D603 Performance Assessment Task 3 Time Series

March 10, 2025

[103]:
```python
# import all necessary packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn
from statsmodels.tsa.stattools import acf
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
import warnings
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima.model import ARIMA
from sklearn.metrics import mean_squared_error
import pmdarima as pm
from datetime import datetime
from pmdarima import auto_arima
warnings.filterwarnings('ignore')
```

[2]:
```python
# Pulling in the data [In text citation: Bowne-Anderson, H. (n.d)]
time_data = pd.read_csv("C:/Users/cfman/OneDrive/Desktop/WGUClasses/
↪D603MachineLearning/Task3/churn_clean.csv")
```

[3]:
```python
time_data.head()
```

[3]:

	Day	Revenue
0	1	0.000000
1	2	0.000793
2	3	0.825542
3	4	0.320332
4	5	1.082554

[4]:
```python
## Adding an index that consists of the date in datetime form
time_data['Date'] = (pd.date_range(start=datetime(2022, 1, 1),
            periods=time_data.shape[0], freq='24H'))
# Set the Date as an index
time_data.set_index('Date',inplace=True)
time_data
```

[4]:
```
            Day    Revenue
Date
2022-01-01    1   0.000000
2022-01-02    2   0.000793
2022-01-03    3   0.825542
2022-01-04    4   0.320332
2022-01-05    5   1.082554
…           …        …
2023-12-28  727  16.931559
2023-12-29  728  17.490666
2023-12-30  729  16.803638
2023-12-31  730  16.194813
2024-01-01  731  16.620798

[731 rows x 2 columns]
```

0.1 D1

[5]:
```
## Plotting the initial line plot for Part D1
seaborn.set(style = 'dark')
plot = seaborn.lineplot(data = time_data, x = time_data.index, y = 'Revenue')
plot.set(title = 'Company Revenue Over Time')
plt.xlabel('Time')
plt.xticks(rotation = 45)
plt.grid()
plt.show()
```

0.2 D2

[6]:
```python
## Some sanity checks of the data set
print("Length of time_data: ", len(time_data))
print("First instance: ", time_data['Day'].iloc[0])
print("Last instance: ", time_data['Day'].iloc[-1])
```

```
Length of time_data:  731
First instance:  1
Last instance:  731
```

[7]:
```python
# Before we get started into the actual cleaning, I want to check for duplicate␣
↪values in case we need to edit those first
# I am also going to check the shape of the dataframe to verify the rows and␣
↪columns
# [In text citation: Bowne-Anderson, H. (n.d)]
print(time_data.shape)
```

```
duplicates = time_data.duplicated(keep = False)
duplicates.value_counts()
```

```
(731, 2)
```

[7]:
```
False    731
dtype: int64
```

[8]:
```
# I want to check which have missing values here
# [In text citation: Bowne-Anderson, H. (n.d)]
time_data.isna().sum()
#time_data.isnull().sum()
```

[8]:
```
Day        0
Revenue    0
dtype: int64
```

[9]:
```
## Checking for no duplicate occurences in the Day column. Should match length␣
↪of the dataframe
print("Count of unique values in Day column: ", time_data['Day'].nunique())
```

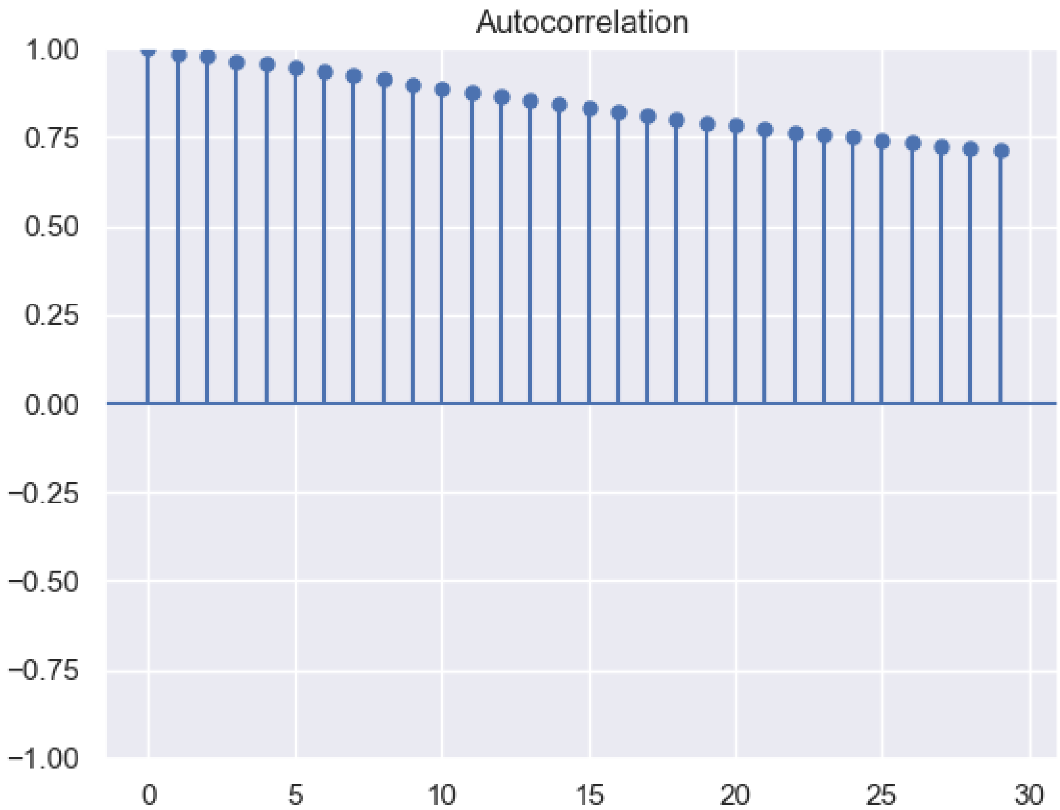
```
Count of unique values in Day column:  731
```

0.3 D3

[10]:
```
# Compute the acf array of HRB
acf_array = acf(time_data['Revenue'])
print(acf_array)

# Plot the acf function
plot_acf(time_data['Revenue'], alpha = 1)
plt.grid()
plt.show()
```

```
[1.         0.98375067 0.97679376 0.96584953 0.95545834 0.94515237
 0.93469998 0.92356074 0.91259674 0.9007179  0.88986482 0.87820542
 0.86783127 0.85669648 0.84568014 0.83433728 0.82369773 0.81266219
 0.80261913 0.79240124 0.78364098 0.77521764 0.76563762 0.75985141
 0.75066964 0.742678   0.73537013 0.72746968 0.72092685]
```



[11]:
```
#perform augmented Dickey-Fuller test
adfuller(time_data['Revenue'])
```

[11]:
```
(-1.9246121573101809,
 0.32057281507939783,
 1,
 729,
 {'1%': -3.4393520240470554,
  '5%': -2.8655128165959236,
  '10%': -2.5688855736949163},
 965.0609576707513)
```

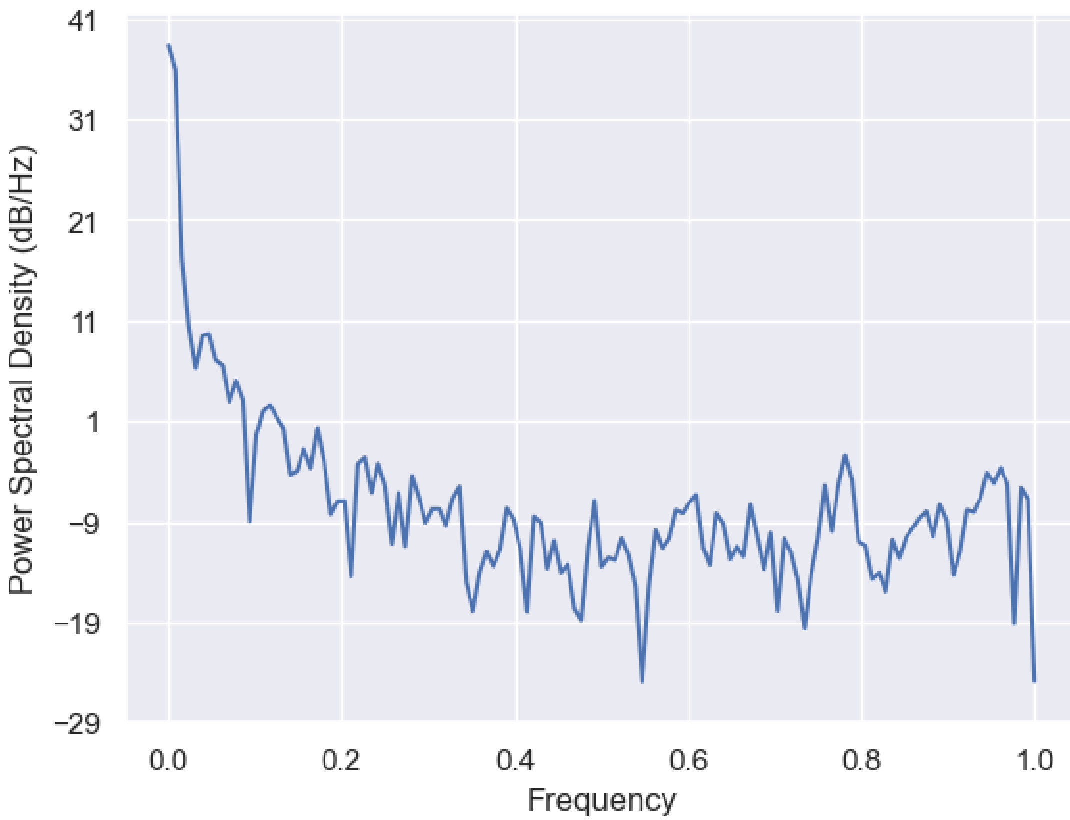
0.4 D3/E1C

[12]:
```
## Plotting the spectral density
plt.psd(time_data['Revenue'])
```

[12]:
```
(array([6.97387711e+03, 3.91439441e+03, 5.47611144e+01, 1.10791953e+01,
        4.25439684e+00, 8.90767103e+00, 9.32436541e+00, 5.07699490e+00,
        4.50820186e+00, 1.97142105e+00, 3.17930644e+00, 2.05465010e+00,
        1.27856111e-01, 9.12357670e-01, 1.58928494e+00, 1.82600308e+00,
```

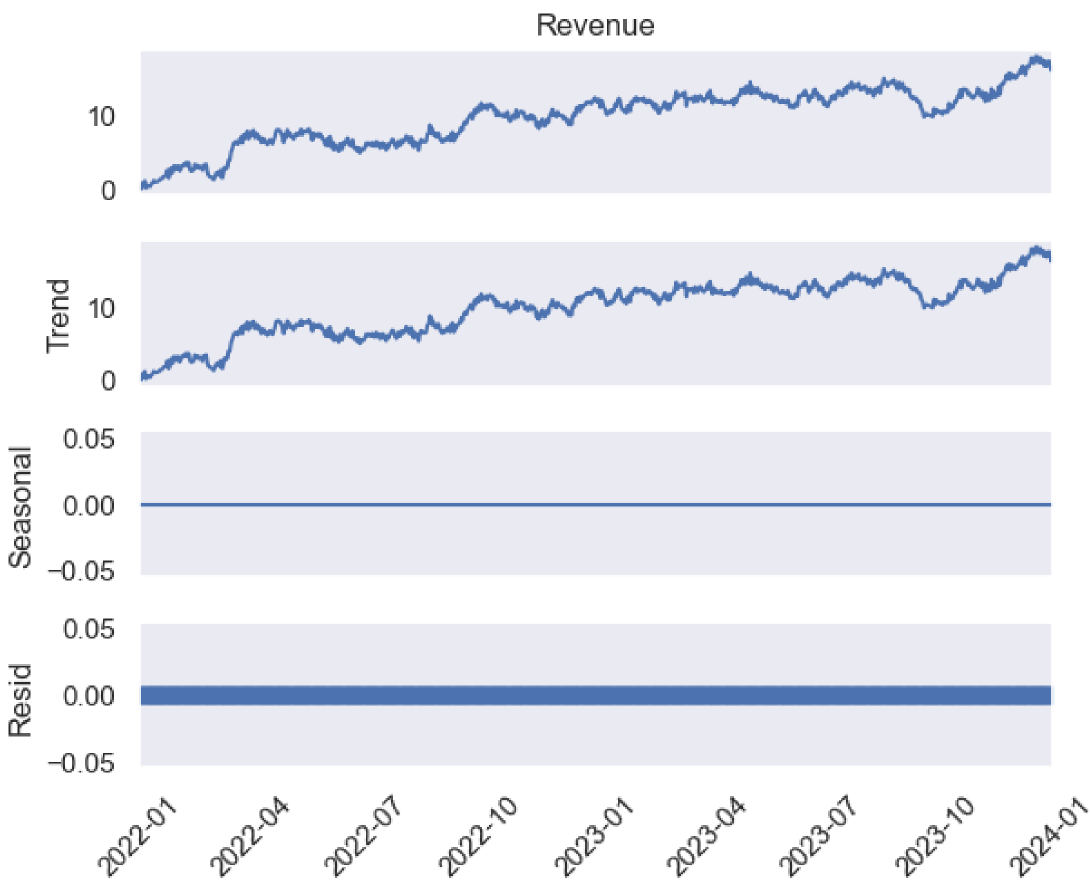
```
       1.37345831e+00, 1.08602805e+00, 3.71664573e-01, 4.01447712e-01,
       6.67019362e-01, 4.28134847e-01, 1.08303874e+00, 5.05461139e-01,
       1.49612021e-01, 2.01156666e-01, 2.01929017e-01, 3.62482513e-02,
       4.74386760e-01, 5.46619399e-01, 2.44460915e-01, 4.75225583e-01,
       2.85007556e-01, 7.59631735e-02, 2.43183009e-01, 7.19503505e-02,
       3.60475608e-01, 2.18971480e-01, 1.23421972e-01, 1.69234109e-01,
       1.69371921e-01, 1.14748035e-01, 2.14144408e-01, 2.82736486e-01,
       3.17059554e-02, 1.62458482e-02, 3.98324693e-02, 6.38895068e-02,
       4.57061837e-02, 6.54718394e-02, 1.72598760e-01, 1.32802101e-01,
       6.74716970e-02, 1.61089989e-02, 1.42266347e-01, 1.23640967e-01,
       4.30737122e-02, 8.17619769e-02, 3.95102804e-02, 4.74833201e-02,
       1.73085344e-02, 1.32485092e-02, 7.14595244e-02, 2.04348554e-01,
       4.51051907e-02, 5.55609932e-02, 5.26341931e-02, 8.68327071e-02,
       5.86340275e-02, 2.88124318e-02, 3.23398046e-03, 2.85277556e-02,
       1.03859619e-01, 6.85761289e-02, 8.53467489e-02, 1.65815053e-01,
       1.53444767e-01, 1.97776090e-01, 2.33413022e-01, 6.95161339e-02,
       4.68937467e-02, 1.53353843e-01, 1.21391009e-01, 5.34892233e-02,
       7.17250734e-02, 5.68272472e-02, 1.87708648e-01, 8.92439810e-02,
       4.25453689e-02, 9.79721726e-02, 1.63975102e-02, 8.61687997e-02,
       6.31239429e-02, 3.34886298e-02, 1.09875888e-02, 3.83717041e-02,
       8.81504336e-02, 2.89832055e-01, 1.01621880e-01, 2.88982189e-01,
       5.81784635e-01, 3.29483923e-01, 8.00152896e-02, 7.31104074e-02,
       3.39216170e-02, 3.93543037e-02, 2.55189466e-02, 8.36160222e-02,
       5.48857747e-02, 8.78315482e-02, 1.10183077e-01, 1.37188728e-01,
       1.60562556e-01, 9.01524835e-02, 1.87554502e-01, 1.29971998e-01,
       3.71787543e-02, 6.31650566e-02, 1.65356465e-01, 1.58164919e-01,
       2.17681939e-01, 3.85248684e-01, 3.04221135e-01, 4.32125608e-01,
       2.94750935e-01, 1.22618052e-02, 2.74845469e-01, 2.10555482e-01,
       3.28604031e-03]),
array([0.       , 0.0078125, 0.015625 , 0.0234375, 0.03125  , 0.0390625,
       0.046875 , 0.0546875, 0.0625   , 0.0703125, 0.078125 , 0.0859375,
       0.09375  , 0.1015625, 0.109375 , 0.1171875, 0.125    , 0.1328125,
       0.140625 , 0.1484375, 0.15625  , 0.1640625, 0.171875 , 0.1796875,
       0.1875   , 0.1953125, 0.203125 , 0.2109375, 0.21875  , 0.2265625,
       0.234375 , 0.2421875, 0.25     , 0.2578125, 0.265625 , 0.2734375,
       0.28125  , 0.2890625, 0.296875 , 0.3046875, 0.3125   , 0.3203125,
       0.328125 , 0.3359375, 0.34375  , 0.3515625, 0.359375 , 0.3671875,
       0.375    , 0.3828125, 0.390625 , 0.3984375, 0.40625  , 0.4140625,
       0.421875 , 0.4296875, 0.4375   , 0.4453125, 0.453125 , 0.4609375,
       0.46875  , 0.4765625, 0.484375 , 0.4921875, 0.5      , 0.5078125,
       0.515625 , 0.5234375, 0.53125  , 0.5390625, 0.546875 , 0.5546875,
       0.5625   , 0.5703125, 0.578125 , 0.5859375, 0.59375  , 0.6015625,
       0.609375 , 0.6171875, 0.625    , 0.6328125, 0.640625 , 0.6484375,
       0.65625  , 0.6640625, 0.671875 , 0.6796875, 0.6875   , 0.6953125,
       0.703125 , 0.7109375, 0.71875  , 0.7265625, 0.734375 , 0.7421875,
       0.75     , 0.7578125, 0.765625 , 0.7734375, 0.78125  , 0.7890625,
       0.796875 , 0.8046875, 0.8125   , 0.8203125, 0.828125 , 0.8359375,
```

```
0.84375   , 0.8515625, 0.859375 , 0.8671875, 0.875    , 0.8828125,
0.890625  , 0.8984375, 0.90625  , 0.9140625, 0.921875 , 0.9296875,
0.9375    , 0.9453125, 0.953125 , 0.9609375, 0.96875  , 0.9765625,
0.984375  , 0.9921875, 1.       ]))
```



0.5 D3/E1D

[13]:
```
## Plotting the decomposed time series.
result = seasonal_decompose(time_data['Revenue'], model='additive', period=1)
result.plot()
plt.xticks(rotation = 45)
plt.show()
```



[14]:
```
## Creating a new dataframe of the differenced data to make the time series␣
↪stationary
time_data_st = time_data.diff().dropna().drop('Day', axis = 1)
print(time_data_st.head(10))
```

```
             Revenue
Date
2022-01-02  0.000793
2022-01-03  0.824749
2022-01-04 -0.505210
2022-01-05  0.762222
2022-01-06 -0.974900
2022-01-07  0.386248
2022-01-08 -0.117203
2022-01-09 -0.072624
2022-01-10  0.287673
2022-01-11  0.139271
```

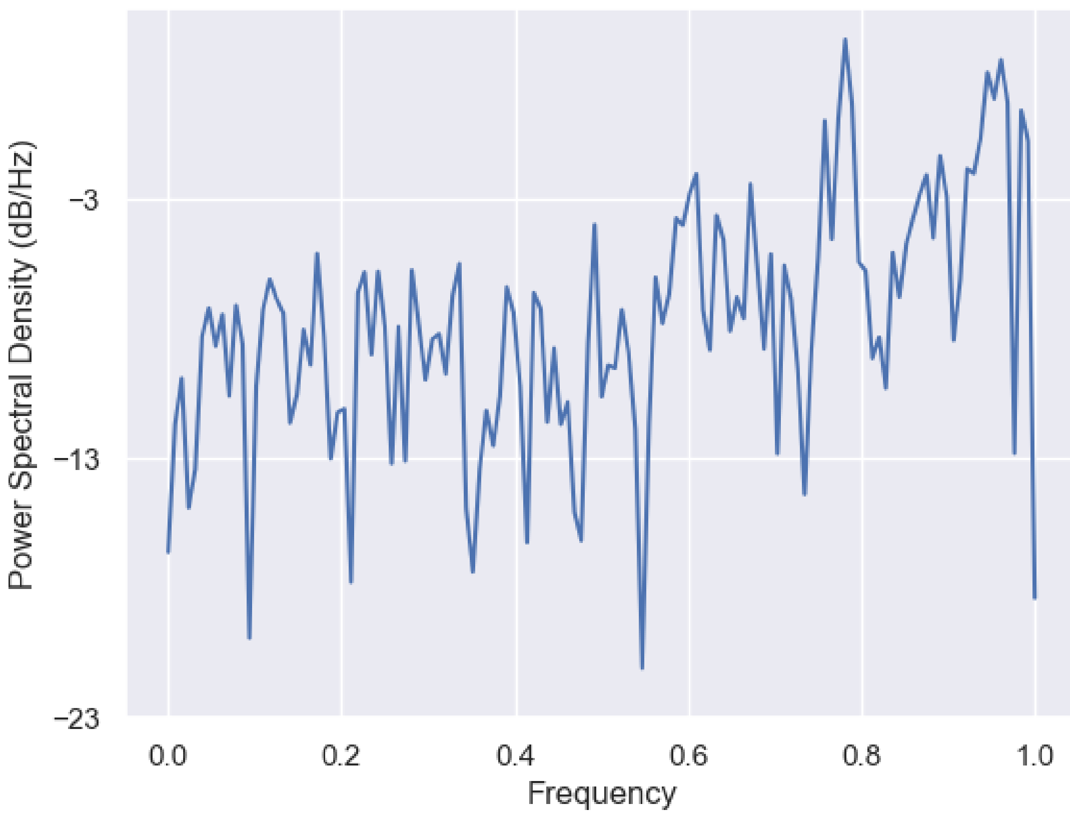
[15]:
```python
# Check for stationarity again
result_st = adfuller(time_data_st['Revenue'])
print('ADF Statistic:', result_st[0])
print('p-value:', result_st[1])
```

```
ADF Statistic: -44.874527193875984
p-value: 0.0
```

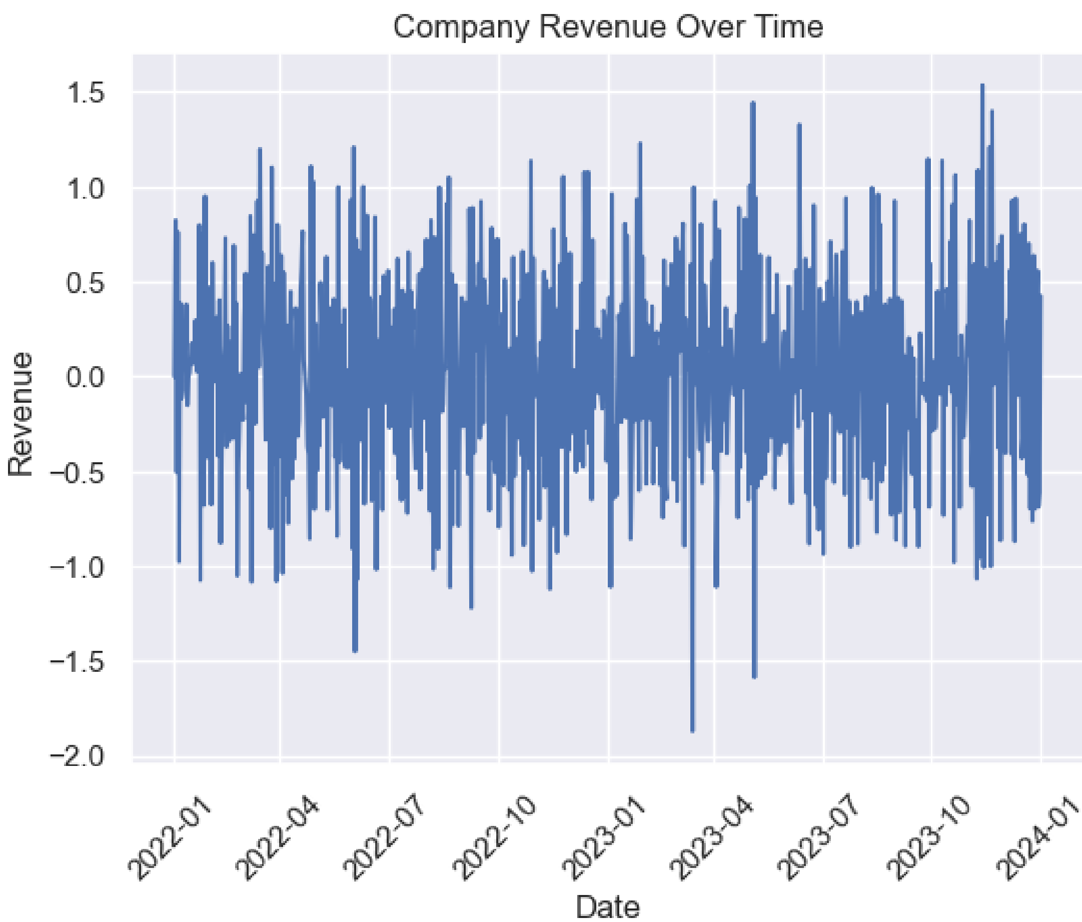
[35]:
```python
## Plotting the spectral density of the differenced data
plt.psd(time_data_st['Revenue'])
```

[35]:
```
(array([0.02187272, 0.06794442, 0.10193972, 0.03234469, 0.04559398,
        0.14815969, 0.19015645, 0.13556855, 0.17930696, 0.08735005,
        0.19447871, 0.13755823, 0.01020268, 0.09512023, 0.18805592,
        0.24517453, 0.20614533, 0.18171205, 0.06867317, 0.08851738,
        0.15697924, 0.11499682, 0.30811184, 0.14944065, 0.05001068,
        0.07531453, 0.07789058, 0.01673615, 0.21814643, 0.26177285,
        0.12608986, 0.26308319, 0.16121103, 0.04795751, 0.1613957 ,
        0.04902511, 0.26613974, 0.16274877, 0.10082987, 0.14421741,
        0.15131872, 0.10587624, 0.21235552, 0.28259121, 0.03221009,
        0.01834018, 0.04559876, 0.07659406, 0.05610271, 0.08696616,
        0.22968462, 0.1827402 , 0.09436169, 0.0237871 , 0.21783212,
        0.18875707, 0.06917957, 0.13361489, 0.06783029, 0.08269334,
        0.03112555, 0.02422832, 0.13926167, 0.39890482, 0.08670748,
        0.11441993, 0.11177872, 0.18740897, 0.12808631, 0.06475304,
        0.0077952 , 0.06857082, 0.24922119, 0.16611852, 0.21520259,
        0.42182914, 0.39754655, 0.52374424, 0.62512939, 0.18726519,
        0.13111332, 0.43322009, 0.34818613, 0.1553137 , 0.21013366,
        0.17361039, 0.57161035, 0.27249034, 0.13228186, 0.30658681,
        0.05239826, 0.27755592, 0.20651373, 0.10902572, 0.03644409,
        0.13032478, 0.30017193, 1.00082731, 0.35010851, 1.02384551,
        2.06051044, 1.16975022, 0.28636722, 0.26486441, 0.12188377,
        0.14693411, 0.09351289, 0.31160769, 0.21024373, 0.3346019 ,
        0.42069878, 0.52108706, 0.61996018, 0.35376199, 0.73309531,
        0.50884317, 0.1431019 , 0.24463468, 0.65323034, 0.62892871,
        0.85769067, 1.5358352 , 1.2161517 , 1.716833  , 1.1779039 ,
        0.05256369, 1.09660037, 0.83633103, 0.01455748]),
 array([0.       , 0.0078125, 0.015625 , 0.0234375, 0.03125  , 0.0390625,
        0.046875 , 0.0546875, 0.0625   , 0.0703125, 0.078125 , 0.0859375,
        0.09375  , 0.1015625, 0.109375 , 0.1171875, 0.125    , 0.1328125,
        0.140625 , 0.1484375, 0.15625  , 0.1640625, 0.171875 , 0.1796875,
        0.1875   , 0.1953125, 0.203125 , 0.2109375, 0.21875  , 0.2265625,
        0.234375 , 0.2421875, 0.25     , 0.2578125, 0.265625 , 0.2734375,
        0.28125  , 0.2890625, 0.296875 , 0.3046875, 0.3125   , 0.3203125,
        0.328125 , 0.3359375, 0.34375  , 0.3515625, 0.359375 , 0.3671875,
        0.375    , 0.3828125, 0.390625 , 0.3984375, 0.40625  , 0.4140625,
        0.421875 , 0.4296875, 0.4375   , 0.4453125, 0.453125 , 0.4609375,
        0.46875  , 0.4765625, 0.484375 , 0.4921875, 0.5      , 0.5078125,
```

```
0.515625 , 0.5234375, 0.53125  , 0.5390625, 0.546875 , 0.5546875,
0.5625   , 0.5703125, 0.578125 , 0.5859375, 0.59375  , 0.6015625,
0.609375 , 0.6171875, 0.625    , 0.6328125, 0.640625 , 0.6484375,
0.65625  , 0.6640625, 0.671875 , 0.6796875, 0.6875   , 0.6953125,
0.703125 , 0.7109375, 0.71875  , 0.7265625, 0.734375 , 0.7421875,
0.75     , 0.7578125, 0.765625 , 0.7734375, 0.78125  , 0.7890625,
0.796875 , 0.8046875, 0.8125   , 0.8203125, 0.828125 , 0.8359375,
0.84375  , 0.8515625, 0.859375 , 0.8671875, 0.875    , 0.8828125,
0.890625 , 0.8984375, 0.90625  , 0.9140625, 0.921875 , 0.9296875,
0.9375   , 0.9453125, 0.953125 , 0.9609375, 0.96875  , 0.9765625,
0.984375 , 0.9921875, 1.       ]))
```



[16]:
```
## Plotting linegraph of differenced data
seaborn.set(style = 'dark')
plot = seaborn.lineplot(data = time_data_st, x = time_data_st.index, y =
    'Revenue')
plot.set(title = 'Company Revenue Over Time')
plt.grid()
plt.xticks(rotation = 45)
plt.show()
```



0.6 D4

[23]:
```
## Checking where to split the data
print(len(time_data) * 0.8)
```

```
584.8000000000001
```

[24]:
```
## Finding the date where the split should occur.
data_for_split = time_data[time_data['Day']==585]
print(data_for_split)
```

```
            Day    Revenue
Date
2023-08-08  585  13.684826
```

[25]:
```
train = time_data.loc[:'2023-08-08']
test = time_data.loc['2023-08-09':]
```

[26]:
```
print(len(train) + len(test))
```

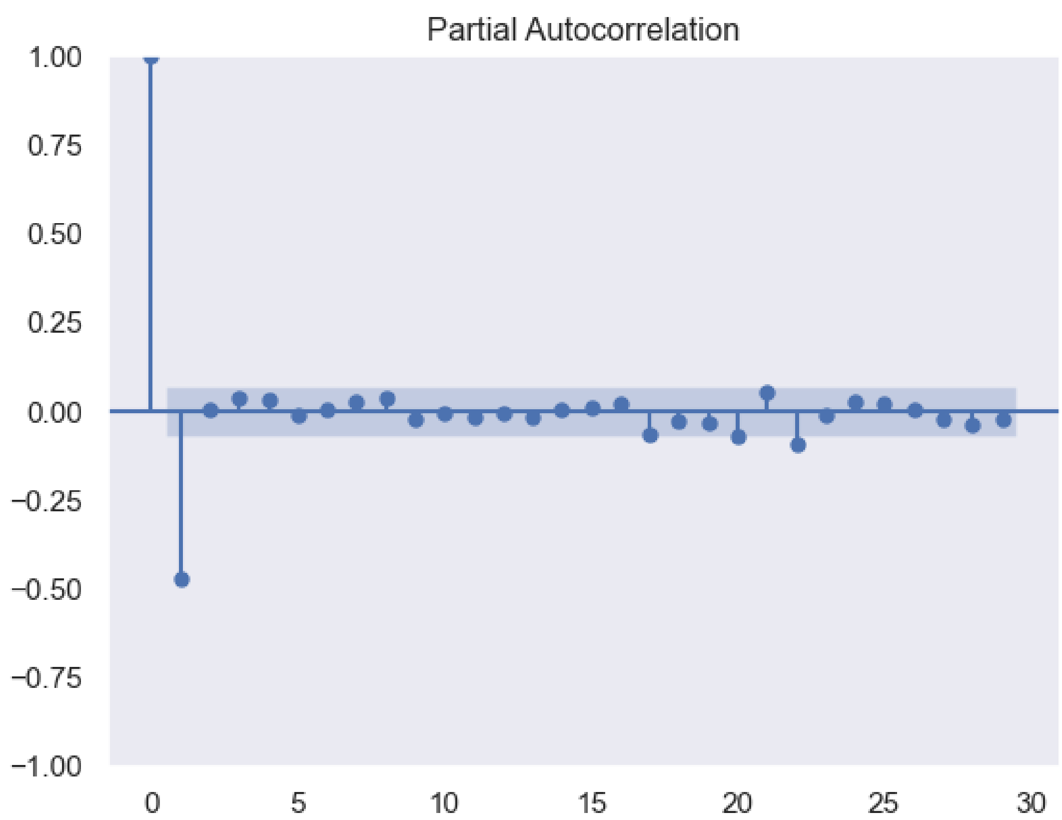
731

0.7 D5

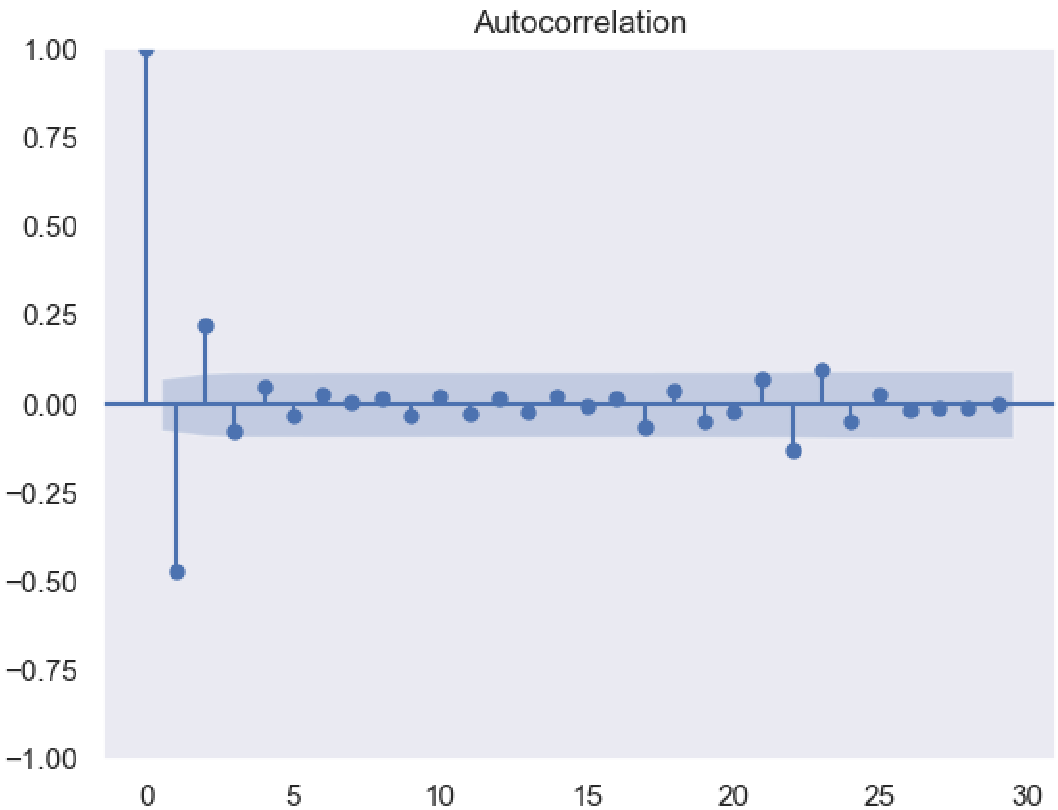
[27]:
```
train.to_csv('C:/Users/cfman/OneDrive/Desktop/WGUClasses/D603MachineLearning/
↪Task3/train.csv')
test.to_csv('C:/Users/cfman/OneDrive/Desktop/WGUClasses/D603MachineLearning/
↪Task3/test.csv')
time_data.to_csv('C:/Users/cfman/OneDrive/Desktop/WGUClasses/
↪D603MachineLearning/Task3/time_data.csv')
```

0.8 E2

[31]:
```
## Determing the p value in the ARIMA model. Looks to be 1
plot_pacf(time_data_st)
plt.show()
```



[32]:
```
## Determing the q value in the ARIMA model. Looks to be 2
plot_acf(time_data_st)
plt.show()
```



[54]:
```python
# Fit the ARIMA(1, 1, 2) model
# Data was differenced once so using 1 for d
# Going to use 1,1,2 for p,d,q as mentioned
model = ARIMA(train['Revenue'], order=(1, 1, 2))
results = model.fit()

# Print the model summary
print(results.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  585
Model:                 ARIMA(1, 1, 2)   Log Likelihood                -384.530
Date:                Sun, 09 Mar 2025   AIC                            777.060
Time:                        22:38:24   BIC                            794.540
Sample:                    01-01-2022   HQIC                           783.873
                         - 08-08-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
```

```
ar.L1         -0.2722      0.174     -1.565      0.118      -0.613       0.069
ma.L1         -0.1829      0.175     -1.045      0.296      -0.526       0.160
ma.L2          0.1146      0.083      1.389      0.165      -0.047       0.276
sigma2         0.2184      0.014     15.705      0.000       0.191       0.246
================================================================================
===
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):
2.69
Prob(Q):                              0.88   Prob(JB):
0.26
Heteroskedasticity (H):               0.97   Skew:
-0.10
Prob(H) (two-sided):                  0.84   Kurtosis:
2.74
================================================================================
===

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

[129]:

```python
# Fit the ARIMA(1, 1, 2) model
# Data was differenced once so using 1 for d
# Going to use 1,1,2 for p,d,q as mentioned
model = ARIMA(train['Revenue'], order=(0, 1, 1))
results = model.fit()

# Print the model summary
print(results.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  585
Model:                 ARIMA(0, 1, 1)   Log Likelihood                -398.550
Date:                Mon, 10 Mar 2025   AIC                            801.099
Time:                        22:39:33   BIC                            809.839
Sample:                    01-01-2022   HQIC                           804.506
                         - 08-08-2023
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.3841      0.036    -10.680      0.000      -0.455      -0.314
sigma2         0.2292      0.014     16.076      0.000       0.201       0.257
================================================================================
===
Ljung-Box (L1) (Q):                   4.46   Jarque-Bera (JB):
1.44
```

```
Prob(Q):                              0.03   Prob(JB):
0.49
Heteroskedasticity (H):               0.90   Skew:
-0.07
Prob(H) (two-sided):                  0.47   Kurtosis:
2.80
================================================================================
===

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

0.9 E4

[138]:
```python
six_month_prediction = results.forecast(len(test) + 180)
```

[ ]:
```python
#print(test)
```

[139]:
```python
six_month_prediction = six_month_prediction.to_frame()
```

[140]:
```python
six_month_prediction = six_month_prediction.loc['2024-01-02':]
print(six_month_prediction)
```

```
            predicted_mean
2024-01-02       13.587929
2024-01-03       13.587929
2024-01-04       13.587929
2024-01-05       13.587929
2024-01-06       13.587929
…                    …
2024-06-25       13.587929
2024-06-26       13.587929
2024-06-27       13.587929
2024-06-28       13.587929
2024-06-29       13.587929

[180 rows x 1 columns]
```

[141]:
```python
mae = np.mean(np.abs(results.resid))
print("Mean Absolute Error: ", mae)
```

```
Mean Absolute Error:  0.38748811354643076
```

[142]:
```python
## https://www.datacamp.com/tutorial/arima
forecast = results.forecast(steps=len(test))
forecast = forecast[:len(test)]
test_close = test['Revenue'][:len(forecast)]
```

```
# Calculate RMSE
rmse = np.sqrt(mean_squared_error(test_close, forecast))
print(f"RMSE: {rmse:.4f}")
```

```
RMSE: 2.1674
```

0.10 E3

[99]:
```
## Forecasting
diff_forecast = results.get_forecast(steps=len(test))

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']
```

[73]:
```
## Trouble shooting line. Uncomment to check to see what each variable looks␣
↪like

#print(diff_forecast)
#print(mean_forecast)
#print(confidence_intervals)
#print(lower_limits)
#print(upper_limits)
```

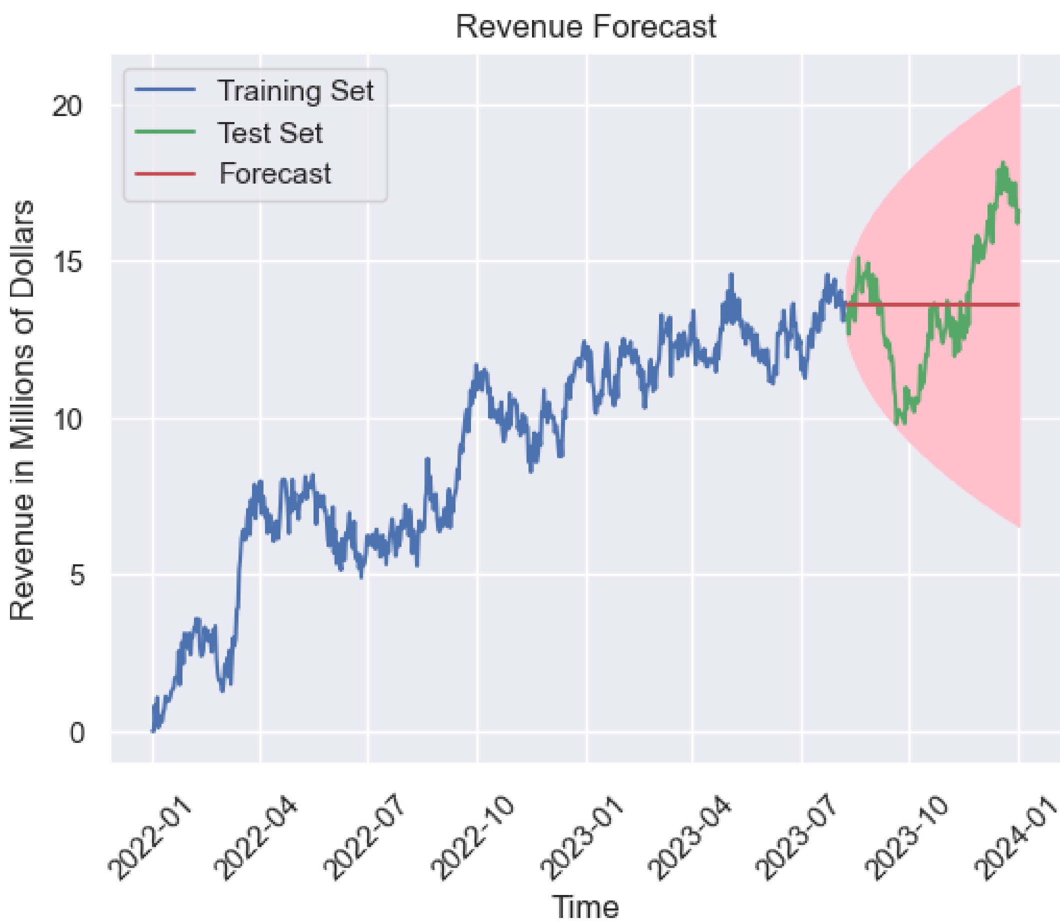
0.11 E4

[100]:
```
## Plotting the training, test, and forecast

plt.plot(train.index, train['Revenue'], label = 'Training Set')
plt.plot(test.index, test['Revenue'], label = 'Test Set', color = 'g')
plt.plot(mean_forecast.index, mean_forecast, color = 'r', label = 'Forecast')

plt.fill_between(lower_limits.index, lower_limits, upper_limits, color = 'pink')

plt.title('Revenue Forecast')
plt.xlabel('Time')
plt.ylabel('Revenue in Millions of Dollars')
plt.xticks(rotation = 45)
plt.legend(loc = 'upper left')
plt.grid()
plt.show()
```



0.12 F

[134]:
```
# Fit the ARIMA(1, 1, 2) model
# Data was differenced once so using 1 for d
# Going to use 1,1,2 for p,d,q as mentioned
model_test = ARIMA(test['Revenue'], order=(0, 1, 1))
results_test = model_test.fit()

# Print the model summary
#print(results_test.summary())

prediction = results_test.get_prediction(-len(test) + 1)

mean_prediction = prediction.predicted_mean

confidence_intervals = prediction.conf_int()

lower_limits = confidence_intervals.loc[:, 'lower Revenue']
```

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

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  146
Model:                 ARIMA(0, 1, 1)   Log Likelihood                -109.115
Date:                Mon, 10 Mar 2025   AIC                            222.229
Time:                        22:45:01   BIC                            228.183
Sample:                    08-09-2023   HQIC                           224.648
                         - 01-01-2024
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ma.L1         -0.3865      0.084     -4.592      0.000      -0.552      -0.222
sigma2         0.2634      0.034      7.713      0.000       0.196       0.330
================================================================================
===
Ljung-Box (L1) (Q):                   2.15   Jarque-Bera (JB):
2.34
Prob(Q):                              0.14   Prob(JB):
0.31
Heteroskedasticity (H):               1.30   Skew:
0.23
Prob(H) (two-sided):                  0.37   Kurtosis:
2.58
================================================================================
===

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

[137]:
```
#print(confidence_intervals)
#print(lower_limits)
#print(upper_limits)
```

[136]:
```
plt.figure(figsize = (12,4))
plt.plot(test.index, test['Revenue'], label = 'Observed (Test Set)')

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('Forecasting Comparing with Test Data')
plt.xlabel('Date')
```

```python
plt.ylabel('Revenue in Millions of Dollars')
plt.xticks(rotation = 45)
plt.legend(loc = 'upper left')
plt.grid()
plt.show()
```