D213-AdvancedDataAnalyticsPA1_v2

August 1, 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

[4]:
```
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.arima.model import ARIMA
from statsmodels.tsa.statespace.sarimax import SARIMAX
import pmdarima as pm
from pmdarima import auto_arima
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')
# import warnings filter
from warnings import simplefilter
# ignore all future warnings
simplefilter(action='ignore', category=FutureWarning)
#!pip install joblib
```

```
import joblib
%matplotlib inline
%time
%timeit
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 3.1 µs
```

[5]:
```
#%lsmagic
```

0.0.4 Load Data From medical_time_series.csv

[6]:
```
# 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)
```

[6]:

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

[7]:
```
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
```

[8]:
```
initial_df.describe()
```

[8]:

	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
```

[9]:
```
# Any Null Values?
initial_df.isnull().any()
```

[9]:
```
Revenue    False
dtype: bool
```

0.0.5 Check for Missing Values

[10]:
```
# Mapping to view missing data...none present.
sns.heatmap(initial_df.isnull(), yticklabels=False, cbar=False, cmap='viridis');
```

[11]:
```
initial_df.columns
```

[11]:
```
Index(['Revenue'], dtype='object')
```

[12]:
```
# 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
```

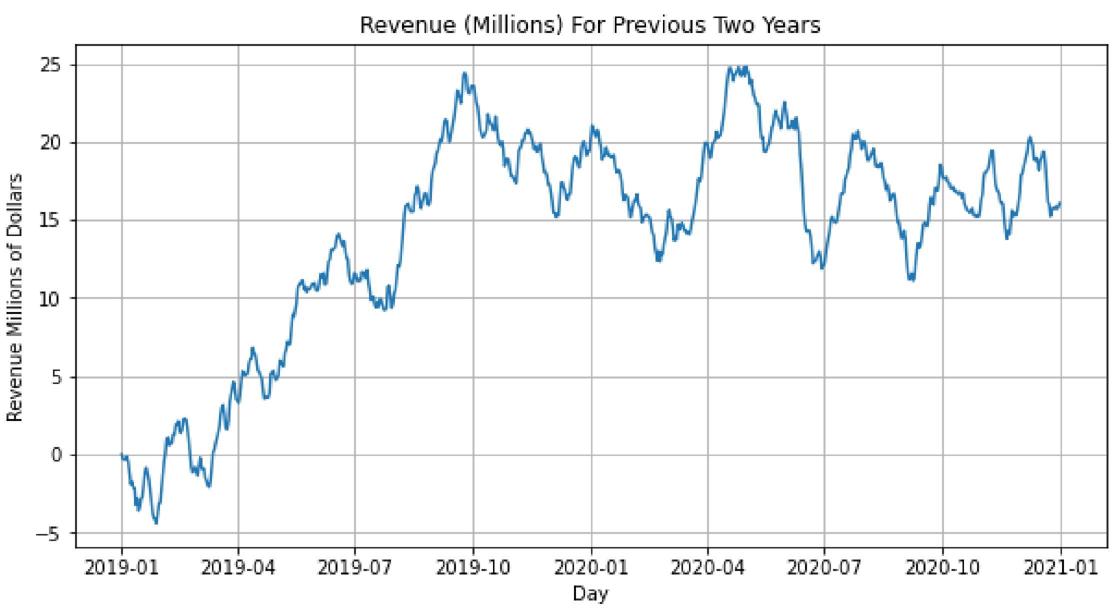
[12]:

```
             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 x 1 columns]
```

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

[13]:

```
#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()
```



[14]:
```
# Drop any null columns
df = initial_df.dropna()
df
```

[14]:
```
              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]
```

[15]:
```
# Export cleaned data
pd.DataFrame(df).to_csv("df_cleaned.csv")
```

0.2 C3 - Make Time Series Stationary

[16]:
```
# 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}
```

[17]:
```
# 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!
```

[18]:
```python
# Make time series stationary
df_stationary = df.diff().dropna()

# View
df_stationary.head()
```

[18]:
```
             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
```

[19]:
```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("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.
```

0.3 Train, Test, and Split

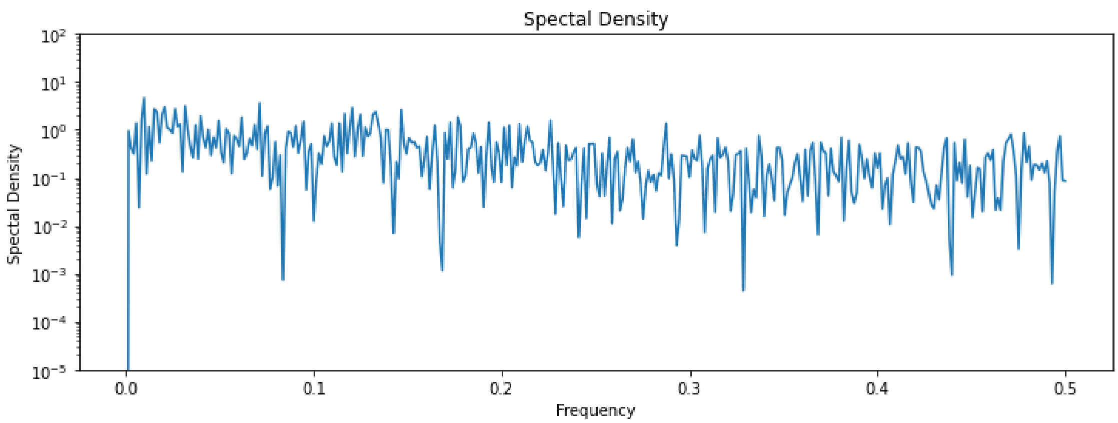
[20]:
```python
# Split for Training and Testing

X_train = df_stationary.loc[:'2020-09-30'] # Get all but the last 90 days for␣
↪training
X_test = df_stationary['2020-10-01':] # Get last 90 days of data to test

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)
```

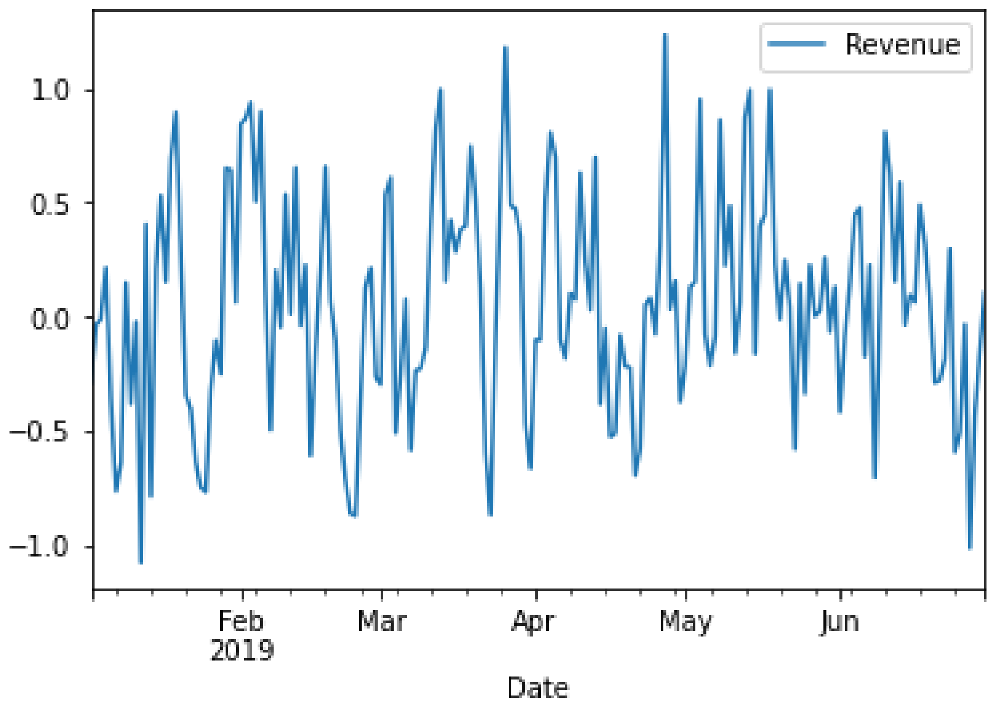
0.4 C5 - Prepared Dataset

[21]:
```
# Export stationary data
pd.DataFrame(df_stationary).to_csv("df_cleaned_stationary.csv")
```

[22]:
```
# 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()
```

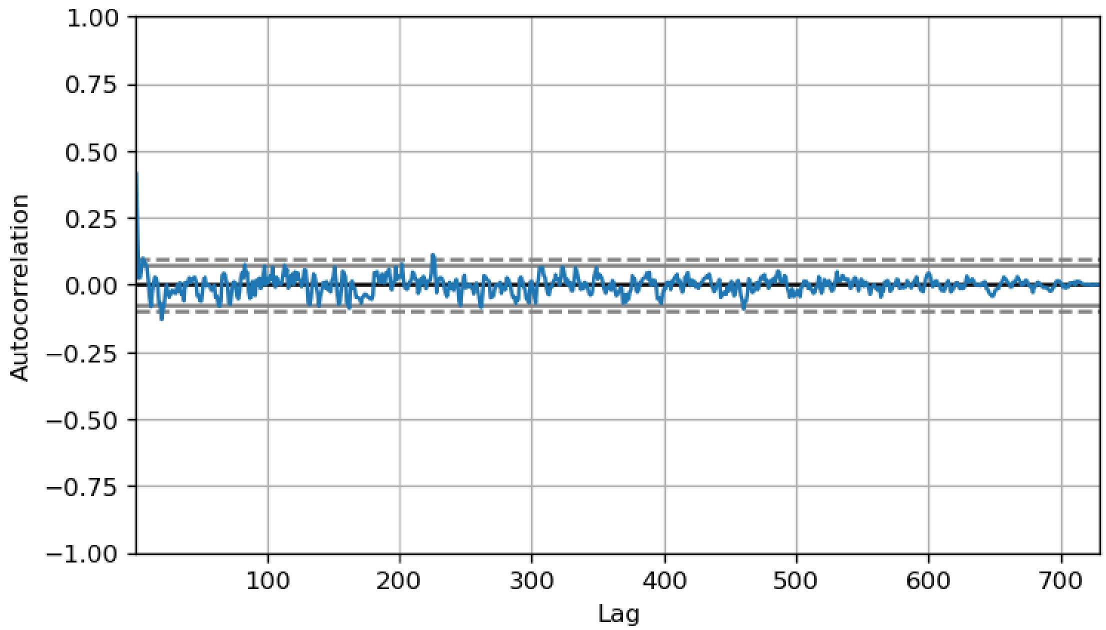


[23]:
```
# Some seasonality visible in data
df_stationary.loc[:'2019-06-30'].plot()
plt.figure(figsize=(12,4))
plt.show();
```

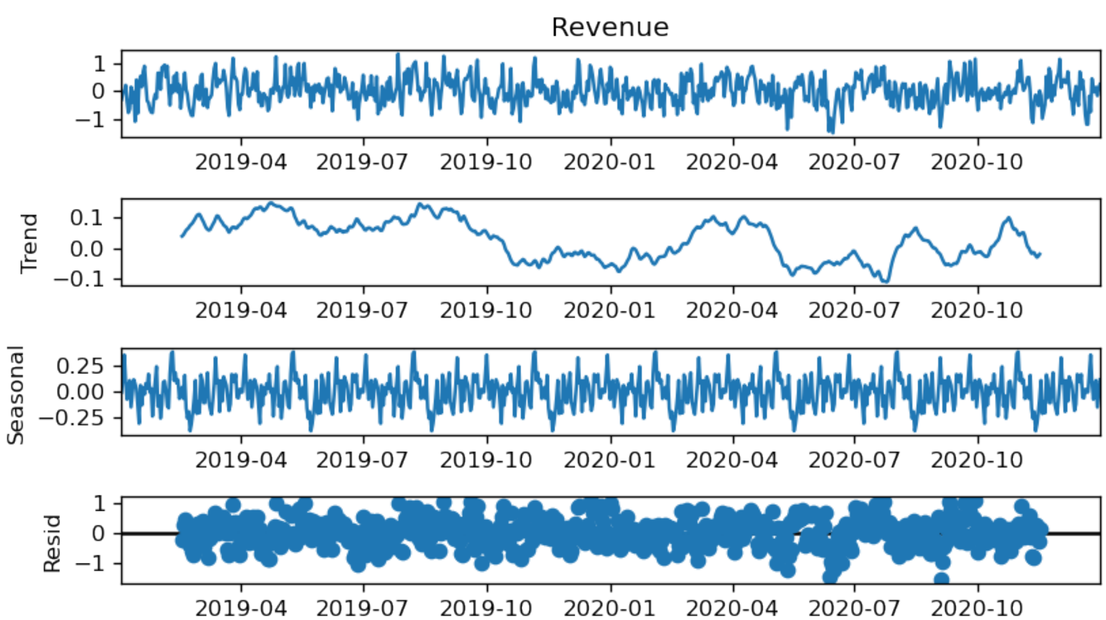



```
<Figure size 864x288 with 0 Axes>
```

[24]:
```
# Continue looking for seasonality
plt.rcParams.update({'figure.figsize':(7,4), 'figure.dpi':120})

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

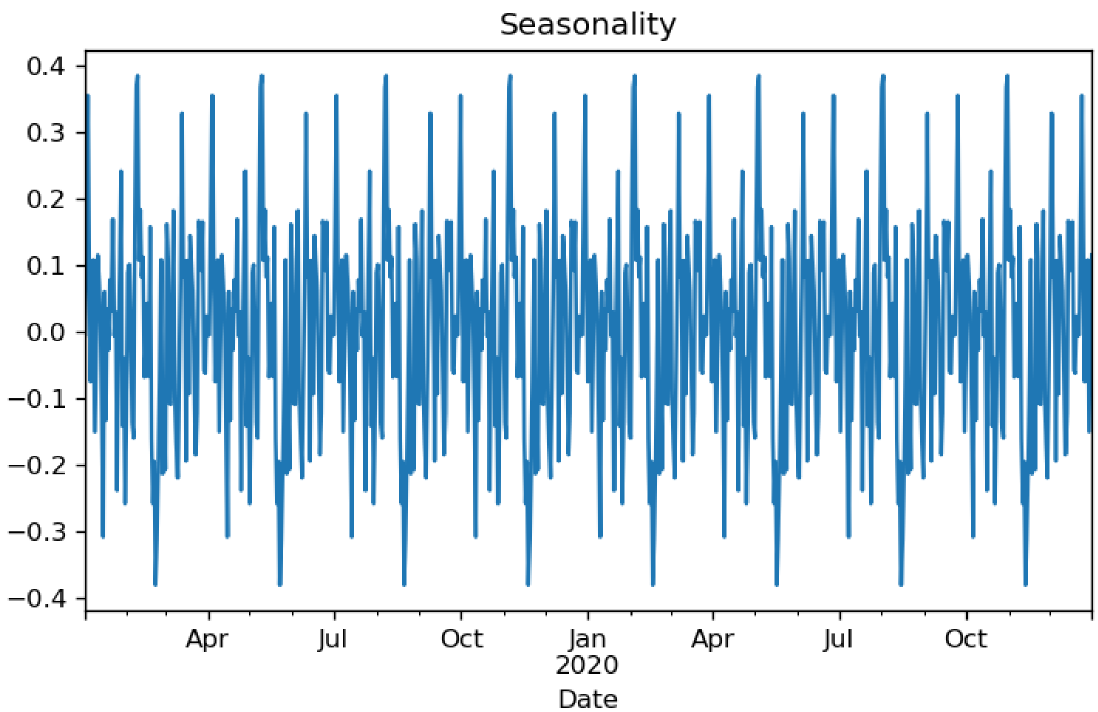


[25]:
```
# Decomposition
decomp = seasonal_decompose(df_stationary['Revenue'],period=90)

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



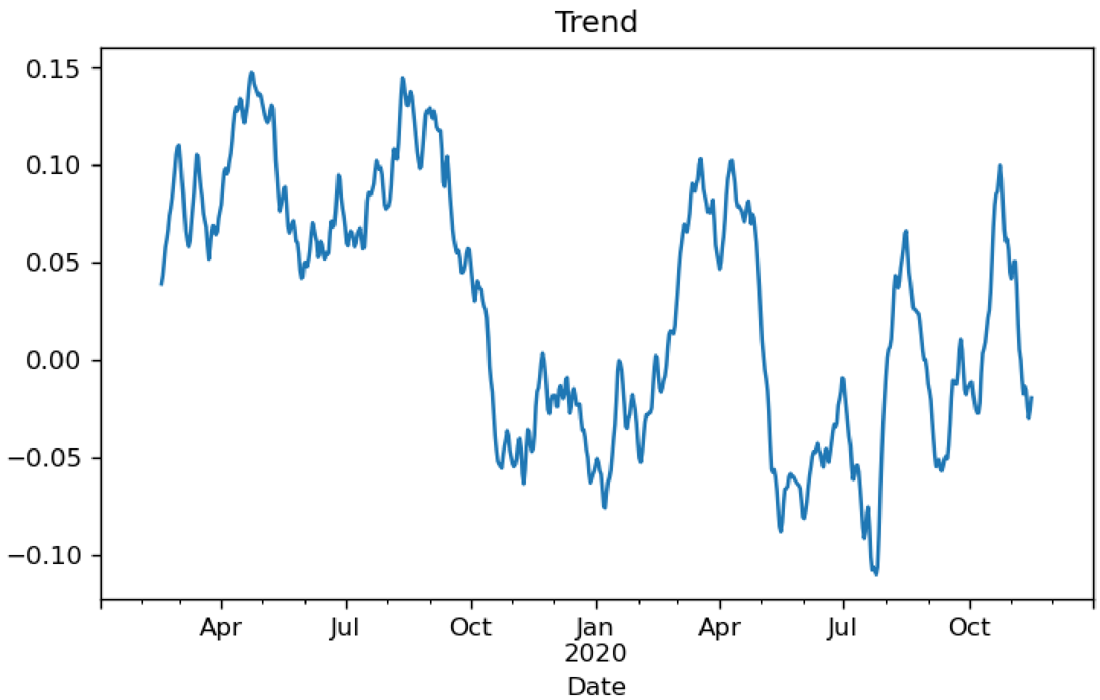
<Figure size 1440x480 with 0 Axes>

[26]:
```
# Plot Seasonality

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



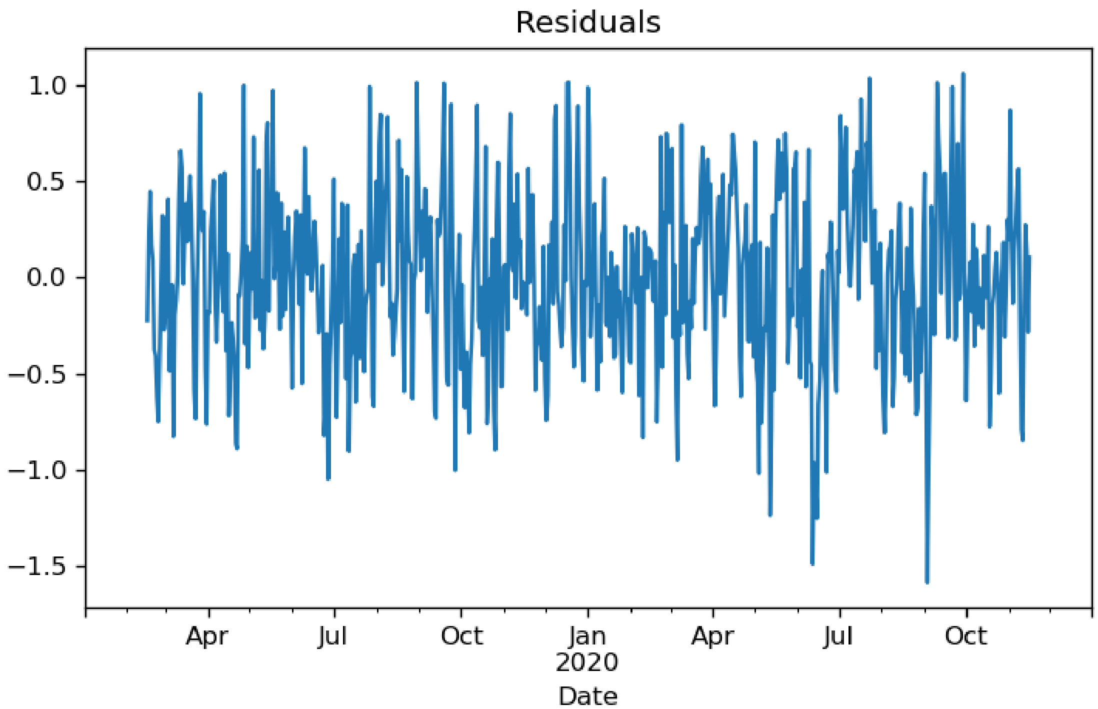
<Figure size 1440x480 with 0 Axes>

[27]:
```
# View Trend
plt.title('Trend')
decomp.trend.plot()
plt.figure(figsize=(12,4))
plt.show();
```



<Figure size 1440x480 with 0 Axes>

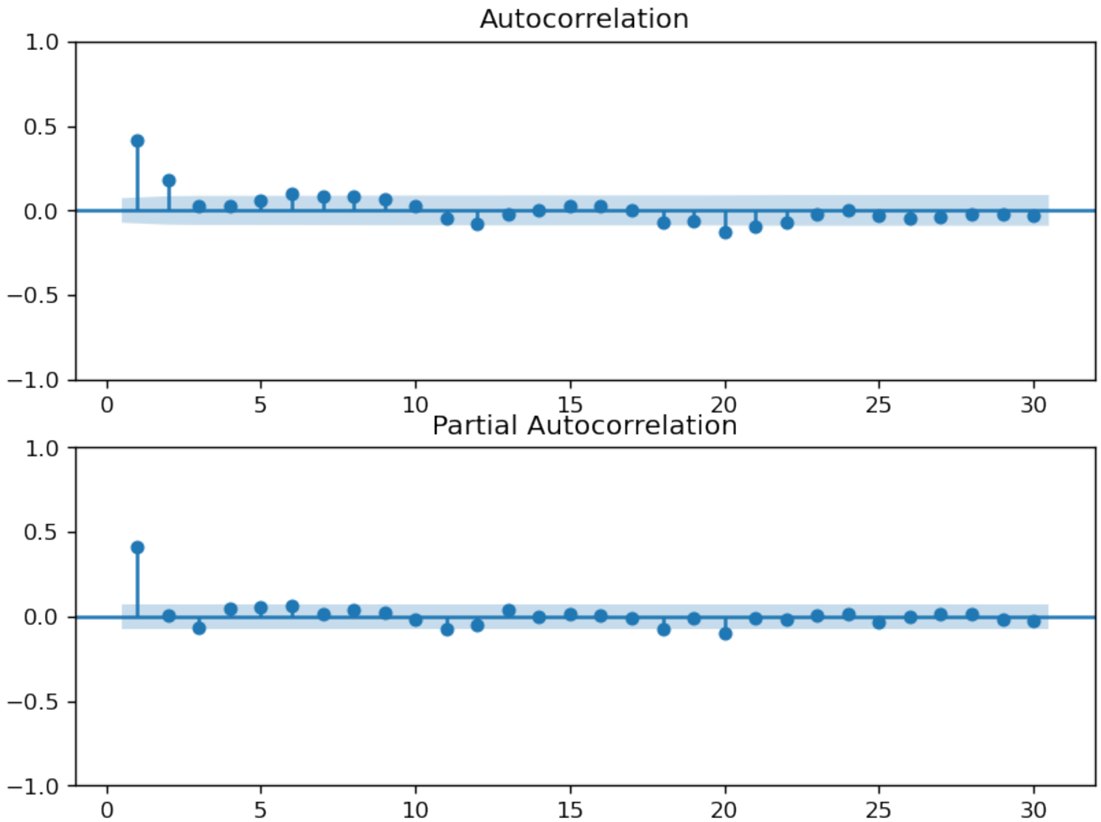
[28]:
```
# Plot Residual
plt.title('Residuals')
decomp.resid.plot()
plt.figure(figsize=(12,4))
plt.show();
```




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

[29]:

```python
# ACF and PACF Autocorrelation Plots

# fig size
fig, (ax1, ax2) = plt.subplots(2,1, figsize=(8,6));

# 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();
```



[30]:
```
# 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:
```

```python
            print(p,q, None, None)

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

```
 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.
```

```
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

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

 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.

           * * *

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

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
```

```
 This problem is unconstrained.
 This problem is unconstrained.
```

```
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

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

 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.

           * * *

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

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 Auto ARIMA; Takes > 120 min

[31]:
```
# Use Auto ARIMA to Find best model_1
# https://www.machinelearningplus.codf_stationary-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, max_q=2,
#                      max_P=2, max_Q=2,
#                      trace=True,
#                      error_action='ignore',
#                      suppress_warnings=True)
```

[32]:
```
#print(model.summary())
```

1.1 SARIMAX Model Using Actual Dataset

[33]:
```
# Create Time Series Model
%time
tqdm.pandas()

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

```
CPU times: user 1 µs, sys: 1e+03 ns, total: 2 µs
Wall time: 3.81 µs
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
```

[33]:
```
<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:                           Mon, 01 Aug 2022   AIC
1249.964
Time:                                   20:56:14   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).
"""
```

1.2 SARIMAX Model Using Training Dataset

[34]:
```python
# Create Time Series Model using Training Data
%time
tqdm.pandas()

s_model_train = SARIMAX(X_train, order=(1,1,0),seasonal_order=(1,1,0,90))
s_results_train = s_model_train.fit()
s_results_train.summary()
```

```
CPU times: user 2 µs, sys: 1e+03 ns, total: 3 µs
Wall time: 4.29 µs
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=  8.41037D-01    |proj g|=  2.84168D-01

 This problem is unconstrained.


At iterate    5    f=  8.26802D-01    |proj g|=  3.46430D-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   1.286D-05   8.268D-01
  F =   0.82679879652093957

CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
```

[34]:

```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
==========
Dep. Variable:                               Revenue   No. Observations:
638
Model:             SARIMAX(1, 1, 0)x(1, 1, 0, 90)   Log Likelihood
-527.498
Date:                               Mon, 01 Aug 2022   AIC
1060.995
Time:                                       20:57:03   BIC
1073.909
Sample:                                   01-02-2019   HQIC
1066.043
                                        - 09-30-2020
Covariance Type:                                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3278      0.040     -8.205      0.000      -0.406      -0.249
ar.S.L90      -0.4743      0.042    -11.325      0.000      -0.556      -0.392
sigma2         0.3862      0.024     16.199      0.000       0.339       0.433
================================================================================
===
Ljung-Box (L1) (Q):                   1.54   Jarque-Bera (JB):
0.67
Prob(Q):                              0.22   Prob(JB):
```

```
0.72
Heteroskedasticity (H):                1.07   Skew:
0.08
Prob(H) (two-sided):                   0.66   Kurtosis:
2.95
================================================================================
===

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

[35]:
```
stepwise_fit=auto_arima(df_stationary['Revenue'], trace=True,␣
 ↪suppress_warnings=True)
stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=883.277, Time=0.47 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=1015.972, Time=0.05 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=881.359, Time=0.04 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=906.199, Time=0.05 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=1015.481, Time=0.03 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=883.300, Time=0.06 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=883.314, Time=0.08 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=883.348, Time=0.20 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=879.982, Time=0.03 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=881.911, Time=0.04 sec
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=881.927, Time=0.05 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=905.166, Time=0.02 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=881.947, Time=0.12 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0]
Total fit time: 1.252 seconds
```

[35]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  730
Model:               SARIMAX(1, 0, 0)   Log Likelihood                -437.991
Date:                Mon, 01 Aug 2022   AIC                            879.982
Time:                        20:57:04   BIC                            889.168
Sample:                             0   HQIC                           883.526
                                - 730
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).
"""
```

1.3 ARIMA Model using Dataset

[36]:
```
# Create Time Series Test Model
%time
tqdm.pandas()

a_model = ARIMA(df_stationary['Revenue'],
 ↪order=(1,1,0),seasonal_order=(1,1,0,90))
a_model = a_model.fit()
a_model.summary()
```

```
CPU times: user 8 µs, sys: 4 µs, total: 12 µs
Wall time: 3.81 µs
```

[36]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                                     SARIMAX Results
================================================================================
========
Dep. Variable:                           Revenue   No. Observations:
730
Model:             ARIMA(1, 1, 0)x(1, 1, 0, 90)   Log Likelihood
-621.982
Date:                           Mon, 01 Aug 2022   AIC
1249.964
```

```
Time:                             20:58:05   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).
"""
```

1.4 ARIMA Model using Training Dataset

[37]:
```python
# Create Time Series Test Model
%time
tqdm.pandas()

a_model_train = ARIMA(X_train['Revenue'],
 ↪order=(1,1,0),seasonal_order=(1,1,0,90))
a_model_train = a_model_train.fit()
a_model_train.summary()
```

```
CPU times: user 1e+03 ns, sys: 0 ns, total: 1e+03 ns
Wall time: 3.81 µs
```

[37]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
```

```
==============================================================================
========
Dep. Variable:                           Revenue   No. Observations:
638
Model:             ARIMA(1, 1, 0)x(1, 1, 0, 90)   Log Likelihood
-527.498
Date:                           Mon, 01 Aug 2022   AIC
1060.995
Time:                                   20:58:49   BIC
1073.909
Sample:                               01-02-2019   HQIC
1066.043
                                    - 09-30-2020
Covariance Type:                             opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3278      0.040     -8.205      0.000      -0.406      -0.249
ar.S.L90      -0.4743      0.042    -11.325      0.000      -0.556      -0.392
sigma2         0.3862      0.024     16.199      0.000       0.339       0.433
==============================================================================
===
Ljung-Box (L1) (Q):                   1.54   Jarque-Bera (JB):
0.67
Prob(Q):                              0.22   Prob(JB):
0.72
Heteroskedasticity (H):               1.07   Skew:
0.08
Prob(H) (two-sided):                  0.66   Kurtosis:
2.95
==============================================================================
===

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

1.5 ARIMA Model using Test Dataset

[38]:
```
# Create Time Series Test Model
%time
tqdm.pandas()

a_model_test = ARIMA(X_test['Revenue'], order=(1,1,0))
a_model_test = a_model_test.fit()
a_model_test.summary()
```

```
CPU times: user 9 µs, sys: 0 ns, total: 9 µs
Wall time: 4.05 µs
```

[38]:
```
<class 'statsmodels.iolib.summary.Summary'>
"""
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                   92
Model:                 ARIMA(1, 1, 0)   Log Likelihood                 -57.847
Date:                Mon, 01 Aug 2022   AIC                            119.695
Time:                        20:58:49   BIC                            124.717
Sample:                    10-01-2020   HQIC                           121.721
                         - 12-31-2020
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.3458      0.108     -3.212      0.001      -0.557      -0.135
sigma2         0.2085      0.030      7.054      0.000       0.151       0.266
================================================================================
===
Ljung-Box (L1) (Q):                   0.16   Jarque-Bera (JB):
0.22
Prob(Q):                              0.69   Prob(JB):
0.90
Heteroskedasticity (H):               2.82   Skew:
-0.06
Prob(H) (two-sided):                  0.01   Kurtosis:
3.20
================================================================================
===

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

[39]:
```
print("Params for Training Data: ",a_model_train.params)
print("*********"*5)
```

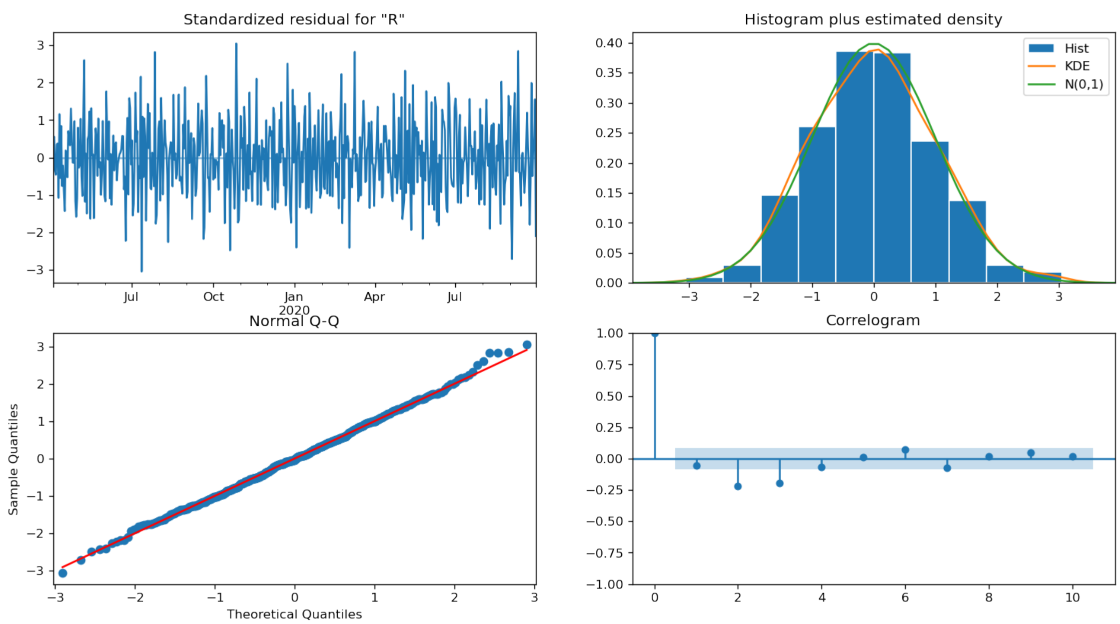
```
print("Params for Data: ",a_model.params)
```

```
Params for Training Data:  ar.L1      -0.327766
ar.S.L90   -0.474285
sigma2      0.386229
dtype: float64
**********************************************
Params for Data:  ar.L1      -0.308376
ar.S.L90   -0.472576
sigma2      0.395774
dtype: float64
```

[40]:
```
# 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
```

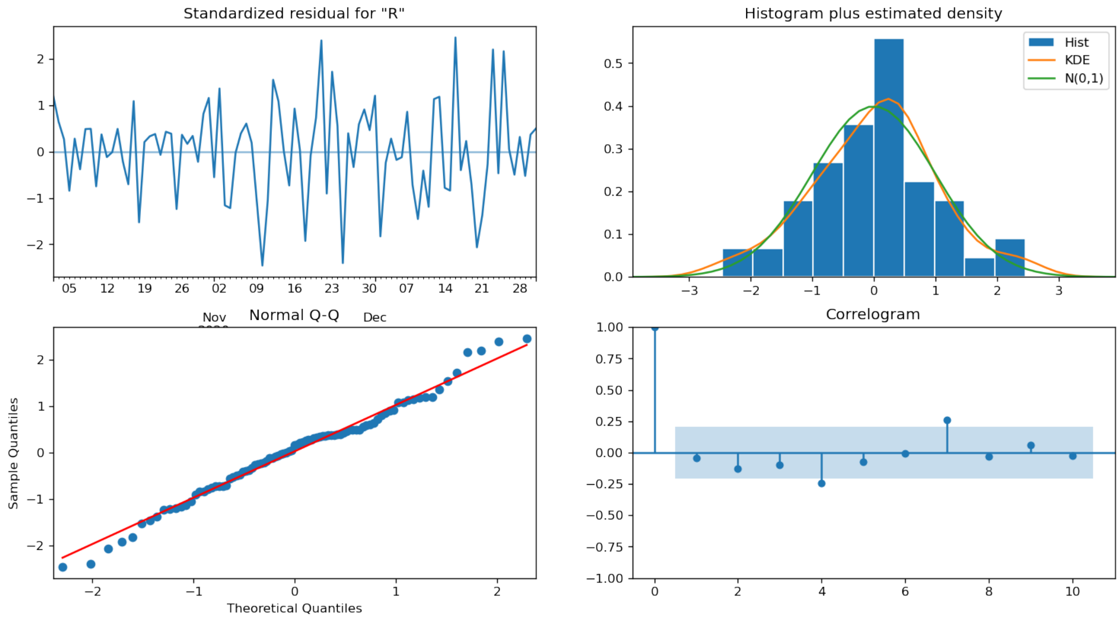
1.5.1 Four Diagnostic Plots using Training Data

[41]:
```
# Create the 4 diagnostics plots
a_model_train.plot_diagnostics(figsize=(15,8)).show()
```



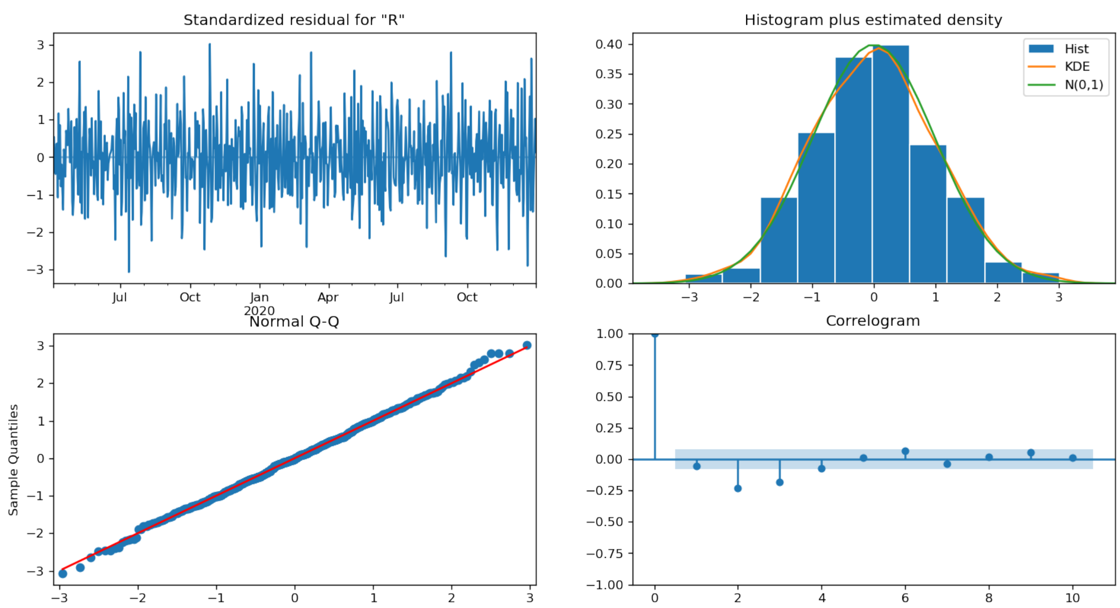
1.5.2 Four Diagnostic Plots using Test Data

[42]:
```python
# Create the 4 diagnostics plots
a_model_test.plot_diagnostics(figsize=(15,8)).show()
```



1.5.3 Four Diagnostic Plots using Data

[43]:
```python
# Create the 4 diagnostics plots
results.plot_diagnostics(figsize=(15,8)).show()
```



[44]:

```python
# Validate w/Test Set

# 90 day prediction range
#prediction = results.get_prediction(start=-90)
prediction = results.get_prediction(start=-90)
prediction_train = a_model_train.get_prediction(start=-90)
prediction_test = a_model_test.get_prediction(start=-90)



# Prediction Mean
mean_prediction = prediction.predicted_mean
mean_prediction_train = prediction_train.predicted_mean
mean_prediction_test = prediction_test.predicted_mean



# Confidence Intervals of Predictions
confidence_intervals = prediction.conf_int()
confidence_intervals_train = prediction_train.conf_int()
confidence_intervals_test = prediction_test.conf_int()



# Upper & lower conf limits
lower_limits = confidence_intervals.loc[:,'lower Revenue']
upper_limits = confidence_intervals.loc[:,'upper Revenue']
lower_limits_train = confidence_intervals_train.loc[:,'lower Revenue']
upper_limits_train = confidence_intervals_train.loc[:,'upper Revenue']
lower_limits_test = confidence_intervals_test.loc[:,'lower Revenue']
upper_limits_test = confidence_intervals_test.loc[:,'upper Revenue']

# Print predictions (best estimate)
print("Mean Pred: ",mean_prediction)
print("***********"*5)
print("Training Mean Pred: ",mean_prediction_train)
print("***********"*5)
print("Testing Mean Pred: ",mean_prediction_test)
```

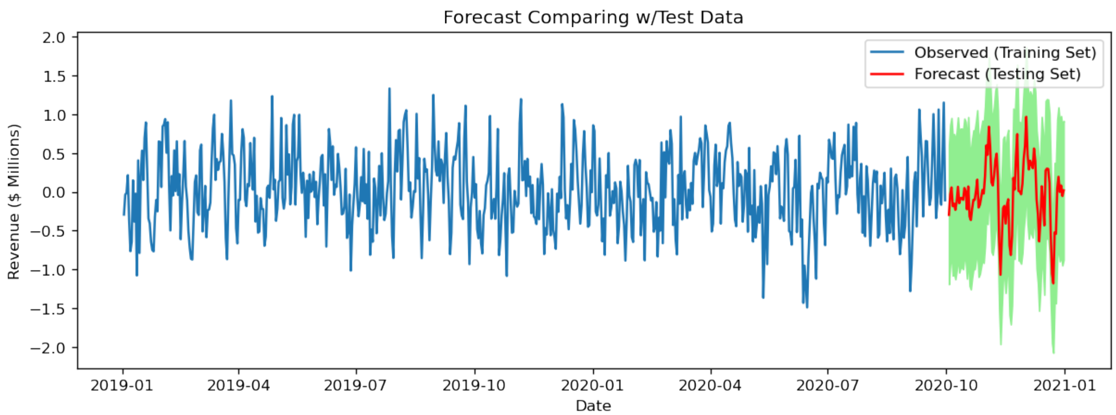
```
Mean Pred:  2020-10-03   -0.516314
2020-10-04    0.111306
2020-10-05   -0.037037
2020-10-06   -0.907660
2020-10-07    0.027541
                 …
```

```
2020-12-27    0.883200
2020-12-28    0.541385
2020-12-29   -0.219236
2020-12-30   -0.530855
2020-12-31    0.164098
Freq: D, Name: predicted_mean, Length: 90, dtype: float64
*******************************************************
Training Mean Pred:  2020-07-03    0.961768
2020-07-04    1.189731
2020-07-05    0.084454
2020-07-06    0.054223
2020-07-07    0.904694
                …
2020-09-26    0.296067
2020-09-27   -0.213579
2020-09-28    0.156638
2020-09-29    0.185135
2020-09-30    1.199903
Freq: D, Name: predicted_mean, Length: 90, dtype: float64
*******************************************************
Testing Mean Pred:  2020-10-03   -0.295152
2020-10-04   -0.033150
2020-10-05    0.057697
2020-10-06   -0.183911
2020-10-07   -0.150564
                …
2020-12-27    0.194839
2020-12-28   -0.000107
2020-12-29    0.082740
2020-12-30   -0.052876
2020-12-31    0.020257
Freq: D, Name: predicted_mean, Length: 90, dtype: float64
```

[45]:

```
# Plot Training Data
plt.figure(figsize=(12,4))
plt.plot(np.array(X_train.index), np.array(X_train[['Revenue']]),
 ↪label='Observed (Training Set)')

# plot your mean predictions
plt.plot(mean_prediction_test.index, mean_prediction_test, color='r',
 ↪label='Forecast (Testing Set)')

# shade upper conf. limit area
#plt.fill_between(lower_limits.index, lower_limits, upper_limits, color='pink')
plt.fill_between(upper_limits_test.index, upper_limits_test, lower_limits_test,
 ↪color='lightgreen')
```

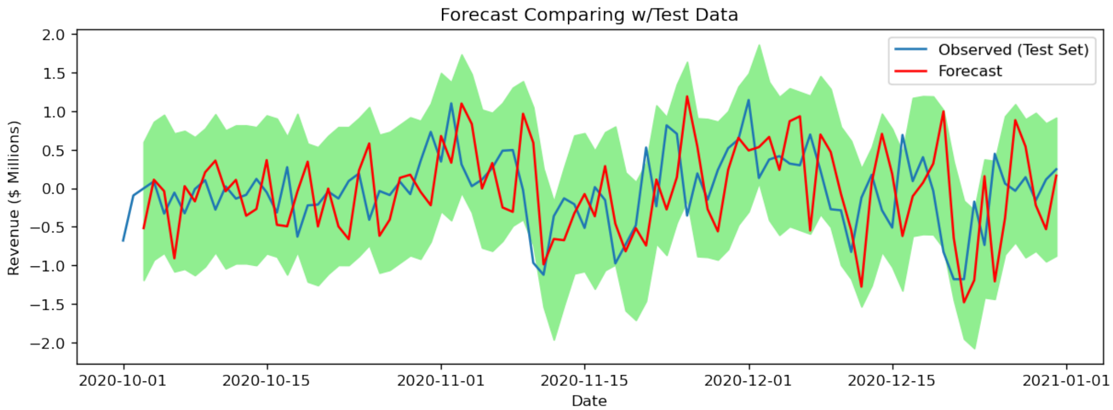
```
## plot mean predictions
#plt.fill_between(mean_prediction.index, mean_prediction, color='brown',␣
↪label='forecast')

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



[46]:

```
# Plot Test 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='Observed␣
↪(Test Set)')

# 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='pink')
plt.fill_between(upper_limits_test.index, upper_limits_test, lower_limits_test,␣
↪color='lightgreen')

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

# Annotations: Labels and Legends
plt.title('Forecast Comparing w/Test Data')
plt.xlabel('Date')
```

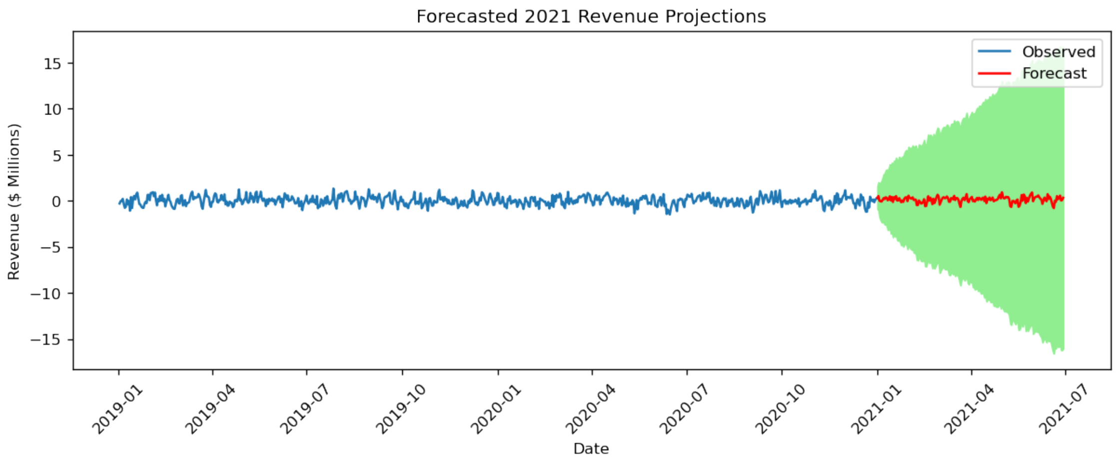
```
plt.ylabel('Revenue ($ Millions)')
plt.legend()
plt.show()
```



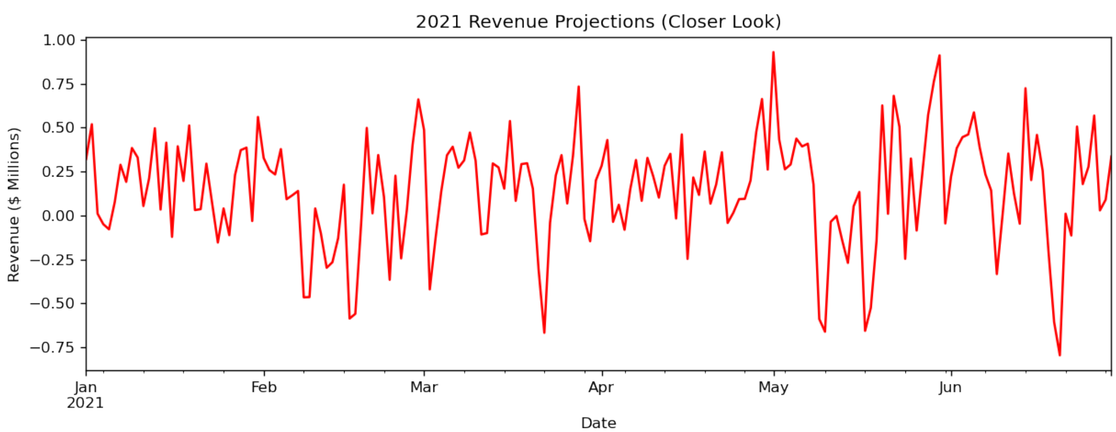
[47]:
```
# Perform forecast
diff_forecast = results.get_forecast(steps=180)
mean_forecast = diff_forecast.predicted_mean

# 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']
```

[48]:
```
# 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']].
  ↪reset_index(drop=True)), label='Observed')

# 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='pink')
plt.fill_between(upper_limits.index, upper_limits, lower_limits,␣
  ↪color='lightgreen')

# 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()
```



[49]:
```
# Mean Forecast Plot
plt.figure(figsize=(12,4))
plt.title('2021 Revenue Projections (Closer Look)')
plt.xlabel('Date')
plt.ylabel('Revenue ($ Millions)')
mean_forecast.plot(color='r');
```

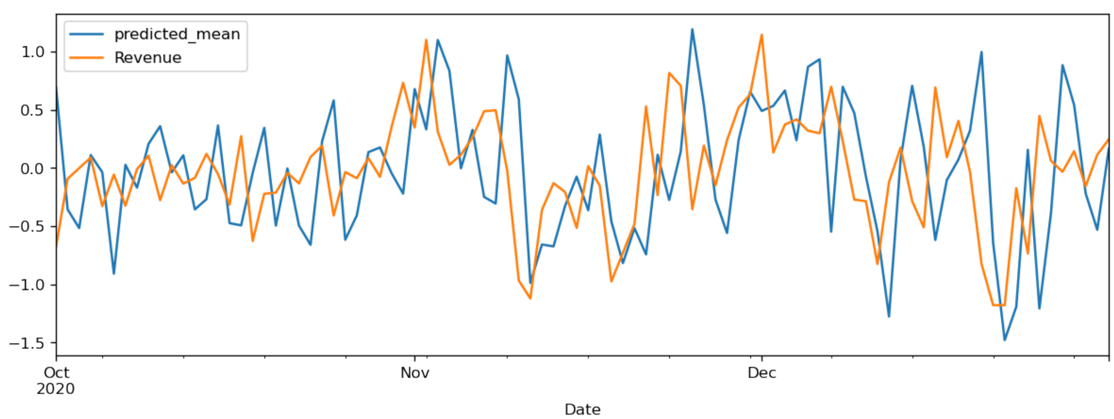


1.5.4 A model run using the data only from the training set and forecasted out to the test set

[50]:
```
start=len(X_train) # A model run using the data only from the training set
end=len(X_train)+len(X_test)-1
pred=results.predict(start=start, end=end, typ='levels') # forecasted out to␣
↪the test set
# Set index for plotting
pred.index=df_stationary.index[start:end+1]
print(pred)
```

```
Date
2020-10-01    0.691880
2020-10-02   -0.355448
2020-10-03   -0.516314
2020-10-04    0.111306
2020-10-05   -0.037037
                ...
2020-12-27    0.883200
2020-12-28    0.541385
2020-12-29   -0.219236
2020-12-30   -0.530855
2020-12-31    0.164098
Name: predicted_mean, Length: 92, dtype: float64
```

[51]:
```
pred.plot(legend=True)
X_test['Revenue'].plot(figsize=(12,4),legend=True);
```



[52]:
```
# Split for Training and Testing

#X_train = df_stationary.loc[:'2020-09-30'] # Get all but the last 90 days for␣
↪training
#X_test = df_stationary['2020-10-01':] # Get last 90 days of data to test
```

```
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)
```

[53]:
```
print("Train Mean: ",X_train['Revenue'].mean())
print("Test Mean: ",X_test['Revenue'].mean())
print("Actual Mean: ",df_stationary['Revenue'].mean())
```

```
Train Mean:  0.02888145484326019
Test Mean:  -0.025618899021739146
Actual Mean:  0.02201291709589041
```

1.5.5 Standard Error Metric: Train, Test and Actual Data

[54]:
```
# Print mean absolute error
mae = np.mean(np.abs(a_model.resid))
mae_train = np.mean(np.abs(a_model_train.resid))
mae_test = np.mean(np.abs(a_model_test.resid))

print("Actual - Mean Absolute Error Data: ", mae)
print("Actual - Mean Absolute Error Training Data: ", mae_train)
print("Actual - Mean Absolute Error Test Data: ", mae_test)
```

```
Actual - Mean Absolute Error Data:  0.49873094599791684
Actual - Mean Absolute Error Training Data:  0.49425653996990915
Actual - Mean Absolute Error Test Data:  0.35953135297954647
```

[55]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(pred,X_test['Revenue']))
print("RMSE of test data: ", rmse)
```

```
RMSE of test data:  0.6713765772122239
```

A visualization that shows a true out-of-sample forecast over the test-set horizon, as well as the test-set actuals, is not readily evident.

Please provide a chart that compares out-of-sample predictions to actuals. out-of-sample = future dates

[56]:
```
# Save model
#joblib.dump(model, "time_series_model.pkl")
```

2 Terminal: nbconvert –to pdf D213_PA1.ipynb

2.1 End