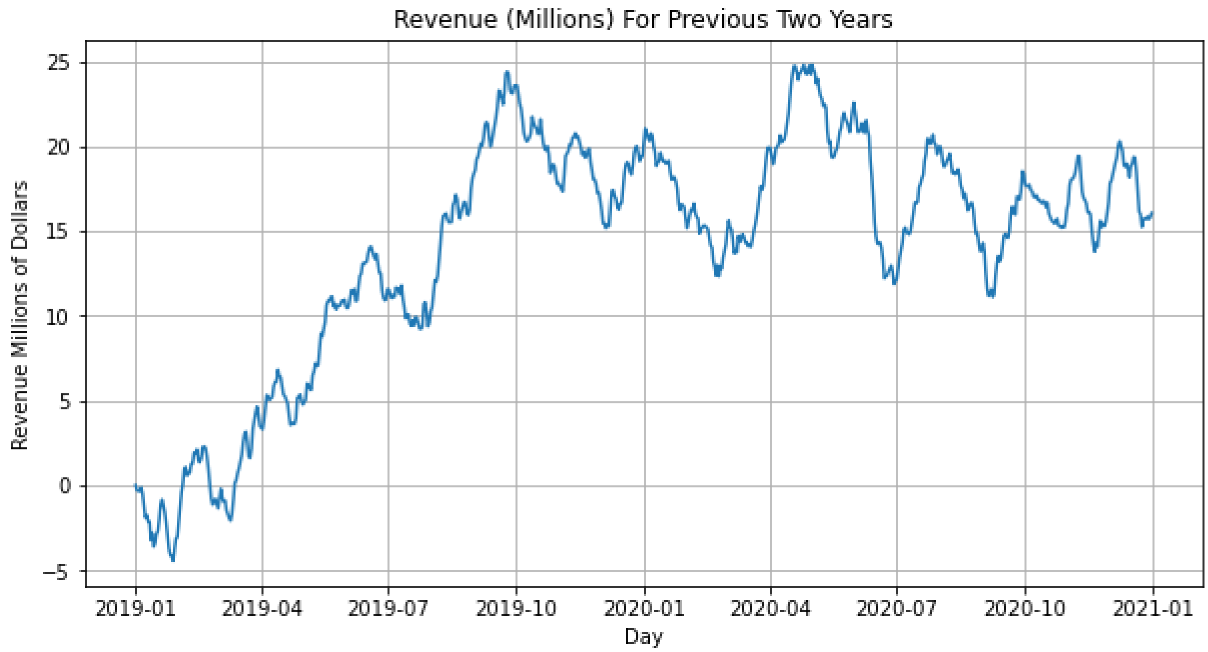
Date	Revenue
2019-01-01	0.000000
2019-01-02	-0.292356
2019-01-03	-0.327772
2019-01-04	-0.339987
2019-01-05	-0.124888
...	...
2020-12-27	15.722056
2020-12-28	15.865822
2020-12-29	15.708988
2020-12-30	15.822867
2020-12-31	16.069429

731 rows × 1 columns

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

In [11]:

```
#https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=efceba6c-e8ef-
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()
```



In [12]:

```
# Drop any null columns
df = initial_df.dropna()
df
```

Out[12]:

	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 × 1 columns

In [13]:

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

C3 - Make Time Series Stationary

In [14]:

```
# 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}
```

In [15]:
```
# Accept or reject null hypothesis
if result[1] <= 0.05: #Compare result against threshold
    print("Time series data is stationary.")
else:
    print("Time series data is non-stationary!")
```

```
Time series data is non-stationary!
```

In [16]:
```
# Make time series stationary
df_stationary = df.diff().dropna()

# View
df_stationary.head()
```

Out[16]:

Date	Revenue
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

In [17]:
```
# 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("Time series data is stationary.")
else:
    print("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}
Time series data is stationary.
```

Train, Test, and Split

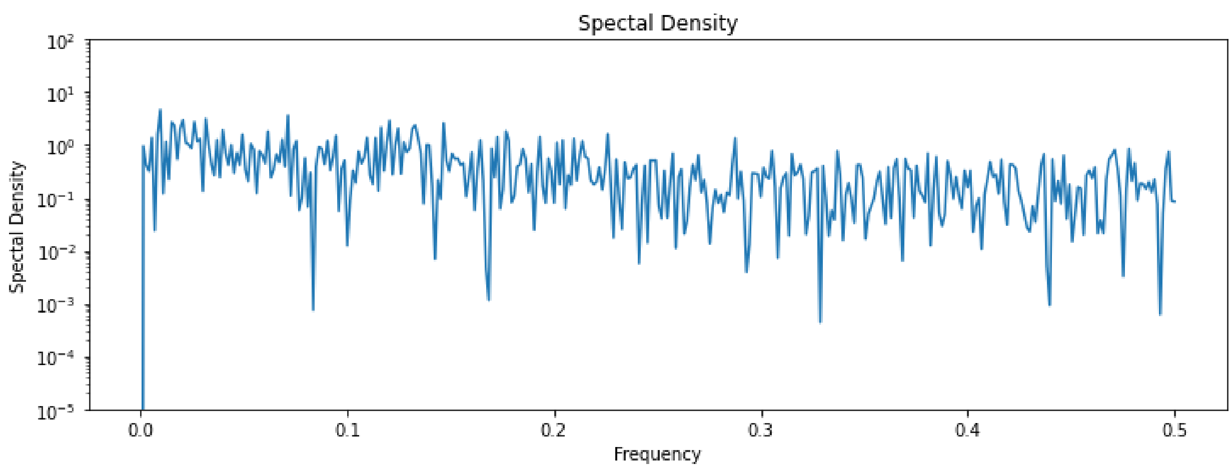
In [18]:
```
# Split for Training and Testing

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

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

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

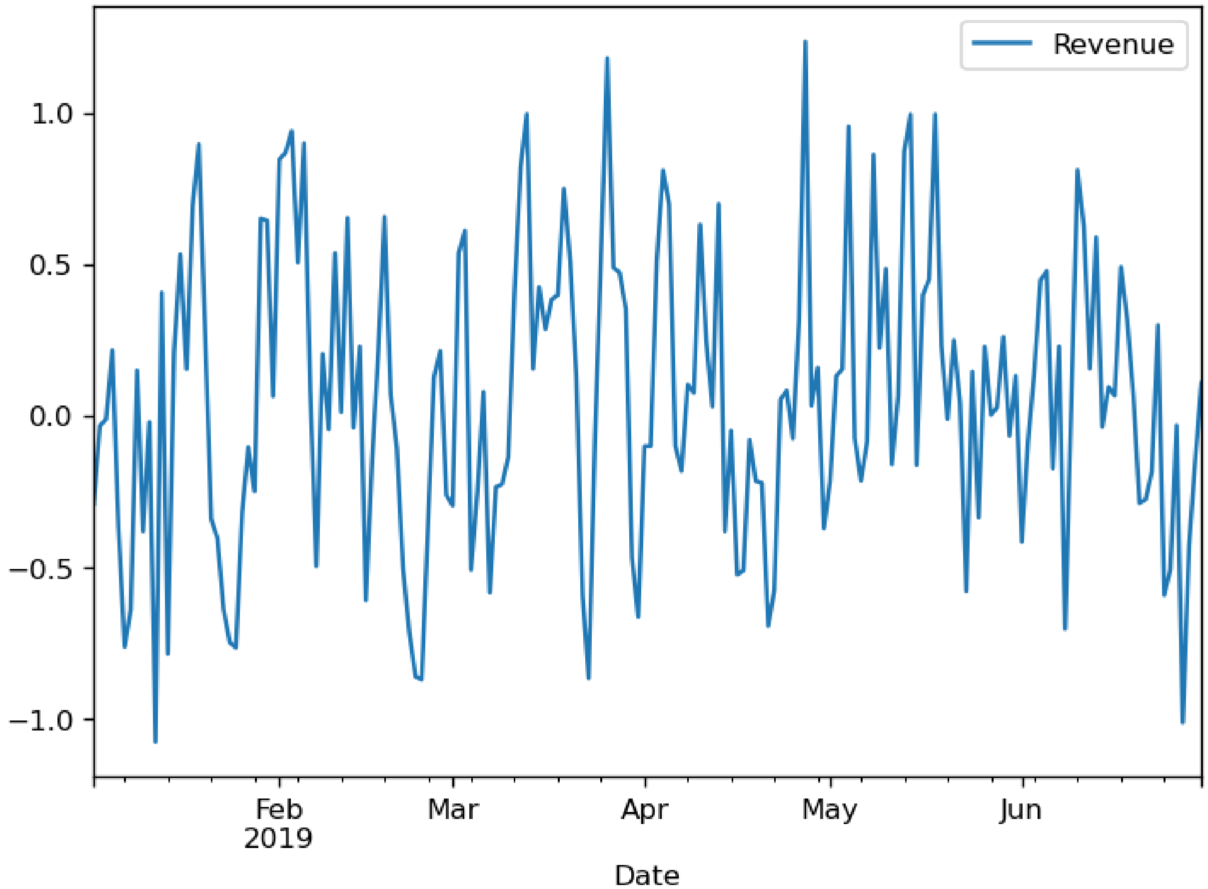
C5 - Prepared Dataset

In [19]:
```
# Export stationary data
pd.DataFrame(df_stationary).to_csv("df_cleaned_stationary.csv")
```

In [25]:
```
# Spectal Density

f, Pxx_den=signal.periodogram(df_stationary['Revenue'])
plt.figure(figsize=(12,4))
plt.semilogy(f,Pxx_den)
plt.ylim([1e-5,1e2])
plt.title('Spectal Density')
plt.xlabel('Frequency')
plt.ylabel('Spectal Density')
plt.show()
```

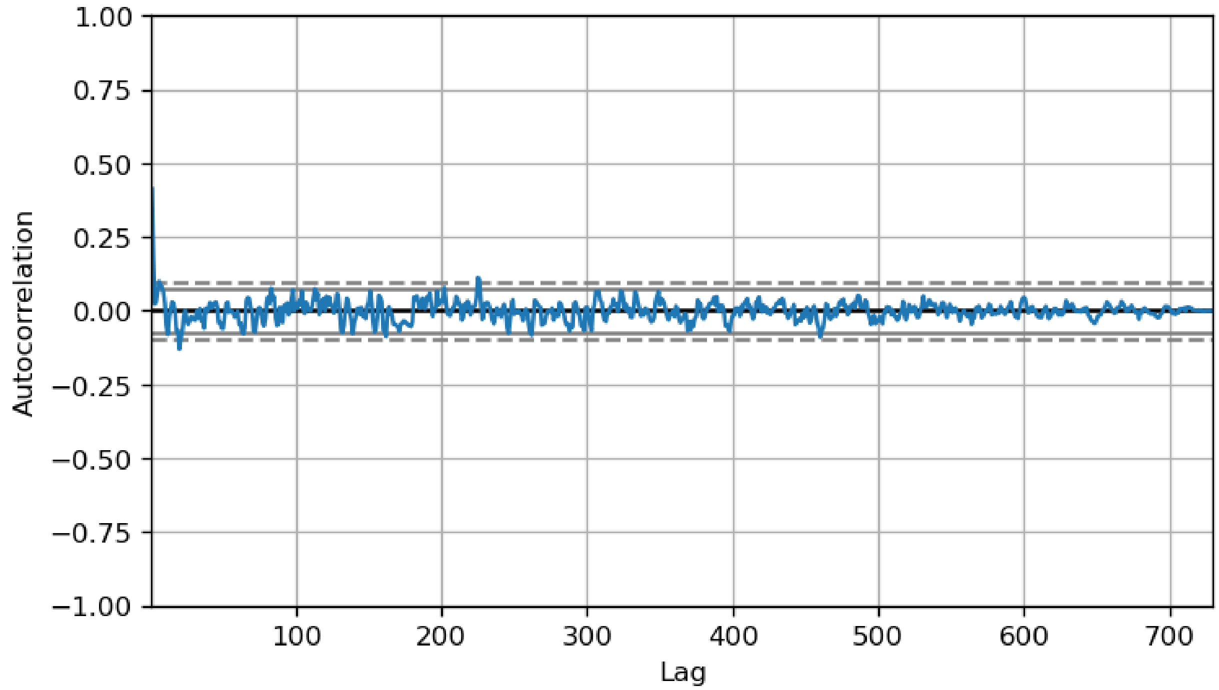


In [32]:
```
# Some seasonality visible in data
df_stationary.loc[:'2019-06-30'].plot()
plt.figure(figsize=(12,4))
plt.show();
```

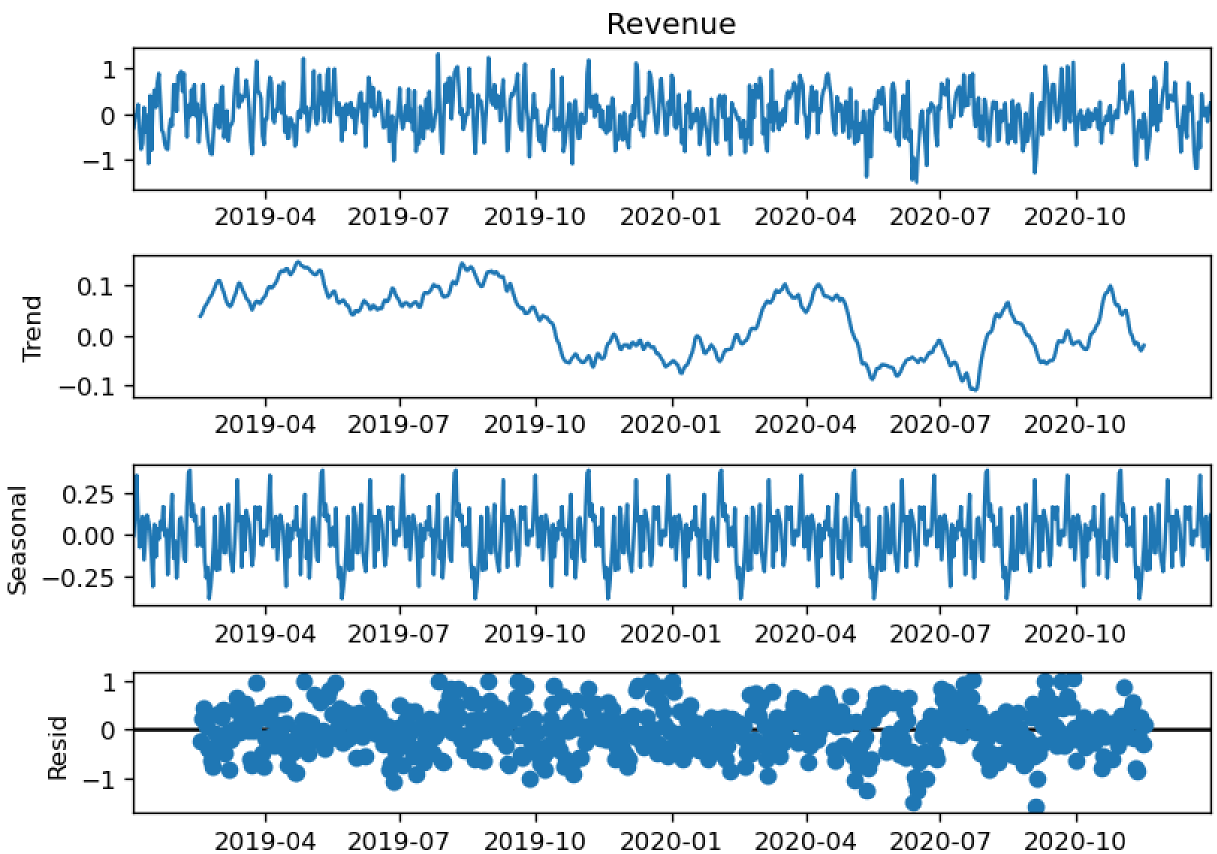



```
<Figure size 1440x480 with 0 Axes>
```

In [37]:
```
# Continue looking for seasonality
plt.rcParams.update({'figure.figsize':(7,4), 'figure.dpi':120})

autocorrelation_plot(df_stationary.Revenue.tolist());
```

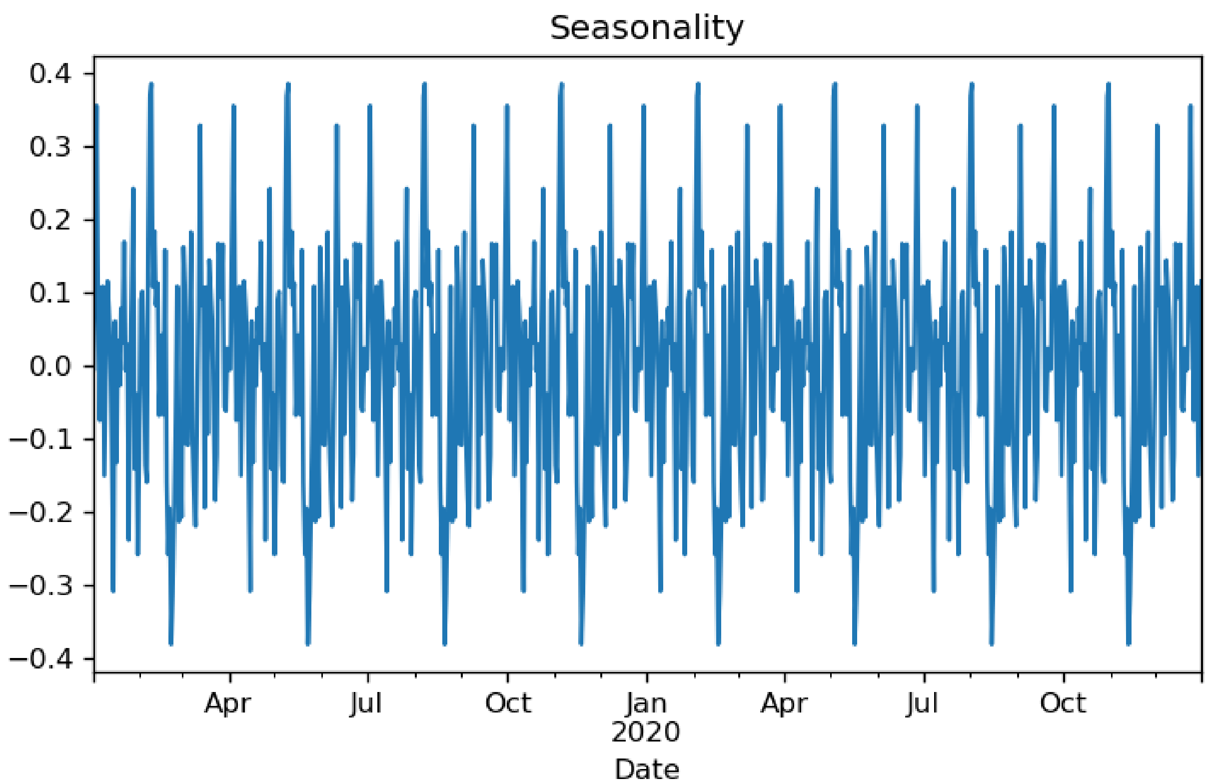


In [31]:
```
# Decomposition
decomp = seasonal_decompose(df_stationary['Revenue'],period=90)

# Plot decomposition
decomp.plot()
plt.figure(figsize=(12,4))
plt.show()
```



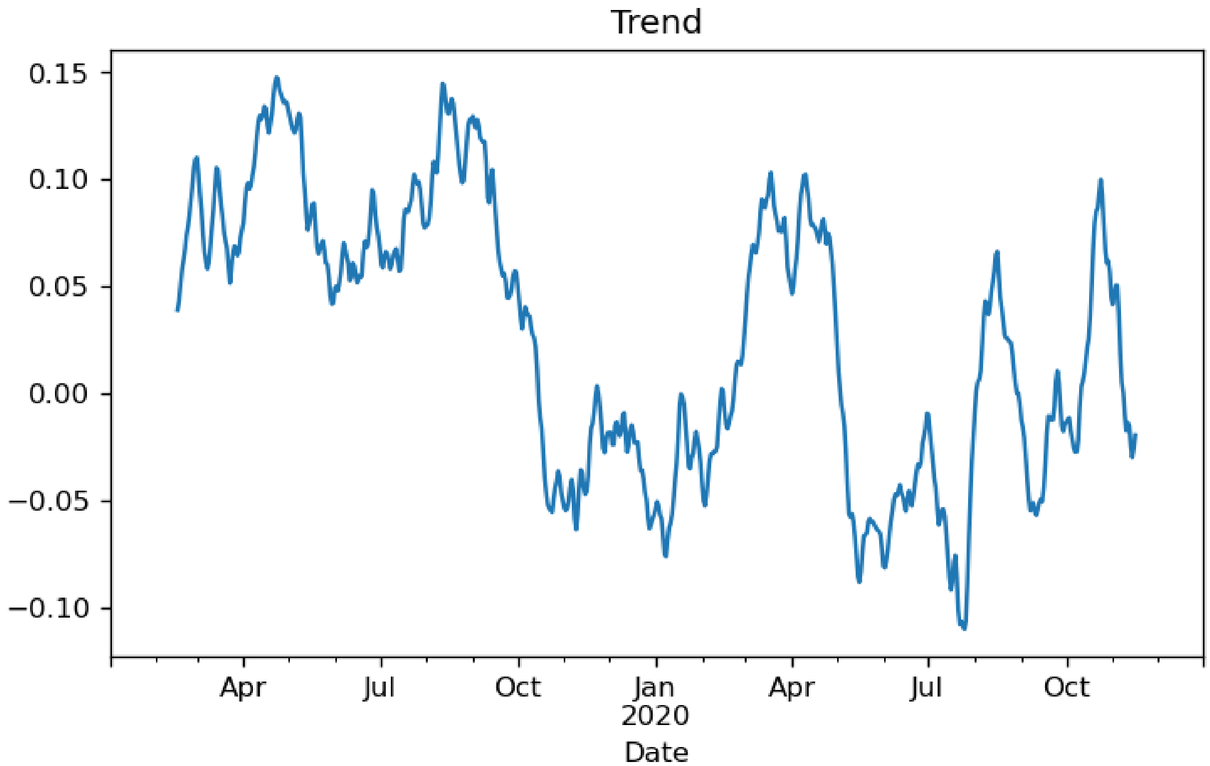

```
<Figure size 1440x480 with 0 Axes>
```

In [38]:
```
# Plot Seasonality

plt.title('Seasonality')
decomp.seasonal.plot()
plt.figure(figsize=(12,4))
plt.show();
```



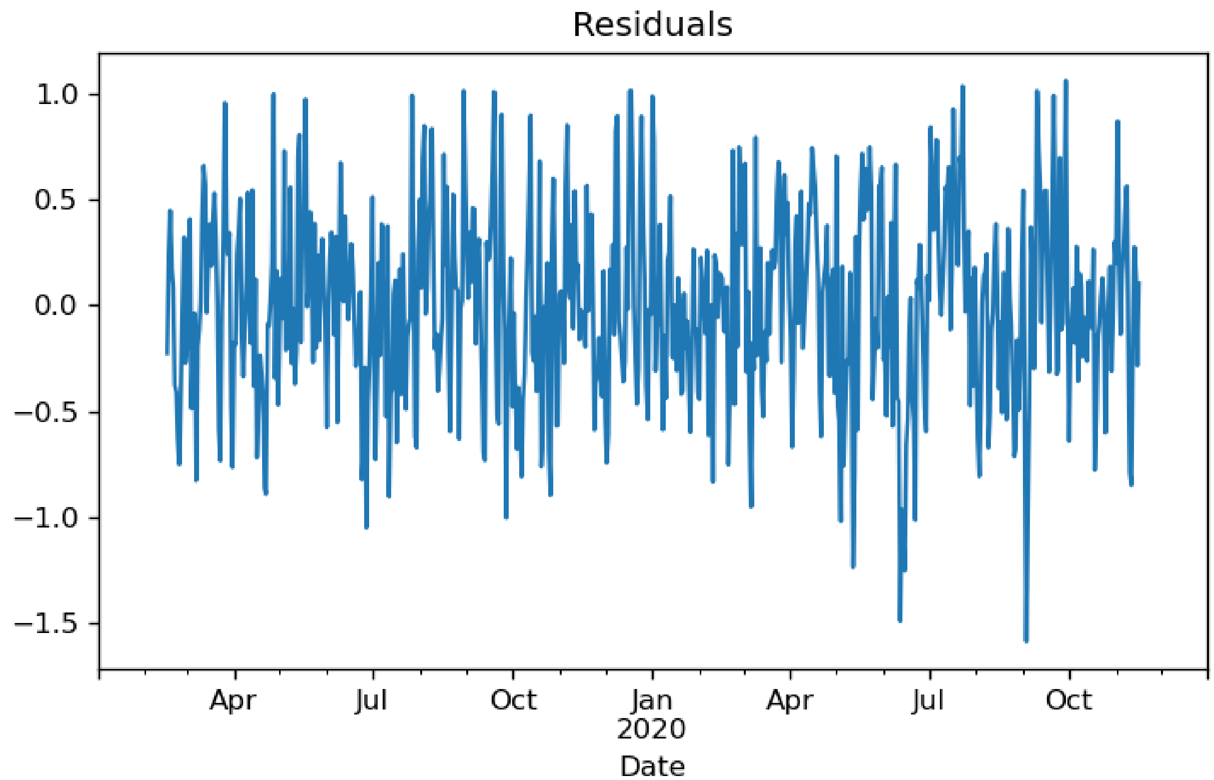

```
<Figure size 1440x480 with 0 Axes>
```

In [39]:
```
# View Trend
plt.title('Trend')
decomp.trend.plot()
plt.figure(figsize=(12,4))
plt.show();
```




```
<Figure size 1440x480 with 0 Axes>
```

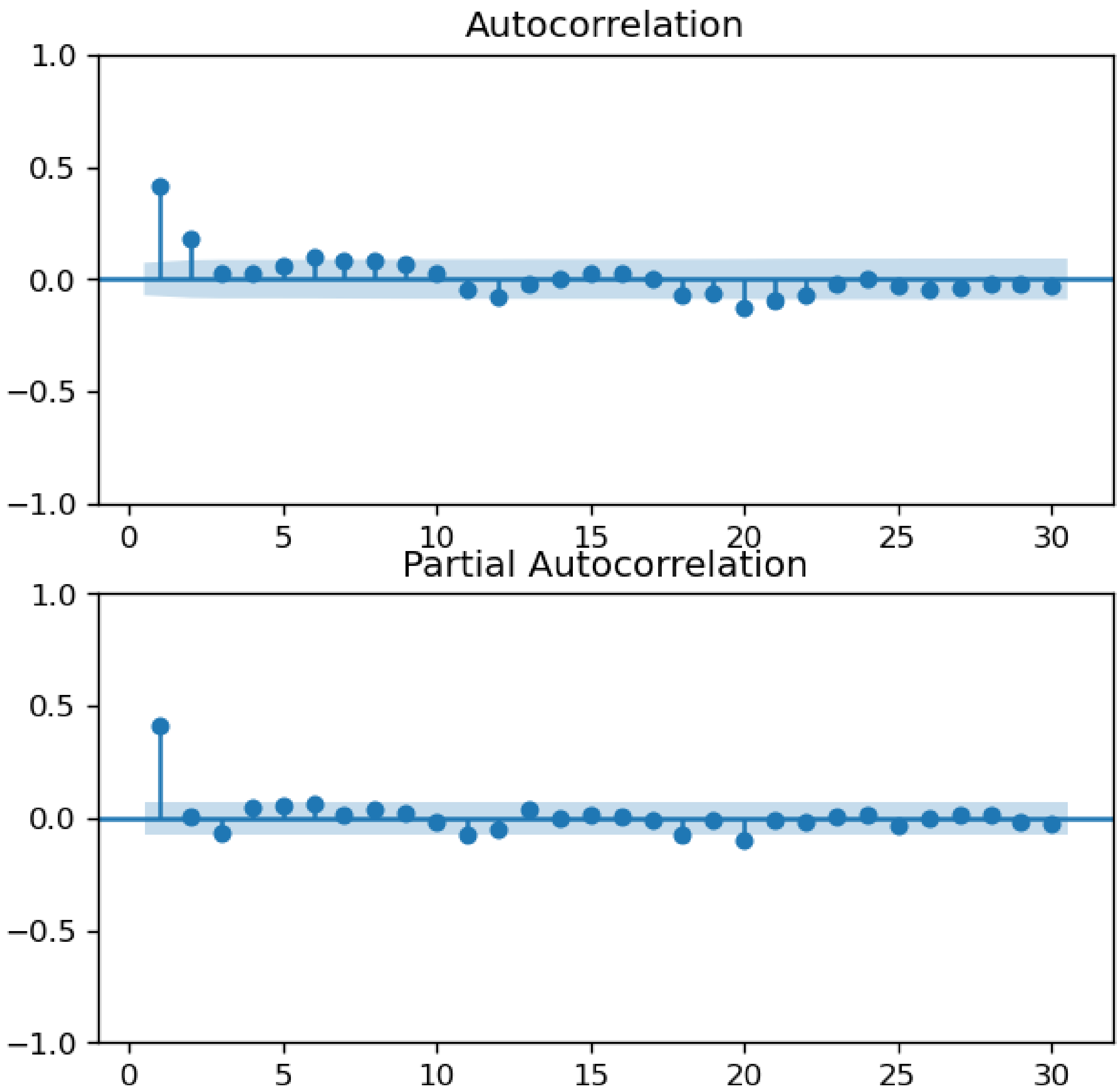
In [40]:
```
# Plot Residual
plt.title('Residuals')
decomp.resid.plot()
plt.figure(figsize=(12,4))
plt.show();
```




```
<Figure size 1440x480 with 0 Axes>
```

In [41]:

```
# ACF and PACF Autocorrelation Plots

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

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

# Plot df PACF
plot_pacf(df_stationary, lags=30, zero=False, ax=ax2)
plt.figure(figsize=(12,4));
plt.show();
```



<Figure size 1440x480 with 0 Axes>

In [42]:
```python
# 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
    for q in rng: #loop over q
        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: # value swap
                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))
```

```
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
```

Auto ARIMA; Takes > 120 min

In [43]:
```python
# Use Auto ARIMA to Find best model
# https://www.machinelearningplus.com/time-series/arima-model-time-series-f

%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, max_q=2,
                      max_P=2, max_Q=2,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True)
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 5.01 µs
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(1,1,1)[90]             : AIC=inf, Time=741.05 sec
 ARIMA(0,1,0)(0,1,0)[90]             : AIC=1448.607, Time=9.03 sec
 ARIMA(1,1,0)(1,1,0)[90]             : AIC=inf, Time=64.66 sec
 ARIMA(0,1,1)(0,1,1)[90]             : AIC=inf, Time=399.31 sec
 ARIMA(0,1,0)(1,1,0)[90]             : AIC=inf, Time=39.63 sec
 ARIMA(0,1,0)(0,1,1)[90]             : AIC=inf, Time=178.11 sec
 ARIMA(0,1,0)(1,1,1)[90]             : AIC=inf, Time=347.94 sec
 ARIMA(1,1,0)(0,1,0)[90]             : AIC=1390.122, Time=12.13 sec
 ARIMA(1,1,0)(0,1,1)[90]             : AIC=inf, Time=374.84 sec
 ARIMA(1,1,0)(1,1,1)[90]             : AIC=inf, Time=415.34 sec
 ARIMA(2,1,0)(0,1,0)[90]             : AIC=1366.083, Time=11.46 sec
 ARIMA(2,1,0)(1,1,0)[90]             : AIC=inf, Time=55.43 sec
 ARIMA(2,1,0)(0,1,1)[90]             : AIC=inf, Time=385.00 sec
 ARIMA(2,1,0)(1,1,1)[90]             : AIC=inf, Time=634.66 sec
 ARIMA(2,1,1)(0,1,0)[90]             : AIC=inf, Time=201.70 sec
 ARIMA(1,1,1)(0,1,0)[90]             : AIC=inf, Time=174.64 sec
 ARIMA(2,1,0)(0,1,0)[90] intercept   : AIC=1368.083, Time=28.25 sec

Best model:  ARIMA(2,1,0)(0,1,0)[90]
Total fit time: 4073.193 seconds
```

In [44]:
```
print(model.summary())
```

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                  y   No. Observations:
730
Model:             SARIMAX(2, 1, 0)x(0, 1, 0, 90)   Log Likelihood
-680.041
Date:                            Sat, 23 Jul 2022   AIC
1366.083
Time:                                    18:39:09   BIC
1379.462
Sample:                                         0   HQIC
1371.276
                                            - 730
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
ar.L1         -0.3605      0.039     -9.183      0.000      -0.437      -
0.284
ar.L2         -0.1998      0.040     -4.939      0.000      -0.279      -
0.121
sigma2         0.4918      0.029     17.237      0.000       0.436
0.548
===================================================================================
Ljung-Box (L1) (Q):                   1.28   Jarque-Bera (JB):
0.82
Prob(Q):                              0.26   Prob(JB):
0.66
Heteroskedasticity (H):               1.05   Skew:
0.06
Prob(H) (two-sided):                  0.74   Kurtosis:
2.88
===================================================================================

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

In [2]:
```
# Create Time Series Model

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

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [2], in <cell line: 3>()
      1 # Create Time Series Model
----> 3 model = SARIMAX(df_stationary, order=(1,1,0),seasonal_order=(1,1,0,90))
      4 results = model.fit()
      5 results.summary()

NameError: name 'SARIMAX' is not defined
```

In [3]:
```
print(results.summary())
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
Input In [3], in <cell line: 1>()
----> 1 print(results.summary())

NameError: name 'results' is not defined
```

In [49]:
```
print(results.params)
```

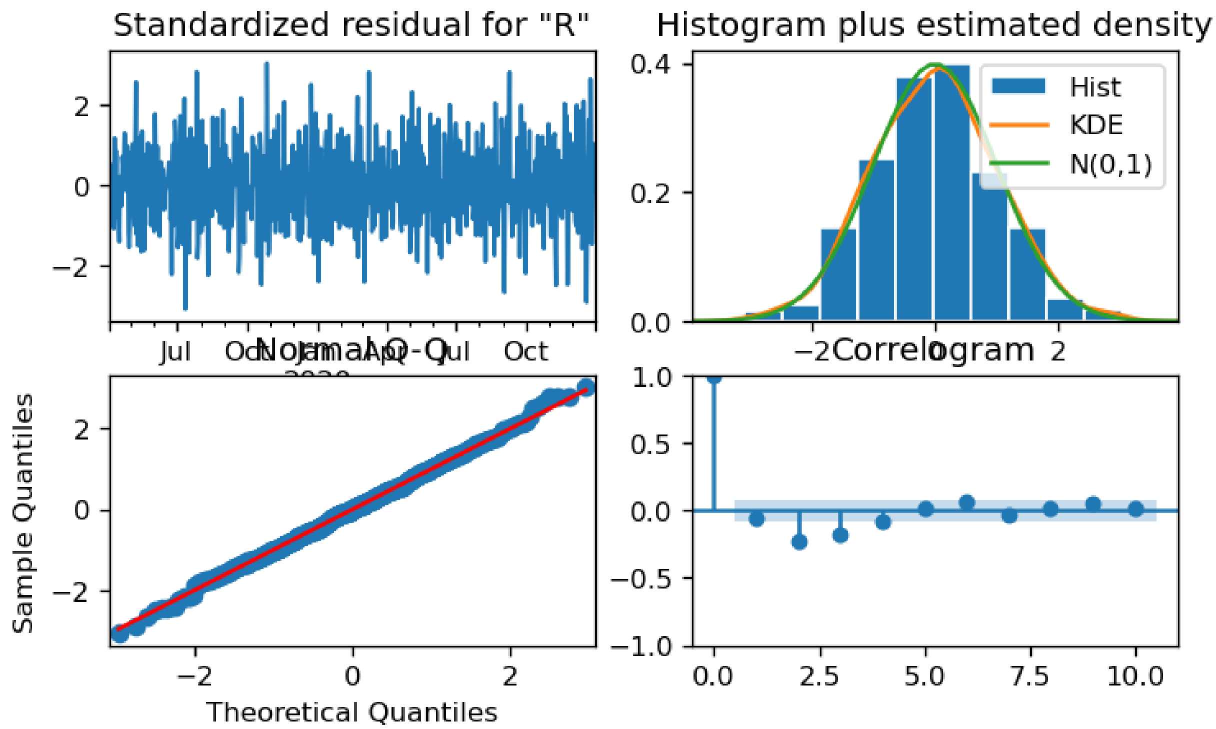
```
ar.L1      -0.308376
ar.S.L90   -0.472576
sigma2      0.395774
dtype: float64
```

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

In [51]:
```
# Print mean absolute error
mae = np.mean(np.abs(results.resid))
print("Mean Absolute Error: ", mae)
```

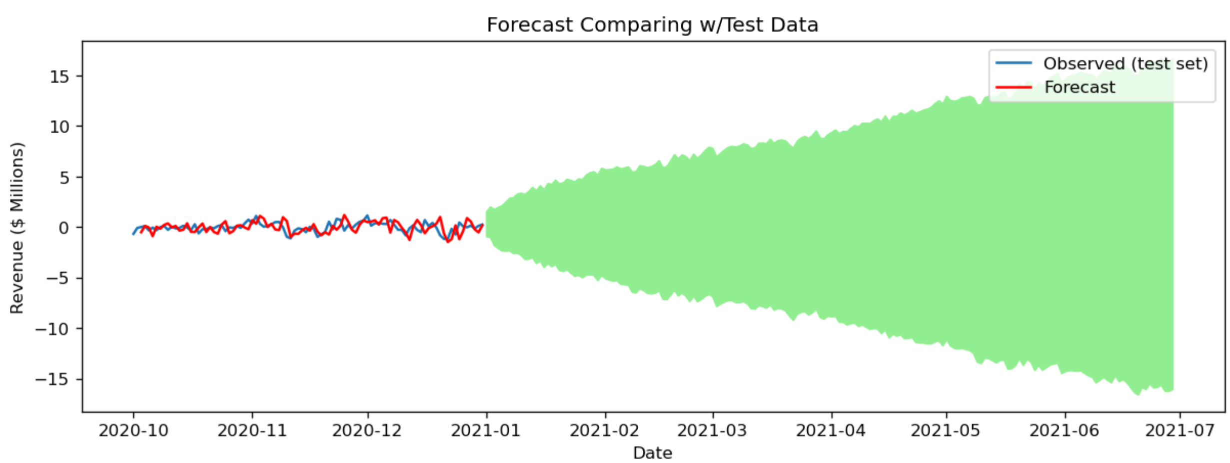
```
Mean Absolute Error:  0.49873094599791684
```

In [52]:
```
# Create the 4 diagnostics plots
results.plot_diagnostics().show()
```



In [53]:
```
# Validate w/Test Set

# 90 day prediction range
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)
```

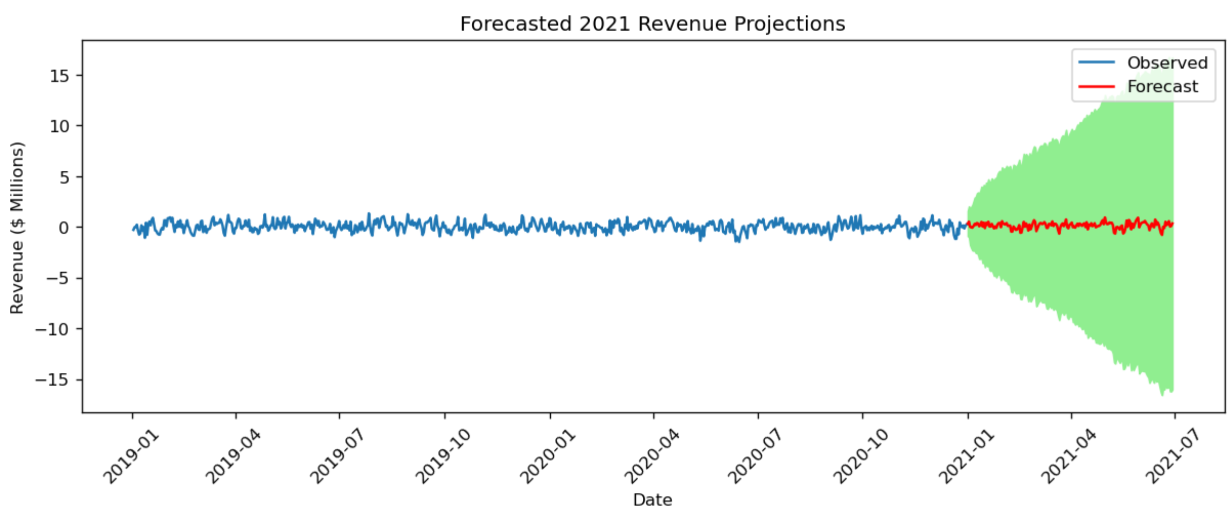
In [65]:
```
# Plot Data
plt.figure(figsize=(12,4))
#plt.plot(X_test.index, X_test, label='Observed X_test')
plt.plot(np.array(X_test.index), np.array(X_test[['Revenue']]), label='Obse

# plot your mean predictions
plt.plot(mean_prediction.index, mean_prediction, color='r', label='Forecast

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

## plot mean predictions
#plt.fill_between(mean_prediction.index, mean_prediction, color='brown', la

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



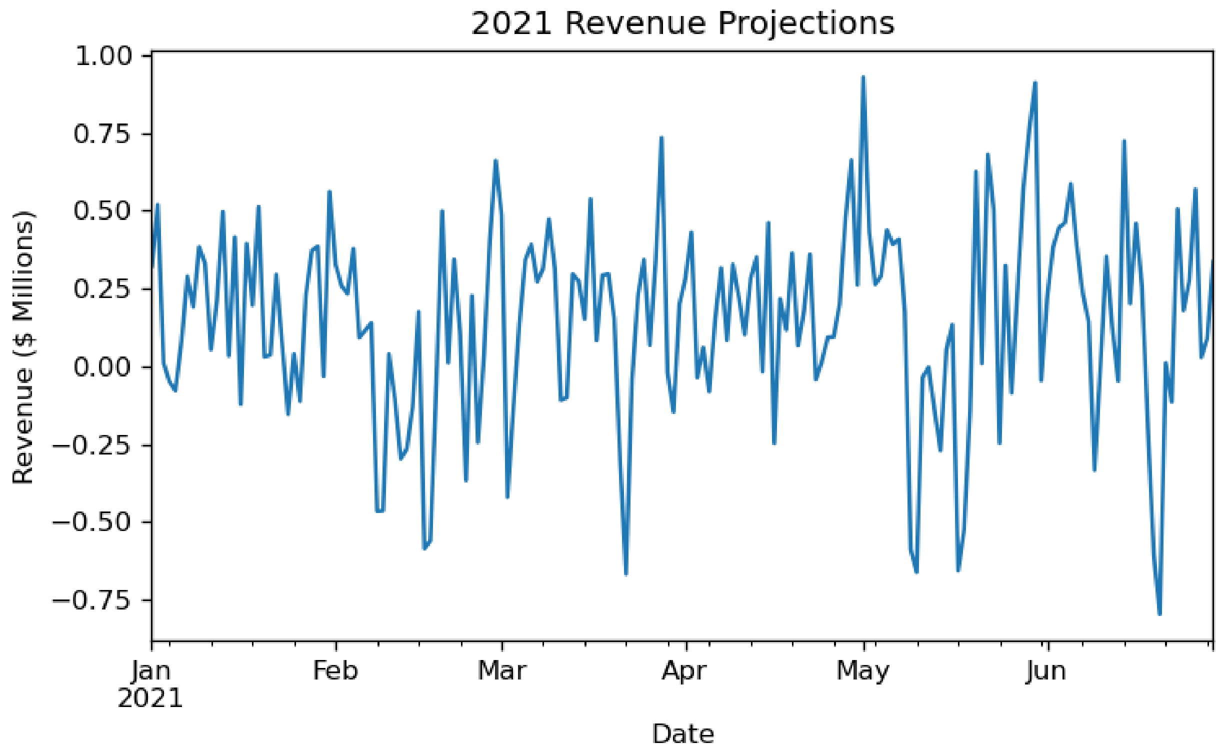
In [59]:
```
# Perform forecast
diff_forecast = results.get_forecast(steps=180)
mean_forecast = diff_forecast.predicted_mean

# 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']
```

In [69]:

```
# Plot forecast
plt.figure(figsize=(12,4))
#plt.plot(df_stationary.index, df_stationary, label='Observed')
plt.plot(np.array(df_stationary.index), np.array(df_stationary[['Revenue']]

# Plot mean predictions

plt.plot(mean_forecast.index, mean_forecast, color='r', label='Forecast')

# shade conf. limit area
#plt.fill_between(lower_limits.index, lower_limits, upper_limits, color='pi
plt.fill_between(upper_limits.index, upper_limits, lower_limits, color='lig



# Annotations: Labels and Legends
plt.title('Forecasted 2021 Revenue Projections')
plt.xlabel('Date')
plt.ylabel('Revenue ($ Millions)')
plt.xticks(rotation=45)
plt.legend()
plt.show()
```



In [63]:
```
# Mean Forecast Plot
plt.title('2021 Revenue Projections')
plt.xlabel('Date')
plt.ylabel('Revenue ($ Millions)')
mean_forecast.plot();
```



In [64]:
```
# Save model
joblib.dump(model, "time_series_model.pkl")
```

Out[64]:
```
['time_series_model.pkl']
```

Terminal: nbconvert --to pdf D213_PA1.ipynb

End