Teleco revenue predictions and forecasting report:

Dataset includes 730 consecutive days of company 'Revenue' which we will use to forecast future revenue

In [1]:
```
#pip install pmdarima
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.pyplot import figure
import os
dirpath = os.getcwd()
print(dirpath)
import warnings
warnings.filterwarnings("ignore")
```

```
C:\Users\mikep\Documents\WGU\D213 Advanced Data Analytics
```

In [2]:
```
df = pd.read_csv('teleco_time_series.csv') #change date to day or time?
df=df.dropna()
print('Shape of data', df.shape)
df.tail()
```

```
Shape of data (731, 2)
```

Out[2]:

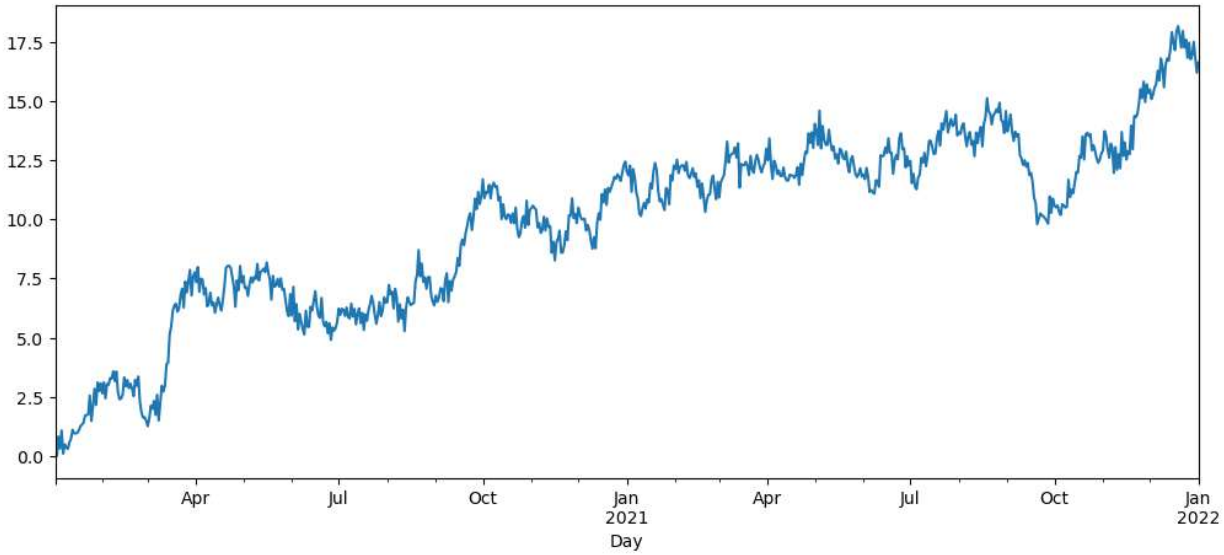
	Day	Revenue
726	727	16.931559
727	728	17.490666
728	729	16.803638
729	730	16.194813
730	731	16.620798

In [3]:
```
df.set_index('Day',inplace=True)
```

In [4]:
```
df.index=pd.to_datetime(df.index, unit = 'D', origin = '2020-01-01')
```

In [6]:
```
df['Revenue'].plot(figsize=(12,5))
```

Out[6]:
```
<AxesSubplot:xlabel='Day'>
```



In [7]:
```
df.isna().sum()
```

Out[7]:
```
Revenue    0
dtype: int64
```

We use the Augmented Dickey-Fuller method to check if our data is stationary. Our data is not as we see the Test Statistic is not lower than the Critical Values and the p-value is high

In [8]:
```
from statsmodels.tsa.stattools import adfuller
print ('Results of ADF test: ')
dftest = adfuller(df['Revenue'],autolag='AIC')
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','No.
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s) '%key] = value
print(dfoutput) #if p-value is > 5, it is not stationary
```

```
Results of ADF test:
Test Statistic            -1.924612
p-value                    0.320573
#Lags Used                 1.000000
No. Observations         729.000000
Critical Value (1%)       -3.439352
Critical Value (5%)       -2.865513
Critical Value (10%)      -2.568886
dtype: float64
```

We use .diff() to make the data stationary (we remove the upward trend in the data). Diff calculates the difference between the values for each row.

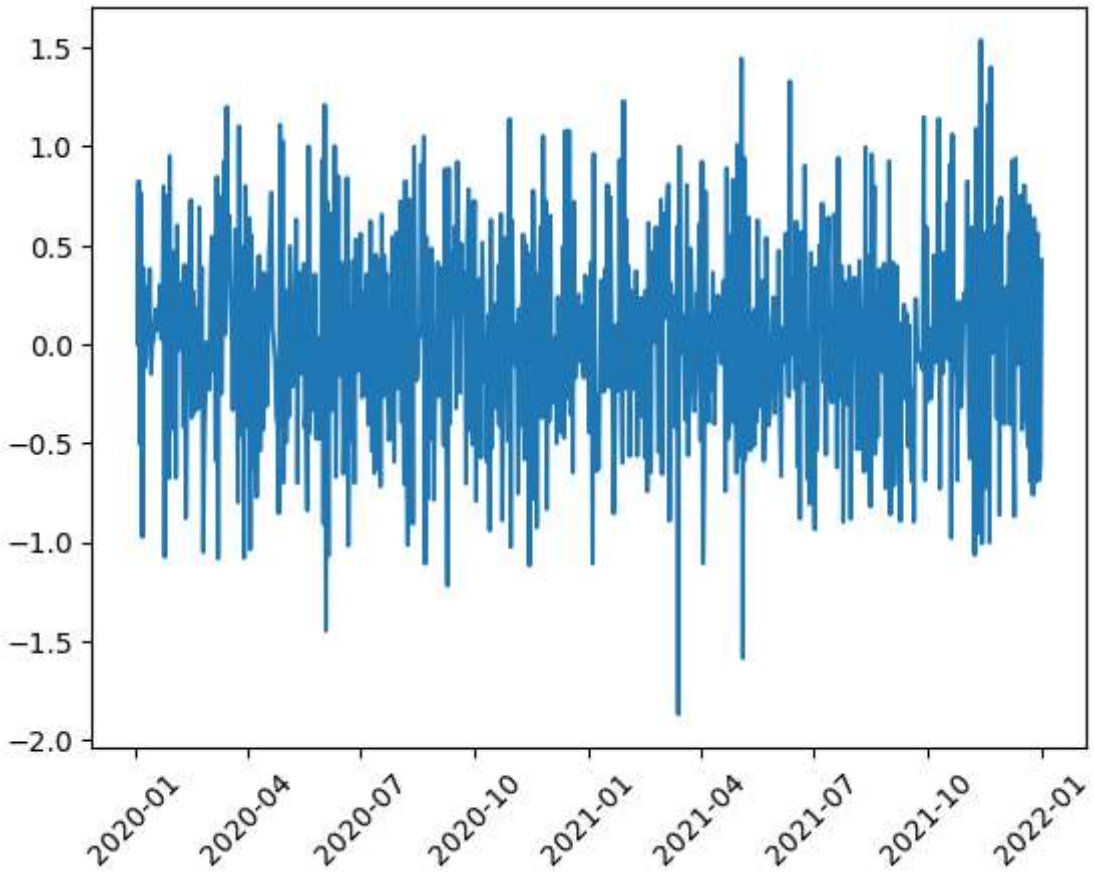
In [9]:
```
diff_df = df.diff().dropna()
diff_df.head()
```

Out[9]:

Day	Revenue
2020-01-03	0.000793
2020-01-04	0.824749
2020-01-05	-0.505210
2020-01-06	0.762222
2020-01-07	-0.974900

We see the data is now stationary

In [10]:
```
import matplotlib.pyplot as plt
plt.plot(diff_df)
plt.xticks(rotation=45)
plt.show()
```



In [11]:
```
diff_df.tail(5)
```

Out[11]:

Day	Revenue
2021-12-28	0.170280
2021-12-29	0.559108
2021-12-30	-0.687028
2021-12-31	-0.608824
2022-01-01	0.425985

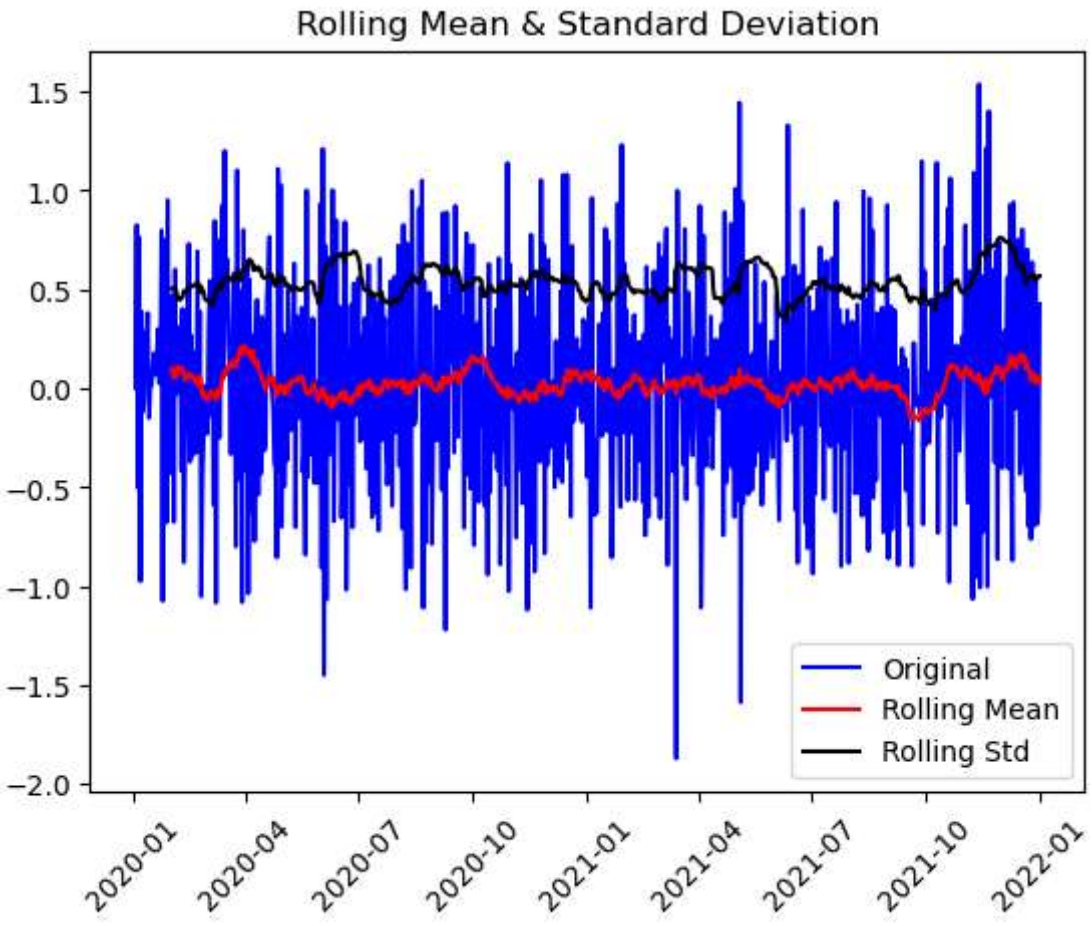
After using .diff() to make our data stationary, our Test Statistic is lower than the Critical Values and our p-value is 0 which is ideal

In [31]:
```
from statsmodels.tsa.stattools import adfuller
print ('Results of ADF test: ')
dftest = adfuller(diff_df['Revenue'],autolag='AIC')
dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used','No.
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s) '%key] = value
print(dfoutput) #if p-value is > 5, it is not stationary
```

```
Results of ADF test:
Test Statistic           -44.874527
p-value                    0.000000
#Lags Used                 0.000000
No. Observations         729.000000
Critical Value (1%)       -3.439352
Critical Value (5%)       -2.865513
Critical Value (10%)      -2.568886
dtype: float64
```

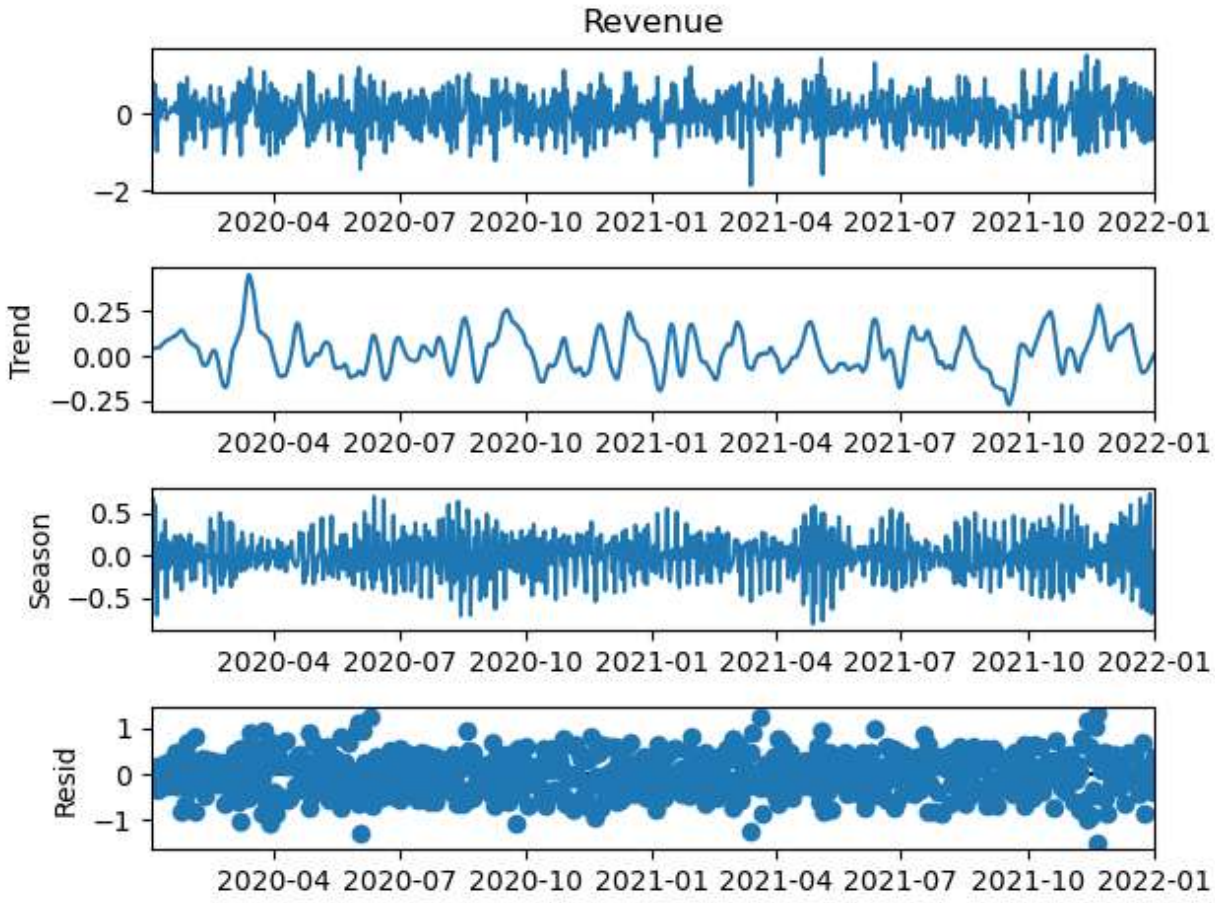
We also can see that the data is stationary with Rolling Mean

In [13]:
```
rolmean = diff_df.rolling(window=30).mean()
rolstd = diff_df.rolling(window=30).std()
orig = plt.plot(diff_df, color='blue', label='Original')
mean = plt.plot(rolmean, color='red', label='Rolling Mean')
std = plt.plot(rolstd, color='black', label='Rolling Std')
plt.legend(loc='best')
plt.title('Rolling Mean & Standard Deviation')
plt.xticks(rotation=45)
plt.show(block=False)
```



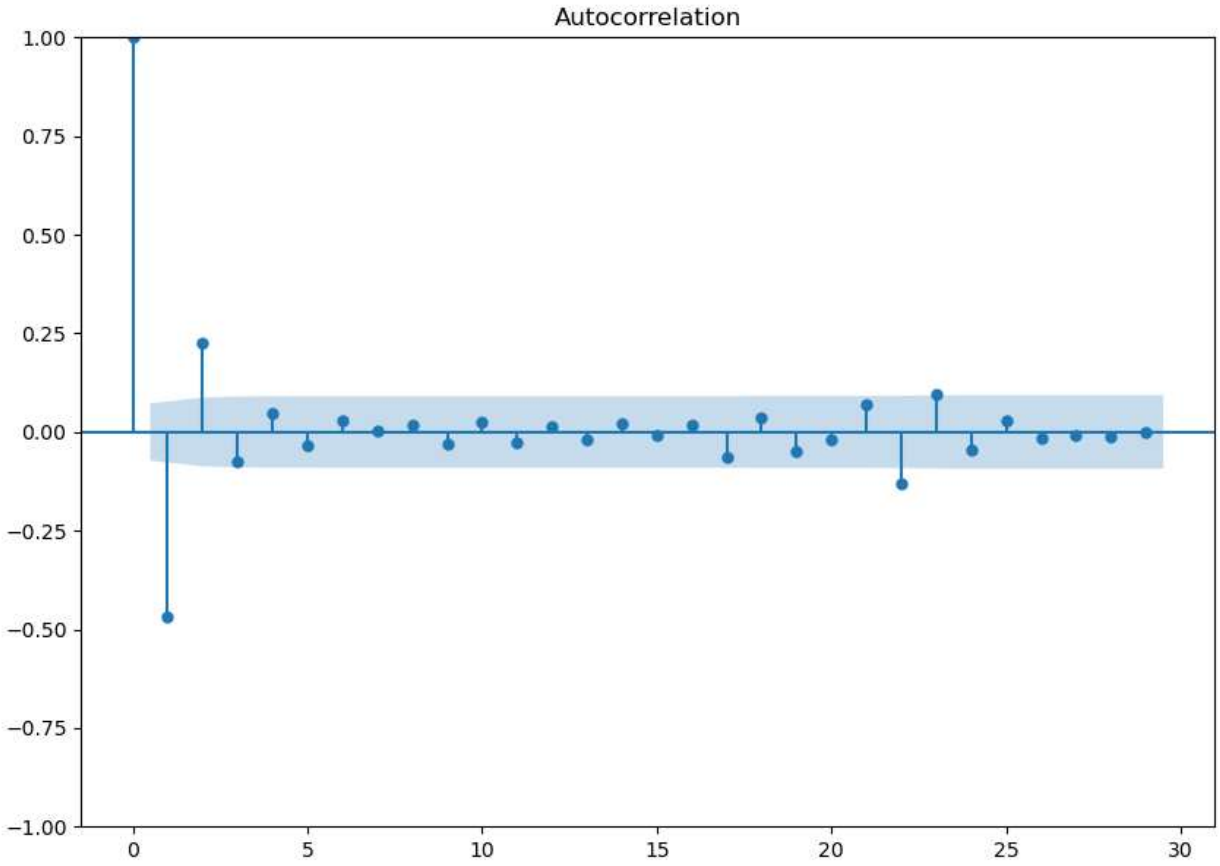
STL (Seasonal-Trend decomposition) to see the decomposition of our time series

In [14]:
```
from statsmodels.tsa.seasonal import STL #default is Additive (vs Multiplicative)
stl = STL(diff_df['Revenue'], period=7)
res = stl.fit()
fig = res.plot()
```

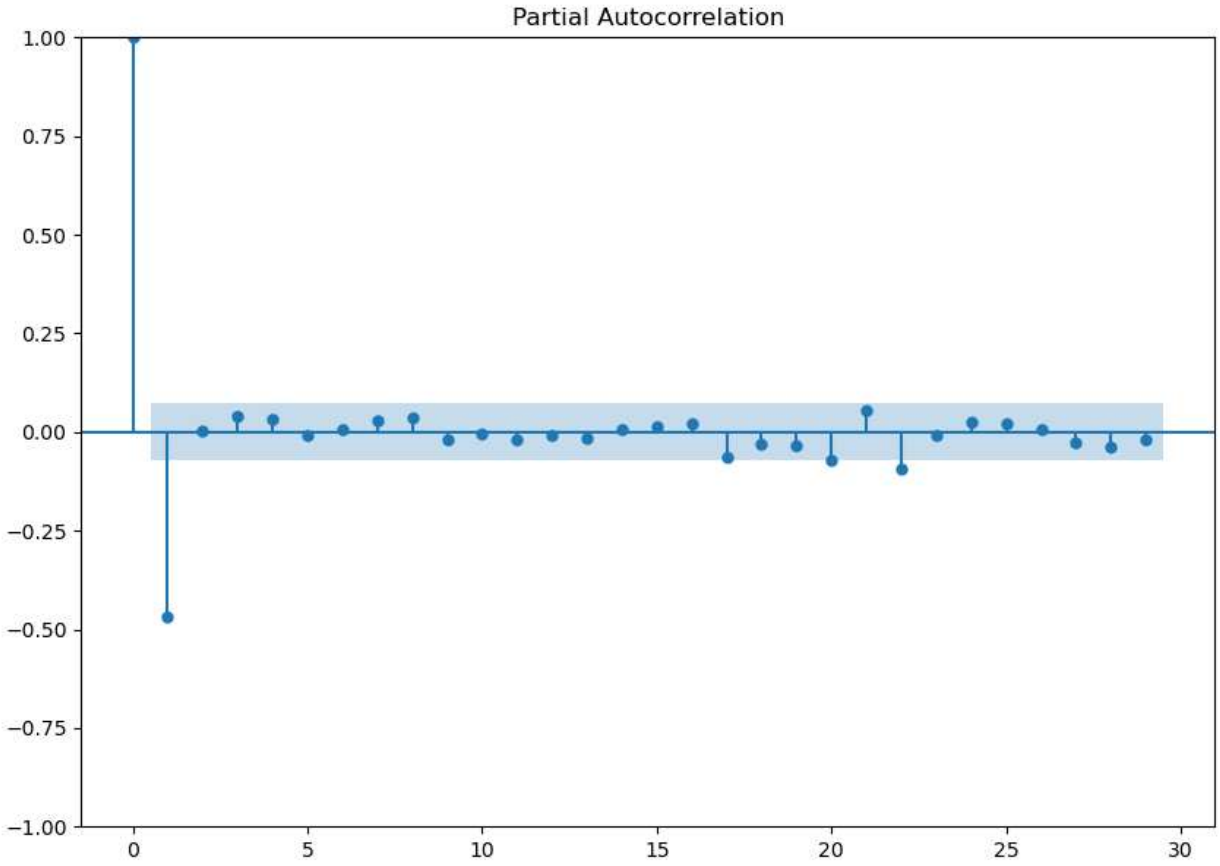


Autocorrelation (ACF) and Partial Autocorrelation (PACF) to identify initial p and q values

In [15]:
```
# Autocorrelation Plot - # of lagged forecast errors (q)
from statsmodels.graphics.tsaplots import plot_acf
fig = plot_acf(diff_df)
fig.set_size_inches(10, 7)
plt.show(fig)
```



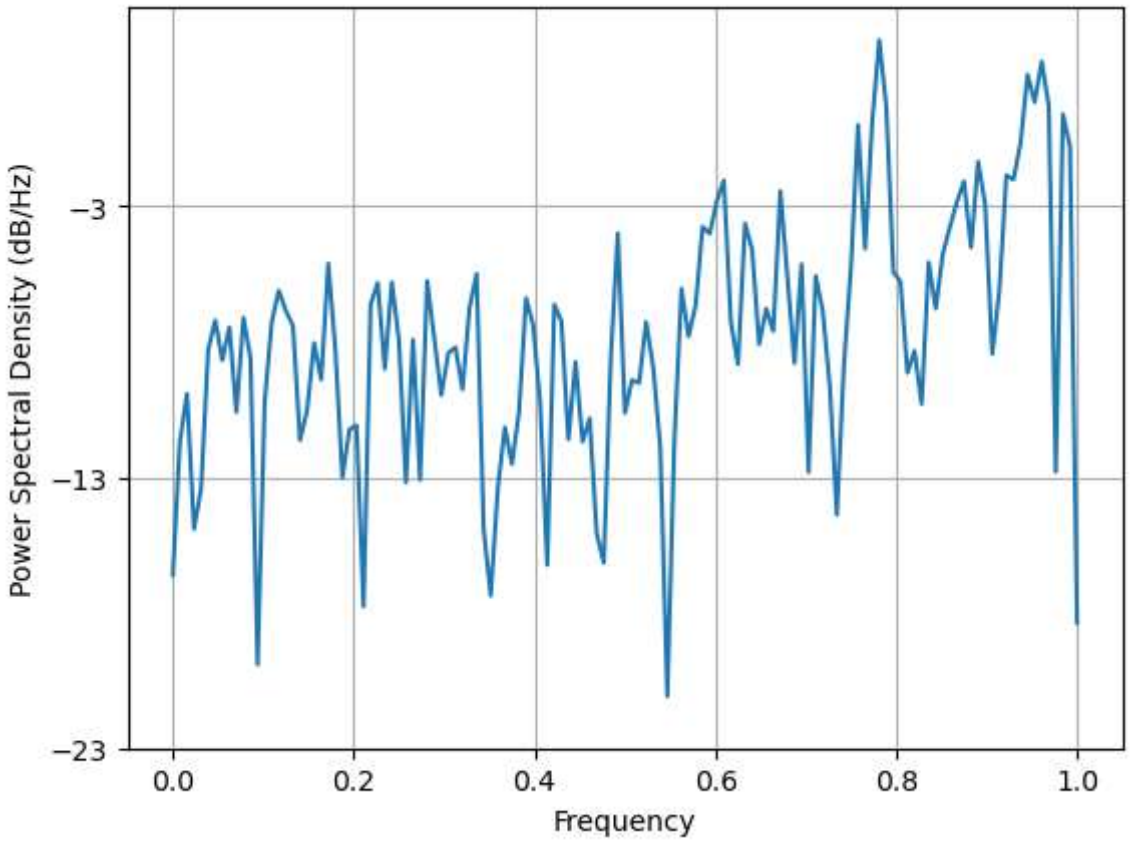
In [16]:
```
# Partial Autocorrelation Plot - # of lag observations (p)
from statsmodels.graphics.tsaplots import plot_pacf
fig = plot_pacf(diff_df)
fig.set_size_inches(10, 7)
plt.show(fig)
```



Spectral Density to help identify patterns (not required here as we learn later with ARIMA that our data is not seasonal)

In [17]:
```
plt.psd(diff_df['Revenue'])
plt.show
```

Out[17]:
```
<function matplotlib.pyplot.show(close=None, block=None)>
```



In [18]:
```
#Export to csv
diff_df.to_csv('telco_cleaned_data.csv', index = False)
```

Split the train and test datasets

In [19]:
```
train = df.iloc[:-250]
test = df.iloc[-250:]
print(df.shape)
print (train.shape, test.shape)
```

```
(731, 1)
(481, 1) (250, 1)
```

Once data our data is stationary we need to find out the p, d, q and do we have seasonality (consistent trends) or not. We use auto_arima to help us identify the best model as it automatically looks for the lowest AIC score. We also verify this by running a manual ARIMA model with the best p, d, q values shown from the auto_arima model

In [20]:
```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(train['Revenue'], order=(1,1,0)) # Set the best order from previous step
model = model.fit()
model.summary()
```

```
C:\Users\mikep\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\mikep\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\mikep\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
```

Out[20]:

SARIMAX Results

Dep. Variable:	Revenue	**No. Observations:**	481
Model:	ARIMA(1, 1, 0)	**Log Likelihood**	-318.476
Date:	Sun, 26 Feb 2023	**AIC**	640.952
Time:	21:48:13	**BIC**	649.299
Sample:	01-02-2020	**HQIC**	644.233
	- 04-26-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.4576	0.040	-11.456	0.000	-0.536	-0.379
sigma2	0.2206	0.015	14.526	0.000	0.191	0.250

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	1.93
Prob(Q):	0.95	**Prob(JB):**	0.38
Heteroskedasticity (H):	0.94	**Skew:**	-0.13
Prob(H) (two-sided):	0.71	**Kurtosis:**	2.82

Warnings:

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

In [21]:

```
from pmdarima import auto_arima #auto_arima
stepwise_fit=auto_arima(train['Revenue'], trace=True, suppress_warnings=True)
stepwise_fit.summary()
#first set of parenthesis is p, d, q
#second set of parenthesis tells us if we do not have seasonality
#third [] would show value of seasonality, if there is one (i.e. 12 for 12 months)
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=643.884, Time=0.23 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=752.777, Time=0.05 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=639.634, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=663.327, Time=0.10 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=751.999, Time=0.03 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=641.498, Time=0.09 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=641.524, Time=0.12 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=643.041, Time=0.33 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=640.952, Time=0.03 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0] intercept
Total fit time: 1.039 seconds
```

Out[21]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	481
Model:	SARIMAX(1, 1, 0)	**Log Likelihood**	-316.817
Date:	Sun, 26 Feb 2023	**AIC**	639.634
Time:	21:48:15	**BIC**	652.155
Sample:	01-02-2020	**HQIC**	644.556
	- 04-26-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0391	0.022	1.805	0.071	-0.003	0.081
ar.L1	-0.4613	0.040	-11.520	0.000	-0.540	-0.383
sigma2	0.2191	0.015	14.631	0.000	0.190	0.248

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	1.90
Prob(Q):	0.88	**Prob(JB):**	0.39
Heteroskedasticity (H):	0.95	**Skew:**	-0.13
Prob(H) (two-sided):	0.74	**Kurtosis:**	2.82

Warnings:

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

Make our prediction on the test data from the chosen ARIMA model

In [22]:

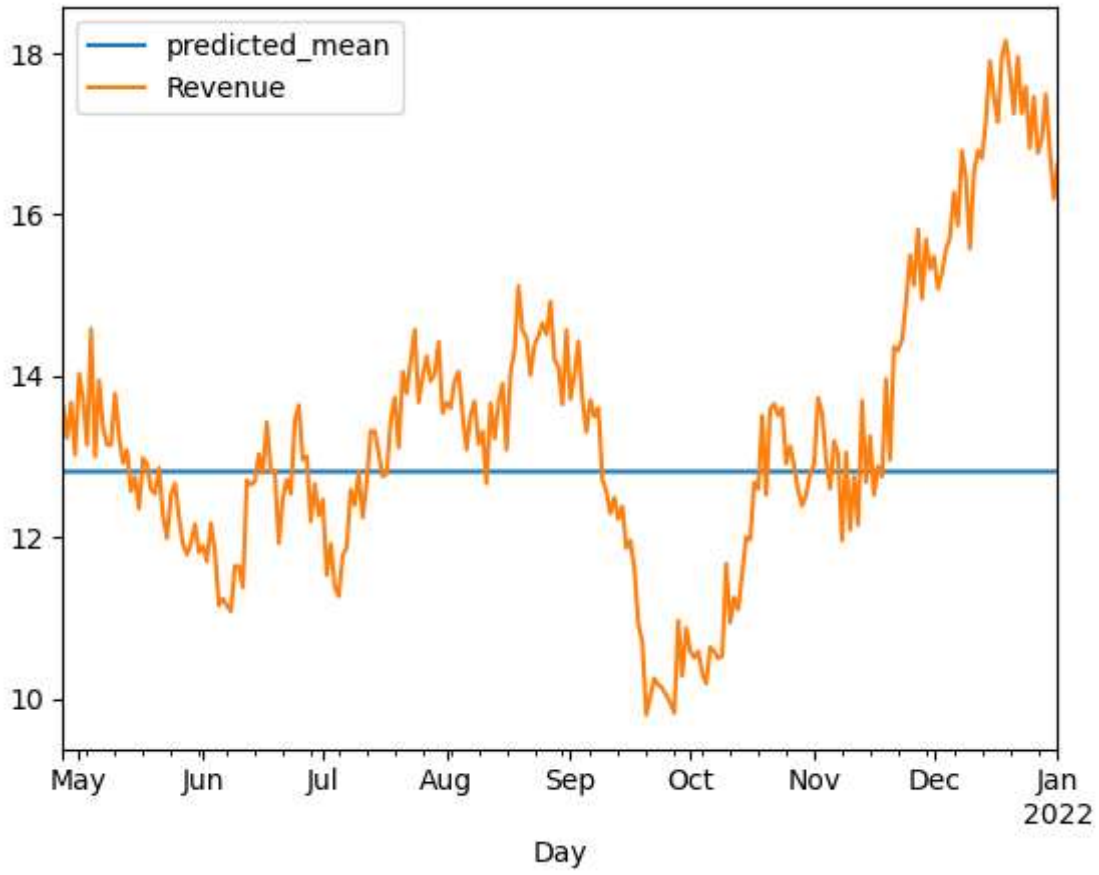
```
start=len(train)
end=len(train)+len(test)-1
pred=model.predict(start=start, end=end, typ='levels')
pred.index=test.index
pred.tail()
```

Out[22]:
```
Day
2021-12-28    12.812678
2021-12-29    12.812678
2021-12-30    12.812678
2021-12-31    12.812678
2022-01-01    12.812678
Name: predicted_mean, dtype: float64
```

We see the mean cuts through the center of the Revenue which is expected

In [23]:
```
pred.plot(legend=True)
test['Revenue'].plot(legend=True)
```

Out[23]:
```
<AxesSubplot:xlabel='Day'>
```



Calculate Root Mean Squared Error (RMSE) which measures average difference between predicted vs actual values

In [24]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
rmse = sqrt(mean_squared_error(pred,test['Revenue']))
print(rmse)
```

```
1.8546541437443795
```

In [25]:
```
start = len(df)
end = len(df) + 30 #representing 30 days
```

In [26]:
```
test['Revenue'].mean()
```

Out[26]:
```
13.308131415140009
```

We fit the ARIMA model for the actual dataset. We remove the 1 representing the d (differencing) in our p,d,q variables as our original dataset does not use .diff() (1,1,0) is now (1,0,0)

In [27]:
```
model2=ARIMA(df['Revenue'], order=(1,0,0))
model2 = model2.fit()
df.tail()
```

```
C:\Users\mikep\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\mikep\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
C:\Users\mikep\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_model.py:471: Val
ueWarning: No frequency information was provided, so inferred frequency D will be use
d.
  self._init_dates(dates, freq)
```

Out[27]:

Day	Revenue
2021-12-28	16.931559
2021-12-29	17.490666
2021-12-30	16.803638
2021-12-31	16.194813
2022-01-01	16.620798

Adding the dates that we will use in our forecast

In [28]:
```
index_future_dates = pd.date_range(start='2022-01-02',end='2022-02-01')
print(index_future_dates)
```

```
DatetimeIndex(['2022-01-02', '2022-01-03', '2022-01-04', '2022-01-05',
               '2022-01-06', '2022-01-07', '2022-01-08', '2022-01-09',
               '2022-01-10', '2022-01-11', '2022-01-12', '2022-01-13',
               '2022-01-14', '2022-01-15', '2022-01-16', '2022-01-17',
               '2022-01-18', '2022-01-19', '2022-01-20', '2022-01-21',
               '2022-01-22', '2022-01-23', '2022-01-24', '2022-01-25',
               '2022-01-26', '2022-01-27', '2022-01-28', '2022-01-29',
               '2022-01-30', '2022-01-31', '2022-02-01'],
              dtype='datetime64[ns]', freq='D')
```

In [29]:
```
pred = model2.predict(start=len(df), end=len(df)+30, typ='levels').rename('ARIMA Predi
pred.index=index_future_dates
pred.head()
```

Out[29]:
```
2022-01-02    16.581221
2022-01-03    16.541861
2022-01-04    16.502720
2022-01-05    16.463795
2022-01-06    16.425085
Freq: D, Name: ARIMA Predictions, dtype: float64
```

Plotting our Revenue data

In [30]:
```
plt.plot(train.index, train['Revenue'], label='Train data')
plt.plot(test.index, test['Revenue'], label='Test data')
plt.plot(pred.index, pred.values, label='Forecast data')
plt.title('Projected Revenue Trend')
plt.xlabel('Date')
plt.ylabel('Revenue')
pred_upper = pred + 0.05 * pred  # Upper bound of forecast (5% error)
pred_lower = pred - 0.05 * pred  # Lower bound of forecast (5% error)
plt.fill_between(pred.index, pred_upper, pred_lower, color='gray', alpha=0.2)
plt.legend()
plt.xticks(rotation=45)
plt.show()
```