Part 1 Research Question

The research question is can I predict with some degree of accuracy future revenues of the company.

The goal is this analysis is to be able to predict future revenues with a high degree of accuracy.

Part 2 Method Justification

One assumption of time series is that the data has stationarity. According to the Engineering Statistics Handbook staionarity can be defined as "but for our purpose we mean a flat looking series, without trend, constant variance over time, a constant autocorrelation structure over time and no periodic fluctuations (seasonality). (https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm (https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm))

Another assumption of time series analysis is that the data is autocorrelated.

Part 3 Data Preperation

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
%matplotlib inline
import warnings
warnings.filterwarnings("ignore")
from matplotlib.pylab import rcParams
rcParams['figure.figsize'] = 10,6
```

In [2]:
```
dataset = pd.read_csv('telco.csv')
```

In [3]:
```
indexedDataset = dataset.set_index(['Day'])
indexedDataset
```

Out[3]:

	Revenue
Day	
1	0.000000
2	0.000793
3	0.825542
4	0.320332
5	1.082554
...	...
727	16.931559
728	17.490666
729	16.803638
730	16.194814
731	16.620798

731 rows × 1 columns

In [4]:
```
indexedDataset.index=pd.to_datetime(indexedDataset.index, unit = 'D', origin = '2
```

In [5]:
```
indexedDataset = indexedDataset[1:]
```

In [6]:
```
indexedDataset.head(5)
```

Out[6]:

	Revenue
Day	
2018-01-03	0.000793
2018-01-04	0.825542
2018-01-05	0.320332
2018-01-06	1.082554
2018-01-07	0.107654

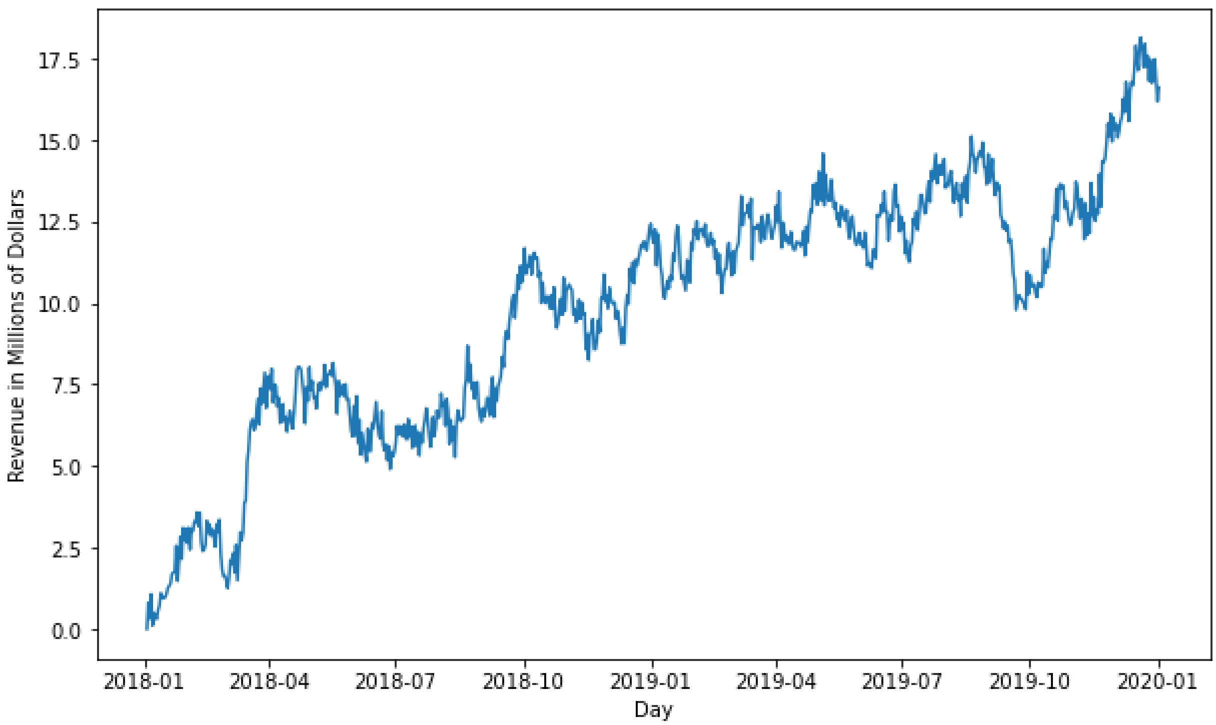
In [7]:
```
indexedDataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
DatetimeIndex: 730 entries, 2018-01-03 to 2020-01-02
Data columns (total 1 columns):
 #   Column   Non-Null Count  Dtype  
---  ------   --------------  -----  
 0   Revenue  730 non-null    float64
dtypes: float64(1)
memory usage: 11.4 KB
```

In [8]:
```
from datetime import datetime
```

In [9]:
```
plt.xlabel('Day')
plt.ylabel('Revenue in Millions of Dollars')
plt.plot(indexedDataset)
```

Out[9]:
```
[<matplotlib.lines.Line2D at 0x1e8257b1df0>]
```



C2) To format my time series data I took the day column and turned it into an index. Since we are just simply counting days I decided I didn't need to change the index to a datetime object. The series is exact meaning there are no gaps and all the data is evenly spaced out. My series has 2 columns and exactly 731 rows.

C4) To clean my dataset I had to make sure there were no missing or nan values in the dataset. After I made sure of that I had to check the stationarity of my data. I found my data to not be stationary so I had to change that. I did that by taking the log of my data. The steps used are in the cells below.

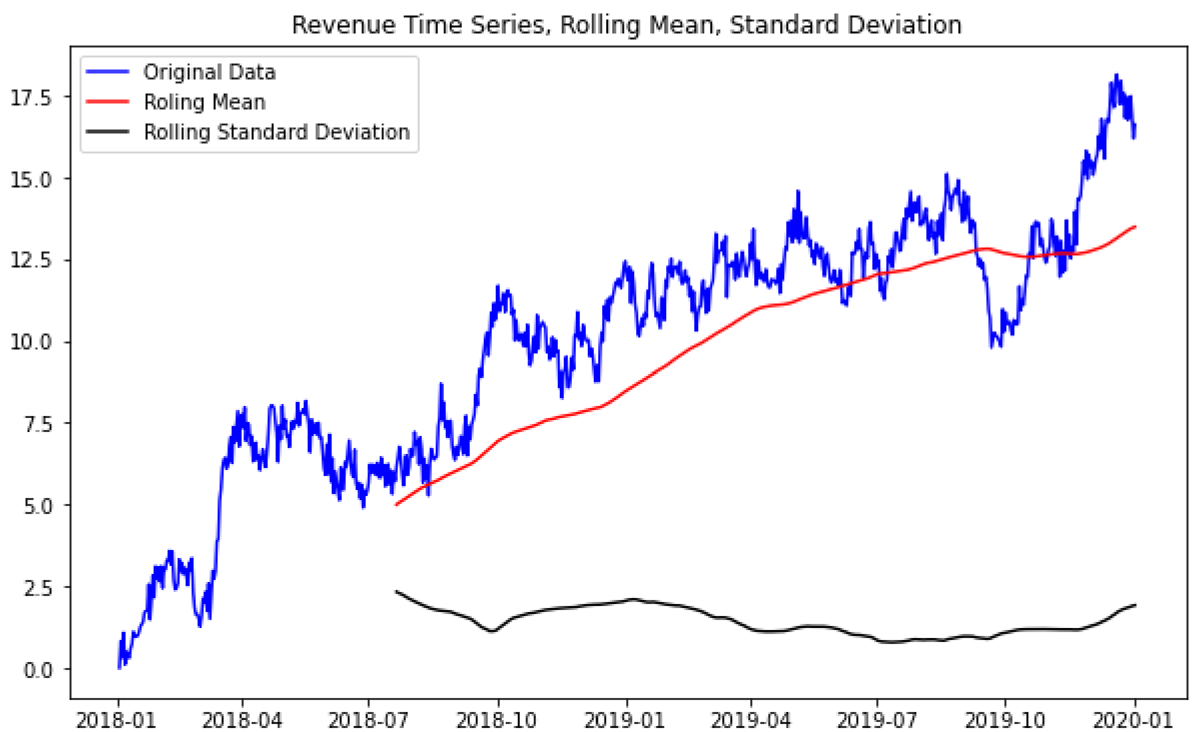
In [10]:
```
indexedDataset.to_csv('indexedDataset.csv')
```

Part 4 Model Identification and Analysis

In [11]:
```
rolling_mean = indexedDataset.rolling(200).mean()
rolling_std = indexedDataset.rolling(200).std()
```

In [12]:
```
plt.plot(indexedDataset, color = "blue", label = "Original Data")
plt.plot(rolling_mean, color ="red", label = "Roling Mean")
plt.plot(rolling_std, color="black", label = "Rolling Standard Deviation")
plt.title("Revenue Time Series, Rolling Mean, Standard Deviation")
plt.legend(loc="best")
```

Out[12]:
```
<matplotlib.legend.Legend at 0x1e827876df0>
```



My data right now is not stationary. We can see that through the visualization above when looking at the rolling meana standard deviation, you can see that they are not constant. There is a gap in the measurement from day 0 to day 200. That is because the window I wanted to use was every 200 days.

In [13]:
```
from statsmodels.tsa.stattools import adfuller
print('Results of Dickey-Fuller Test:')
dftest = adfuller(indexedDataset['Revenue'], autolag='AIC')

dfoutput = pd.Series(dftest[0:4], index=['Test Statistic','p-value','#Lags Used',
for key,value in dftest[4].items():
    dfoutput['Critical Value (%s)' %key] = value

print(dfoutput)
```

```
Results of Dickey-Fuller Test:
Test Statistic                  -1.774638
p-value                          0.393124
#Lags Used                       1.000000
NUmber of Observations Used    728.000000
Critical Value (1%)             -3.439364
Critical Value (5%)             -2.865518
Critical Value (10%)            -2.568888
dtype: float64
```

We can see here again that the data is not stationary because our critical values are less than the test statistic.

In [14]:
```
df_stationary = indexedDataset.diff().dropna()
```

In [15]:
```
df_stationary
```

Out[15]:

	Revenue
Day	
2018-01-04	0.824749
2018-01-05	-0.505210
2018-01-06	0.762222
2018-01-07	-0.974900
2018-01-08	0.386248
...	...
2019-12-29	0.170280
2019-12-30	0.559108
2019-12-31	-0.687028
2020-01-01	-0.608824
2020-01-02	0.425985

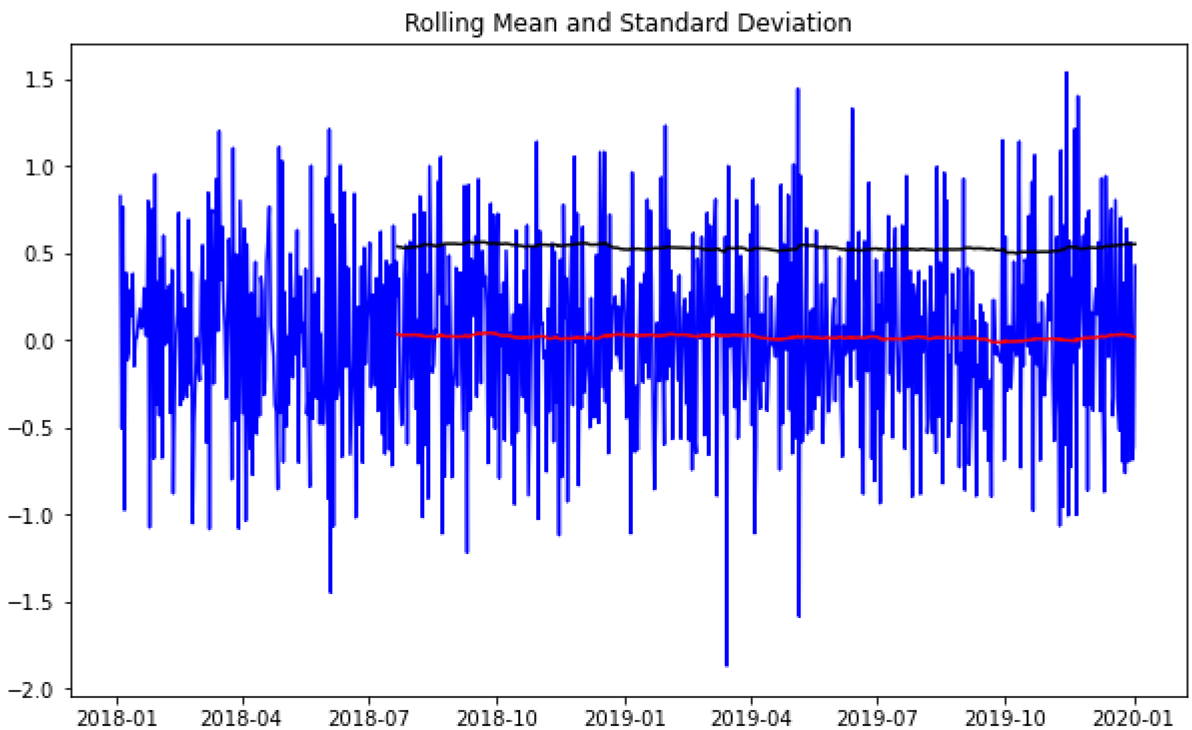
729 rows × 1 columns

In [16]:
```
from statsmodels.tsa.stattools import adfuller
def test_stationarity(timeseries):
    movingAverage = timeseries.rolling(window=200).mean()
    movingSTD = timeseries.rolling(window=200).std()

    orig = plt.plot(timeseries, color ='blue', label='original')
    mean = plt.plot(movingAverage, color = 'red', label = 'Rolling Mean')
    std = plt.plot(movingSTD, color = 'black', label = 'Rolling Std')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)

    print('Results of Dickey-Fuller test:')
    dftest = adfuller(timeseries['Revenue'], autolag='AIC')
    dfoutput = pd.Series(dftest[0:4], index = ['Test Statistic', 'p-value', '#Lag
    for key,value in dftest[4].items():
        dfoutput['Critical Value (%s)'%key] = value
    print(dfoutput)
```

In [17]:
```
test_stationarity(df_stationary)
```




```
Results of Dickey-Fuller test:
Test Statistic                 -44.927782
p-value                          0.000000
#Lags Used                       0.000000
Number of Observations Used    728.000000
Critical Value (1%)             -3.439364
Critical Value (5%)             -2.865518
Critical Value (10%)            -2.568888
dtype: float64
```

You can visually see that the data is now more stationary than it was to start. THe p-value is much smaller than the original and now the 10% criticla value is greater than the test statistic.

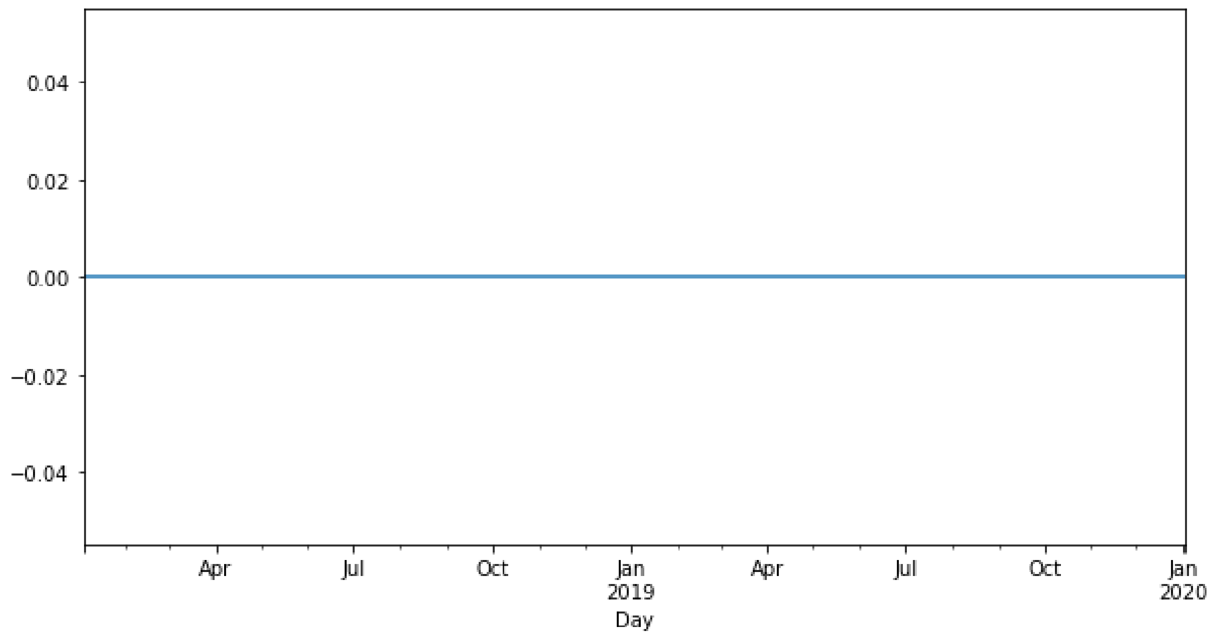
In [18]:
```
df_stationary['Revenue'] = df_stationary['Revenue'].astype('float64')
```

In [19]:
```
df_stationary['Revenue'].dtype
```

Out[19]: dtype('float64')

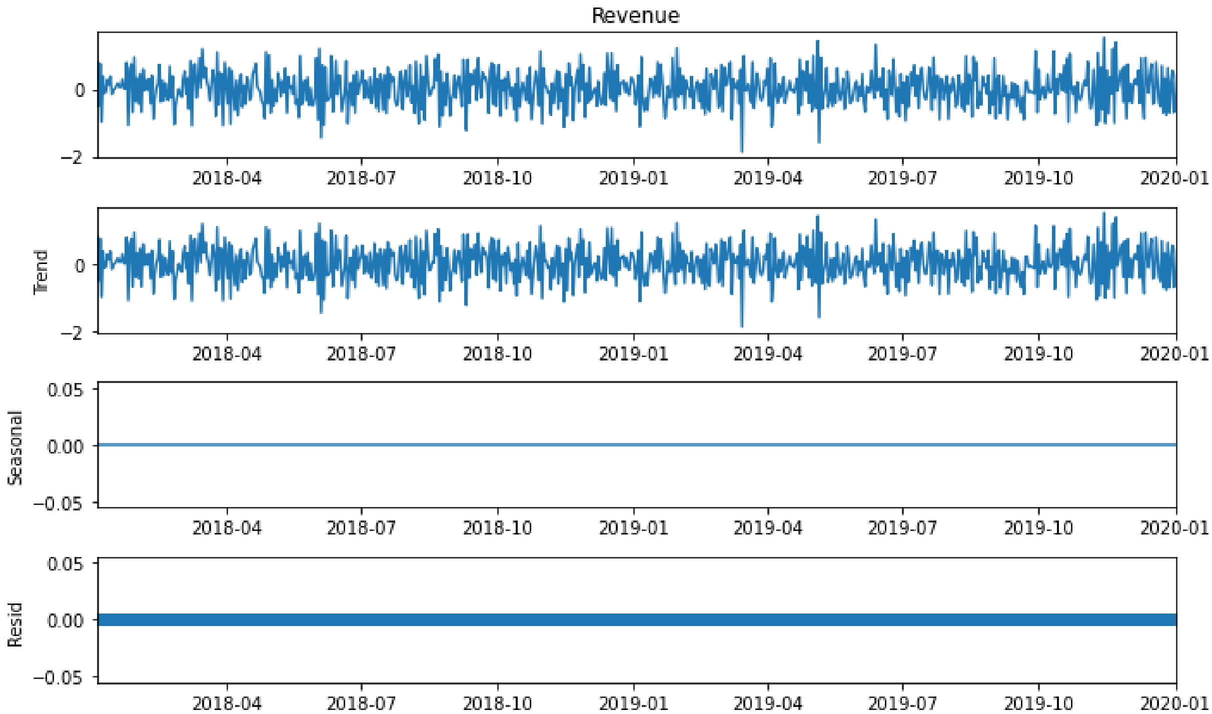
In [20]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
result=seasonal_decompose(df_stationary['Revenue'], model ='additive', period=1)
result.seasonal.plot(figsize=(10,5))
```

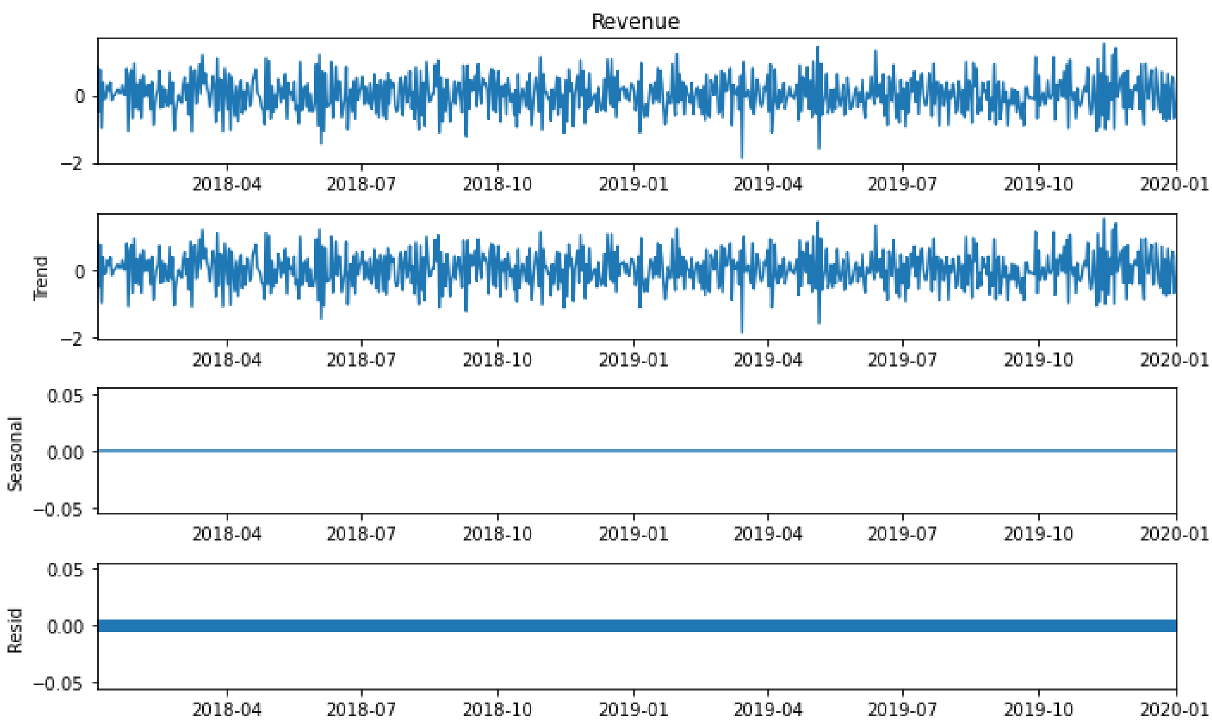
Out[20]: <AxesSubplot:xlabel='Day'>



In [21]:
```
result.plot()
```

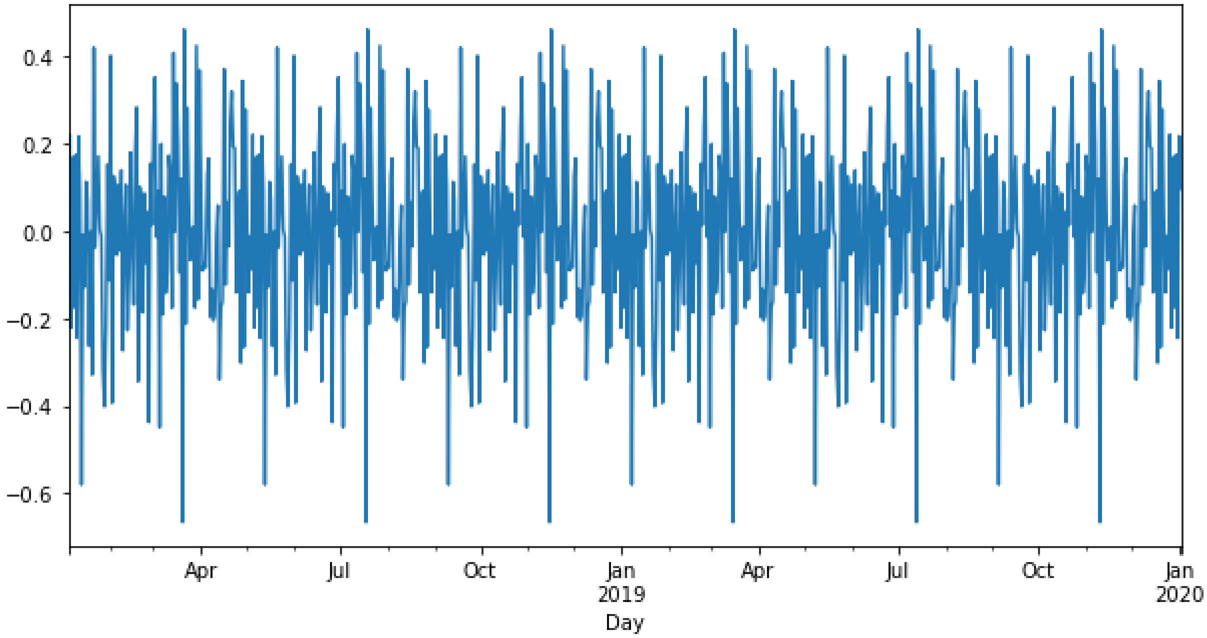
Out[21]:





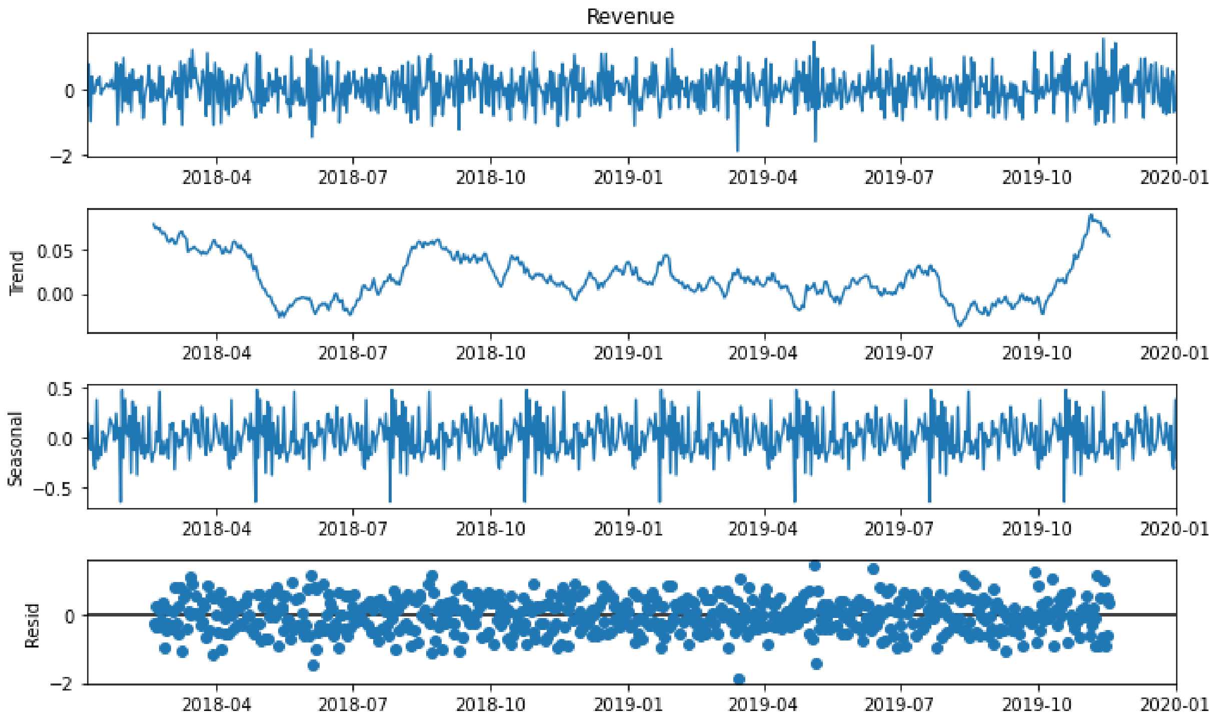
In [22]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
result=seasonal_decompose(df_stationary['Revenue'], model ='additive', period=12(
result.seasonal.plot(figsize=(10,5))
```

Out[22]: <AxesSubplot:xlabel='Day'>



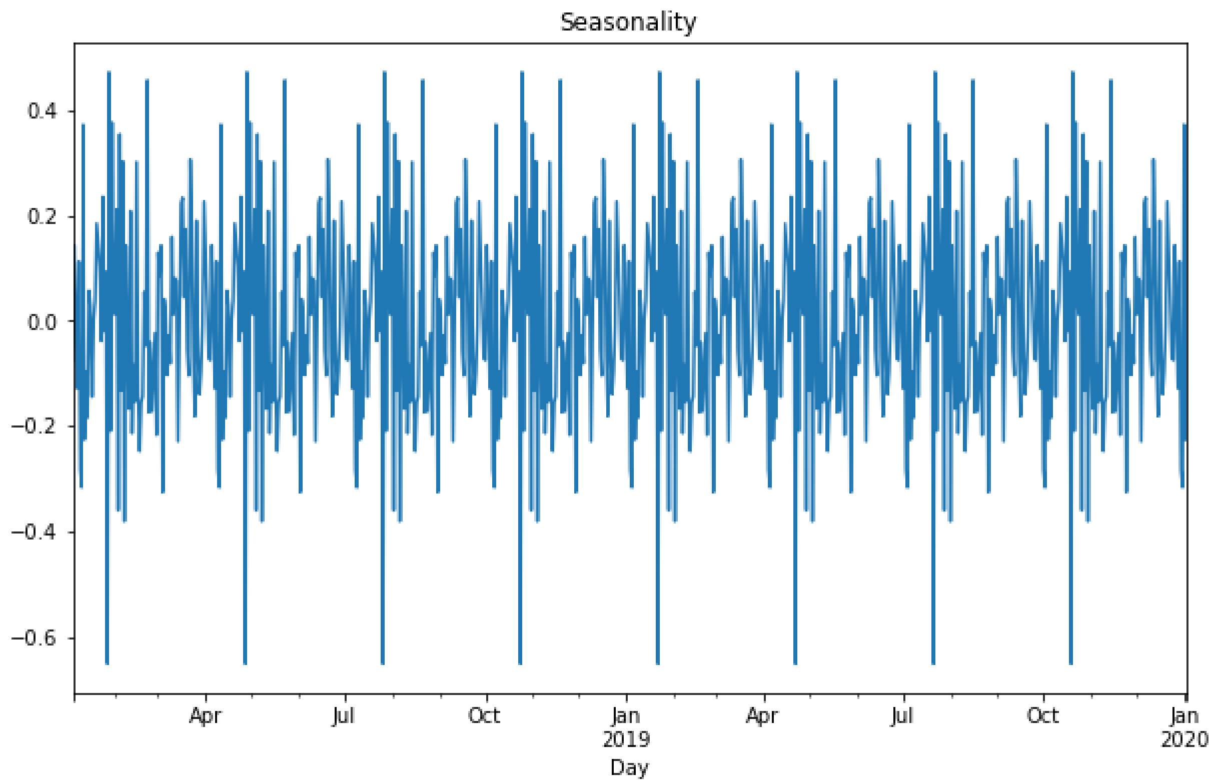
In [23]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
```

In [24]:
```
decomp = seasonal_decompose(df_stationary['Revenue'], period=90)
decomp.plot()
plt.show()
```



In [25]:
```
plt.title('Seasonality')
decomp.seasonal.plot()
```

Out[25]: <AxesSubplot:title={'center':'Seasonality'}, xlabel='Day'>



In [26]:
```
import matplotlib.mlab as mlab
import matplotlib.gridspec as gridspec
```

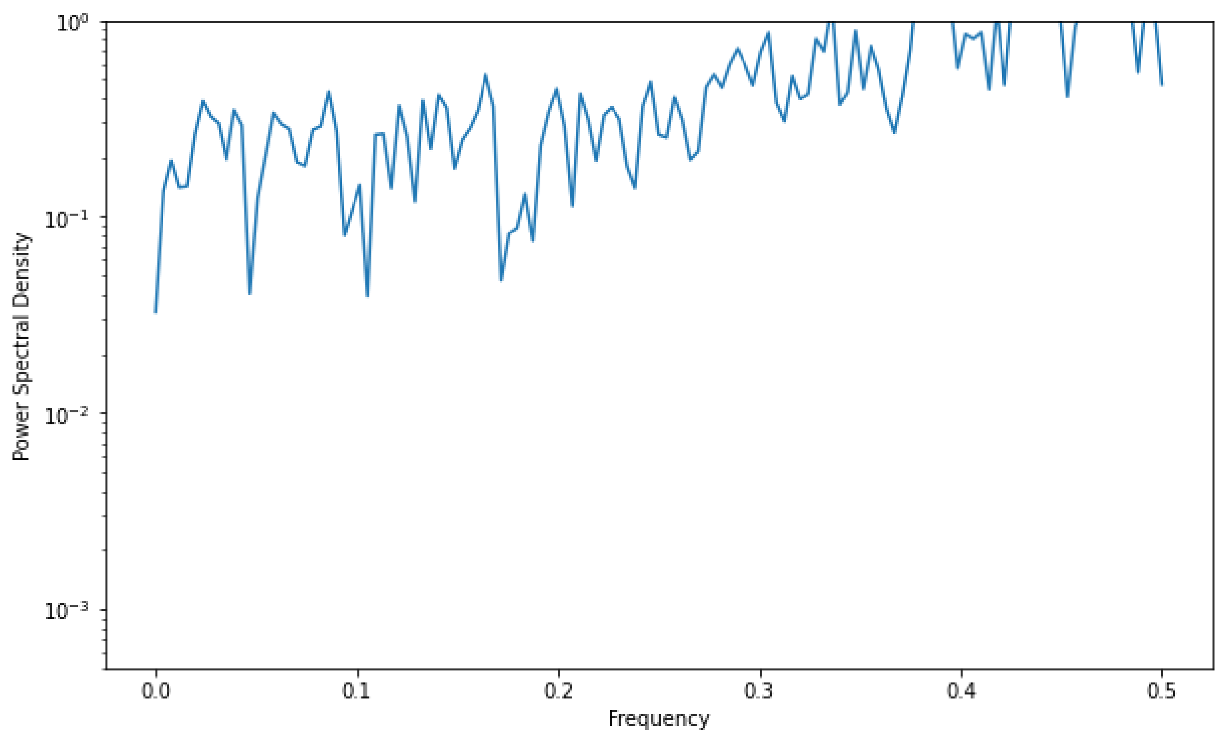
In [27]:
```
np.random.seed(19695601)
```

In [28]:
```
import scipy
from scipy import signal
```

In [29]:
```
from scipy.signal import welch
scipy.signal.welch(df_stationary, fs=1.0, nperseg=1)
```

Out[29]:
```
(array([0.]),
 array([[0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
        [0.],
```

In [30]:
```
f, Pxx_den = signal.welch(df_stationary['Revenue'])
plt.semilogy(f, Pxx_den)
plt.ylim([0.5e-3,1])
plt.xlabel('Frequency')
plt.ylabel('Power Spectral Density')
plt.show()
```



In [31]:
```
df_stationary.isnull()
```

Out[31]:

	Revenue
Day	
2018-01-04	False
2018-01-05	False
2018-01-06	False
2018-01-07	False
2018-01-08	False
...	...
2019-12-29	False
2019-12-30	False
2019-12-31	False
2020-01-01	False
2020-01-02	False

729 rows × 1 columns

In [32]:
```
df_stationary.dropna(inplace=True)
```

In [33]:
```
df_stationary.isna().sum()
```

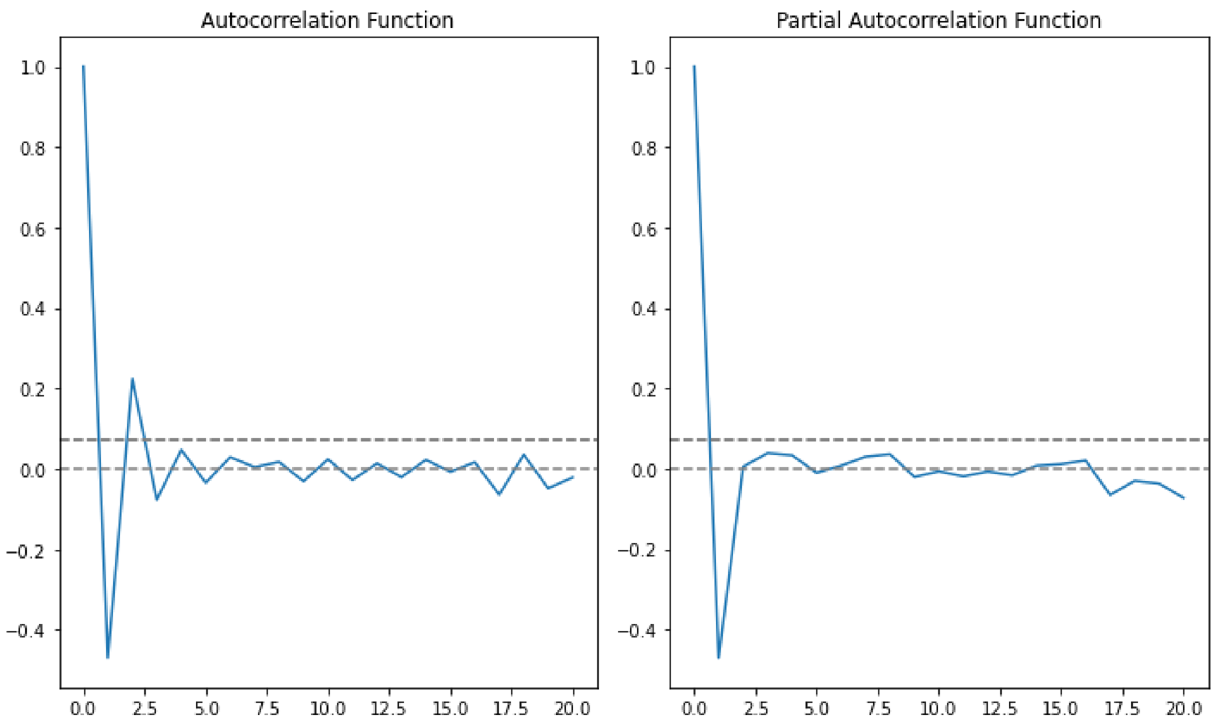
Out[33]:
```
Revenue    0
dtype: int64
```

In [34]:
```
from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(df_stationary, nlags=20)
lag_pacf = pacf(df_stationary, nlags=20, method='ols')

plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0, linestyle = '--', color='gray')
plt.axhline(y=1.96/np.sqrt(len(df_stationary)),linestyle='--', color ='gray')
plt.axhline(y=1.96/np.sqrt(len(df_stationary)), linestyle='--', color='gray')
plt.title('Autocorrelation Function')

plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0,linestyle = '--', color='gray')
plt.axhline(y=1.96/np.sqrt(len(df_stationary)),linestyle='--', color ='gray')
plt.axhline(y=1.96/np.sqrt(len(df_stationary)), linestyle='--', color='gray')
plt.title('Partial Autocorrelation Function')
plt.tight_layout()
```



In [35]:
```
from statsmodels.tsa.arima.model import ARIMA
from pandas import DataFrame
```

In [36]:
```
df_stationary.index = df_stationary.index.to_period('D')
```

In [37]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX
```

In [38]:
```
model = SARIMAX(df_stationary, freq = "D")
model_fit = model.fit()
```

In [39]:
```
print(model_fit.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  729
Model:               SARIMAX(1, 0, 0)   Log Likelihood                -489.851
Date:                Thu, 20 Jan 2022   AIC                            983.702
Time:                        20:36:04   BIC                            992.885
Sample:                    01-04-2018   HQIC                           987.245
                         - 01-02-2020
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -0.4682      0.033    -14.253      0.000      -0.533      -0.404
sigma2         0.2244      0.013     17.752      0.000       0.200       0.249
===============================================================================
====
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
2.12
Prob(Q):                              0.98   Prob(JB):
0.35
Heteroskedasticity (H):               1.02   Skew:                            -
0.02
Prob(H) (two-sided):                  0.88   Kurtosis:
2.74
===============================================================================
====

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

In [40]:
```
from sklearn.metrics import mean_squared_error
from math import sqrt
X = df_stationary.values
size = int(len(X) * 0.66)
train, test = X[0:size], X[size:len(X)]
history = [x for x in train]
predictions = list()
```

In [41]:
```
for t in range(len(test)):
    model = SARIMAX(history, order=(1,0,0))
    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))
```

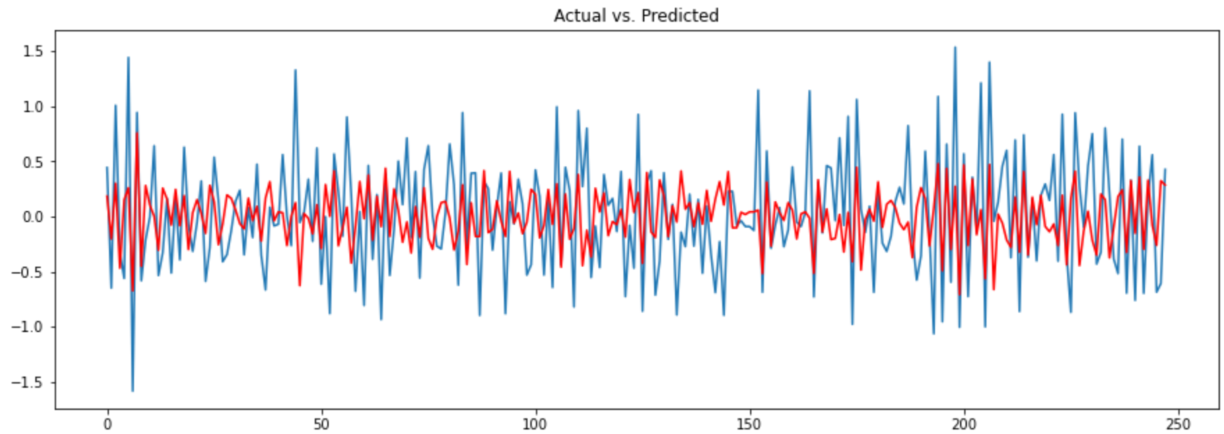
```
predicted=0.415775, expected=0.318120
predicted=-0.147002, expected=0.251187
predicted=-0.115874, expected=-0.305541
predicted=0.141039, expected=0.089945
predicted=-0.041510, expected=0.393701
predicted=-0.181599, expected=-0.882748
predicted=0.408700, expected=0.132842

predicted=-0.061303, expected=-0.067610
predicted=0.031200, expected=0.338037
predicted=-0.156041, expected=0.113264
predicted=-0.052220, expected=-0.531705
predicted=0.245319, expected=-0.437835
predicted=0.201022, expected=0.422243
predicted=-0.194117, expected=0.179940
predicted=-0.082548, expected=-0.531923
predicted=0.244286, expected=0.157387
predicted=-0.072235, expected=-0.644689
predicted=0.296250, expected=0.995057
predicted=-0.460011, expected=-0.438775
predicted=0.202788, expected=0.445385
```

In [42]:
```
rmse = sqrt(mean_squared_error(test, predictions))
print('Test RMSE: %.3f' % rmse)
```

```
Test RMSE: 0.481
```

In [43]:
```
plt.figure(figsize=(15,5))
plt.title('Actual vs. Predicted')
plt.plot(test)
plt.plot(predictions, color='red')
plt.show()
```



Part V: Data Summary and Implications

The ARIMA model that was selected SARIMA using SARIMAX. This model was selected because of the seasonality component to the data.

The prediction interval of the forecast is one day. It is one day because of data is in a daily interval for two years.

The choosen length for the forecast/predictions was to keep it the same as the original dataset. This way you can see how well the predicted values match with the orignal of the dataset.

You can use cell 41 to see the predicted vs actual values and cell 42 to see the RMSE of 0.481. The model that was choosen to be the final compared to the original can be seen as the output of cell 43.

Right off the gate I would recommend that a company not use this analysis right away to make business decisions. However I would continue to run and tweak the model until we had a more favorable RMSE then I could with confidence take this to upper managment ot help in making business decisions.

Online Sources

https://www.machinelearningplus.com/time-series/time-series-analysis-python/ (https://www.machinelearningplus.com/time-series/time-series-analysis-python/)

https://www.youtube.com/watch?v=CAT0Y66nPhs (https://www.youtube.com/watch?v=CAT0Y66nPhs)

https://www.aionlinecourse.com/blog/time-series-analysis-in-python (https://www.aionlinecourse.com/blog/time-series-analysis-in-python)

https://builtin.com/data-science/time-series-python (https://builtin.com/data-science/time-series-python)

https://medium.com/@josemarcialportilla/using-python-and-auto-arima-to-forecast-seasonal-time-series-90877adff03c (https://medium.com/@josemarcialportilla/using-python-and-auto-arima-to-forecast-seasonal-time-series-90877adff03c)

https://www.statology.org/durbin-watson-test-python/ (https://www.statology.org/durbin-watson-test-python/)