Jordan Tompkins

WGU D213, Advanced Data Analytics

Task 1

A1, Research Question: The research question that is being looked at in this assessment is whether or not we can accurately forecast the company's daily revenues?

A2, Objectives or Goals: The goal of this analysis is to be able to take a timeseries data set and determine/create an accurate forecast for the company's daily revenues. To do this, the data will be split into a training set and a testing set at a 80/20 split (80% of the data in the training set and 20% in the testing)

B, Summary of Assumptions: One of the main assumptions for time series data, is that the data is considered stationary. What this means is that the statistical properties of the data will not change over time. So the mean, variance, etc are not changing over time. It also means that the autocorrelation structure of the data does not change either. One wants to assume that the patterns and relationships (ie trends) over time are considered stable. This is important when it comes to using time series analysis techniquies such as forecasting, which will be used later on in this report. An article published by Statistics Solutions titled "The Stationary Data Assumptions in Time Series Analysis" explains this concept very well.

In [1]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
import seaborn as sns
from sklearn.model_selection import train_test_split
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.arima.model import ARIMA
```

In [2]:
```
df = pd.read_csv('teleco_time_series .csv', dtype={'locationid':np.int64})
df.head(5)
```

Out[2]:

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

In [3]:
```
start_date = pd.to_datetime('2020-01-01')
df['Day'] = pd.to_timedelta(df['Day']-1, unit = 'D') + start_date
df.set_index('Day', inplace = True)
```

In [4]:
```
df
```

Out[4]:

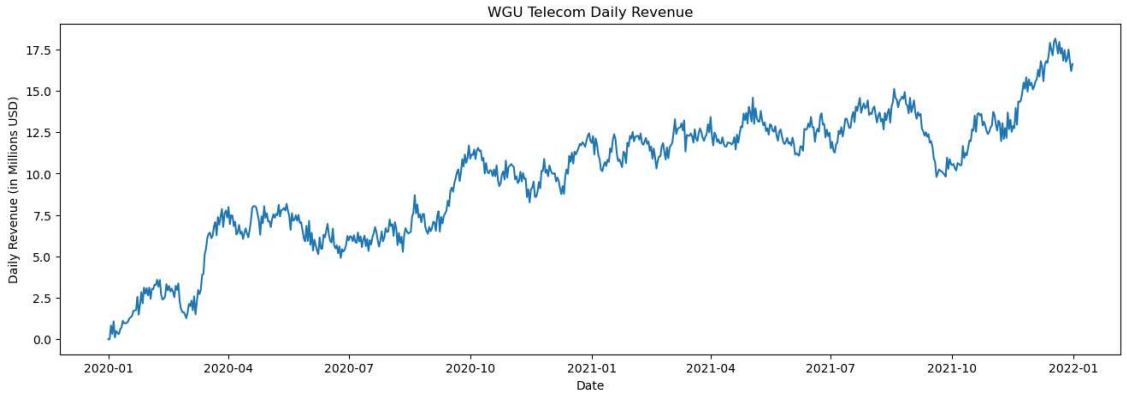
Day	Revenue
2020-01-01	0.000000
2020-01-02	0.000793
2020-01-03	0.825542
2020-01-04	0.320332
2020-01-05	1.082554
...	...
2021-12-27	16.931559
2021-12-28	17.490666
2021-12-29	16.803638
2021-12-30	16.194813
2021-12-31	16.620798

731 rows × 1 columns

C1, Line Graph: The Line graph below is a representation of the time series data provided by WGU that was used for this analysis. The initial data provided was for the Company's first couple years of operation. However, the "Day" column provided was not an actual date, but rather a number corresponding with the day of operation (for example, the first day of operation had a "1" in the day column). To use this data for the analysis, that day column needed to be converted to a date rather than a single number, so the code provided above was used to do just that. There was no indication on what the dates actually were so January 1, 2020 was used as the first day of operation and thus the original "Day" 1 was converted to "2020-01-01" and the remaining days followed suit.

In [5]:
```
plt.figure(figsize = [16,5])
plt.xlabel("Date")
plt.ylabel("Daily Revenue (in Millions USD)")
plt.title("WGU Telecom Daily Revenue")
plt.plot(df)
```

Out[5]:
```
[<matplotlib.lines.Line2D at 0x212ece321c0>]
```



C2, Time Step Formatting: Through examination of the data and the fact that the time series consists of a DateTime index, there appears to be no gaps within the measurements. The DateTime index incremements by single day observations and the data is 731 observations long. This indicated to us that the data consisted of a leap year which will add an extra day onto the data (so instead of 730 for 2 years, it is 731 for the two years). Besides that observation, there was appeared to be no gaps within the data.

C3, Stationairy of Series: There were a few ways that stationarity of the series could be determined. In order to be stationary, the trend has to be zero, which as shown by the graph in **C1**, there is clearly an upwards sloping trend. Because of this prescense of a trend, we can assume that the data is not stationary. Another test for non-stationarity is to use the Augmented Dicky-Fuller (ADF) test below. For this test, there is a null hypothesis that the time series is non-stationary, thus the alternative is that it is stationary. When determining if you would accept or reject the null hypothesis, one of the ways is to look at the p-value from the ADF. If the p-value is less then 0.05, you would be able to reject the null and conclude that the data is infact stationary. In the case shown below, the p-value is 0.32, well above the 0.05 testing threshold, thus failing to reject the null and confirming the data is infact non-stationary.

In [6]:
```
results = adfuller(df['Revenue'])
print("The data's test statistic is :", results[0])
print("The data's p-value is: ", results[1])
```

```
The data's test statistic is : -1.9246121573101842
The data's p-value is:  0.3205728150793961
```

C4, Steps to Prepare the Data: To start working with the data, we first needed to convert the "Day" column to a DateTime as shown in C1 above. From there, that column was turned into the index of the time series. As shown above, the time series data that we have been working with is not stationary. There are a few different ways to make the data stationary, or essentailly transform the data. The main transformation is using the diff() method to create an empty data

point at the beginning of the data, and then using .dropna() to delete that empty data point. After that, another ADF test will be ran and as shown below, the p-value is essentially zero, thus allowing us to reject the null hypothesis and conclude that the data is now stationary. Once the data was transformed to stationary data, train_test_split() was used throwing in the newly created stationary data, a test size of 0.2, and shuffle=False to maintain the order of the data since they are dates. The data was split in order for the model creation and testing conducted later on. The data just needed to be prepared before hand. See below for the code to transform the data and then split the data into the training and testing data sets.

In [7]:
```
df_stationary = df.diff().dropna()
```

In [8]:
```
results_stat = adfuller(df_stationary['Revenue'])
print("The data's test statistic is: ", results_stat[0])
print("The data's p-value is: ", results_stat[1])
```

```
The data's test statistic is:  -44.874527193876
The data's p-value is:  0.0
```

In [9]:
```
train, test = train_test_split(df_stationary, test_size = 0.2, shuffle = F
```

In [10]:
```
print(train)
```

```
             Revenue
Day
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
...              ...
2021-08-03  0.113264
2021-08-04 -0.531705
2021-08-05 -0.437835
2021-08-06  0.422243
2021-08-07  0.179940

[584 rows x 1 columns]
```

In [11]:
```
print(test)
```

```
            Revenue
Day
2021-08-08 -0.531923
2021-08-09  0.157387
2021-08-10 -0.644689
2021-08-11  0.995057
2021-08-12 -0.438775
...              ...
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

[146 rows x 1 columns]
```

C5, Copy of Prepared data: See code below as well as the prepared training and testing data sets provided separately.

In [12]:
```
train.to_csv(r'C:\Users\jorda\OneDrive\Documents\WGU Stuff\D213\JTompkins_
test.to_csv(r'C:\Users\jorda\OneDrive\Documents\WGU Stuff\D213\JTompkins_t
```

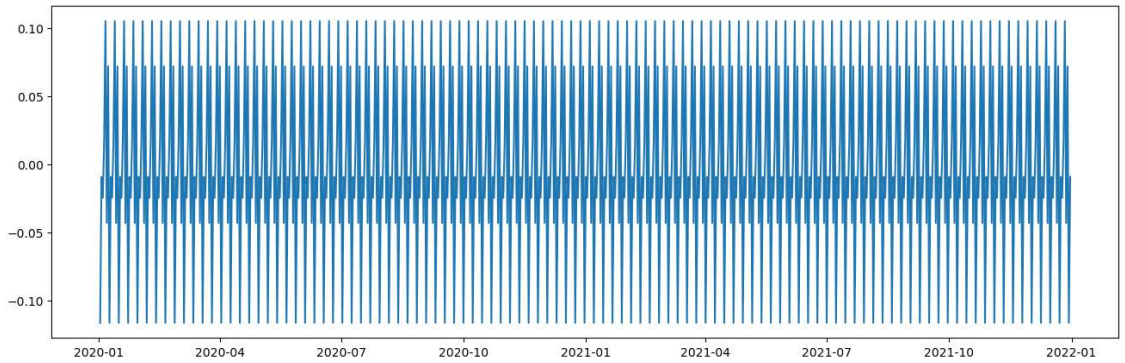
D1, Reporting Findings and Visualizations: See code below for annotated findings with visualizaitons of the data analysis.

Seasonality

In [13]:
```
#First step was decomposing the transformed data with the following code
decomp_results = seasonal_decompose(df_stationary)
type(decomp_results)
```

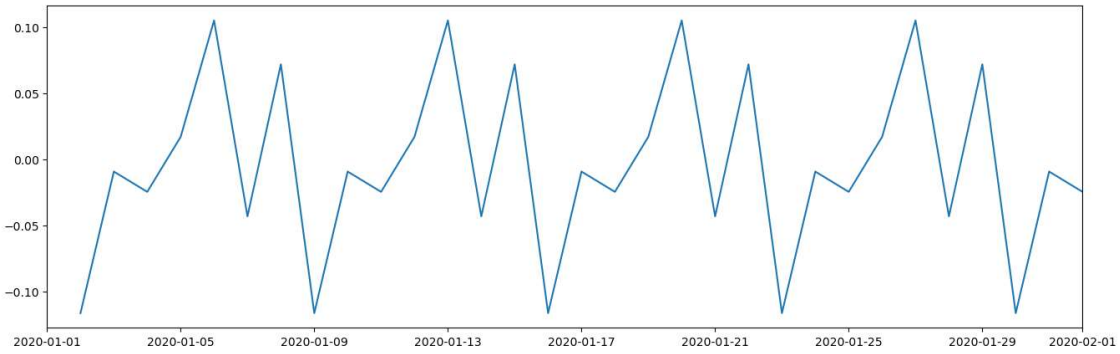
Out[13]: statsmodels.tsa.seasonal.DecomposeResult

In [14]:
```
#Code below allows for the visualization of a seasonal component
plt.figure(figsize=[16,5])
plt.plot(decomp_results.seasonal)
plt.show()
```



The graph above is a visualization for the decomposed data, specifically looking at the seasonal component. Looking at a large overview of the seasonality, we are able to slightly see that there appears to be a seasonal component for this data. The code below takes a closer look at a set range of the data, the month of January 2020, in order to get a better look at the seasonal component of the data.

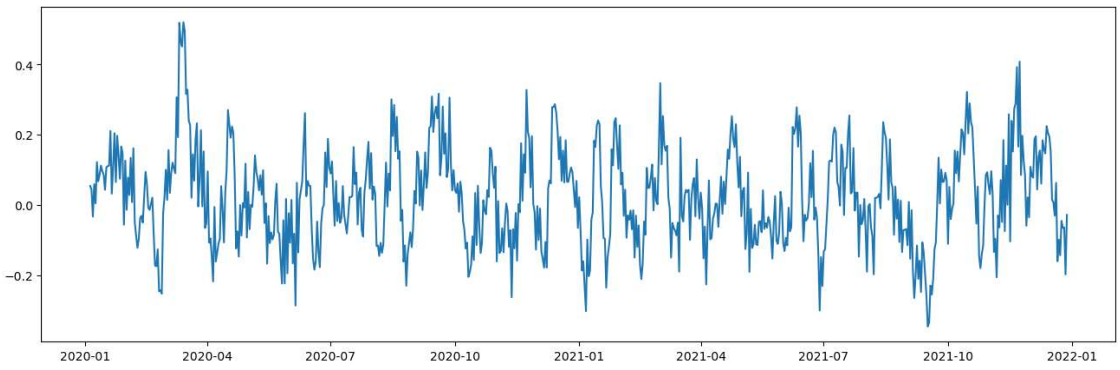
In [15]:
```
plt.figure(figsize = [16,5])
plt.plot(decomp_results.seasonal)
plt.xlim([pd.to_datetime('2020-01-01'), pd.to_datetime('2020-02-01')])
plt.show()
```



The graph above is a plot of the seasonal component of the data limited to just the month of January 2020. Limiting the data to a specific month allows for us to get a better understanding of the seasonal component of the data and can make the observation more clear. As shown by the graph above, it appears that every 7 days, the highest and lowest points appear to occur every 7 days and appear to be the same value. For example, the lowest point of the graph appear to occur on 1-2-2020, 1-9, 1-16 and so on. Between each of the lowest point, the graph increases and decreases at the same rate each and every week.

Trend: See below for the visualization of the trends for the data. As one can see, there appears to be no trends within the dataset.

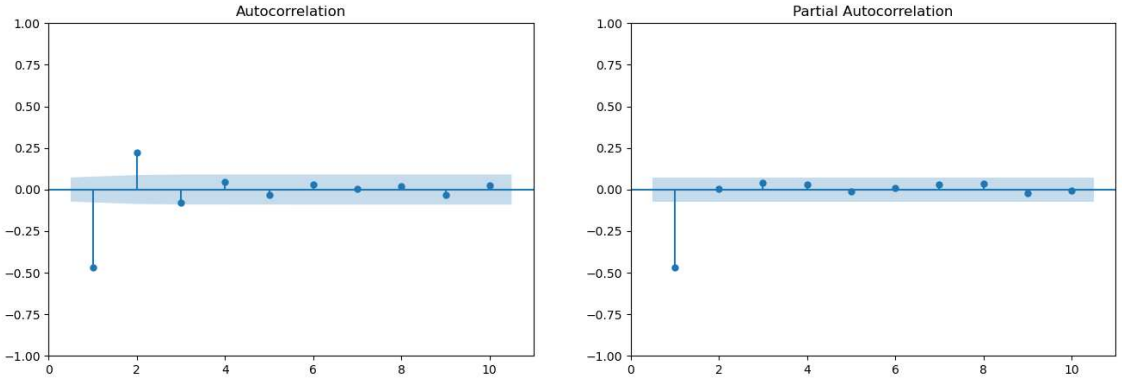
In [16]:
```
plt.figure(figsize = [16,5])
plt.plot(decomp_results.trend)
plt.show()
```



Auto Correlation Functions:

In [17]:
```
fig, (ax1, ax2) = plt.subplots(1,2, figsize=(16,5))
#Plotting the ACF with 10 lags and ignoring zero
plot_acf(df_stationary, ax = ax1, lags = 10, zero=False)
#Plotting the PACF with 10 lags and ignoring zero
plot_pacf(df_stationary, ax = ax2, lags = 10, zero = False)
plt.show()
```

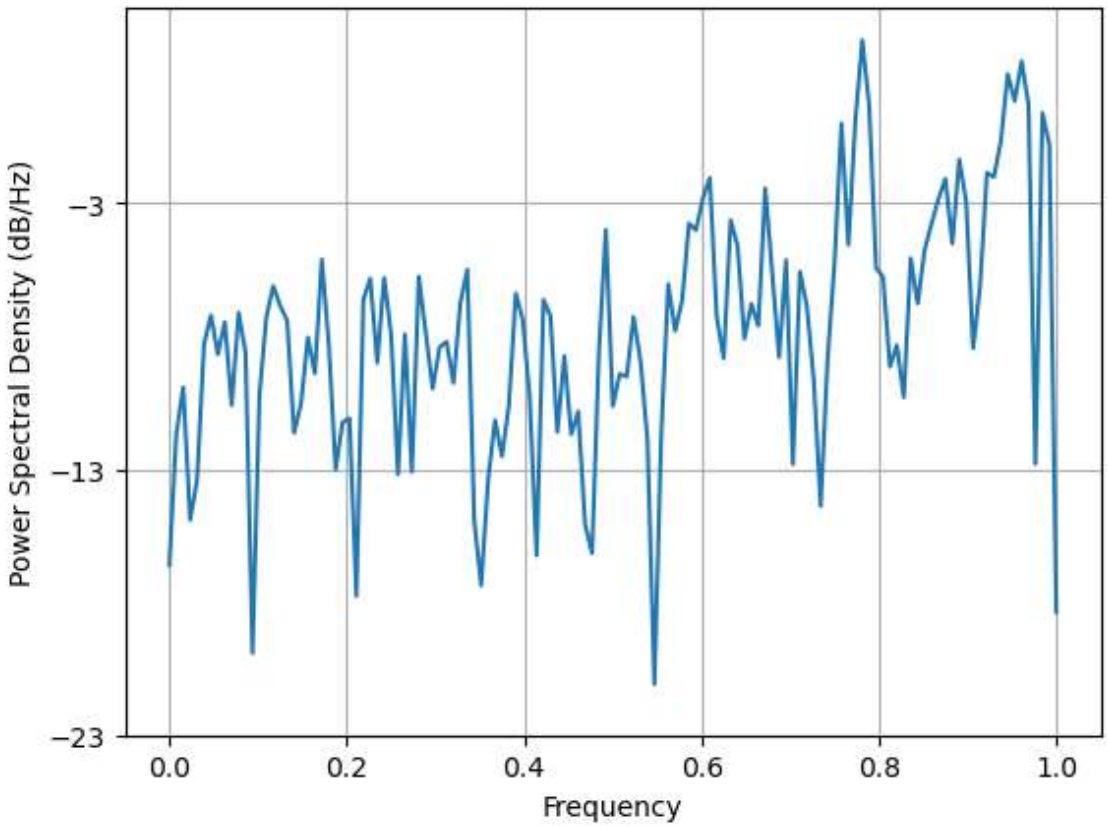
```
C:\Users\jorda\anaconda3\lib\site-packages\statsmodels\graphics\tsaplots.
py:348: FutureWarning: The default method 'yw' can produce PACF values ou
tside of the [-1,1] interval. After 0.13, the default will change tounadj
usted Yule-Walker ('ywm'). You can use this method now by setting method
='ywm'.
  warnings.warn(
```



Using the code above we are able to get visualizations of both the Autocorrelation (ACF) and Partial Autocorrelation (PACF) functions. Using these two functions, we are able to determine whether or not our data is an Autoregression (AR) model, or a Moving Average (MA) model. A way to determine this is to look at where the function appears to "tail" or "cut" off. A funciton that tails off will have data points that begin to slowly get closer to zero and enter into the shaded area, which represents the statistically insignificant results. The autocorrelation function above is an example where it begins to "tail off", in this case, at 1. After 1, it starts to gradually get closer to zero and gradually enter into that shaded area. The PACF visualization is an example of when the datapoints cuts off. After 1, the rest of the datapoints are in that shaded area. What these two visualizations shows is that the model to be used is an AR(1) model. The reason being, a model is an AR model if the ACF tails off after lag p, while the PACF cuts off after lag p. The ACF tails off after 1 while the PACF clearly cuts off after 1 indicating that the model is an AR(1) model.

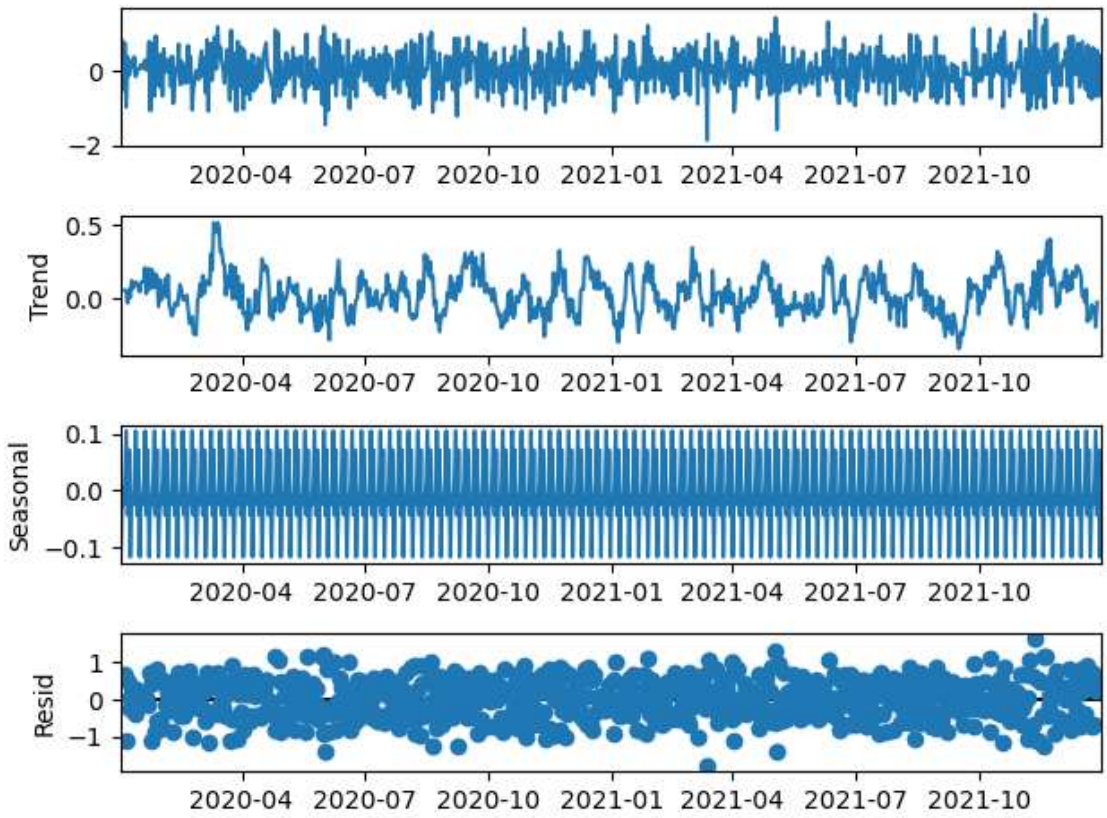
Spectral Density: See below for a plot of the spectral density

In [18]:
```
plt.psd(df_stationary.Revenue)
plt.show()
```



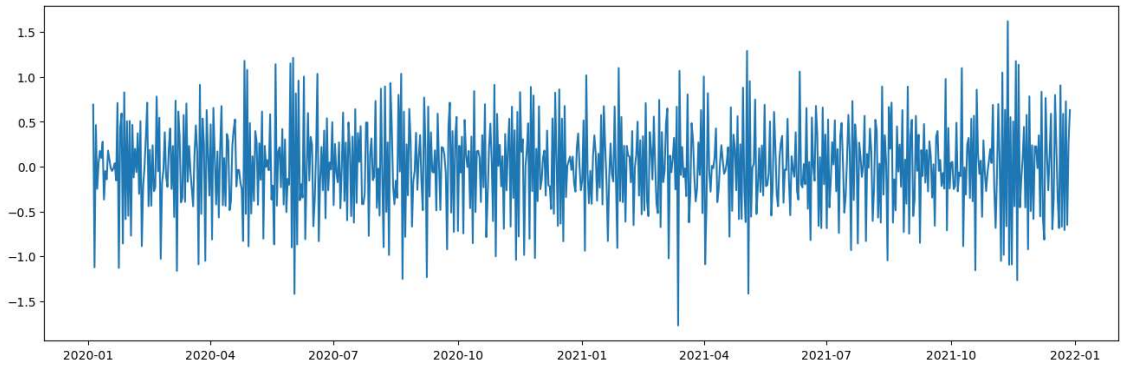
Decomposed Time Series: See below for plots of the decomposed time series. The plots below show the time series, the trend plot, seasonal plot, and even the residual plot.

In [19]:
```
decomp_results.plot()
plt.show()
```



Confirmation of lack of trends in the residuals of the decomposed series: See the plot below for the plot of the residuals of the decomposed series. As shown by the visualization, there appears to be no trend within the residuals of the decomposed series.

In [20]:
```
plt.figure(figsize = [16,5])
plt.plot(decomp_results.resid)
plt.show()
```



D2, ARIMA Model: As stated above, the model for the time series was an AR(1) model. This comes into play when we use the ARIMA method shown below. That lag 1 is used within the order poriton of the method below.

In [21]:
```
model = ARIMA(train, order = (1,0,0))
results = model.fit()
print(results.summary())
```

```
                               SARIMAX Results
==============================================================================
=====
Dep. Variable:                Revenue   No. Observations:
584
Model:                 ARIMA(1, 0, 0)   Log Likelihood                -38
3.946
Date:                Mon, 11 Sep 2023   AIC                            77
3.893
Time:                        19:45:34   BIC                            78
7.002
Sample:                    01-02-2020   HQIC                           77
9.002
                         - 08-07-2021
Covariance Type:                 opg
==============================================================================
=====
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------------------
-----
const          0.0234      0.013      1.758      0.079      -0.003
0.049
ar.L1         -0.4597      0.036    -12.654      0.000      -0.531      -
0.388
sigma2         0.2180      0.014     16.034      0.000       0.191
0.245
==============================================================================
==========
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):
1.84
Prob(Q):                              0.96   Prob(JB):
0.40
Heteroskedasticity (H):               0.97   Skew:
-0.08
Prob(H) (two-sided):                  0.83   Kurtosis:
2.77
==============================================================================
==========

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

```
C:\Users\jorda\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:471: ValueWarning: No frequency information was provided, so inferre
d frequency D will be used.
  self._init_dates(dates, freq)
C:\Users\jorda\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:471: ValueWarning: No frequency information was provided, so inferre
d frequency D will be used.
  self._init_dates(dates, freq)
C:\Users\jorda\anaconda3\lib\site-packages\statsmodels\tsa\base\tsa_mode
l.py:471: ValueWarning: No frequency information was provided, so inferre
d frequency D will be used.
  self._init_dates(dates, freq)
```

The resulting ARIMA equation would be Rt = 0.2180 -0.4597(R(t-1)) + εt

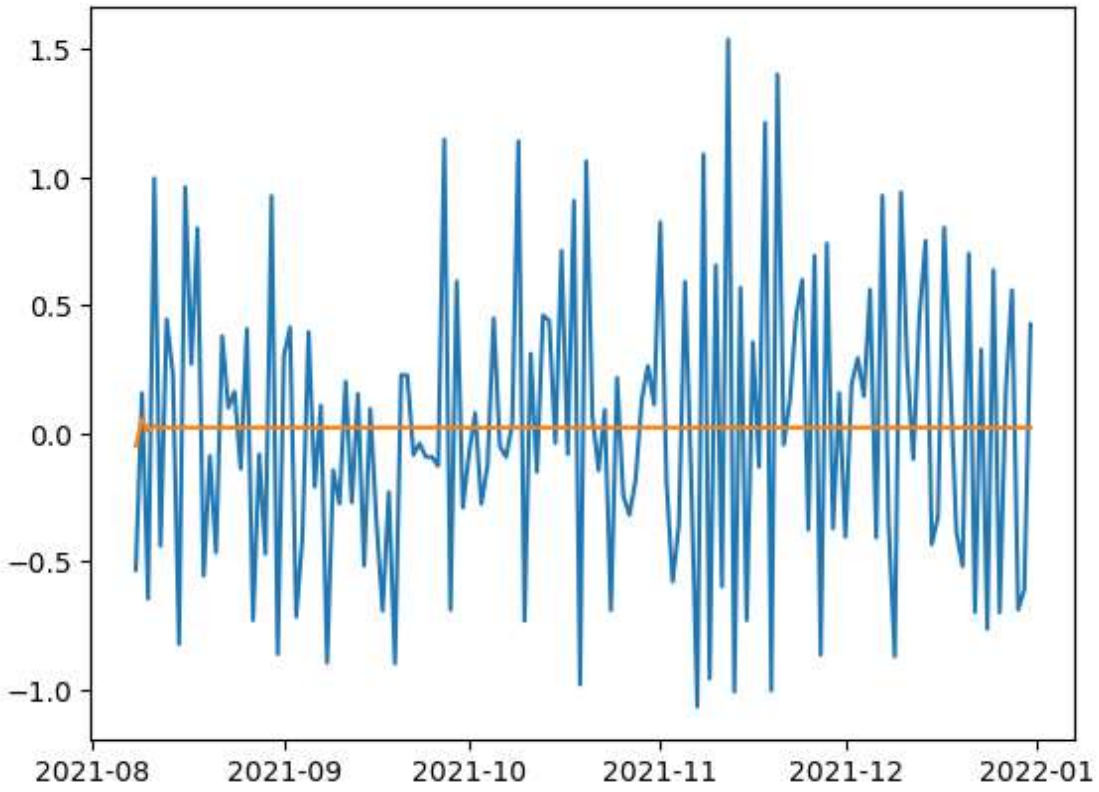
D3, Forecasting using ARIMA Model:

In [22]:
```
forecast = results.get_prediction(start = 584, end = 729, dynamic = True)
mean_forecast = forecast.predicted_mean
print(mean_forecast)
```

```
2021-08-08   -0.048621
2021-08-09    0.056441
2021-08-10    0.008147
2021-08-11    0.030347
2021-08-12    0.020142
                ...
2021-12-27    0.023356
2021-12-28    0.023356
2021-12-29    0.023356
2021-12-30    0.023356
2021-12-31    0.023356
Freq: D, Name: predicted_mean, Length: 146, dtype: float64
```

In [23]:
```
#Plot of the test set and the predicted means of the forecasted value
#to get a comparison of predicted versus actual values
plt.plot(test)
plt.plot(mean_forecast)
```

Out[23]:
```
[<matplotlib.lines.Line2D at 0x212f21cdac0>]
```



In [24]:
```
confidence_interval = forecast.conf_int()
print(confidence_interval)
```

```
            lower Revenue  upper Revenue
2021-08-08      -0.963665       0.866422
2021-08-09      -0.950645       1.063528
2021-08-10      -1.017331       1.033625
2021-08-11      -0.998976       1.059669
2021-08-12      -1.009990       1.050275
...                   ...            ...
2021-12-27      -1.006994       1.053705
2021-12-28      -1.006994       1.053705
2021-12-29      -1.006994       1.053705
2021-12-30      -1.006994       1.053705
2021-12-31      -1.006994       1.053705

[146 rows x 2 columns]
```

The values within the predicted mean and the confidence intervals appear to be small, but that is because these values are associated with the daily difference in the revenues rather than the actual daily revenues. Because of this, the following code can be used to convert the data into the proper format for the daily revenue and then allow us to get those confidence intervals as

In [25]:
```
from numpy import cumsum
mean_forecast = cumsum(mean_forecast) + df.iloc[-1,0]
mean_forecast
```

Out[25]:
```
2021-08-08    16.572177
2021-08-09    16.628618
2021-08-10    16.636766
2021-08-11    16.667112
2021-08-12    16.687254
                 ...
2021-12-27    19.887992
2021-12-28    19.911347
2021-12-29    19.934703
2021-12-30    19.958059
2021-12-31    19.981414
Freq: D, Name: predicted_mean, Length: 146, dtype: float64
```

In [26]:
```
confidence_interval = cumsum(confidence_interval) + df.iloc[-1,0]
confidence_interval
```

Out[26]:

	lower Revenue	upper Revenue
2021-08-08	15.657134	17.487220
2021-08-09	14.706489	18.550748
2021-08-10	13.689158	19.584373
2021-08-11	12.690182	20.644042
2021-08-12	11.680192	21.694317
...	...	...
2021-12-27	-126.276900	166.052883
2021-12-28	-127.283893	167.106588
2021-12-29	-128.290887	168.160293
2021-12-30	-129.297881	169.213998
2021-12-31	-130.304875	170.267704

146 rows × 2 columns

In [27]:
```
plt.figure(figsize = [16,5])
plt.plot(mean_forecast, color = 'green', linestyle = 'dashed')
plt.plot(df, color = 'blue')
plt.fill_between(confidence_interval.index, confidence_interval['lower Rev
plt.ylim(-7,27)
plt.legend(['Predicted', 'Actual'])
plt.show()
```

The plot above shows the company's daily revenue as well as the forecast for the final 20% of the time period. This forecast was created based on the prior 80% (the training data) and shown along with the actual values for the period.

In [28]:
```
residuals = results.resid
mae = np.mean(np.abs(residuals))
print(mae)
```

```
0.37639356837030136
```

The above code was used to calculate the MAE or Mean Absolute Error. This is the average absolute difference between the predicted and actual values of the model. The lower the MAE, the better since a lower score would mean that the average absolute difference between the predicted and actual values was smaller. In this case, 0.38 was the MAE which indicates a good score and model.

D4, Analysis Output and Calculations: See code provided in previous sections for the calculations performed and the outputs

D5, ARIMA Model Code: See code above for the code used in the implementation of the time series model.

E1, Results:

The ARIMA model was selected using the Autocorrelation (ACF) and Partial Autocorrelation (PACF) functions. These two functions allowed for the determination of whether the model was an Autoregression model (AR) or a moving average (MA) model. Within D1 it was determined that the model was an AR(1) model which means that it was an autoregression model with 1 lagged value included. This led to the equation for the dataset being:

$Rt = 0.2180 -0.4597(R(t-1)) + \varepsilon t$

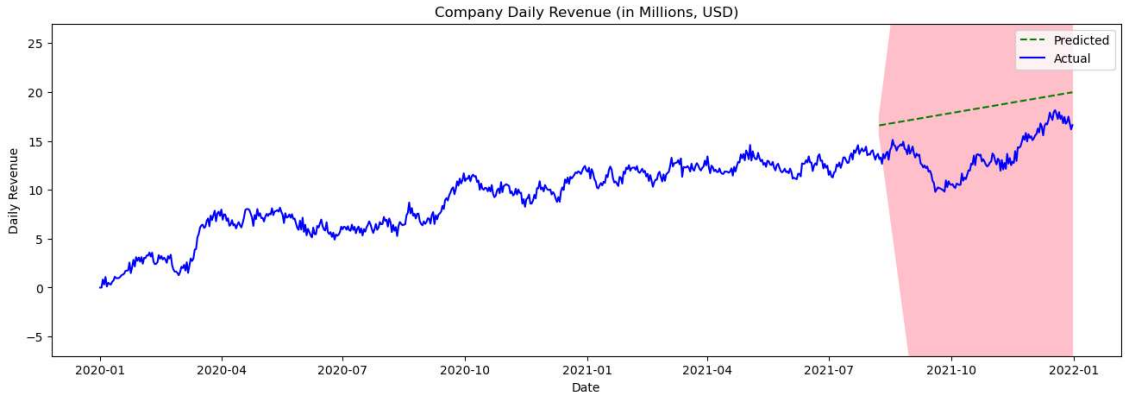
The prediction interval of the forecast was determined when deciding how to split the dataset into the training and testing dataset. The test size that was used was 0.2, or 20%. There is no specific reason for 20%, rather the fact that 20% of the data being split into the testing dataset is typically considered a reasonable amount of the data.

The forecast length was determined by the fact that the prediction interval was 20%. This left us with 80% of the data, or roughly 19 months worth of data to use for the forecast. Throughout those 19 months, the Company continues to grow and develop until it reaches that range where the daily revenue begins to stay roughly the same and if/when they increase, it does so at a much slower rate. We are able to see at the beginning of the Company the highs and lows of starting up before we see them start to pull in a much larger profit. In the long run, this might not be useful due to the fact that it appears that it took roughly 3 months for the Company to even see a considerable daily revenue. Those months, daily revenue was waivering around zero, before it began to steadily increase. For a company that is just starting, this data is fine to include, however, for future analysis, a larger dataset that does not include those initial months due to them being those start up months before the Company saw any steady increase in daily revenue.

The model was evaluated by looking at the Mean Absolute Error. This metric shows us the average absolute difference between the predicted and actual values of the model. In the model created, the MAE was 0.38, which is indicative of a good model. The lower the MAE, the better the model. Another way to evalue the model's performance is by looking at the Akikie Information Criteria and the Bayseian Information Criteria (AIC and BIC respectively). For these two metric, the lower the score the better. The model created had score of 774 and 787 for the AIC and BIC respectively. These scores are decently low, however, they'd be of more use if you were to compare the scores to the scores of different models, so in this analysis, they aren't the most useful due to only one model being created.

E2, Annotated Visualization: See below for the visualization of the observed data provided by WGU and the forecast created from the model, with the confidence intervals dispalyed

In [29]:
```
plt.figure(figsize = [16,5])
plt.plot(mean_forecast, color = 'green', linestyle = 'dashed')
plt.plot(df, color = 'blue')
plt.fill_between(confidence_interval.index, confidence_interval['lower Rev
plt.ylim(-7,27)
plt.title("Company Daily Revenue (in Millions, USD)")
plt.xlabel("Date")
plt.ylabel("Daily Revenue")
plt.legend(['Predicted', 'Actual'])
plt.show()
```



E3, Recommended Action: With the data provided, the company can use the model to help them conduct a time series analysis and create a forecast for daily revenues with little to no issue. As time goes on and more data is collected, however, that data should be updated and they might want to think about omitting the first 4 or so months from the forecasting. The reason being, the data those first months of operation are much lower compared to the rest of the data. There was a spike around month 4 and from there the daily revenues appear to slowly increase day by day and never drop anywhere close to 0. As time goes on the company could consider omitting those months. Further down the line, as the company has been around for a much longer time frame and has much more data to analyze, they can even consider limiting that data to say 5 years and doing 5 year forecasts. For now, the company is able to use the data provided and are able to confidently create a usable forecast for they're analysis.

F, Reporting: This analysis was created using Jupyter Notebook and has been uploaded along side a PDF copy of this executed notebook

G, Sources for Third-Party Code: There was no third-party code used to conduct the analysis.

H, Sources:

Ravelo, C. (2023, March 9). The Stationary Data Assumption in time series analysis. Statistics Solutions. https://www.statisticssolutions.com/stationary-data-assumption-in-time-series-analysis/ (https://www.statisticssolutions.com/stationary-data-assumption-in-time-series-analysis/)

In [ ]: