In [120…

```
#Import pandas and then the csv file into our environment
#Jupyter Lab 3.44, Python 3
import pandas as pd
df = pd.read_csv(r'C:\Users\seans\Documents\WGU\D213\teleco_time_series .csv')

#Basic math imports
import numpy as np
from pandas import DataFrame

#For visualizations and plotting
import matplotlib.pyplot as mpl
%matplotlib inline

#Packages relevant to the time series
import math
import sklearn
from sklearn import datasets
#from sklearn import train_test_split
from sklearn import preprocessing
from sklearn import metrics
from sklearn. preprocessing import StandardScaler
from sklearn.metrics import mean_squared_error
from math import sqrt
from statsmodels.tsa.arima.model import ARIMA
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from dateutil.parser import parse
```

In [121…

```
#View summary information
print("Floats")
print(df.select_dtypes(include="float").info())
print("Integers")
print(df.select_dtypes(include="integer").info())
print("Objects")
print(df.select_dtypes(include="object").info())
print("Dataset Information")
print(df.info)
```

```
Floats
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Revenue  731 non-null    float64
dtypes: float64(1)
memory usage: 5.8 KB
None
Integers
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 1 columns):
 #   Column  Non-Null Count  Dtype
---  ------  --------------  -----
 0   Day     731 non-null    int64
dtypes: int64(1)
memory usage: 5.8 KB
None
Objects
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Empty DataFrameNone
Dataset Information
<bound method DataFrame.info of      Day    Revenue
0      1   0.000000
1      2   0.000793
2      3   0.825542
3      4   0.320332
4      5   1.082554
..   ...        ...
726  727  16.931559
727  728  17.490666
728  729  16.803638
729  730  16.194813
730  731  16.620798

[731 rows x 2 columns]>
```

In [122…

```
#Check for missing values
df.isna().sum()
```

Out[122]:

```
Day        0
Revenue    0
dtype: int64
```

In [123…

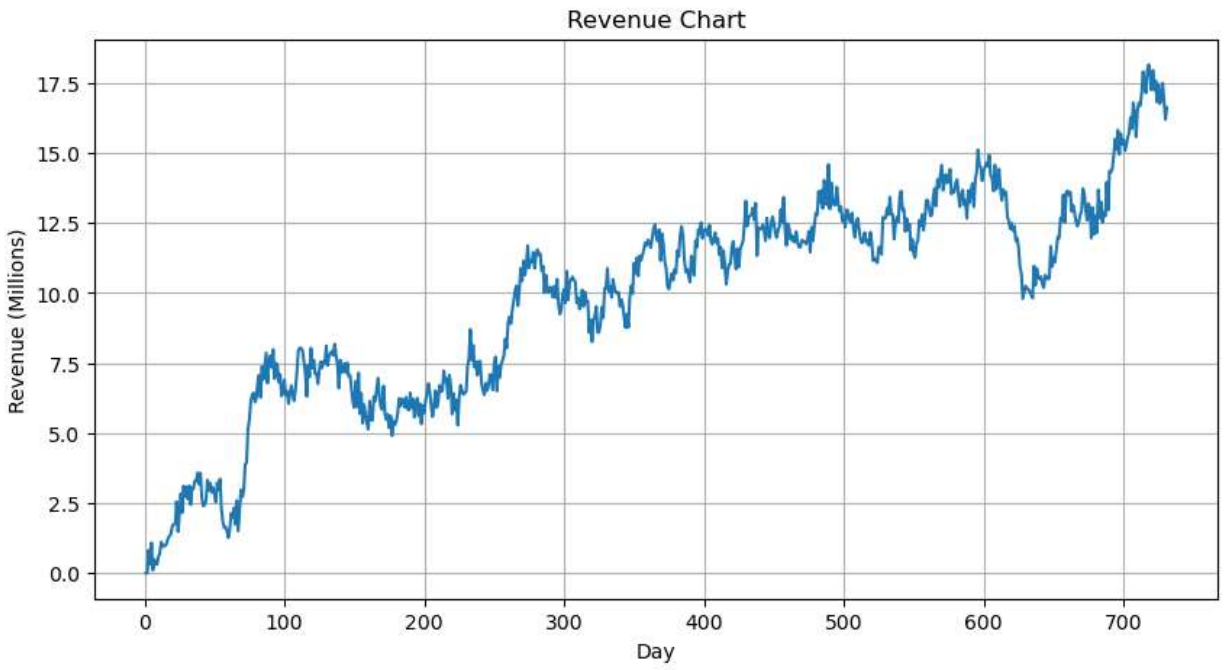
```
#View summary statistics
print("Means")
print(df.mean())
print("Medians")
print(df.median())
```

```
Means
Day        366.000000
Revenue      9.822901
dtype: float64
Medians
Day        366.000000
Revenue     10.785571
dtype: float64
```

In [124…
```
#Index our Day and Revenue
X=df.Day
Y=df.Revenue

df.set_index('Day', inplace=True)
df.index
```

Out[124]:
```
Int64Index([  1,   2,   3,   4,   5,   6,   7,   8,   9,  10,
            ...
            722, 723, 724, 725, 726, 727, 728, 729, 730, 731],
           dtype='int64', name='Day', length=731)
```

In [125…
```
#Create a line graph to visualize the realization of the time series.
mpl.figure(figsize=(10,5))
mpl.plot(df.Revenue)
mpl.title('Revenue Chart')
mpl.xlabel('Day')
mpl.ylabel('Revenue (Millions)')
mpl.grid(True)
mpl.show()
```



In [126…
```
#C2- Time step formatting
from datetime import datetime
df['Date'] = (pd.date_range(start=datetime(2020, 1, 1),
    periods=df.shape[0], freq='24H'))
df.set_index('Date',inplace=True)
df
```

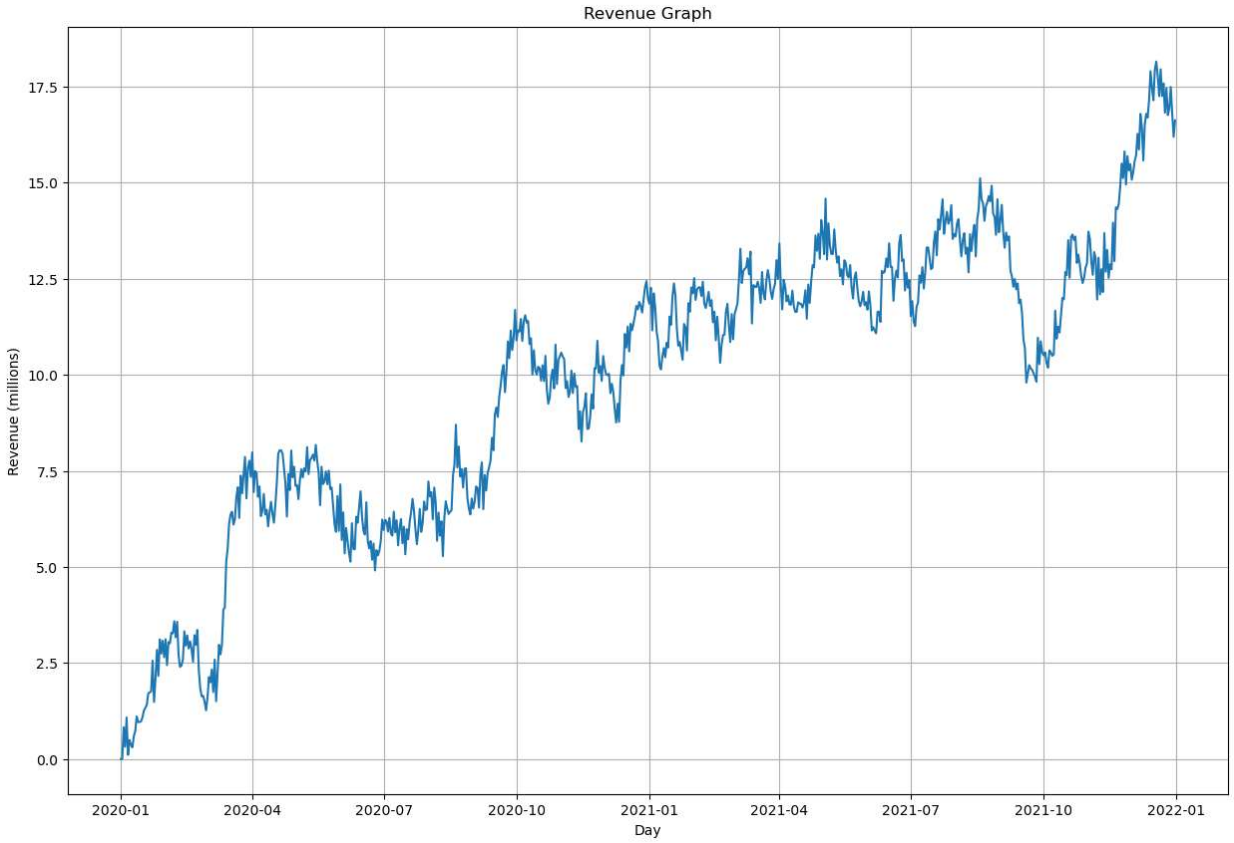
Out[126]:

Date	Revenue
2020-01-01	0.000000
2020-01-02	0.000793
2020-01-03	0.825542
2020-01-04	0.320332
2020-01-05	1.082554
...	...
2021-12-27	16.931559
2021-12-28	17.490666
2021-12-29	16.803638
2021-12-30	16.194813
2021-12-31	16.620798

731 rows × 1 columns

In [127…

```
#C2 data exploration Part 2
mpl.figure(figsize=(15,10))
mpl.plot(df.Revenue)
mpl.title('Revenue Graph')
mpl.xlabel('Day')
mpl.ylabel('Revenue (millions)')
mpl.grid(True)
mpl.show()
```



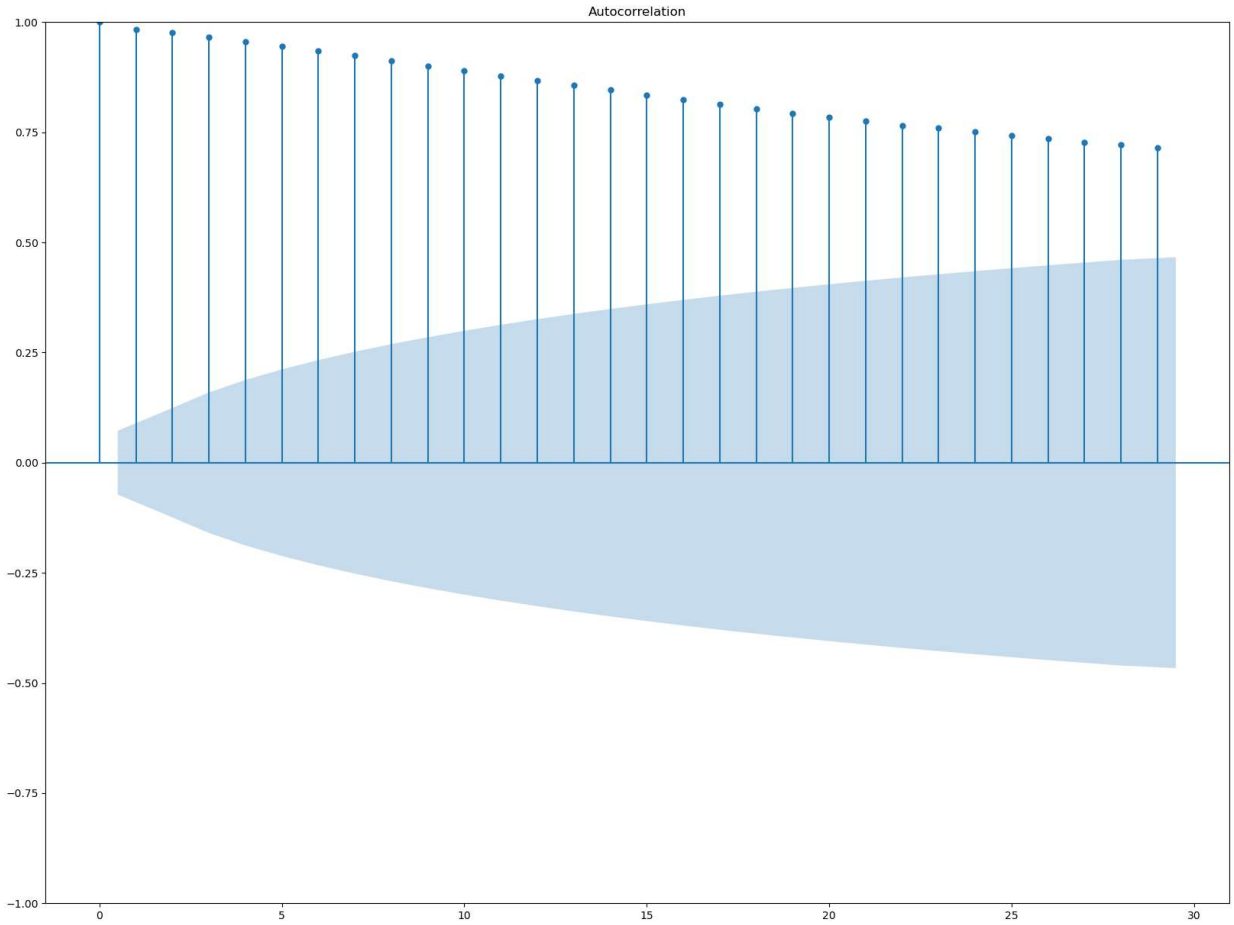
In [128…

```
#C3 Let's see the stationarity of the time series using the ADFuller Test
result = adfuller(df['Revenue'])
print("Test statistics: ", result[0])
print("p-value: ", result[1])
print("Critical values: ", result[4])

#if the p-value is above .5 we fail to reject the null hypothesis, the time series is
#If the p-value is below .5, we reject the null hypothesis, the time series is statior
#Here, the p-value is below .5, but the test statistic is not below the critical value
```

```
Test statistics:  -1.924612157310183
p-value:  0.3205728150793967
Critical values:  {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.56
88855736949163}
```

In [129…

```
#Initial ACF
af = plot_acf(df)
af.set_size_inches(20, 15)
mpl.show(af)
```



In [130…

```
#Use the built in differencing function
df = df.diff().dropna()
df
```

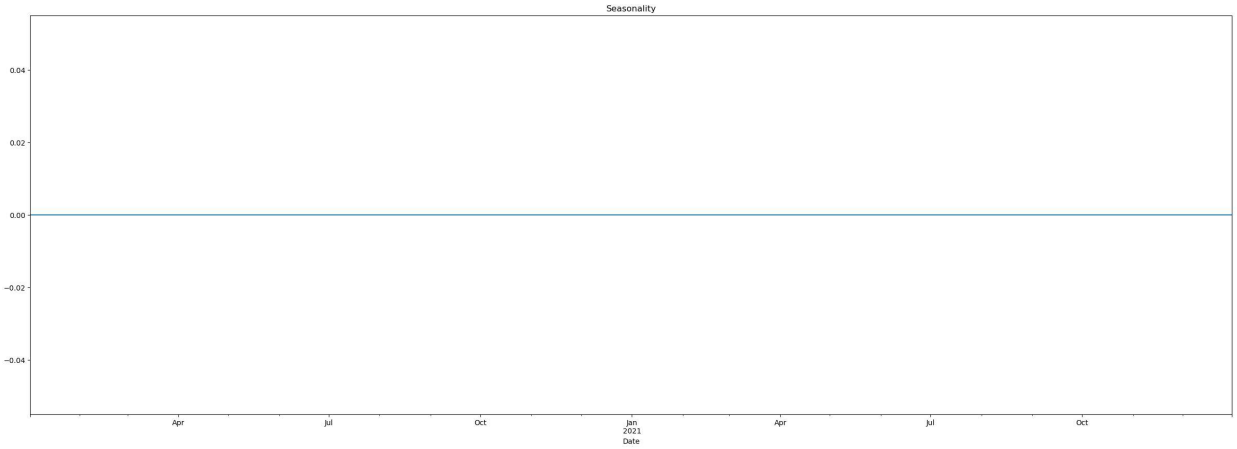
Out[130]:

Date	Revenue
2020-01-02	0.000793
2020-01-03	0.824749
2020-01-04	-0.505210
2020-01-05	0.762222
2020-01-06	-0.974900
...	...
2021-12-27	0.170280
2021-12-28	0.559108
2021-12-29	-0.687028
2021-12-30	-0.608824
2021-12-31	0.425985

730 rows × 1 columns

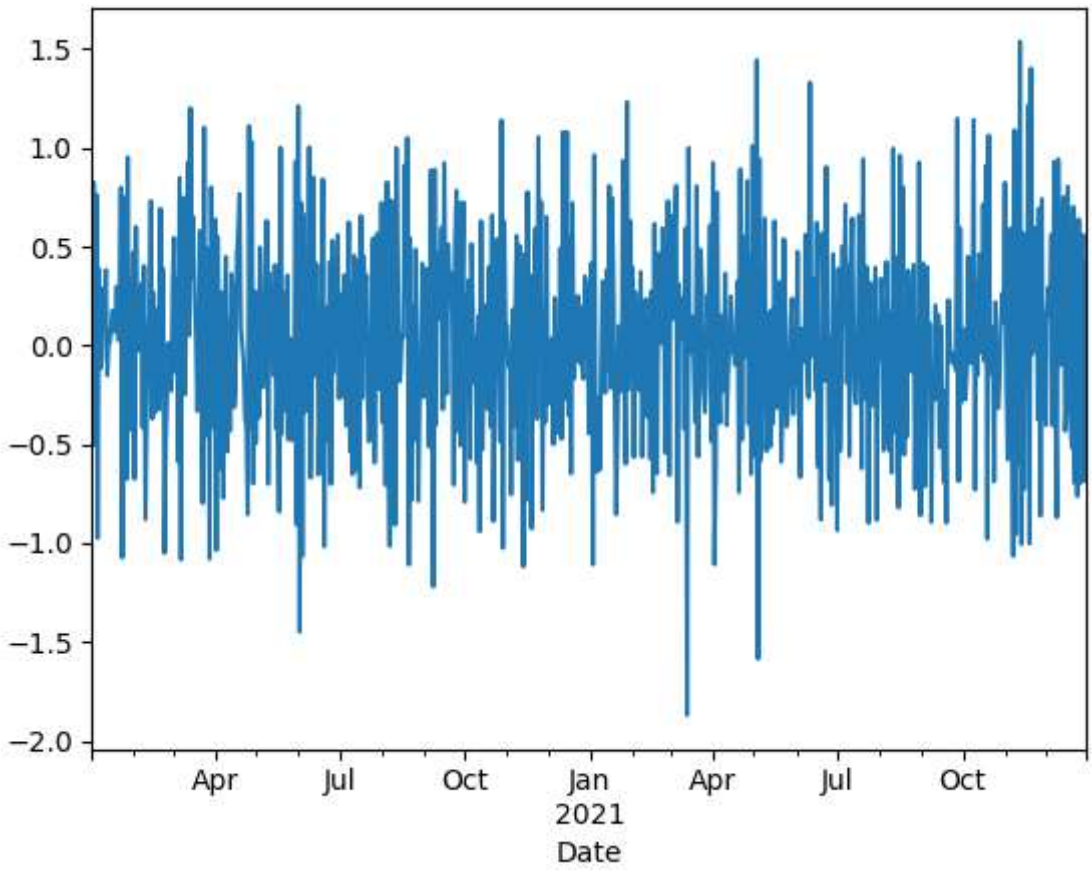
In [131…
```
#D1a: Lets evaluate our model
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df['Revenue'], model='additive', period=1)
mpl.title('Seasonality')
result.seasonal.plot(figsize=(30, 10))
#The straight line here shows us there is no seasonal trend to our data
```

Out[131]:
```
<AxesSubplot:title={'center':'Seasonality'}, xlabel='Date'>
```



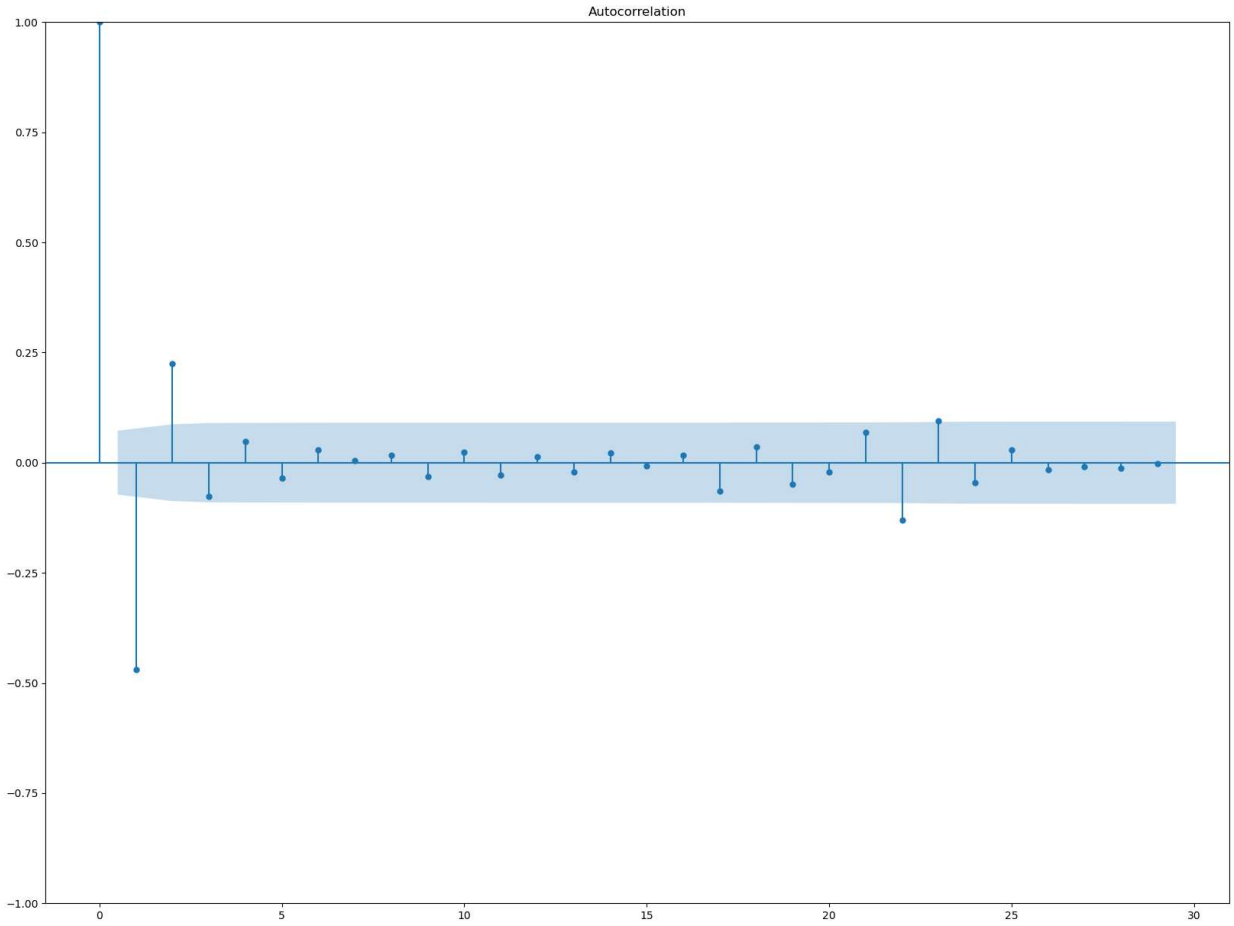
In [132…
```
#D1b: Let's look at trends
result.trend.plot()
```

Out[132]:
```
<AxesSubplot:xlabel='Date'>
```



In [133…
```
#D1c: Autocorrelation function
af = plot_acf(df)
```

```
af.set_size_inches(20, 15)
mpl.show(af)
#If we are stationary there should be a quick drop to 0
```



In [134…

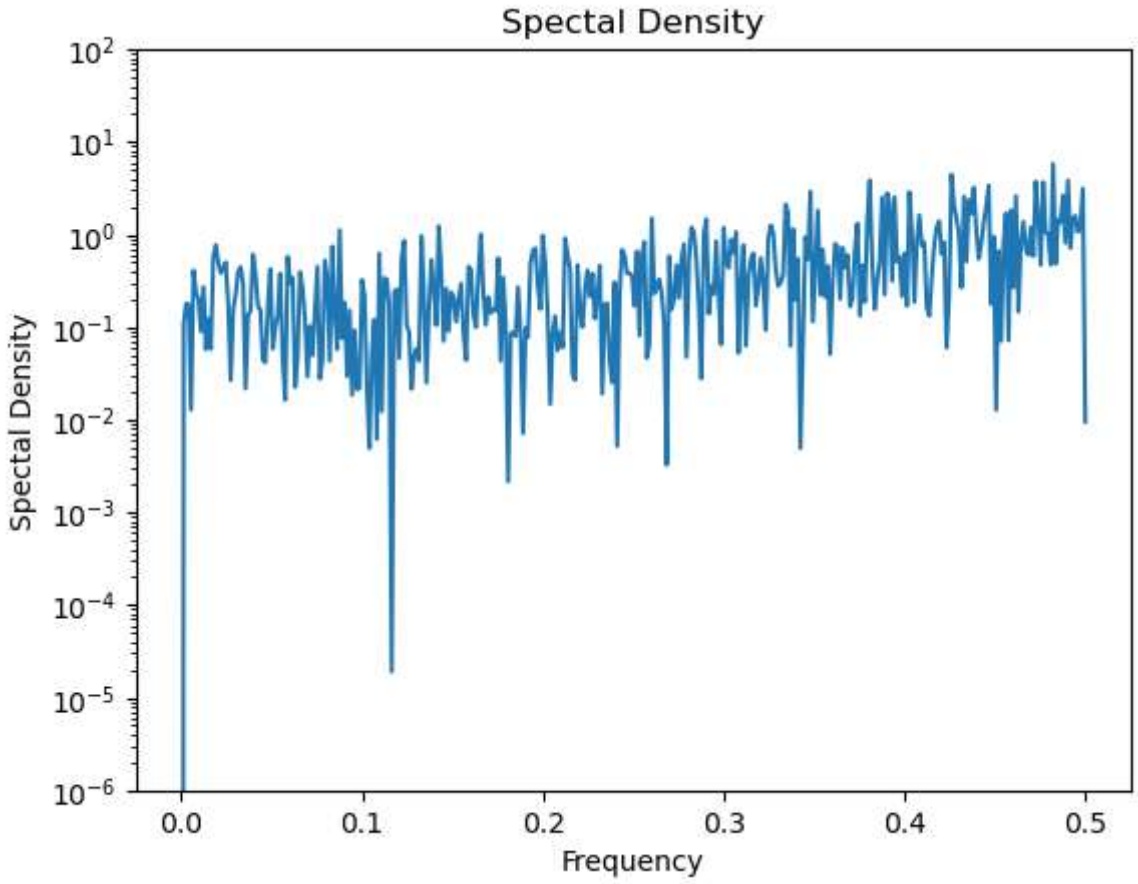
```
#Splitting the data into training and test sets
#Creating training and testing data .75 train to .25 test
X_train = df.iloc[:548]
X_test = df.iloc[549:]
```

In [135…

```
#C5: Extract cleaned data
X_train.to_csv("Task1_Train.csv")
X_test.to_csv("Task1_Test.csv")
df.to_csv("D213_Task1_cleaned.csv")
```

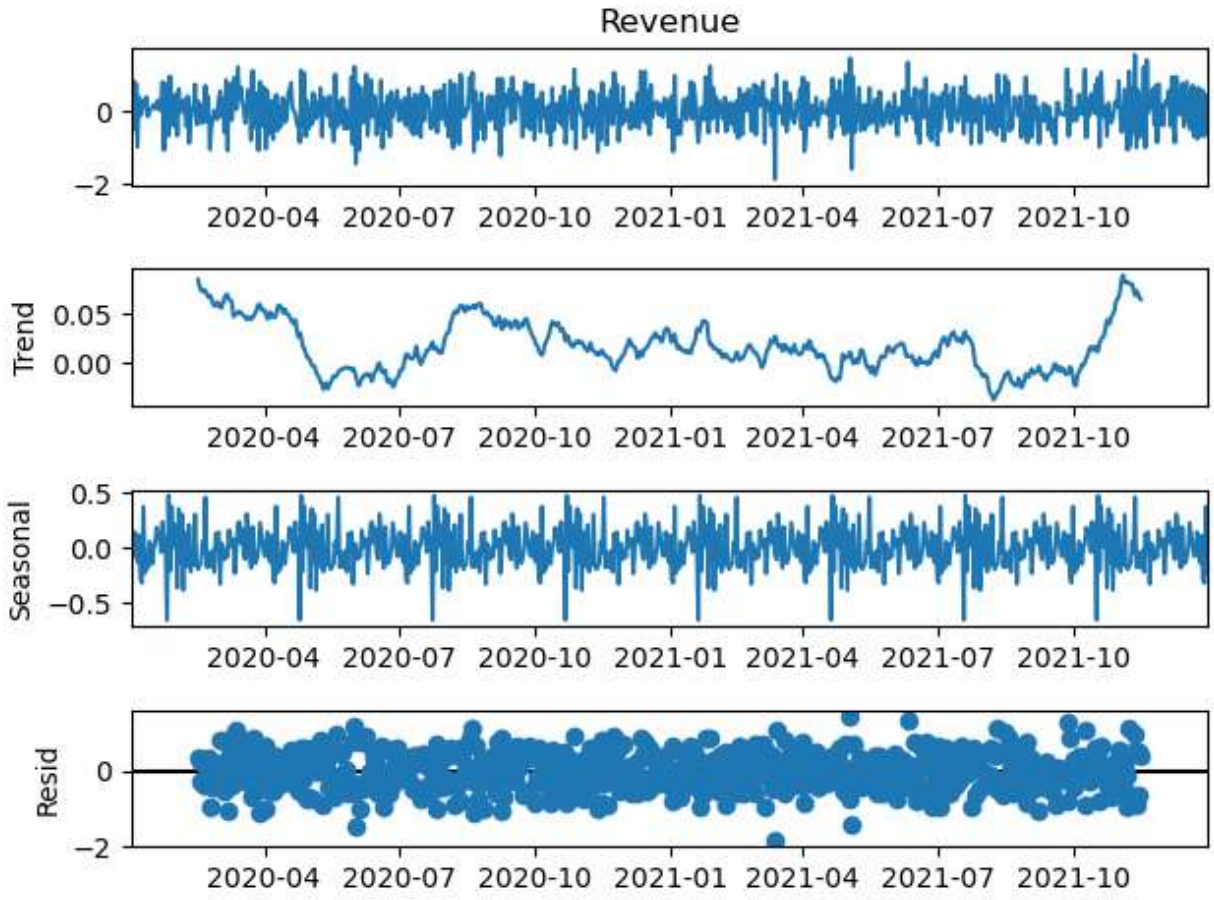
In [136…

```
#D1d: Spectral Density
from scipy import signal
f, Pxx_den = signal.periodogram(df['Revenue'])
mpl.semilogy(f, Pxx_den)
mpl.ylim([1e-6, 1e2])
mpl.title('Spectal Density')
mpl.xlabel('Frequency')
mpl.ylabel('Spectal Density')
mpl.show()
```



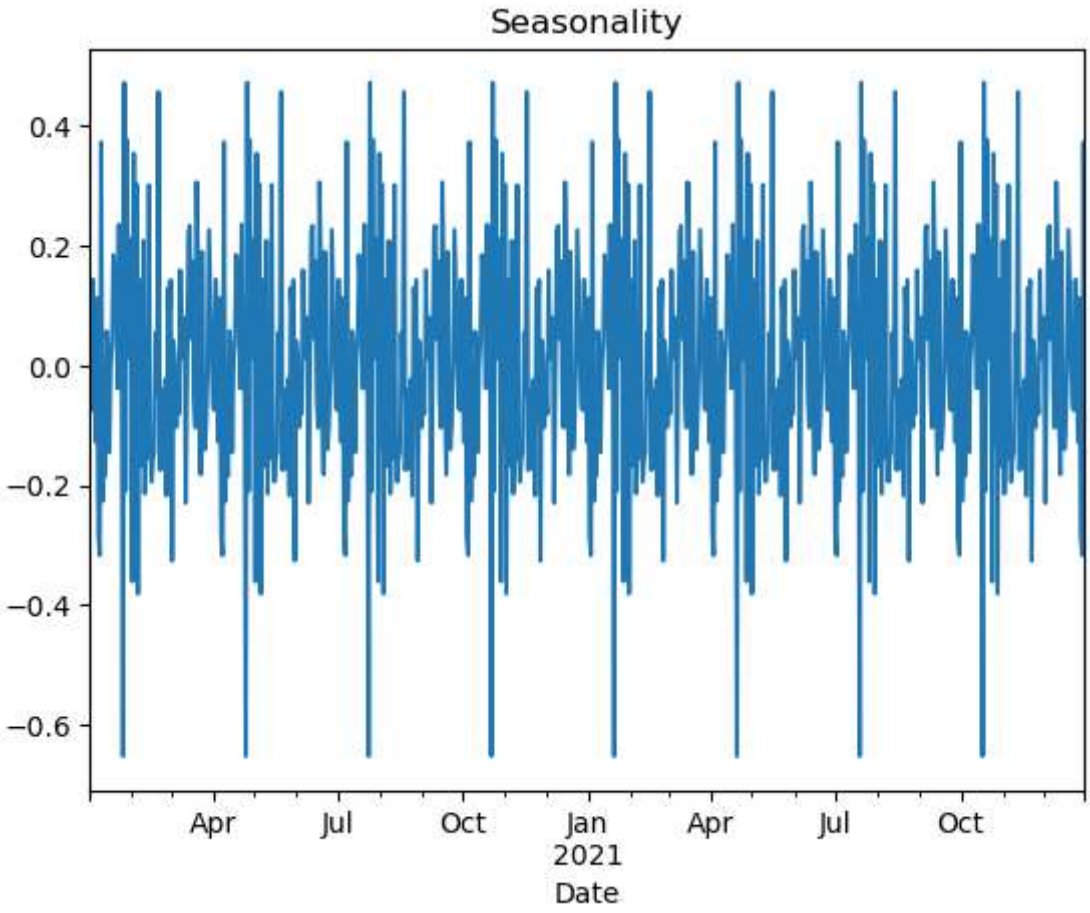
In [137…

```
#D1e: Decomposed Time Series
decomp = seasonal_decompose(df['Revenue'], period=90)
decomp.plot() # Plot decomposition
mpl.show() # Check  for seasonality in the data
```



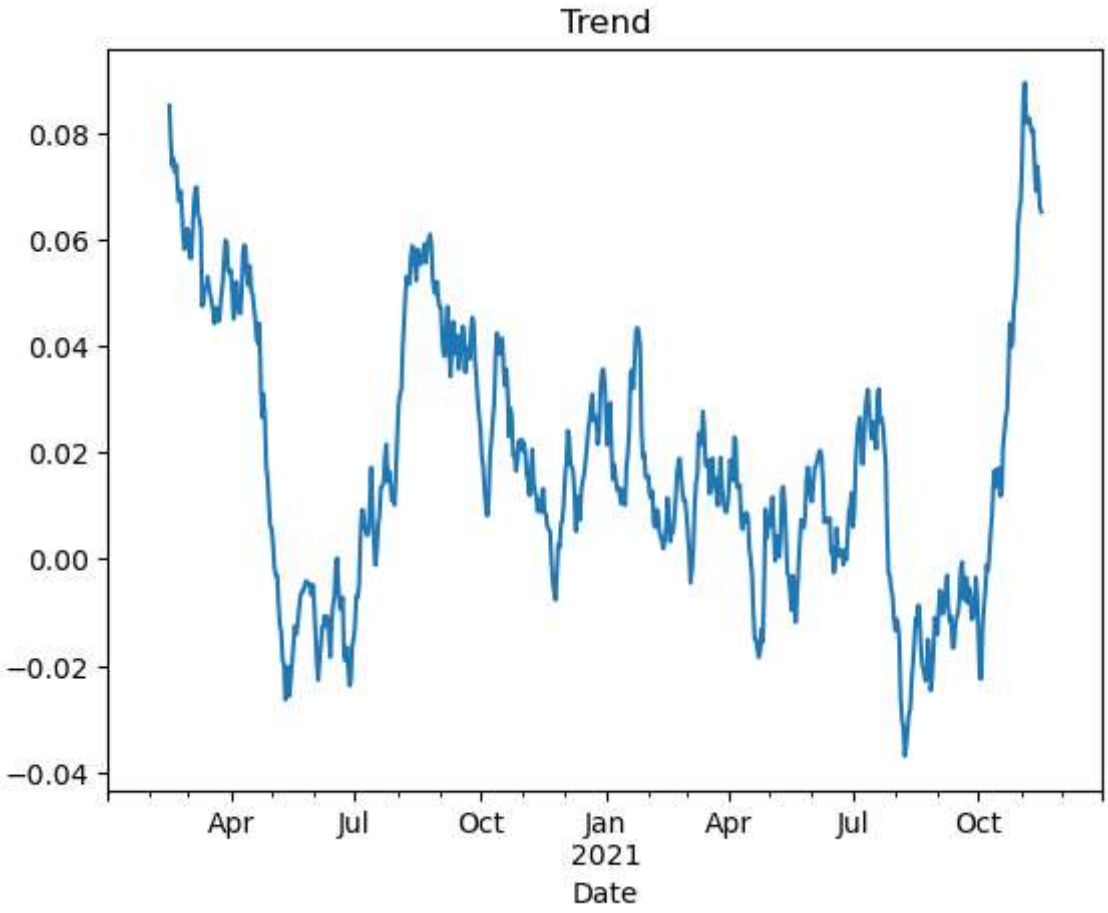
In [138…
```
#Seasonality
mpl.title('Seasonality')
decomp.seasonal.plot()
```

Out[138]:
```
<AxesSubplot:title={'center':'Seasonality'}, xlabel='Date'>
```



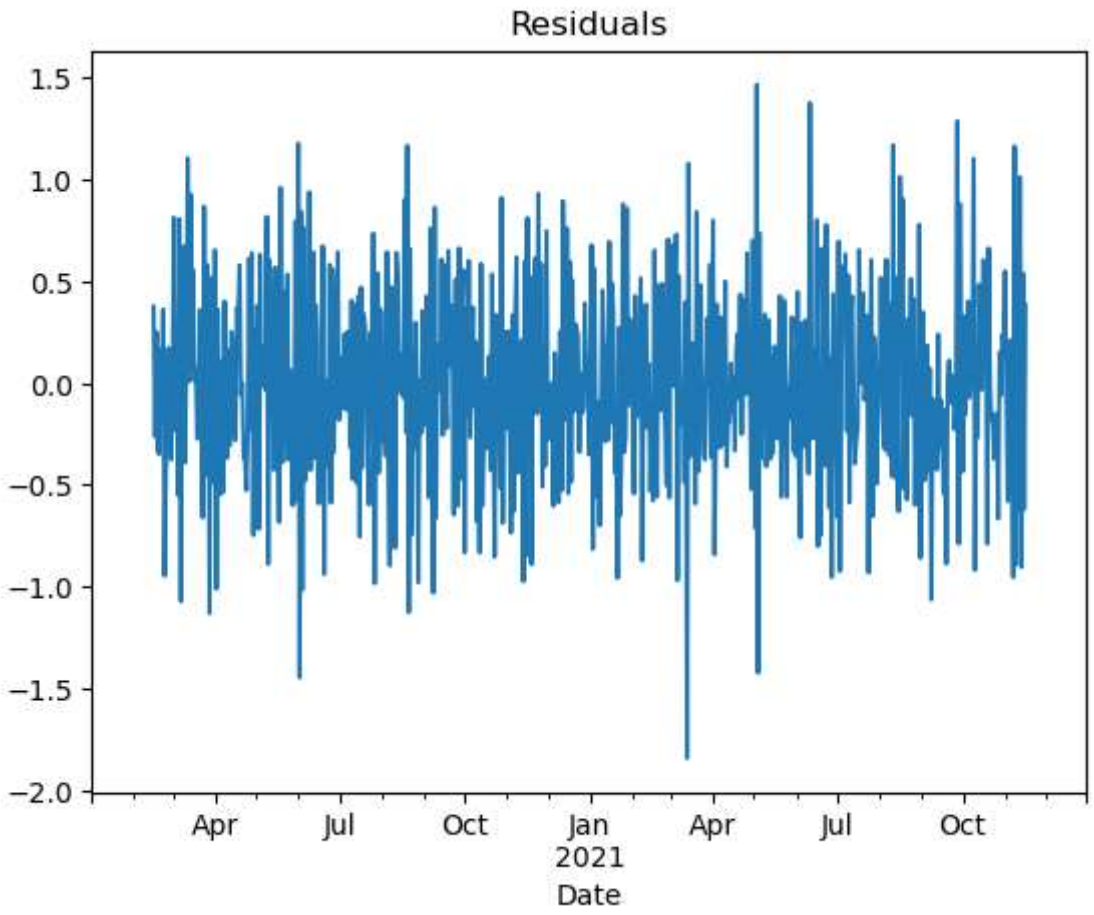
In [139…
```
#Trend
mpl.title('Trend')
decomp.trend.plot()
```

Out[139]:
```
<AxesSubplot:title={'center':'Trend'}, xlabel='Date'>
```



In [140…
```
#D1f: Residuals
mpl.title('Residuals')
decomp.resid.plot()
```

Out[140]:
```
<AxesSubplot:title={'center':'Residuals'}, xlabel='Date'>
```

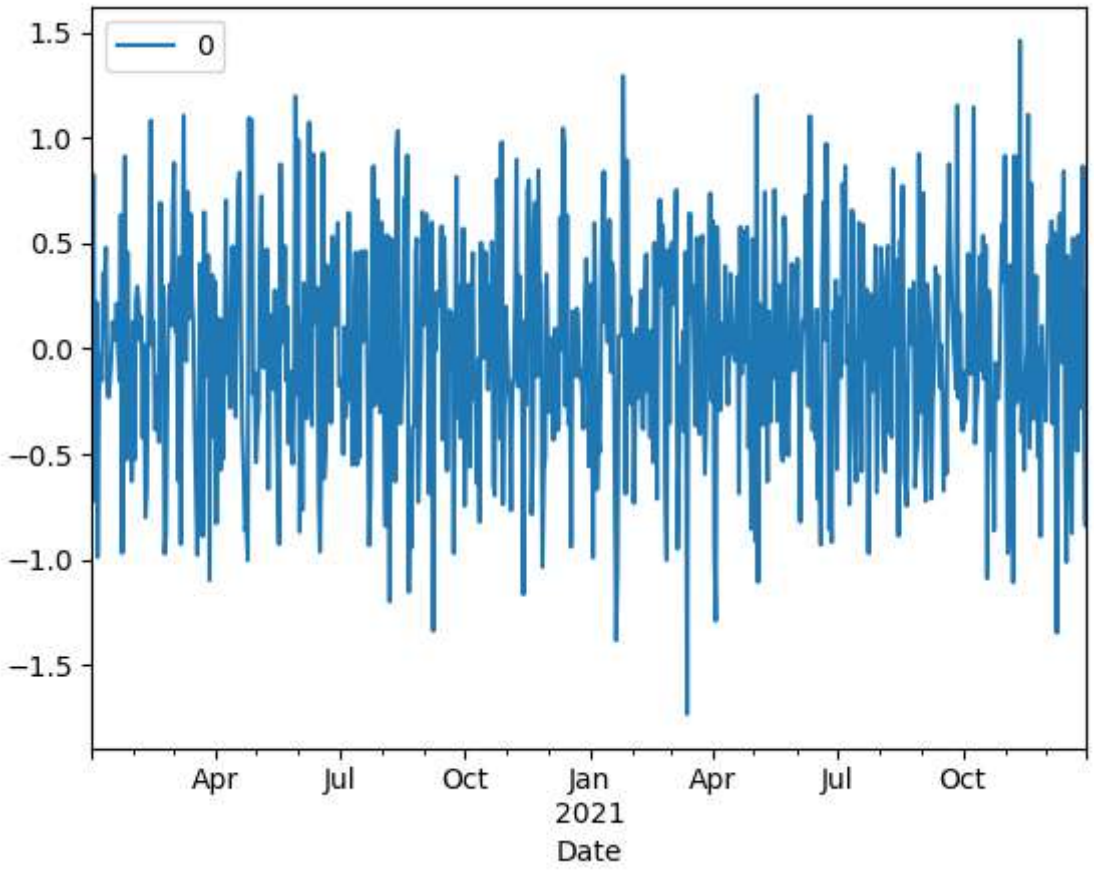


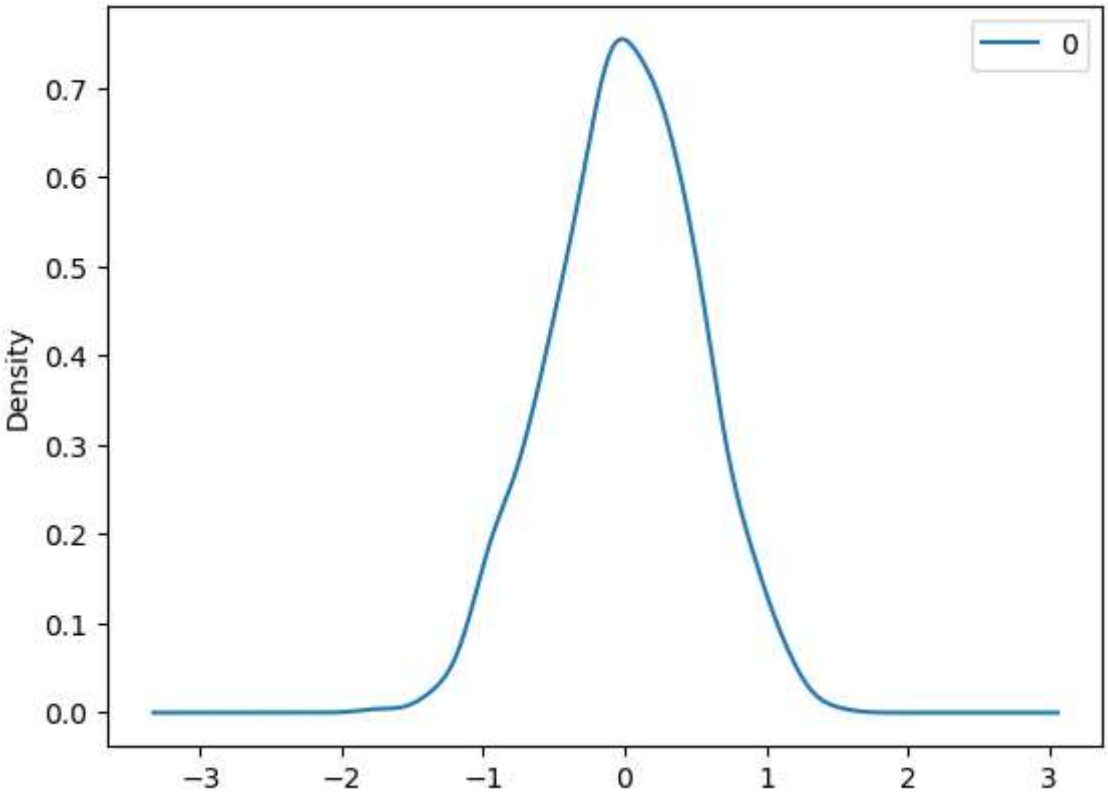
In [ ]:
```
#Rolling average
import seaborn as sb
dfroll = df.copy()
dfroll[ '7day_rolling_avg' ] = dfroll.Revenue.rolling( 7).mean()
# Rolling Average Plot
plt.figure( figsize = ( 15, 5))


sb.lineplot( x = 'Day',
             y = 'Revenue',
             data = dfroll,
             label = 'DailyRevenue')


sb.lineplot( x = 'Day',
             y = '7day_rolling_avg',
             data = dfroll,
             label = 'Rollingavg')

mpl.xlabel('Day')
```

In [141…
```
#D2: Autoregressive integrated moving average (ARIMA).
# fit model
df.index = df.index.to_period('D')
model = ARIMA(df, order=(5,1,0), freq = "D")
model_fit = model.fit()
# summary of fit model
print(model_fit.summary())
# line plot of residuals
```

```
residuals = DataFrame(model_fit.resid)
residuals.plot()
mpl.show()
# density plot of residuals
residuals.plot(kind='kde')
mpl.show()
# summary stats of residuals
print(residuals.describe())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  730
Model:                 ARIMA(5, 1, 0)   Log Likelihood                -545.622
Date:                Wed, 01 Mar 2023   AIC                           1103.243
Time:                        17:10:00   BIC                           1130.793
Sample:                    01-02-2020   HQIC                          1113.873
                         - 12-31-2021
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.2947      0.036    -35.703      0.000      -1.366      -1.224
ar.L2         -1.0175      0.057    -17.802      0.000      -1.130      -0.906
ar.L3         -0.7079      0.065    -10.935      0.000      -0.835      -0.581
ar.L4         -0.4271      0.058     -7.311      0.000      -0.542      -0.313
ar.L5         -0.1813      0.037     -4.882      0.000      -0.254      -0.109
sigma2         0.2609      0.015     17.960      0.000       0.232       0.289
===================================================================================
Ljung-Box (L1) (Q):                   0.74   Jarque-Bera (JB):                 3.05
Prob(Q):                              0.39   Prob(JB):                         0.22
Heteroskedasticity (H):               0.99   Skew:                            -0.12
Prob(H) (two-sided):                  0.92   Kurtosis:                         2.79
===================================================================================

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


0
1.5
1.0
0.5
0.0
−0.5
−1.0
−1.5
Apr
Jul
Oct
Jan
2021
Apr
Jul
Oct
Date


0
Density
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.0
−3
−2
−1
0
1
2
3

```
                0
count  730.000000
mean    -0.002052
std      0.511771
min     -1.734059
25%     -0.354040
50%      0.007620
75%      0.347918
max      1.460947
```

In [146…

```
#Predict
from sklearn.metrics import mean_squared_error
from math import sqrt
X = df.values
size = int(len(X) * 0.66)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
# walk-forward validation
for t in range(len(test)):
    model = ARIMA(history, order=(5,1,0))
    model_fit = model.fit()
    output = model_fit.forecast()
    yhat = output[0]
    predictions.append(yhat)
    obs = test[t]
    history.append(obs)
    print('predicted=%f, expected=%f' % (yhat, obs))
#Evaluate our forecast
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
```

```
predicted=0.073678, expected=-0.398402
predicted=0.493211, expected=0.442802
predicted=0.200065, expected=-0.650088
predicted=0.451605, expected=1.008159
predicted=-0.255253, expected=-0.324824
predicted=0.365546, expected=-0.562324
predicted=0.244546, expected=1.442980
predicted=-0.513372, expected=-1.585790
predicted=0.776759, expected=0.943729
predicted=-0.338858, expected=-0.583497
predicted=0.106102, expected=-0.212997
predicted=0.062260, expected=-0.003891
predicted=-0.167196, expected=0.640939
predicted=-0.455344, expected=-0.535783
predicted=0.334733, expected=-0.329183
predicted=-0.013814, expected=0.163790
predicted=-0.205994, expected=-0.513667
predicted=0.112459, expected=0.178902
predicted=-0.253228, expected=-0.394099
predicted=-0.128893, expected=0.628061
predicted=-0.390155, expected=-0.071771
predicted=0.037969, expected=-0.317071
predicted=0.135298, expected=-0.057014
predicted=-0.015031, expected=0.321697
predicted=-0.150852, expected=-0.589136
predicted=0.264832, expected=-0.280994
predicted=-0.094939, expected=0.537505
predicted=-0.425178, expected=0.143334
predicted=-0.051832, expected=-0.410972
predicted=0.184911, expected=-0.345691
predicted=-0.004449, expected=-0.126817
predicted=-0.084963, expected=0.137300
predicted=-0.171820, expected=0.237650
predicted=-0.230651, expected=-0.348350
predicted=0.031539, expected=0.079178
predicted=-0.104597, expected=-0.191498
predicted=0.038073, expected=0.473420
predicted=-0.176750, expected=-0.351148
predicted=0.157943, expected=-0.667049
predicted=0.153342, expected=0.083698
predicted=-0.248611, expected=-0.086103
predicted=-0.172073, expected=-0.070429
predicted=-0.155547, expected=0.560739
predicted=-0.412026, expected=0.001060
predicted=0.030125, expected=-0.265185
predicted=0.232748, expected=1.328921
predicted=-0.353508, expected=-0.055184
predicted=0.447617, expected=0.039311
predicted=0.436738, expected=0.337561
predicted=0.172477, expected=-0.226087
predicted=0.437229, expected=0.620871
predicted=0.089902, expected=-0.615377
predicted=0.374942, expected=-0.001600
predicted=0.051748, expected=-0.882018
predicted=0.209349, expected=0.566653
predicted=-0.473357, expected=0.223587
predicted=-0.225600, expected=-0.179076
predicted=-0.027922, expected=0.902409
predicted=-0.308548, expected=0.200776
predicted=0.204630, expected=-0.679496
```

```
predicted=0.653911, expected=0.043582
predicted=0.070660, expected=-0.807559
predicted=0.246728, expected=0.462048
predicted=-0.325480, expected=-0.389328
predicted=-0.129126, expected=0.198132
predicted=-0.325634, expected=-0.935939
predicted=0.124846, expected=0.387963
predicted=-0.454457, expected=-0.535211
predicted=0.003310, expected=-0.115931
predicted=-0.255803, expected=0.502020
predicted=-0.442630, expected=0.107100
predicted=-0.112147, expected=0.711919
predicted=-0.088697, expected=-0.191015
predicted=0.312962, expected=0.406537
predicted=0.182773, expected=-0.558972
predicted=0.498995, expected=0.426452
predicted=-0.022853, expected=0.641889
predicted=-0.050167, expected=0.003840
predicted=0.263625, expected=-0.269767
predicted=0.318573, expected=-0.294945
predicted=0.161012, expected=0.035542
predicted=0.042512, expected=0.657522
predicted=-0.212256, expected=0.287393
predicted=-0.001607, expected=-0.623019
predicted=0.345753, expected=0.942162
predicted=-0.240946, expected=-0.268232
predicted=0.400469, expected=0.391809
predicted=0.139954, expected=0.395174
predicted=0.049950, expected=-0.898690
predicted=0.550810, expected=0.318120
predicted=-0.040278, expected=0.251187
predicted=-0.111991, expected=-0.305541
predicted=0.213675, expected=0.089945
predicted=-0.112828, expected=0.393701
predicted=-0.201337, expected=-0.882748
predicted=0.438341, expected=0.132842
predicted=-0.181532, expected=-0.067610
predicted=-0.161195, expected=0.338037
predicted=-0.170655, expected=0.113264
predicted=-0.065481, expected=-0.531705
predicted=0.142026, expected=-0.437835
predicted=0.087640, expected=0.422243
predicted=-0.310560, expected=0.179940
predicted=-0.097491, expected=-0.531923
predicted=0.110944, expected=0.157387
predicted=-0.247729, expected=-0.644689
predicted=0.128149, expected=0.995057
predicted=-0.420362, expected=-0.438775
predicted=0.147215, expected=0.445385
predicted=-0.162580, expected=0.235454
predicted=0.033484, expected=-0.821111
predicted=0.467598, expected=0.960360
predicted=-0.259317, expected=0.271259
predicted=0.033086, expected=0.800479
predicted=0.096822, expected=-0.554332
predicted=0.529489, expected=-0.087464
predicted=0.274902, expected=-0.464173
predicted=0.382660, expected=0.379702
predicted=-0.132501, expected=0.100962
predicted=-0.069676, expected=0.163249
```

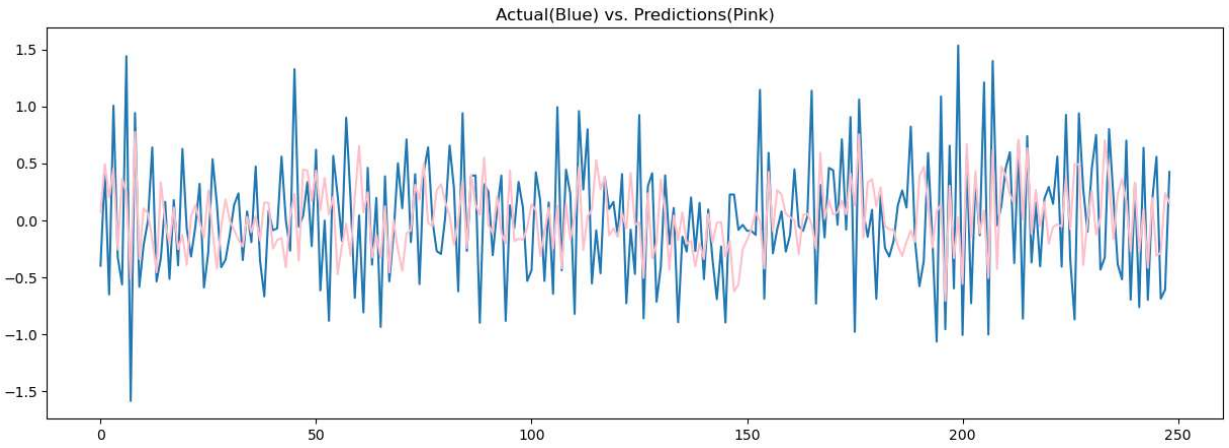
```
predicted=-0.142987, expected=-0.137079
predicted=0.063726, expected=0.407677
predicted=-0.069197, expected=-0.727597
predicted=0.417433, expected=-0.080859
predicted=-0.037378, expected=-0.469428
predicted=-0.019455, expected=0.925701
predicted=-0.505007, expected=-0.860539
predicted=0.237113, expected=0.301026
predicted=-0.335025, expected=0.414290
predicted=-0.202407, expected=-0.714124
predicted=0.357784, expected=-0.404817
predicted=0.058897, expected=0.395940
predicted=-0.429898, expected=-0.207910
predicted=0.042106, expected=0.109283
predicted=-0.173295, expected=-0.894099
predicted=0.070337, expected=-0.143334
predicted=-0.204844, expected=-0.273737
predicted=-0.181446, expected=0.201678
predicted=-0.403265, expected=-0.268963
predicted=-0.198502, expected=0.154077
predicted=-0.337569, expected=-0.515649
predicted=0.055783, expected=0.095988
predicted=-0.192754, expected=-0.345536
predicted=-0.022694, expected=-0.691324
predicted=-0.014368, expected=-0.228499
predicted=-0.317310, expected=-0.897176
predicted=-0.180007, expected=0.227910
predicted=-0.624262, expected=0.226743
predicted=-0.564732, expected=-0.082917
predicted=-0.252908, expected=-0.039153
predicted=-0.172451, expected=-0.091047
predicted=-0.073152, expected=-0.091918
predicted=0.071360, expected=-0.127095
predicted=-0.005392, expected=1.146975
predicted=-0.422470, expected=-0.687688
predicted=0.426273, expected=0.593714
predicted=-0.095722, expected=-0.289182
predicted=0.266978, expected=-0.077812
predicted=0.226469, expected=0.079727
predicted=0.057295, expected=-0.274431
predicted=0.027821, expected=-0.123906
predicted=0.001046, expected=0.449110
predicted=-0.298030, expected=-0.049435
predicted=0.048281, expected=-0.091533
predicted=0.057806, expected=0.035426
predicted=-0.032539, expected=1.139601
predicted=-0.246564, expected=-0.730623
predicted=0.591600, expected=0.311328
predicted=0.018101, expected=-0.149609
predicted=0.177862, expected=0.461114
predicted=0.051141, expected=0.439421
predicted=0.071924, expected=-0.037716
predicted=0.173666, expected=0.712873
predicted=0.053584, expected=-0.080626
predicted=0.412317, expected=0.907834
predicted=0.131649, expected=-0.978916
predicted=0.756128, expected=1.061361
predicted=-0.150710, expected=0.065350
predicted=0.332967, expected=-0.144047
predicted=0.361877, expected=0.092849
```

```
predicted=0.125199, expected=-0.688597
predicted=0.290626, expected=0.217696
predicted=-0.048603, expected=-0.241950
predicted=-0.077321, expected=-0.317428
predicted=-0.084844, expected=-0.180828
predicted=-0.226204, expected=0.133820
predicted=-0.312912, expected=0.263952
predicted=-0.188478, expected=0.113545
predicted=-0.088030, expected=0.823673
predicted=-0.187801, expected=-0.184929
predicted=0.390302, expected=-0.577835
predicted=0.469737, expected=-0.359540
predicted=0.148056, expected=0.592132
predicted=-0.237116, expected=-0.169088
predicted=0.078572, expected=-1.065162
predicted=0.139446, expected=1.088779
predicted=-0.707013, expected=-0.955825
predicted=0.304669, expected=0.656067
predicted=-0.330213, expected=-0.596857
predicted=0.025971, expected=1.536069
predicted=-0.557178, expected=-1.006270
predicted=0.669553, expected=0.568323
predicted=-0.147638, expected=-0.727736
predicted=0.432471, expected=0.355638
predicted=-0.112329, expected=-0.130363
predicted=0.088523, expected=1.211860
predicted=-0.506477, expected=-1.002120
predicted=0.613922, expected=1.400221
predicted=-0.427806, expected=-0.043895
predicted=0.472750, expected=0.128075
predicted=0.392459, expected=0.454894
predicted=0.226490, expected=0.599325
predicted=0.148320, expected=-0.374764
predicted=0.711857, expected=0.693519
predicted=0.008799, expected=-0.862922
predicted=0.634130, expected=0.740440
predicted=-0.118462, expected=-0.369679
predicted=0.269106, expected=0.158926
predicted=-0.053856, expected=-0.403559
predicted=0.162955, expected=0.191899
predicted=-0.205000, expected=0.294106
predicted=-0.057628, expected=0.145749
predicted=-0.034021, expected=0.560893
predicted=-0.053706, expected=-0.405928
predicted=0.371331, expected=0.927333
predicted=-0.075935, expected=-0.345283
predicted=0.491648, expected=-0.869865
predicted=0.493305, expected=0.940542
predicted=-0.392499, expected=0.275525
predicted=-0.001845, expected=-0.099936
predicted=0.258186, expected=0.459377
predicted=-0.124528, expected=0.750848
predicted=0.023465, expected=-0.432122
predicted=0.705158, expected=-0.325217
predicted=0.346282, expected=0.802984
predicted=-0.165041, expected=0.205935
predicted=0.225539, expected=-0.385952
predicted=0.362889, expected=-0.517971
predicted=0.162307, expected=0.701432
predicted=-0.269716, expected=-0.696344
```

```
predicted=0.331193, expected=0.327692
predicted=-0.275620, expected=-0.761868
predicted=0.097753, expected=0.637298
predicted=-0.415888, expected=-0.697778
predicted=0.198563, expected=0.170280
predicted=-0.309482, expected=0.559108
predicted=-0.283029, expected=-0.687028
predicted=0.240091, expected=-0.608824
predicted=0.151032, expected=0.425985
Test RMSE: 0.517
```

In [148…

```
#D3: Forecast using the derived ARIMA model against outcome
mpl.figure(figsize=(15,5))
mpl.title('Actual(Blue) vs. Predictions(Pink)')
mpl.plot(test)
mpl.plot(predictions, color='pink')
mpl.show()
```



In [ ]: