WGU D213 Advanced Data Analytics

Task 1 - Time-Series Modeling

Ednalyn C. De Dios

September 7, 2023

Environment

- Python: 3.9.9
- Jupyter: 7.0.2

Part I. Research Question

A1. Summarize one research question that is relevant to a real-world organizational situation captured in the selected data set and that you will answer using time series modeling techniques.

> ***What is the revenue forecast for the next quarter?***

A2. Define the objectives or goals of the data analysis. Ensure your objectives or goals are reasonable within the scope of the scenario and are represented in the available data.

Analyze two years worth of daily revenue data of the organization and create a predictive model that will forecast the next 90 days of future revenue.

Part II. Method Justification

B. Summarize the assumptions of a time series model including stationarity and autocorrelated data.

Two assumptions of time series analysis include stationarity and autocorrelation. Stationarity means that "the mean, variance, and autocorrelation structure are constant over time" (Statisticssolutions.com, n.d.). In other words, "the statistical properties of a time series do not change over time" (Statisticssolutions.com, n.d.). The other assumption is no autocorrelation. "Autocorrelation occurs when future values in a time series linearly depend on past values" (Pierre, 2021).

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

In [1]:

```
# setting the random seed for reproducibility
import random
random.seed(493)

# for manipulating dataframes
import pandas as pd
import numpy as np
from datetime import datetime

# for visualizations
%matplotlib inline
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
sns.set(style="whitegrid")
from IPython.display import Image

from statsmodels.tsa.stattools import adfuller
from sklearn.model_selection import train_test_split
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from pmdarima.arima import auto_arima
from pmdarima.arima import ADFTest
from sklearn.metrics import r2_score
from sklearn.metrics import mean_squared_error
from math import sqrt

from scipy import signal
from statsmodels.tsa.stattools import acf, pacf
import statsmodels.tsa.stattools as ts
from statsmodels.tsa.arima.model import ARIMA

# to print out all the outputs of the cell
from IPython.core.interactiveshell import InteractiveShell
InteractiveShell.ast_node_interactivity = "all"

# set display options
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
```

```
pd.set_option('display.max_rows', None)
pd.set_option('display.max_colwidth', None)
```

In [2]:
```
# read the time series data set
df =  pd.read_csv('../data/teleco_time_series.csv')
```

In [3]:
```
df.head().T
df.tail().T
```

Out[3]:

	0	1	2	3	4
Day	1.0	2.000000	3.000000	4.000000	5.000000
Revenue	0.0	0.000793	0.825542	0.320332	1.082554

Out[3]:

	726	727	728	729	730
Day	727.000000	728.000000	729.000000	730.000000	731.000000
Revenue	16.931559	17.490666	16.803638	16.194813	16.620798

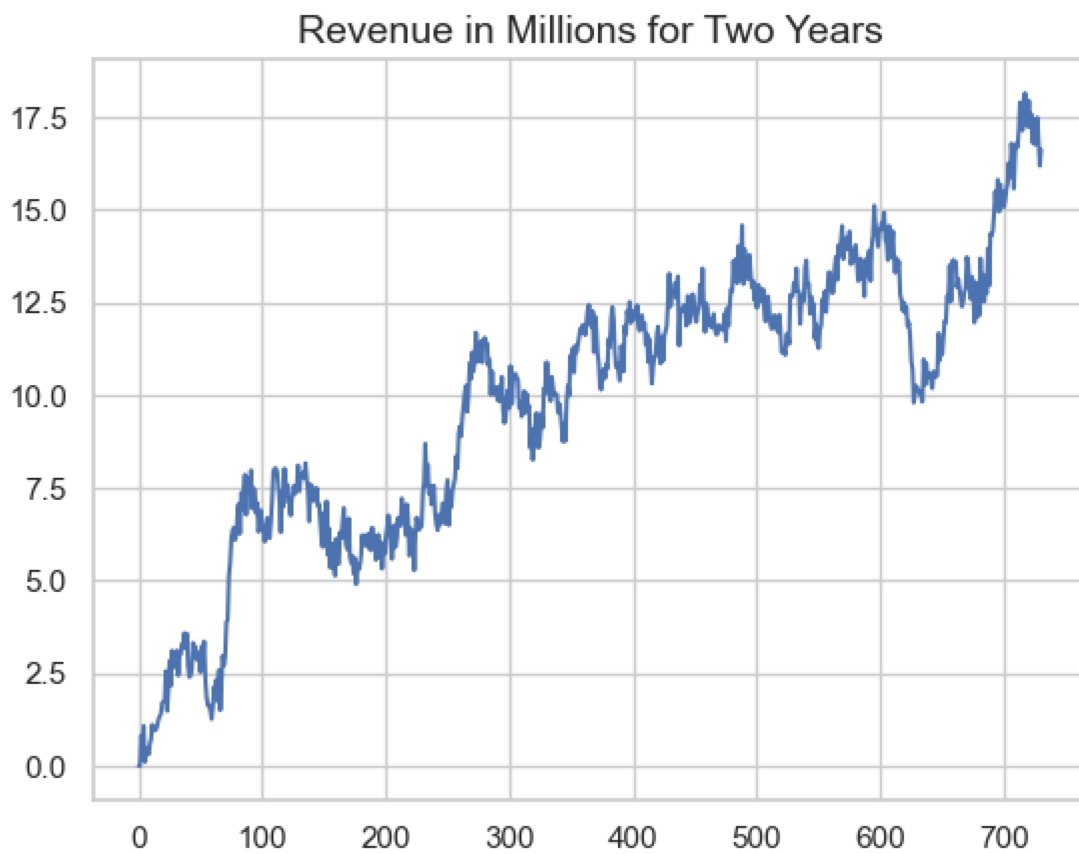
In [4]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Day      731 non-null    int64  
 1   Revenue  731 non-null    float64
dtypes: float64(1), int64(1)
memory usage: 11.5 KB
```

In [5]:
```
# plotting the realization of the time series
plt.plot(df.index, df['Revenue'])
plt.title('Revenue in Millions for Two Years', fontsize=14)
```

Out[5]: `[<matplotlib.lines.Line2D at 0x243bb8aa400>]`

Out[5]: `Text(0.5, 1.0, 'Revenue in Millions for Two Years')`



C2. Describe the time step formatting of the realization, including any gaps in measurement and the length of the sequence.

tk

In [6]:
```
# drop zero values
df= df[df['Revenue'] != 0]
```

In [7]:
```
def show_missing(df):
    """
    Takes a dataframe and returns a dataframe with stats
    on missing and null values with their percentages.
    """
    null_count = df.isnull().sum()
    null_percentage = (null_count / df.shape[0]) * 100
    empty_count = pd.Series(((df == ' ') | (df == '')).sum())
    empty_percentage = (empty_count / df.shape[0]) * 100
    nan_count = pd.Series(((df == 'nan') | (df == 'NaN')).sum())
    nan_percentage = (nan_count / df.shape[0]) * 100
    dfx = pd.DataFrame({'num_missing': null_count, 'missing_percentage': null_perce
                        'num_empty': empty_count, 'empty_percentage': empty_percen
                        'nan_count': nan_count, 'nan_percentage': nan_percentage})
    return dfx

show_missing(df)
```

Out[7]:

	num_missing	missing_percentage	num_empty	empty_percentage	nan_count	na
Day	0	0.0	0	0.0	0	
Revenue	0	0.0	0	0.0	0	

In [8]:

```
# add Date column
df['Date'] = pd.date_range(start = datetime(2019,1,1),
                           periods = df.shape[0],
                           freq = '24H'
                          )

# set Date column as index
df.set_index('Date', inplace=True)
df.drop(columns=['Day'], inplace=True)
```

In [9]:

```
plt.plot(df.Revenue)
plt.title('Revenue Chart')
plt.xlabel('Date')
plt.ylabel('Revenue in Millions $')
plt.xticks(rotation=30, fontsize=10)
```

Out[9]:

```
[<matplotlib.lines.Line2D at 0x243bb8f82b0>]
```

Out[9]:

```
Text(0.5, 1.0, 'Revenue Chart')
```

Out[9]:

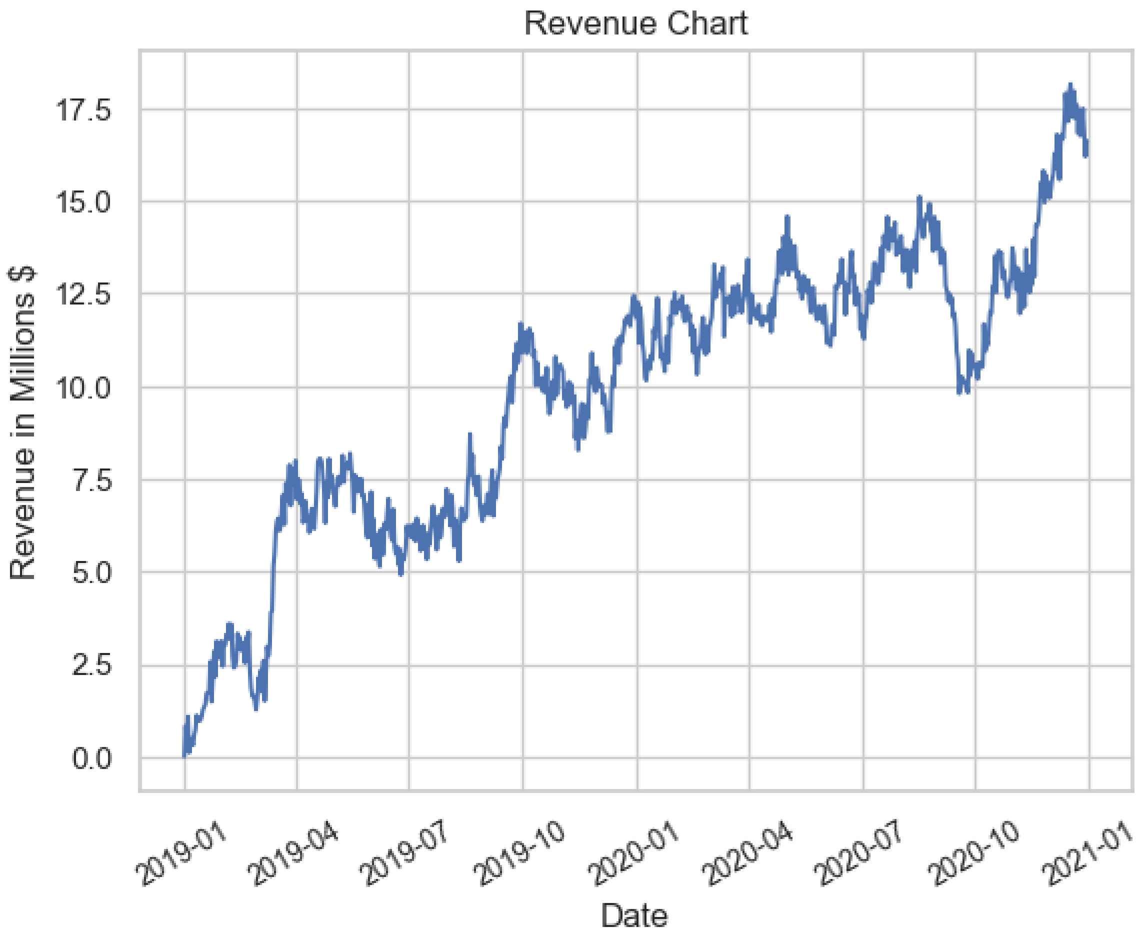
```
Text(0.5, 0, 'Date')
```

Out[9]:

```
Text(0, 0.5, 'Revenue in Millions $')
```

Out[9]:

```
(array([17897., 17987., 18078., 18170., 18262., 18353., 18444., 18536.,
        18628.]),
 [Text(17897.0, 0, '2019-01'),
  Text(17987.0, 0, '2019-04'),
  Text(18078.0, 0, '2019-07'),
  Text(18170.0, 0, '2019-10'),
  Text(18262.0, 0, '2020-01'),
  Text(18353.0, 0, '2020-04'),
  Text(18444.0, 0, '2020-07'),
  Text(18536.0, 0, '2020-10'),
  Text(18628.0, 0, '2021-01')])
```



C3. Evaluate the stationarity of the time series.

In [10]:
```
result = adfuller(df['Revenue'])
print('Test statistics: ', result[0])
print('P-value: ', result[1])
print('Critical value: ', result[4])
print('-----------------------------------------------------------------------------

if result[1] >= 0.05:
    print('Reject the null hypothesis. The time series is stationary. No further ac
else:
    print('Fail to reject the null hypothesis. The time series is not stationary. Y
```

```
Test statistics:  -1.7746383121968732
P-value:  0.3931237595029723
Critical value:  {'1%': -3.4393644334758475, '5%': -2.8655182850048306, '10%': -2.56
8888486973192}
------------------------------------------------------------------------------------
----------
Reject the null hypothesis. The time series is stationary. No further action require
d.
```

C4. Explain the steps you used to prepare the data for analysis, including the training and test set split.

tk

In [11]:
```
# use the last 30 days for testing
train = df.iloc[:-30]
test = df.iloc[-30:]
print('Training set: {}'.format(train.shape))
print('Testing set: {}'.format(test.shape))
```

```
Training set: (700, 1)
Testing set: (30, 1)
```

C5. Provide a copy of the cleaned data set.

In [12]:
```
# save the cleaned data set
df.to_csv('../data/teleco_cleaned1.csv', index=False)
train.to_csv('../data/teleco_cleaned1_train.csv', index=False)
test.to_csv('../data/teleco_cleaned1_test.csv', index=False)
```

Part IV. Model Identification and Analysis

D1. Report the annotated findings with visualizations of your data analysis, including the following elements:

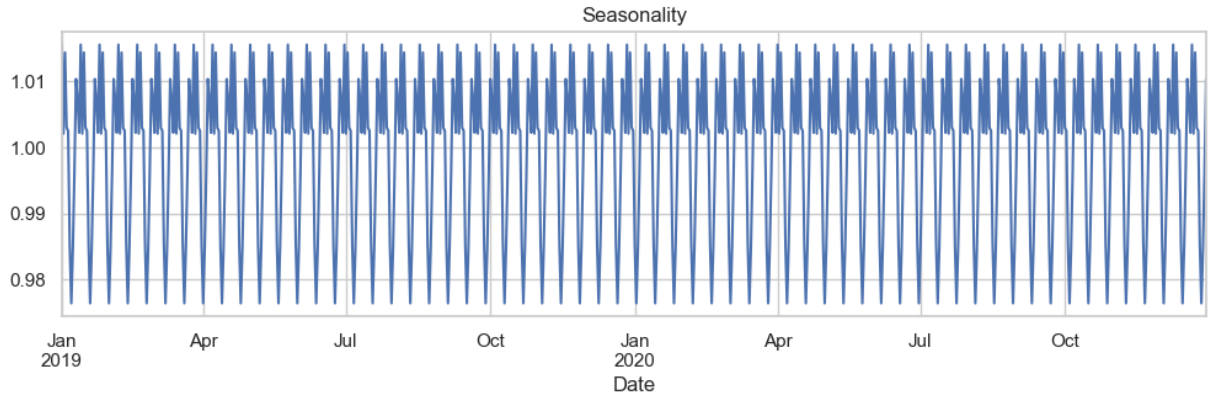
- the presence or lack of a seasonal component
- trends
- the autocorrelation function
- the spectral density
- the decomposed time series
- confirmation of the lack of trends in the residuals of the decomposed series

In [13]:
```
result = seasonal_decompose(df['Revenue'], model='multiplicable', period=12)
```

In [14]:
```
plt.title('Seasonality')
result.seasonal.plot(figsize=(12, 3))
```

Out[14]:
```
Text(0.5, 1.0, 'Seasonality')
```

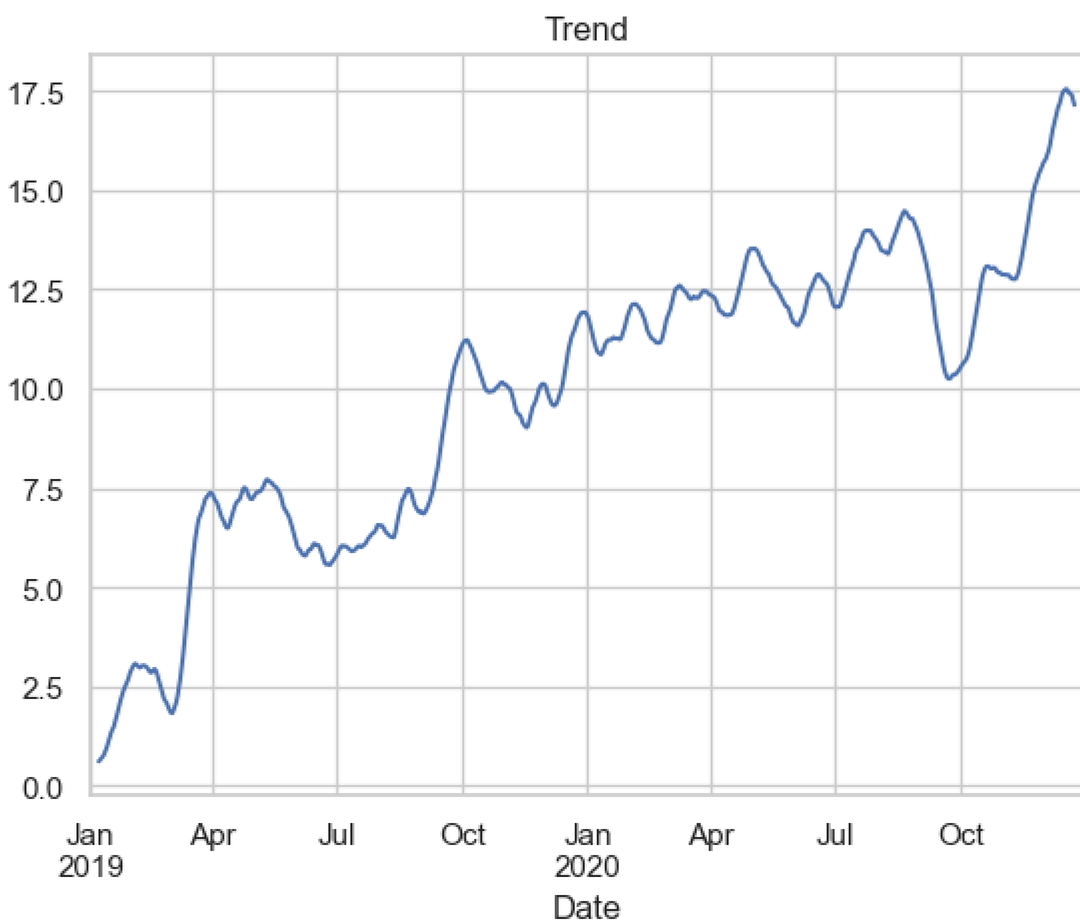
Out[14]:
```
<Axes: title={'center': 'Seasonality'}, xlabel='Date'>
```



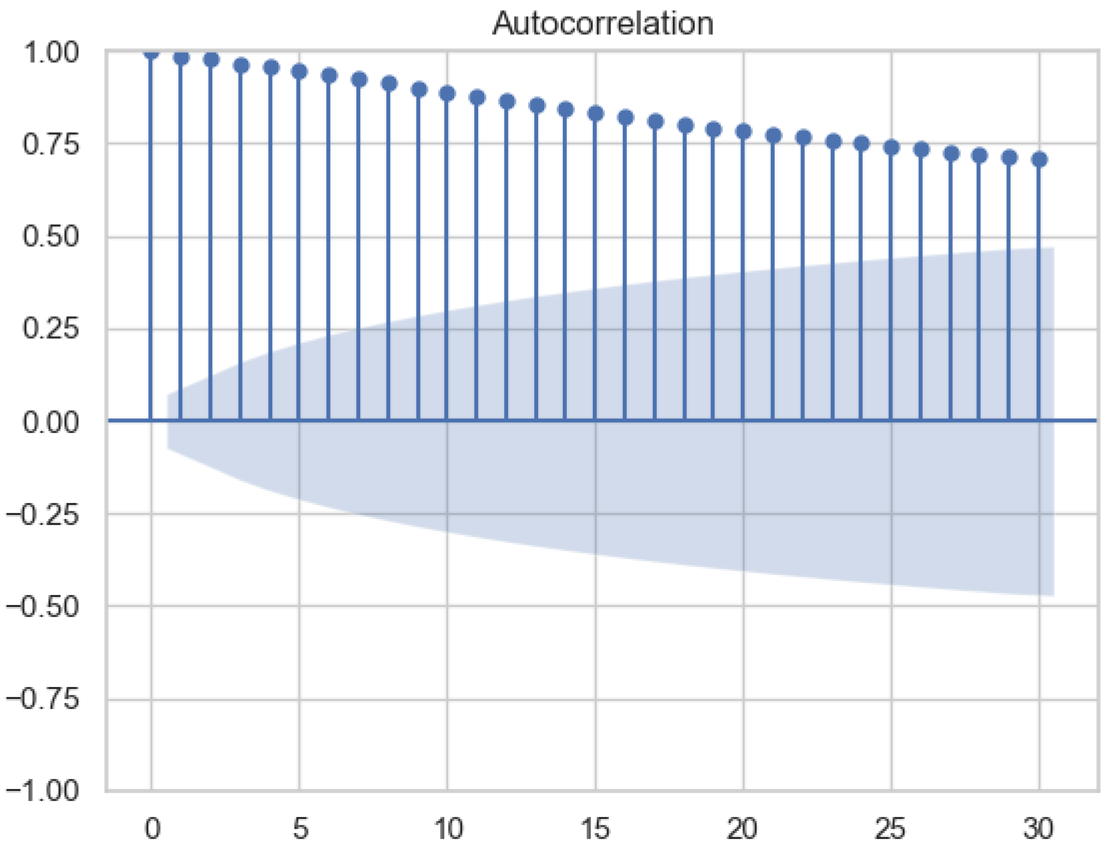
In [15]:
```
plt.title('Trend')
result.trend.plot()
```

Out[15]: `Text(0.5, 1.0, 'Trend')`

Out[15]: `<Axes: title={'center': 'Trend'}, xlabel='Date'>`



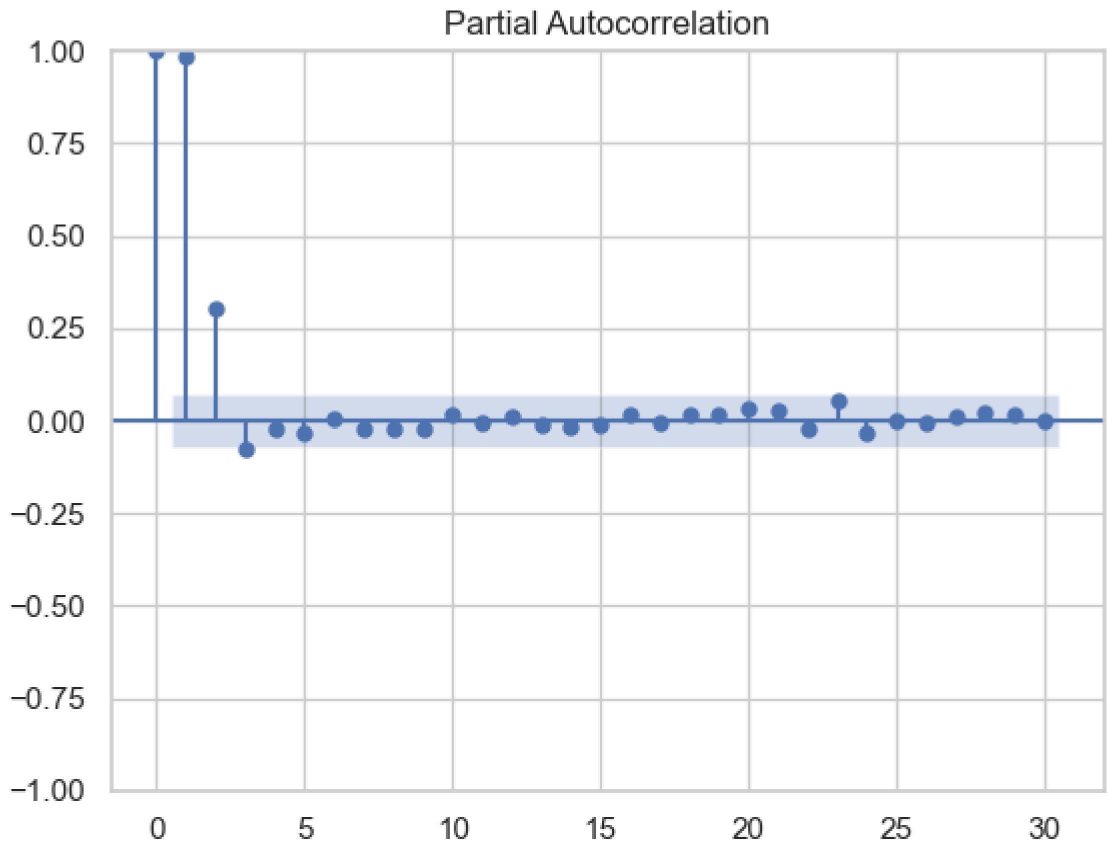
In [16]:
```
# calculate acf
acf_values = acf(df['Revenue'])

# keeping lag as 30
plot_acf(df['Revenue'], lags=30);
```

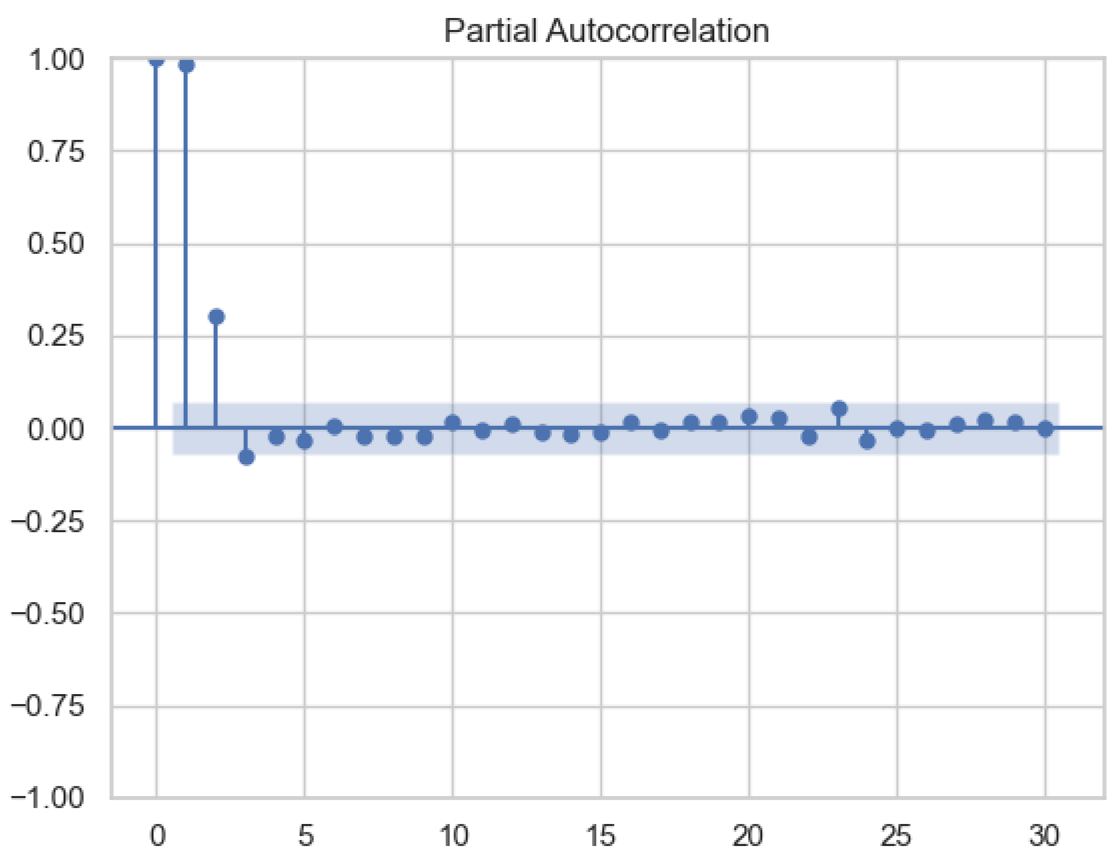


In [17]:

```
# PACF
pacf_values = (df['Revenue'])

# plot pacf
plot_pacf(df['Revenue'], lags=30)
```

Out[17]:





In [18]:

```
f, Pxx_den = signal.periodogram(df['Revenue'])
plt.semilogy(f, Pxx_den)
plt.ylim(1e-6, 1e2)
```

```python
plt.title('Spectral Density')
plt.xlabel('Frequency')
plt.ylabel('Spectral Density')
```

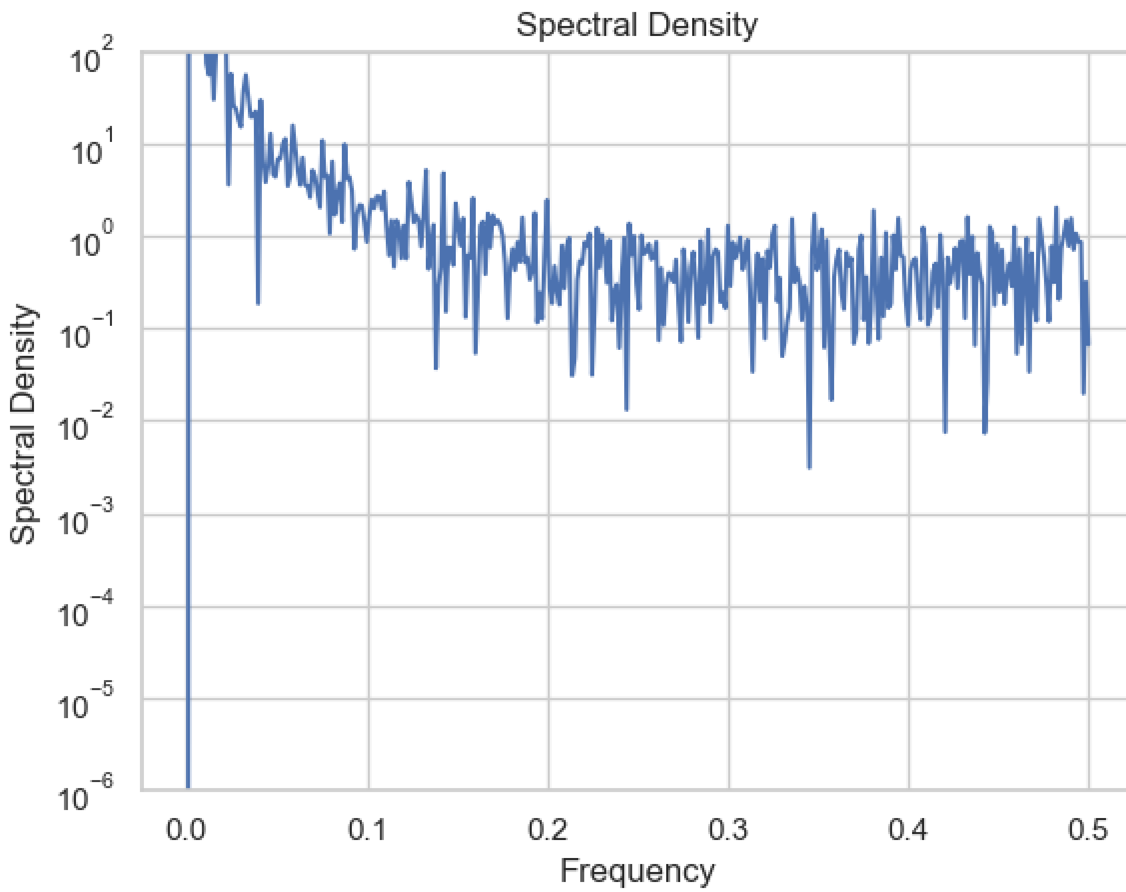
Out[18]: [<matplotlib.lines.Line2D at 0x243be06fbb0>]

Out[18]: (1e-06, 100.0)

Out[18]: Text(0.5, 1.0, 'Spectral Density')

Out[18]: Text(0.5, 0, 'Frequency')

Out[18]: Text(0, 0.5, 'Spectral Density')

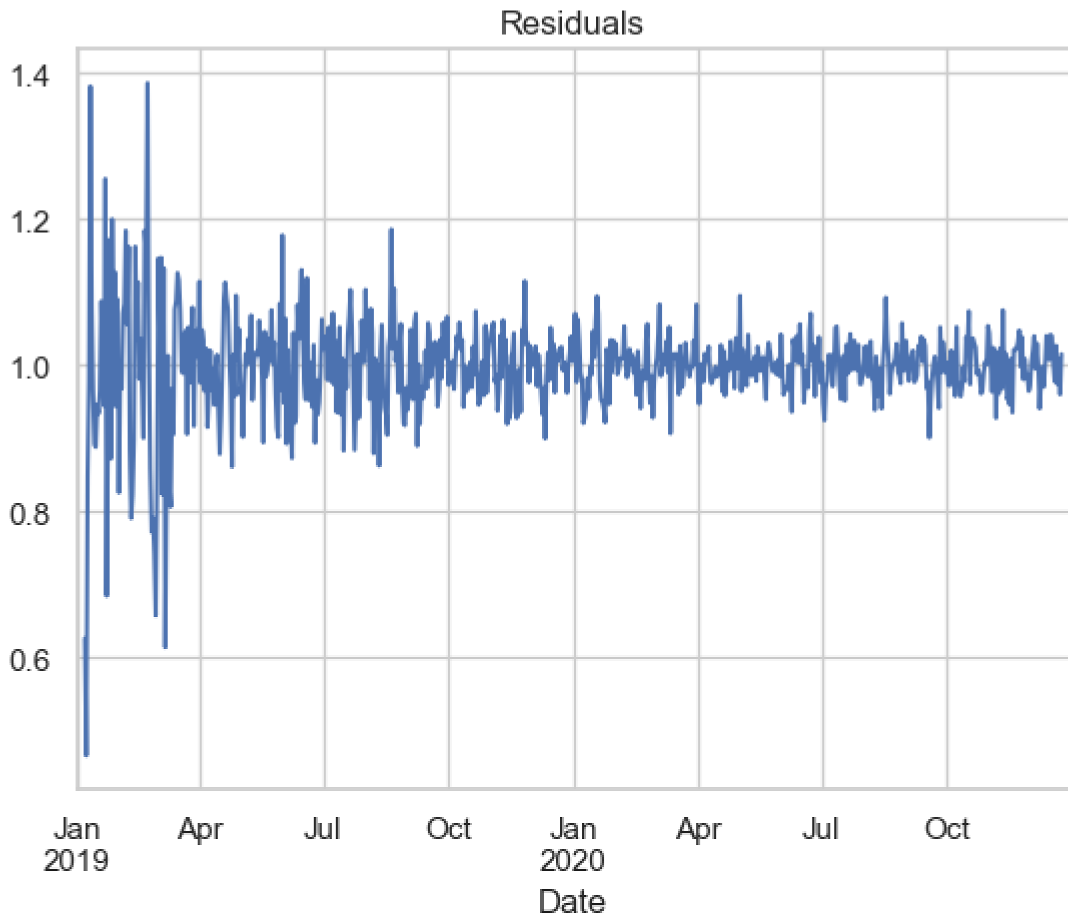


In [19]:
```python
decomp = seasonal_decompose(df['Revenue'], period=90)
```

In [20]:
```python
plt.title('Residuals')
result.resid.plot()
```

Out[20]: Text(0.5, 1.0, 'Residuals')

Out[20]: <Axes: title={'center': 'Residuals'}, xlabel='Date'>



D2. Identify an autoregressive integrated moving average (ARIMA) model that accounts for the observed trend and seasonality of the time series data.

In [21]:
```
adf_test=ADFTest(alpha=0.05)
adf_test.should_diff(df)
```

Out[21]:
```
(0.02237291702715709, False)
```

In [22]:
```
model=auto_arima(train,start_p=0,d=1,start_q=0,
          max_p=5,max_d=5,max_q=5, start_P=0,
          D=1, start_Q=0, max_P=5,max_D=5,
          max_Q=5, m=12, seasonal=True,
          error_action='warn',trace=True,
          supress_warnings=True,stepwise=True,
          random_state=493,n_fits=50)
```

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,1,0)[12]             : AIC=1550.191, Time=0.11 sec
 ARIMA(1,1,0)(1,1,0)[12]             : AIC=1200.240, Time=0.20 sec
 ARIMA(0,1,1)(0,1,1)[12]             : AIC=inf, Time=0.68 sec
 ARIMA(1,1,0)(0,1,0)[12]             : AIC=1394.111, Time=0.04 sec
 ARIMA(1,1,0)(2,1,0)[12]             : AIC=1086.815, Time=0.37 sec
 ARIMA(1,1,0)(3,1,0)[12]             : AIC=1053.850, Time=0.91 sec
 ARIMA(1,1,0)(4,1,0)[12]             : AIC=1041.732, Time=1.61 sec
 ARIMA(1,1,0)(5,1,0)[12]             : AIC=1026.291, Time=4.70 sec
 ARIMA(1,1,0)(5,1,1)[12]             : AIC=inf, Time=31.28 sec
 ARIMA(1,1,0)(4,1,1)[12]             : AIC=inf, Time=10.45 sec
 ARIMA(0,1,0)(5,1,0)[12]             : AIC=1199.175, Time=3.11 sec
 ARIMA(2,1,0)(5,1,0)[12]             : AIC=1028.127, Time=5.65 sec
 ARIMA(1,1,1)(5,1,0)[12]             : AIC=1028.161, Time=6.05 sec
 ARIMA(0,1,1)(5,1,0)[12]             : AIC=1065.220, Time=4.18 sec
 ARIMA(2,1,1)(5,1,0)[12]             : AIC=1027.416, Time=13.30 sec
 ARIMA(1,1,0)(5,1,0)[12] intercept   : AIC=1028.287, Time=19.89 sec

Best model:  ARIMA(1,1,0)(5,1,0)[12]
Total fit time: 102.539 seconds
```

In [23]:

```
model.summary()
```

Out[23]: SARIMAX Results

Dep. Variable:	y	**No. Observations:**	700
Model:	SARIMAX(1, 1, 0)x(5, 1, 0, 12)	**Log Likelihood**	-506.146
Date:	Thu, 07 Sep 2023	**AIC**	1026.291
Time:	08:29:48	**BIC**	1058.017
Sample:	01-01-2019	**HQIC**	1038.566
	- 11-30-2020		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.4779	0.035	-13.760	0.000	-0.546	-0.410
ar.S.L12	-0.8505	0.039	-21.914	0.000	-0.927	-0.774
ar.S.L24	-0.7018	0.052	-13.424	0.000	-0.804	-0.599
ar.S.L36	-0.4564	0.059	-7.686	0.000	-0.573	-0.340
ar.S.L48	-0.2886	0.052	-5.526	0.000	-0.391	-0.186
ar.S.L60	-0.1658	0.040	-4.150	0.000	-0.244	-0.087
sigma2	0.2509	0.015	17.169	0.000	0.222	0.280

Ljung-Box (L1) (Q):	0.03	**Jarque-Bera (JB):**	1.69
Prob(Q):	0.87	**Prob(JB):**	0.43
Heteroskedasticity (H):	1.05	**Skew:**	0.03
Prob(H) (two-sided):	0.71	**Kurtosis:**	2.76

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

In [24]:

```
# create final model
model = ARIMA(df['Revenue'], order=(1,1,0), seasonal_order=(5, 1, 0, 12))
results = model.fit()
results.summary()
```

Out[24]:

SARIMAX Results

Dep. Variable:	Revenue	**No. Observations:**	730
Model:	ARIMA(1, 1, 0)x(5, 1, 0, 12)	**Log Likelihood**	-535.993
Date:	Thu, 07 Sep 2023	**AIC**	1085.987
Time:	08:29:53	**BIC**	1118.012
Sample:	01-01-2019	**HQIC**	1098.353
	- 12-30-2020		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	-0.4781	0.034	-14.173	0.000	-0.544	-0.412
ar.S.L12	-0.8591	0.039	-22.126	0.000	-0.935	-0.783
ar.S.L24	-0.7058	0.052	-13.577	0.000	-0.808	-0.604
ar.S.L36	-0.4672	0.058	-8.071	0.000	-0.581	-0.354
ar.S.L48	-0.3017	0.050	-6.017	0.000	-0.400	-0.203
ar.S.L60	-0.1672	0.039	-4.304	0.000	-0.243	-0.091
sigma2	0.2565	0.015	17.414	0.000	0.228	0.285

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	2.12
Prob(Q):	0.95	**Prob(JB):**	0.35
Heteroskedasticity (H):	1.08	**Skew:**	0.00
Prob(H) (two-sided):	0.54	**Kurtosis:**	2.73

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

D3. Perform a forecast using the derived ARIMA model identified in part D2.

In [25]:

```
# make forecast outside of sample
results.forecast(90)
```

```
Out[25]: 2020-12-31    16.594268
         2021-01-01    16.201477
         2021-01-02    16.416889
         2021-01-03    16.375951
         2021-01-04    16.497808
         2021-01-05    16.215819
         2021-01-06    16.601740
         2021-01-07    16.464448
         2021-01-08    16.659518
         2021-01-09    16.859056
         2021-01-10    17.032650
         2021-01-11    17.195227
         2021-01-12    17.189922
         2021-01-13    16.697272
         2021-01-14    16.955515
         2021-01-15    17.086065
         2021-01-16    17.123561
         2021-01-17    16.911494
         2021-01-18    17.440188
         2021-01-19    17.226752
         2021-01-20    17.413404
         2021-01-21    17.581965
         2021-01-22    17.832530
         2021-01-23    17.763408
         2021-01-24    17.766160
         2021-01-25    17.422238
         2021-01-26    17.640157
         2021-01-27    17.704545
         2021-01-28    17.754207
         2021-01-29    17.687338
         2021-01-30    18.038688
         2021-01-31    17.988914
         2021-02-01    18.165736
         2021-02-02    18.340504
         2021-02-03    18.336846
         2021-02-04    18.451854
         2021-02-05    18.428935
         2021-02-06    18.066007
         2021-02-07    18.322782
         2021-02-08    18.371551
         2021-02-09    18.300440
         2021-02-10    18.290960
         2021-02-11    18.531198
         2021-02-12    18.477955
         2021-02-13    18.651189
         2021-02-14    18.837363
         2021-02-15    18.775749
         2021-02-16    18.993823
         2021-02-17    18.857959
         2021-02-18    18.520401
         2021-02-19    18.750903
         2021-02-20    18.882170
         2021-02-21    18.789713
         2021-02-22    18.843384
         2021-02-23    19.152852
         2021-02-24    19.014700
```

```
2021-02-25    19.163831
2021-02-26    19.331993
2021-02-27    19.363840
2021-02-28    19.415336
2021-03-01    19.302271
2021-03-02    19.099974
2021-03-03    19.179057
2021-03-04    19.291289
2021-03-05    19.174392
2021-03-06    19.198109
2021-03-07    19.385064
2021-03-08    19.320260
2021-03-09    19.560646
2021-03-10    19.594409
2021-03-11    19.551563
2021-03-12    19.701363
2021-03-13    19.654443
2021-03-14    19.298735
2021-03-15    19.513304
2021-03-16    19.582240
2021-03-17    19.568270
2021-03-18    19.490834
2021-03-19    19.815723
2021-03-20    19.712809
2021-03-21    19.897855
2021-03-22    20.057780
2021-03-23    20.104047
2021-03-24    20.222006
2021-03-25    20.161331
2021-03-26    19.810963
2021-03-27    20.023301
2021-03-28    20.120105
2021-03-29    20.079333
2021-03-30    20.038843
Freq: D, Name: predicted_mean, dtype: float64
```

In [26]:
```
index_future_dates = pd.date_range(start='2020-12-02', end='2021-01-01')
print(index_future_dates)
```

```
DatetimeIndex(['2020-12-02', '2020-12-03', '2020-12-04', '2020-12-05',
               '2020-12-06', '2020-12-07', '2020-12-08', '2020-12-09',
               '2020-12-10', '2020-12-11', '2020-12-12', '2020-12-13',
               '2020-12-14', '2020-12-15', '2020-12-16', '2020-12-17',
               '2020-12-18', '2020-12-19', '2020-12-20', '2020-12-21',
               '2020-12-22', '2020-12-23', '2020-12-24', '2020-12-25',
               '2020-12-26', '2020-12-27', '2020-12-28', '2020-12-29',
               '2020-12-30', '2020-12-31', '2021-01-01'],
              dtype='datetime64[ns]', freq='D')
```

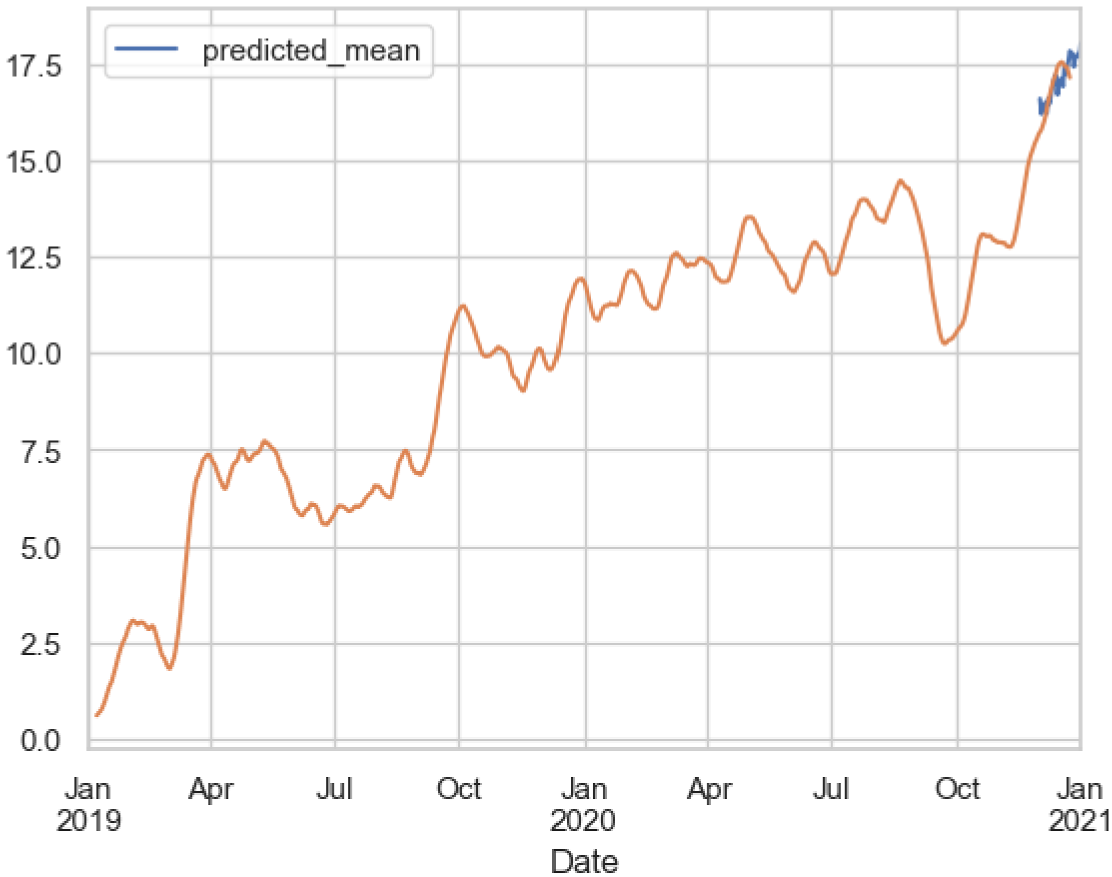
In [27]:
```
pred = results.predict(start=len(df), end=len(df)+30, typ='levels')

pred.index = index_future_dates
print(pred)
```

```
2020-12-02    16.594268
2020-12-03    16.201477
2020-12-04    16.416889
2020-12-05    16.375951
2020-12-06    16.497808
2020-12-07    16.215819
2020-12-08    16.601740
2020-12-09    16.464448
2020-12-10    16.659518
2020-12-11    16.859056
2020-12-12    17.032650
2020-12-13    17.195227
2020-12-14    17.189922
2020-12-15    16.697272
2020-12-16    16.955515
2020-12-17    17.086065
2020-12-18    17.123561
2020-12-19    16.911494
2020-12-20    17.440188
2020-12-21    17.226752
2020-12-22    17.413404
2020-12-23    17.581965
2020-12-24    17.832530
2020-12-25    17.763408
2020-12-26    17.766160
2020-12-27    17.422238
2020-12-28    17.640157
2020-12-29    17.704545
2020-12-30    17.754207
2020-12-31    17.687338
2021-01-01    18.038688
Freq: D, Name: predicted_mean, dtype: float64
```

In [28]:
```
pred.plot(legend=True)
result.trend.plot()
```

Out[28]:
```
<Axes: >
```

Out[28]:
```
<Axes: xlabel='Date'>
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [29]:

```
prediction = pd.DataFrame(results.predict(n_periods = 12),index=test.index)
prediction.columns = ['Revenue']
prediction
```

Out[29]:

Date	Revenue
2020-12-01	15.439003
2020-12-02	15.205029
2020-12-03	15.562525
2020-12-04	15.895255
2020-12-05	16.238106
2020-12-06	16.282606
2020-12-07	16.916224
2020-12-08	16.241115
2020-12-09	15.931532
2020-12-10	16.064233
2020-12-11	16.859898
2020-12-12	16.429369
2020-12-13	17.124037
2020-12-14	17.586317
2020-12-15	17.889969
2020-12-16	17.600538
2020-12-17	17.916328
2020-12-18	18.304870
2020-12-19	18.371859
2020-12-20	17.005974
2020-12-21	17.634535
2020-12-22	17.680886
2020-12-23	17.772028
2020-12-24	16.995105
2020-12-25	17.577145
2020-12-26	17.292523
2020-12-27	16.927520
2020-12-28	17.677000
2020-12-29	17.537108

	Revenue
Date	
2020-12-30	16.709108

In [30]:
```
plt.figure(figsize=(8,5))
plt.plot(train,label="Training")
plt.plot(test,label="Test")
plt.plot(prediction,label="Predicted")
plt.legend(loc = 'upper left')
```

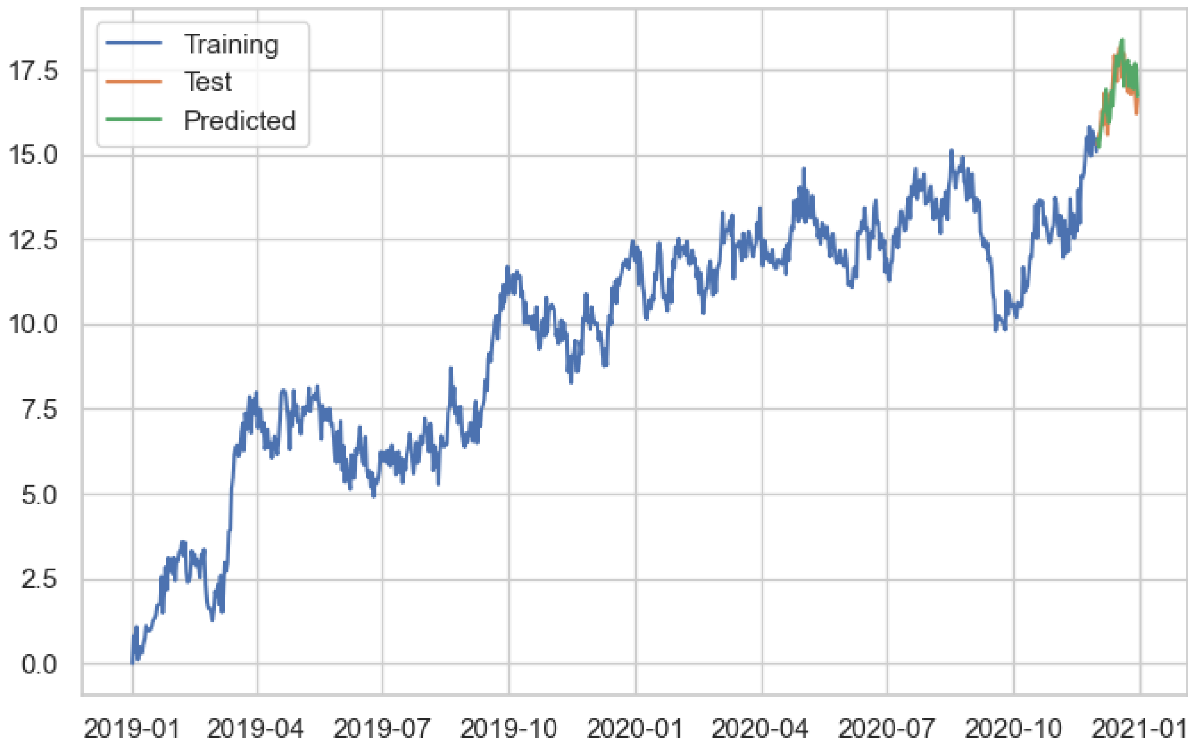
Out[30]: <Figure size 800x500 with 0 Axes>

Out[30]: [<matplotlib.lines.Line2D at 0x244b8476790>]

Out[30]: [<matplotlib.lines.Line2D at 0x243be794520>]

Out[30]: [<matplotlib.lines.Line2D at 0x243be794760>]

Out[30]: <matplotlib.legend.Legend at 0x244780e9550>



In [31]:
```
test['predicted_revenue'] = prediction
r2_score(test['Revenue'], test['predicted_revenue'])

# R2 Score for test data set
```

Out[31]: 0.3381274048582671

In [32]:
```
diff_forecast = results.get_forecast(steps=30)
mean_forecast = diff_forecast.predicted_mean
confidence_intervals = diff_forecast.conf_int()
lower_limits = confidence_intervals.loc[:, 'lower Revenue']
```

```
upper_limits = confidence_intervals.loc[:, 'upper Revenue']

prediction = results.get_prediction(start=len(df), end=len(df)+90)
mean_prediction = prediction.predicted_mean
confidence_intervals = prediction.conf_int()
lower_limits = confidence_intervals.loc[:, 'lower Revenue']
upper_limits = confidence_intervals.loc[:, 'upper Revenue']

plt.plot(test.index, test, label='Observed')
plt.plot(mean_prediction.index, mean_prediction, color='r', label='Forecast')
plt.fill_between(lower_limits.index, lower_limits, upper_limits, color='pink')
plt.title('Forecast comparing with test data')
plt.xlabel('Date')
plt.ylabel('Revenue in Million $')
plt.xticks(rotation=30, fontsize=10)
plt.legend()
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x244f94cbb80>,
 <matplotlib.lines.Line2D at 0x243cef23580>]
```

Out[32]:
```
[<matplotlib.lines.Line2D at 0x243cef23ac0>]
```

Out[32]:
```
<matplotlib.collections.PolyCollection at 0x243cef23790>
```

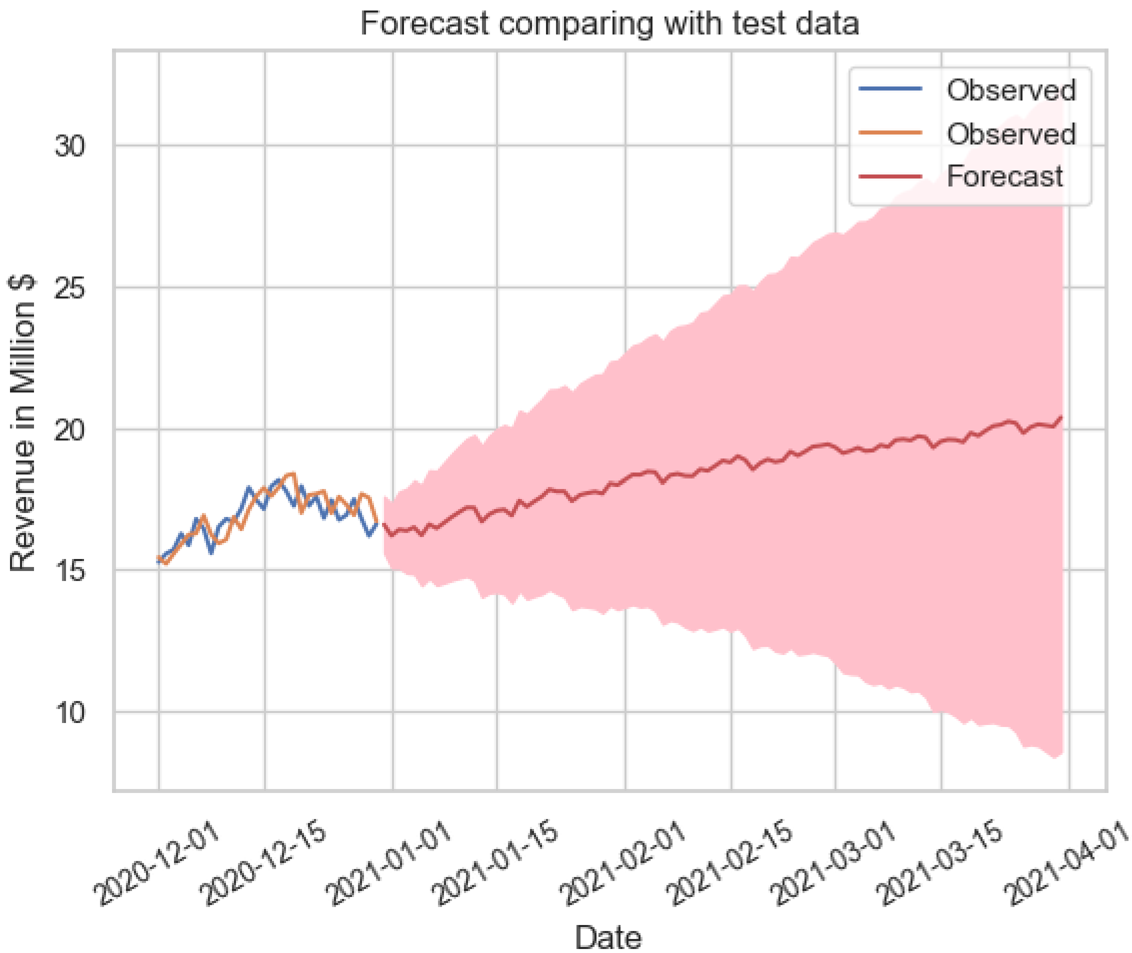
Out[32]:
```
Text(0.5, 1.0, 'Forecast comparing with test data')
```

Out[32]:
```
Text(0.5, 0, 'Date')
```

Out[32]:
```
Text(0, 0.5, 'Revenue in Million $')
```

Out[32]:
```
(array([18597., 18611., 18628., 18642., 18659., 18673., 18687., 18701.,
        18718.]),
 [Text(18597.0, 0, '2020-12-01'),
  Text(18611.0, 0, '2020-12-15'),
  Text(18628.0, 0, '2021-01-01'),
  Text(18642.0, 0, '2021-01-15'),
  Text(18659.0, 0, '2021-02-01'),
  Text(18673.0, 0, '2021-02-15'),
  Text(18687.0, 0, '2021-03-01'),
  Text(18701.0, 0, '2021-03-15'),
  Text(18718.0, 0, '2021-04-01')])
```

Out[32]:
```
<matplotlib.legend.Legend at 0x243cef23fd0>
```



In [33]:
```
mean_prediction.tail()
```

Out[33]:
```
2021-03-27    20.023301
2021-03-28    20.120105
2021-03-29    20.079333
2021-03-30    20.038843
2021-03-31    20.362539
Freq: D, Name: predicted_mean, dtype: float64
```

D4. Provide the output and calculations of the analysis you performed.

Filename: "D213 Performance Assessment Task 1 (Rev. 2).ipynb"

D5. Provide the code used to support the implementation of the time series model.

Filename: "D213 Performance Assessment Task 1 (Rev. 2).ipynb"

Part V. Data Summary and Implications

E1. Discuss the results of your data analysis, including the following points:

- the selection of an ARIMA model
- the prediction interval of the forecast
- a justification of the forecast length
- the model evaluation procedure and error metric

The final ARIMA model was based on the results of Auto ARIMA (Best model: ARIMA(1,1,0)(5,1,0)[12]) which takes into account trend and seasonality of the data set. The prediction interval of the forecast is 180 days and can be made using the .predict() or .forecast() methods. Forecast length of 180 is enough information to make changes in preparation for the next quarter or so. The final model was evaluated with R2. It "measures the strength of the relationship between your model and the dependent variable" (Frost, 2018). Although 33.81 is a low result, a low R2 doesn't necessarily mean the model is bad (Frost, 2018).

E2. Provide an annotated visualization of the forecast of the final model compared to the test set.

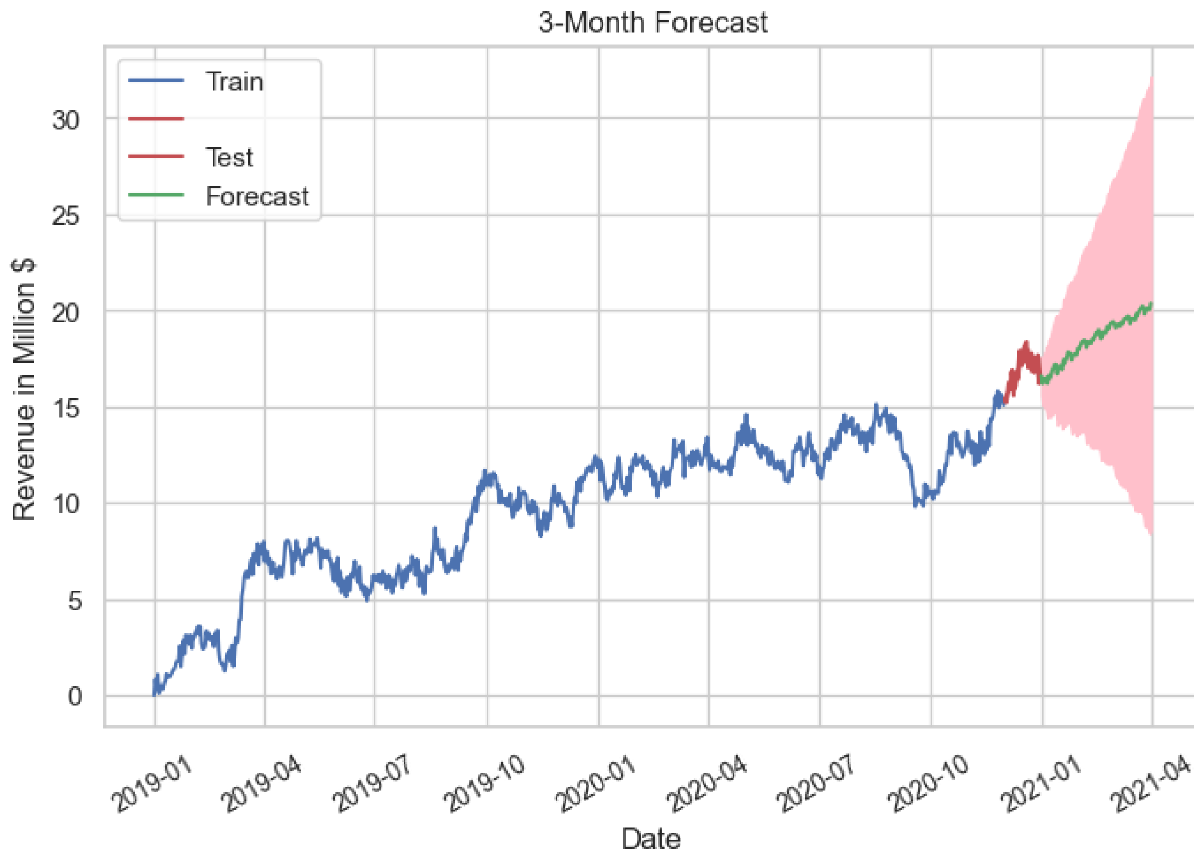
In [53]:

```
plt.figure(figsize=(8,5))
plt.plot(train, color='b', label='Train')
plt.plot(test, color='r', label='Test')
plt.plot(mean_prediction, color='g', label='Forecast')
plt.fill_between(lower_limits.index, lower_limits, upper_limits, color='pink')
plt.title('3-Month Forecast')
plt.xlabel('Date')
plt.ylabel('Revenue in Million $')
plt.xticks(rotation=30, fontsize=10)
plt.legend(['Train', '', 'Test', 'Forecast'], loc='upper left')
```

```
Out[53]: <Figure size 800x500 with 0 Axes>
Out[53]: [<matplotlib.lines.Line2D at 0x243ea336f70>]
Out[53]: [<matplotlib.lines.Line2D at 0x243ea345160>,
          <matplotlib.lines.Line2D at 0x243ea345190>]
Out[53]: [<matplotlib.lines.Line2D at 0x243ea3456d0>]
Out[53]: <matplotlib.collections.PolyCollection at 0x243ea3457f0>
Out[53]: Text(0.5, 1.0, '3-Month Forecast')
Out[53]: Text(0.5, 0, 'Date')
Out[53]: Text(0, 0.5, 'Revenue in Million $')
```

```
Out[53]: (array([17897., 17987., 18078., 18170., 18262., 18353., 18444., 18536.,
                18628., 18718.]),
          [Text(17897.0, 0, '2019-01'),
           Text(17987.0, 0, '2019-04'),
           Text(18078.0, 0, '2019-07'),
           Text(18170.0, 0, '2019-10'),
           Text(18262.0, 0, '2020-01'),
           Text(18353.0, 0, '2020-04'),
           Text(18444.0, 0, '2020-07'),
           Text(18536.0, 0, '2020-10'),
           Text(18628.0, 0, '2021-01'),
           Text(18718.0, 0, '2021-04')])
```

```
Out[53]: <matplotlib.legend.Legend at 0x243ea249370>
```



E3. Recommend a course of action based on your results.

The forecast data estimates that revenue will be at $20.36 million. Visual inspection of the plot also reveals an upward trend for the forecasted months. As such, I recommend a conservative approach to configuring organization operations for this quarter.

Part VI. Reporting

F. With the information from part E, create your report using an industry-relevant interactive development environment (e.g., an R Markdown document, a Jupyter Notebook). Include a PDF or HTML document of your executed notebook presentation.

Filename: "D213 Performance Assessment Task 1 (Rev. 2).pdf"

G. Cite the web sources you used to acquire third-party code to support the application.

- https://github.com/ecdedios/code-snippets/blob/main/notebooks/master.ipynb
- https://www.datacamp.com/tutorial/matplotlib-time-series-line-plot
- https://towardsdatascience.com/finding-seasonal-trends-in-time-series-data-with-python-ce10c37aa861
- https://towardsdatascience.com/time-series-decomposition-in-python-8acac385a5b2
- https://analyticsindiamag.com/what-are-autocorrelation-and-partial-autocorrelation-in-time-series-data/
- https://github.com/mkosaka1/AirPassengers_TimeSeries/blob/master/Time_Series.ipynb

H. Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

- https://www.statisticssolutions.com/stationary-data-assumption-in-time-series-analysis/
- https://builtin.com/data-science/time-series-python
- https://statisticsbyjim.com/regression/interpret-r-squared-regression/

In [54]:
```
print('Succesful run!')
```

```
Succesful run!
```

In [ ]: