Time Series Analysis

Charles Rowe, ID# 006539689, WGU D213 Advanced Data Analytics, Task 1

A1: Research Question

For this analysis, I would like to answer if there are any significant trends in revenue, and if I can accurately forecast these numbers compared to the observed values.

A2: Goals of Analysis

The goal of this analysis is to find if there are significant trends in the revenue and if an accurate forecast can be made.

B: Summary of Assumptions

For our ARIMA (autoregressive integrated moving average) model, the assumptions we are going to be focused on are the assumptions of stationarity and autocorrelation.

- Stationarity is the assumption that there is a constant mean and variance within the data, and that there are no trends or seasonality.
- Autocorrelation (or rather the assumption that there is no autocorrelation) is the assumption that the data has not been autocorrelated.

C1: Time Series Visualization

Below is the visualization of our time series, including the preprocessing needed to convert values into dates.

Import all libraries and read head of dataframe

In [2]:
```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.dates as mdates
import seaborn as sns
from sklearn.model_selection import train_test_split
from sklearn.metrics import mean_squared_error
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

#Printing head of med_df (time series data)

med_df = pd.read_csv('medical_time_series .csv')
print(med_df.head)
```

```
<bound method NDFrame.head of      Day    Revenue
0      1   0.000000
1      2  -0.292356
2      3  -0.327772
3      4  -0.339987
4      5  -0.124888
..   ...        ...
726  727  15.722056
727  728  15.865822
728  729  15.708988
729  730  15.822867
730  731  16.069429

[731 rows x 2 columns]>
```

Converting "day" column to date

The "Day" column contains little information that we can use for time series analysis. The document received along with the data set states that this column contains information on the "first two years" of operations, but does not say which year operations began. As such, I can take this at my own discretion. As the data set contains 731 observations, we know that there is a leap year to take into consideration, so I will either need to start on a leap year, or a year before a leap year. Since the documnet provided mentions that 78% of hospitals were penalized by the Centers for Medicare and Medicaid Services in FY 2015, I would like to start on a date after this. So, I will start on January 1, 2020. This is a leap year and is after 2015, so this meets my criteria.

(Below uses code from William Townsend)

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

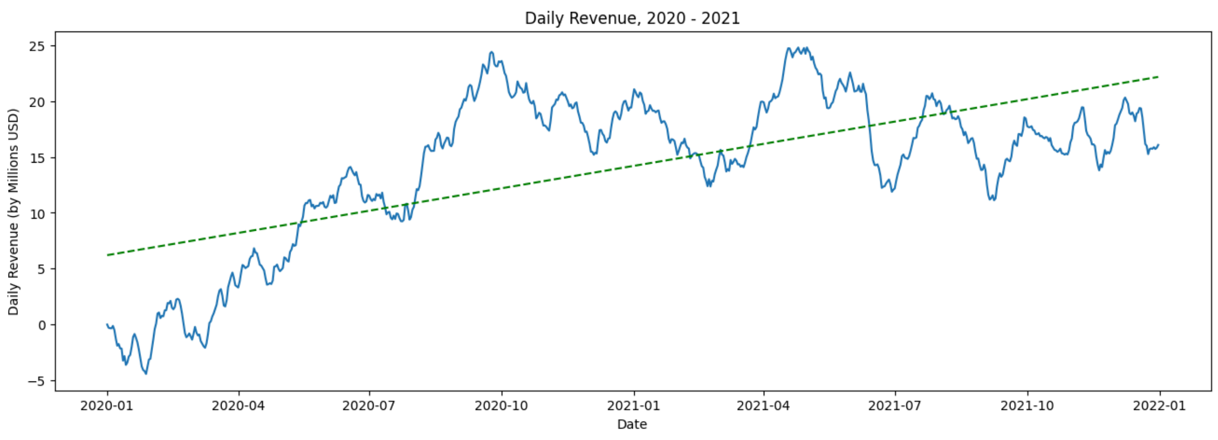
Out[3]:

	Revenue
Day	
2020-01-01	0.000000
2020-01-02	-0.292356
2020-01-03	-0.327772
2020-01-04	-0.339987
2020-01-05	-0.124888
...	...
2021-12-27	15.722056
2021-12-28	15.865822
2021-12-29	15.708988
2021-12-30	15.822867
2021-12-31	16.069429

731 rows × 1 columns

In [4]:

```python
plt.figure(figsize = [16,5])
plt.title('Daily Revenue, 2020 - 2021')
plt.xlabel('Date')
plt.ylabel('Daily Revenue (by Millions USD)')
plt.plot(med_df)

x = mdates.date2num(med_df.index)
y = med_df.Revenue
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
plt.plot(x, p(x), 'g--')
plt.show()
```



Looking at the above, we can see an obvious upward trend in revenue from the first two years of operation, made more obvious by the gree trendline. Further anaylsis will be needed to determine if there are any seasonality trends.

C2: Description of Time Step Formatting

The provided dataset contains two columns: 'Day' and 'Revenue'. It is stated that the 'Day' pertains to each day in the span of the first two years of operation of this hospital. This being said, the document provided with the dataset does not provide the year that operations began. Looking over the 'Day' column, it consists of 731 data points. This coincides with the number of days in two years if one of the years was a leap year. As stated above, since the document provided mentions that 78% of hospitals were penalized by the Centers for Medicare and Medicaid Services in FY 2015, I started on a date after this. As such, I incremented a datetime index by days (starting with January 1, 2020) and replaced the original integers within the 'Day' column.

C3: Stationarity of Series

Due to the obvious upward trend in the graph above, this data is not stationary. For the first few months of operation, revenue stayed negative most of the time. However, around April of 2020, revenue began to increase, reaching peaks around October of 2020 and May of 2021.

C4: Data Preparation and Explanation

There are four total steps I will need to take to prepare data for analysis. Two have already been completed above (1 and 2 below)

1. Convert "Day" column to time series
2. Reindex dataframe using "Day" colum
3. Convert non-stationary data into stationary data
4. Separate data into train and test sets

Before continuing, and even though there is an obvious upward trend, I would like to confirm that our data is non-stationary. To do this, I will employ an Augmented Dicky-Fuller test on the "Revenue" column. The Dicky-Fuller test will test the null hypothesis (that our data is non-stationary) using unit root analysis and giving us a p-value, of which we are looking for a value <0.05.

```
In [5]: fuller_test = adfuller(med_df.Revenue)

        print(f"Test Statistic: {round(fuller_test[0], 3)}\np-value: {round(fuller_t
```

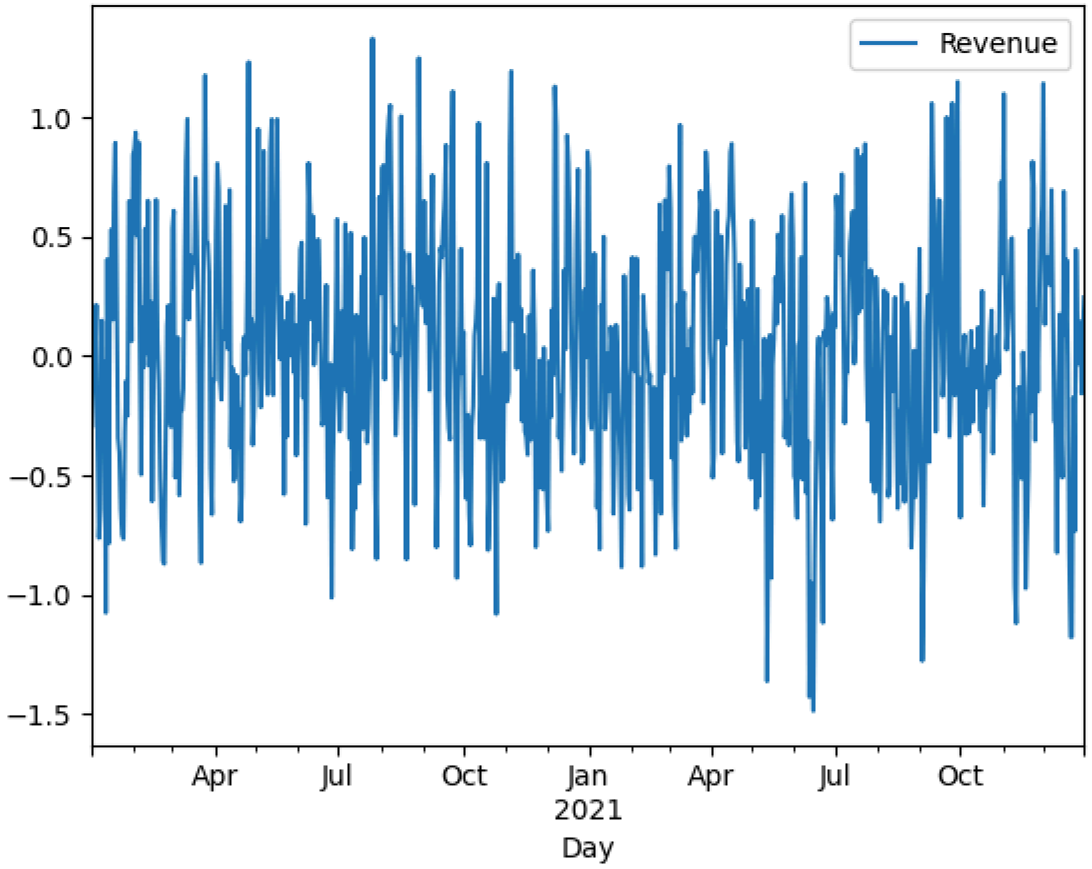
```
Test Statistic: -2.218
p-value: 0.2
```

In running the above code, the Dicky-Fuller test shows a p-value of 0.2, meaning that I am unable to reject the null hypothesis. As a result, I can conclude that this data is not stationary and I will need to make it stationary to continue the analysis. To begin transforming the data, I will need to choose a method to stabilize the mean. For simplicity, I will choose the "differencing" method. Using this method, I will calculate the difference between each consecutive value, using the pandas library's .diff method. This will, however, create a blank value due to the first value trying to difference from a value that is not available. I will need to drop the first difference, which is done easily with .dropna().

```
In [6]: med_df_diff = med_df.diff().dropna()
        fuller_test = adfuller(med_df_diff.Revenue)
        print(f"Test Statistic: {round(fuller_test[0], 3)}\np-value: {round(fuller_t
        med_df_diff.plot()
```

```
Test Statistic: -17.375
p-value: 0.0
```

```
Out[6]: <Axes: xlabel='Day'>
```



Based on the above Dicky-Fuller test and the succeeding graph, the data is now stationary and is able to be used for analysis. I will need to split the data into training and testing sets. To do so, I will use scikit-learn's train_test_split function. However, I will need to feed the "shuffle=False" variable into the function so that the data is not rearranged. I will then call both variables to confirm.

In [7]:
```
X_train, X_test = train_test_split(med_df_diff, test_size=0.3, shuffle=False
X_train
```

Out[7]:

	Revenue
Day	
2020-01-02	-0.292356
2020-01-03	-0.035416
2020-01-04	-0.012215
2020-01-05	0.215100
2020-01-06	-0.366702
...	...
2021-05-22	0.228260
2021-05-23	0.589730
2021-05-24	0.276029
2021-05-25	-0.342315
2021-05-26	-0.188423

511 rows × 1 columns

In [8]:
```
X_test
```

```
Out[8]:
```

Day	Revenue
2021-05-27	-0.234087
2021-05-28	-0.374122
2021-05-29	0.549773
2021-05-30	0.681126
2021-05-31	0.480611
...	...
2021-12-27	-0.032693
2021-12-28	0.143766
2021-12-29	-0.156834
2021-12-30	0.113880
2021-12-31	0.246562

219 rows × 1 columns

C5: Copy of Prepared Data Set

I will save both of the sets above as separate CSVs and will provide them in the submission.

```
In [9]: X_train.to_csv('train_set.csv')
        X_test.to_csv('test_set.csv')
```

D1: Annotated Findings and Visualizations

Below I will discuss the decomposition of the data. This will include visualizations of the seasonality, trends, auto correlation, spectral density, and the fully decomposed data.

Seasonality

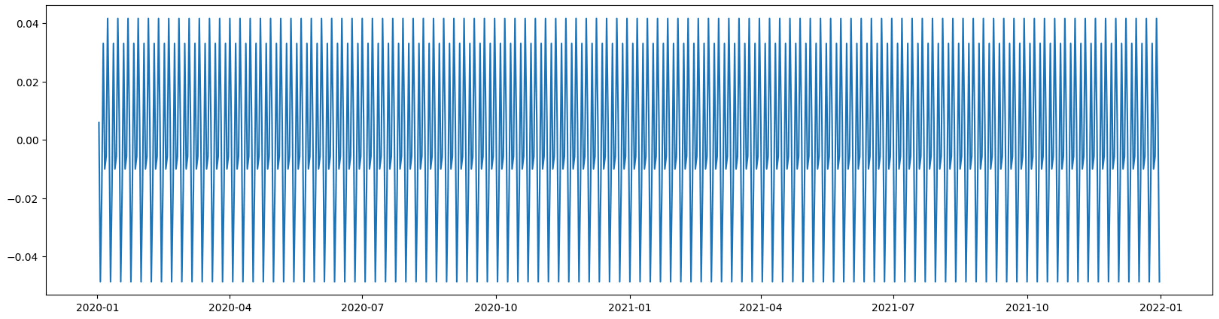
Seasonality is a predictable change that occurs within a specific time frame. Take electronics sales for example. Electronics sales usually peak at certain times during the year, normally through Spring (tax season), Late Summer (Back-to-

School season), and Late Fall to Early Winter (Holiday season). This is an example of seasonality.

Below, I will analyze any seasonality within the stationary data.

```
In [10]: med_df_decomp = seasonal_decompose(med_df_diff)
         plt.figure(figsize=[20,5])
         plt.plot(med_df_decomp.seasonal)
```

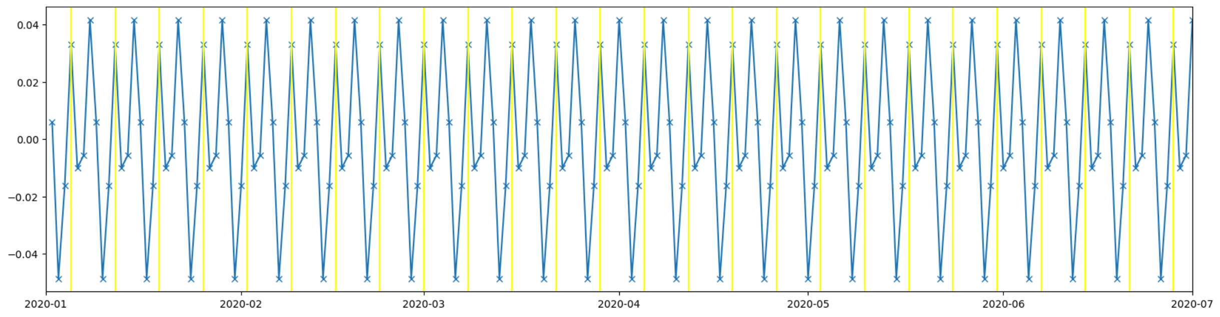
```
Out[10]: [<matplotlib.lines.Line2D at 0x28c40dfdbd0>]
```



The above figure shows the decomposed data's seasonality component. There is a pattern with peaks and troughs, indicating that there does seem to be a seasonality with the data. Because the data consists of daily revenue, I would like to see this pattern on a smaller scale. As such, I will create a visualzation showing 6 months worth of data with a line indicating the start of a new week (Sunday).

```
In [11]: plt.figure(figsize=[20,5])
         plt.plot(med_df_decomp.seasonal, marker='x')
         plt.xlim(pd.to_datetime('2020-01-01'), pd.to_datetime('2020-07-01'))
         plt.axvline(x=pd.to_datetime('2020-01-05'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-01-12'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-01-19'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-01-26'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-02-02'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-02-09'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-02-16'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-02-23'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-03-01'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-03-08'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-03-15'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-03-22'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-03-29'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-04-05'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-04-12'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-04-19'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-04-26'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-05-03'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-05-10'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-05-17'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-05-24'), color='yellow')
         plt.axvline(x=pd.to_datetime('2020-05-31'), color='yellow')
plt.axvline(x=pd.to_datetime('2020-06-07'), color='yellow')
plt.axvline(x=pd.to_datetime('2020-06-14'), color='yellow')
plt.axvline(x=pd.to_datetime('2020-06-21'), color='yellow')
plt.axvline(x=pd.to_datetime('2020-06-28'), color='yellow')
```

```
Out[11]:  <matplotlib.lines.Line2D at 0x28c40e61090>
```



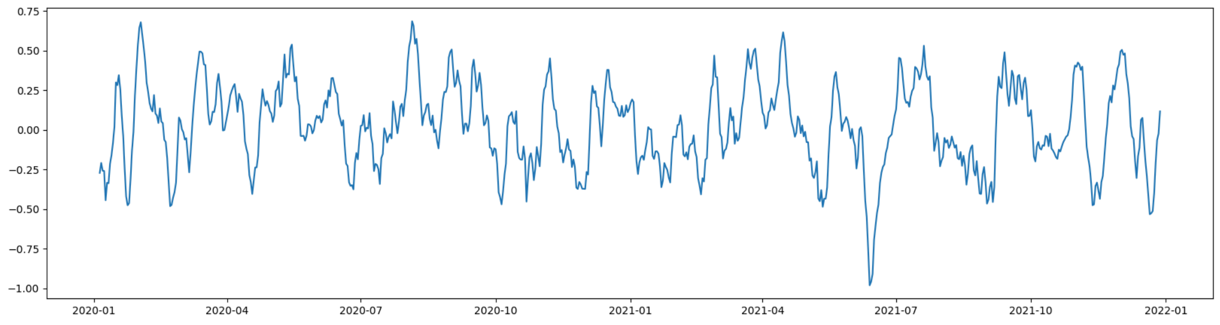
Looking over the above data, there does seem to be an inherent pattern. Peaks seem to occur on Sunday and Wednesday while the throughs seems to occur on Mondays and Fridays. While there are theories that could be made from the patterns above, more research and data would be needed to reach any concrete conclusion.

Trends

Below is a visualization of trends within the decomposed data.

```
In [12]:  plt.figure(figsize=[20,5])
          plt.plot(med_df_decomp.trend)
```

```
Out[12]:  [<matplotlib.lines.Line2D at 0x28c40ec9f90>]
```



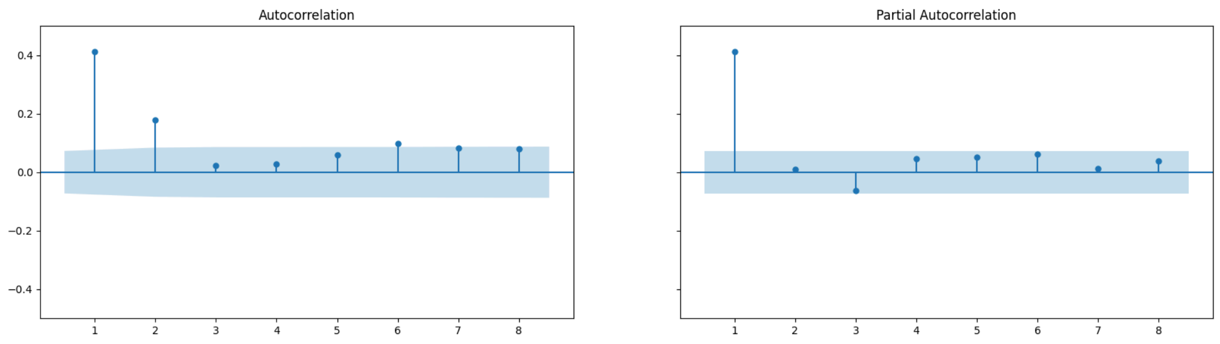
There do not seem to be any apparent trends apart from a large dip during June of 2021.

Autocorrelation Functions

The Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) are used to determine if a dataset should be used with an autoregression model (AR) or a moving average (MA) model. Below is a visualization of these said functions.

```
In [13]: fig, (ax1, ax2) = plt.subplots(1, 2, figsize=[20,5], sharey=True)
         plot_acf(med_df_diff, lags=8, zero=False, ax=ax1)
         plot_pacf(med_df_diff, lags=8, zero=False,ax=ax2)
         plt.ylim(-0.5, 0.5)
```

```
Out[13]: (-0.5, 0.5)
```



According to IBM, when selecting a model to use, general guidelines are as follows:

AR Model:

- ACF plots show autocorrelation decaying towards zero
- PACF plot cuts off quickly towards zero
- ACF of a stationary series shows positive at Lag - 1

MA Model:

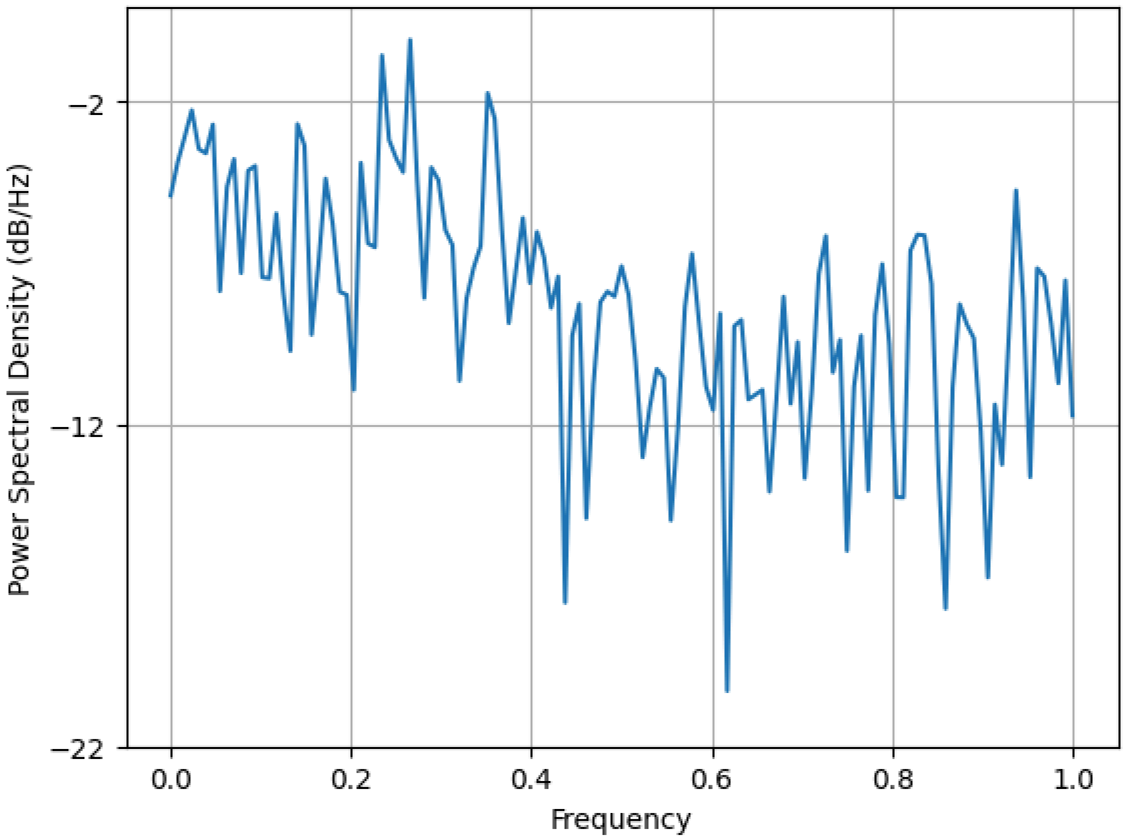
- Negatively Autocorrelated at Lag - 1
- ACF that drops sharply after a few lags
- PACF decreases gradually rather than suddenly

Looking at the above figure, ACF shows a gradual decay towards 0 from the first lag and PACF shows a quick cut off after the first lag. As such, the model I will be using is an AR(1) model.

Spectral Density

Below is a visualization of the power spectral density of the data.

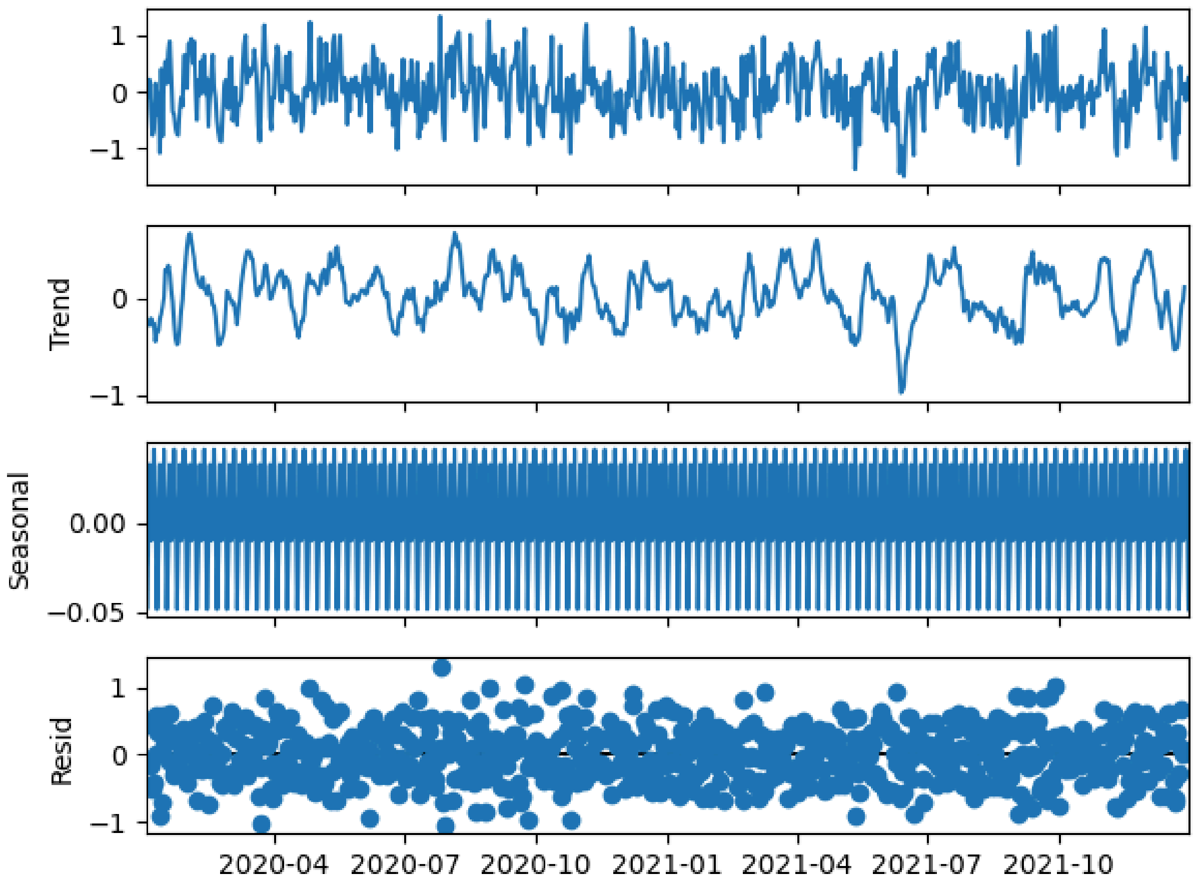
```
In [14]: plt.psd(x=med_df_diff.Revenue);
```



Decomposed Time Series

Below is the full contents of the decomposed data.

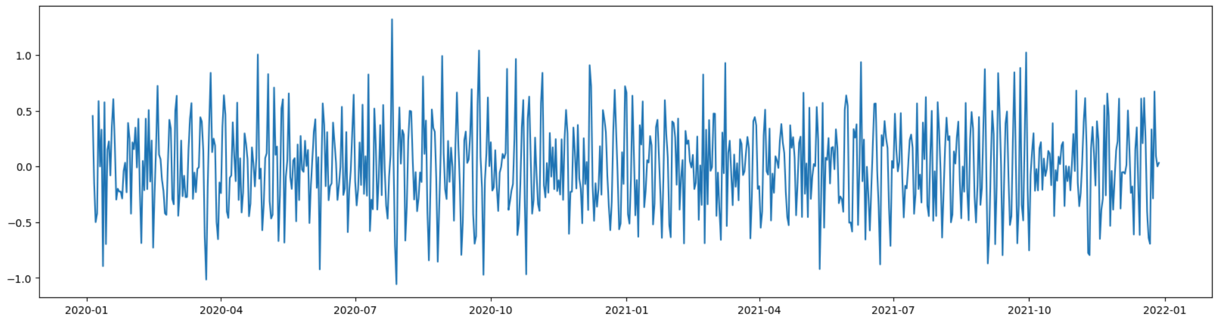
```
In [15]: med_df_decomp.plot();
```



Lack of Trends in Residuals

Below is a visualization showing the apparent lack of trends in the residuals of the decomposed time series.

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



D2: ARIMA Model of Time Series Data

Below is the code and output of the AR(1) model of the time series data.

```
In [17]: model = ARIMA(X_train, order=(1,0,0), freq='D')
         res = model.fit()
         print(res.summary())
```

```
                               SARIMAX Results
=============================================================================
==
Dep. Variable:                Revenue   No. Observations:                   5
11
Model:                 ARIMA(1, 0, 0)   Log Likelihood                 -303.7
83
Date:                Thu, 01 May 2025   AIC                             613.5
67
Time:                        14:38:26   BIC                             626.2
76
Sample:                    01-02-2020   HQIC                            618.5
49
                         - 05-26-2021
Covariance Type:                  opg
=============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
-----------------------------------------------------------------------------
--
const          0.0413      0.032      1.309      0.190      -0.021       0.1
03
ar.L1          0.3820      0.043      8.942      0.000       0.298       0.4
66
sigma2         0.1922      0.013     14.793      0.000       0.167       0.2
18
=============================================================================
=======
Ljung-Box (L1) (Q):                   0.03   Jarque-Bera (JB):
1.78
Prob(Q):                              0.85   Prob(JB):
0.41
Heteroskedasticity (H):               0.96   Skew:
0.02
Prob(H) (two-sided):                  0.78   Kurtosis:
2.71
=============================================================================
=======

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

```
c:\Users\charlesrowe\AppData\Local\Programs\Python\Python313\Lib\site-packag
es\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency informa
tion was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

I will also create another model to compare based on AR(2).

```
In [18]: model2 = ARIMA(X_train, order=(2,0,0), freq='D')
         res = model2.fit()
         print(res.summary())
```

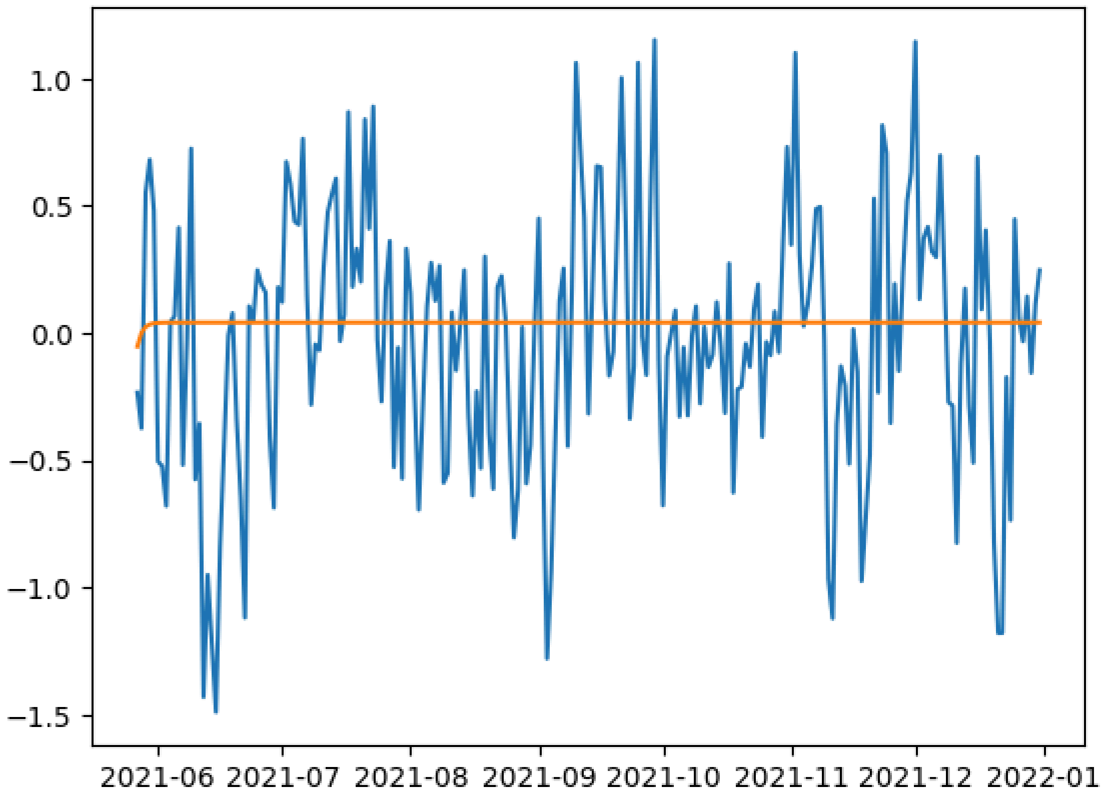
```
                               SARIMAX Results                                
==============================================================================
==
Dep. Variable:                Revenue   No. Observations:                    5
11
Model:                 ARIMA(2, 0, 0)   Log Likelihood                  -303.6
59
Date:                Thu, 01 May 2025   AIC                              615.3
18
Time:                        14:38:26   BIC                              632.2
63
Sample:                    01-02-2020   HQIC                             621.9
61
                         - 05-26-2021                                         
Covariance Type:                  opg                                         
==============================================================================
==
                 coef    std err          z      P>|z|      [0.025      0.97
5]
------------------------------------------------------------------------------
--
const          0.0412      0.032      1.272      0.203      -0.022       0.1
05
ar.L1          0.3735      0.045      8.389      0.000       0.286       0.4
61
ar.L2          0.0220      0.047      0.466      0.641      -0.071       0.1
15
sigma2         0.1921      0.013     14.663      0.000       0.166       0.2
18
==============================================================================
=======
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):              
1.95
Prob(Q):                              0.96   Prob(JB):                      
0.38
Heteroskedasticity (H):               0.96   Skew:                          
0.03
Prob(H) (two-sided):                  0.77   Kurtosis:                      
2.70
==============================================================================
=======

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

```
c:\Users\charlesrowe\AppData\Local\Programs\Python\Python313\Lib\site-packag
es\statsmodels\tsa\base\tsa_model.py:473: ValueWarning: No frequency informa
tion was provided, so inferred frequency D will be used.
  self._init_dates(dates, freq)
```

D3: Forecast

I will be attempting to forecast revenue from the start of the testing data set.

```
In [19]: forecast_med_rev = res.get_prediction(start=511, end=729, dynamic = True)
         plt.plot(X_test)
         plt.plot(forecast_med_rev.predicted_mean);
```



```
In [20]: print(forecast_med_rev.predicted_mean)
```

```
2021-05-27   -0.053011
2021-05-28    0.000960
2021-05-29    0.024104
2021-05-30    0.033938
2021-05-31    0.038121
                ...
2021-12-27    0.041218
2021-12-28    0.041218
2021-12-29    0.041218
2021-12-30    0.041218
2021-12-31    0.041218
Freq: D, Name: predicted_mean, Length: 219, dtype: float64
```

Looking at the above, it seems like the model was not very effective at predicting future values and plateaus at an additive value of 0.04129 (the same value as the constant in the ARIMA model). This means that it is forecasting for revenue to gradually increase at the same rate per day, which is not what we see in our actual values.

Even though it does not seem to be effective, I will still continue to plot predicted vs observed data including the confidence intervals.

```
In [21]: med_df_forecast = pd.DataFrame(forecast_med_rev.predicted_mean)
         med_df_forecast.rename(columns={'predicted_mean' : 'Revenue'}, inplace=True)
         forecast_df = pd.concat([X_train.copy(), med_df_forecast.copy()])
         forecast_df = forecast_df.cumsum()
         forecast_df
```

Out[21]:

	Revenue
2020-01-02	-0.292356
2020-01-03	-0.327772
2020-01-04	-0.339987
2020-01-05	-0.124888
2020-01-06	-0.491590
...	...
2021-12-27	30.148396
2021-12-28	30.189614
2021-12-29	30.230832
2021-12-30	30.272049
2021-12-31	30.313267

730 rows × 1 columns

```
In [22]: conf_int = forecast_med_rev.conf_int()
         conf_int
```

Out[22]:

	lower Revenue	upper Revenue
2021-05-27	-0.912056	0.806033
2021-05-28	-0.916055	0.917975
2021-05-29	-0.903354	0.951562
2021-05-30	-0.895389	0.963265
2021-05-31	-0.891544	0.967787
...	...	...
2021-12-27	-0.888522	0.970958
2021-12-28	-0.888522	0.970958
2021-12-29	-0.888522	0.970958
2021-12-30	-0.888522	0.970958
2021-12-31	-0.888522	0.970958

219 rows × 2 columns

In [23]:
```
untrans_row = pd.DataFrame({'lower Revenue':[21.6392907], 'upper Revenue':[2
untrans_row['date'] = pd.to_datetime(untrans_row['date'])
untrans_row.set_index('date',inplace=True)
untrans_row
```

Out[23]:

	lower Revenue	upper Revenue
date		
2021-05-26	21.639291	21.639291

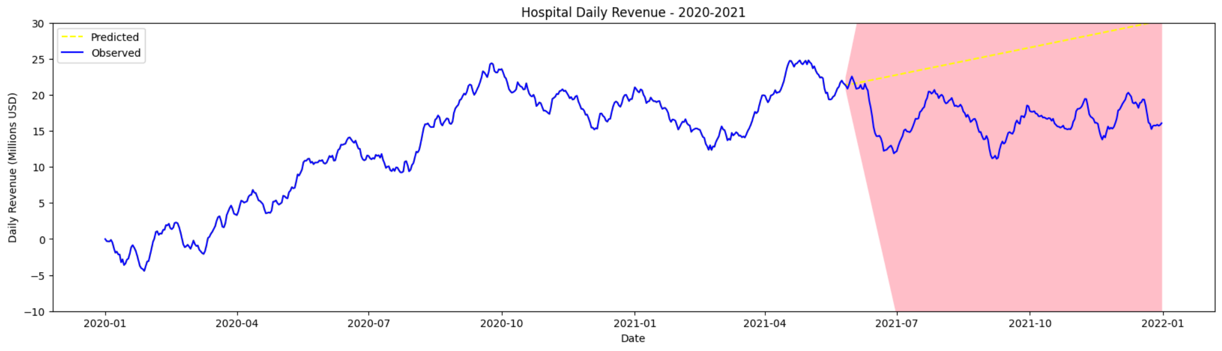
In [24]:
```
conf_int = pd.concat([untrans_row, conf_int])
conf_int = conf_int.cumsum()
conf_int = conf_int.loc['2021-05-27' : '2021-12-31']
conf_int
```

Out[24]:

	lower Revenue	upper Revenue
2021-05-27	20.727235	22.445324
2021-05-28	19.811180	23.363300
2021-05-29	18.907825	24.314862
2021-05-30	18.012436	25.278127
2021-05-31	17.120892	26.245914
...	...	...
2021-12-27	-169.471049	230.144688
2021-12-28	-170.359571	231.115646
2021-12-29	-171.248093	232.086603
2021-12-30	-172.136615	233.057561
2021-12-31	-173.025137	234.028518

219 rows × 2 columns

In [25]:
```
plt.figure(figsize=[20,5])
plt.title("Hospital Daily Revenue - 2020-2021")
plt.xlabel("Date")
plt.ylabel("Daily Revenue (Millions USD)")
plt.plot(forecast_df, color='yellow', linestyle='dashed')
plt.plot(med_df, color='blue')
plt.fill_between(conf_int.index, conf_int['lower Revenue'], conf_int['upper
plt.ylim(-10,30)
plt.legend(['Predicted','Observed'])
plt.show();
```



The above shows the predicted values vs the observed values. It is obvious from looking that the model did not accurately predict future values. It also seems that the confidence intervals are astronomically large and are pretty unreliable as well. To evaluate further, I would like to look at the model's root mean squared error and the diagnostics.

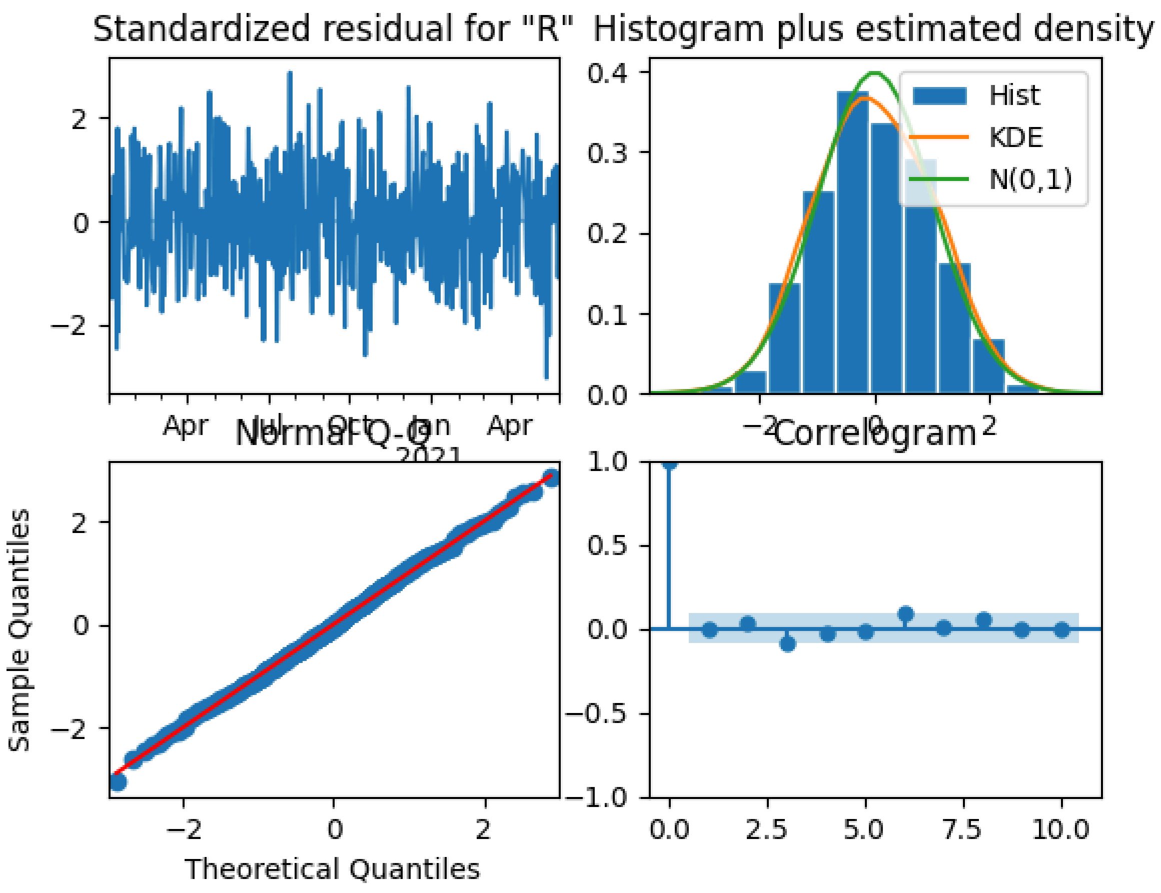
In [26]:
```
rmse = mean_squared_error(med_df.loc['2021-05-27' : '2021-12-31'], forecast_
rmse = np.sqrt(rmse)
```

```
print(f"RMSE: {round(rmse, 5)}")
```

```
RMSE: 9.80814
```

```
In [27]: plt.figure(figsize = [30,30])
         res.plot_diagnostics();
```

```
<Figure size 3000x3000 with 0 Axes>
```



D4: Analysis Output and Calculation

All outputs and calculations are above.

D5: ARIMA Model Code

All code used for the ARIMA model are above.

E1: Results of Analysis

Looking over the above sections, I would like to discuss the selection of the ARIMA model, the prediction interval (the test and split data), the justification of the forecast length, and the evaluation of the model, including the error metric.

First, the selection of the ARIMA model was founded on the observed ACF and PACF of the data. Since the ACF trails off beginning at 2 and the PACF drops suddenly at 1, I believed that it was best to stick with an AR(1) model. Arguements could be made that, because the ACF dropped at 2, I should have used an AR(2) model. However, I will speak as to me selection of the AR(1) model more below.

Second, the prediction interval was 30% of the data starting at May 27th 2021. When testing and splitting data, a normal interval is 20-30%. I felt that I should go with the maximum of this ratio so that the model could be fit to the maximum amount of data.

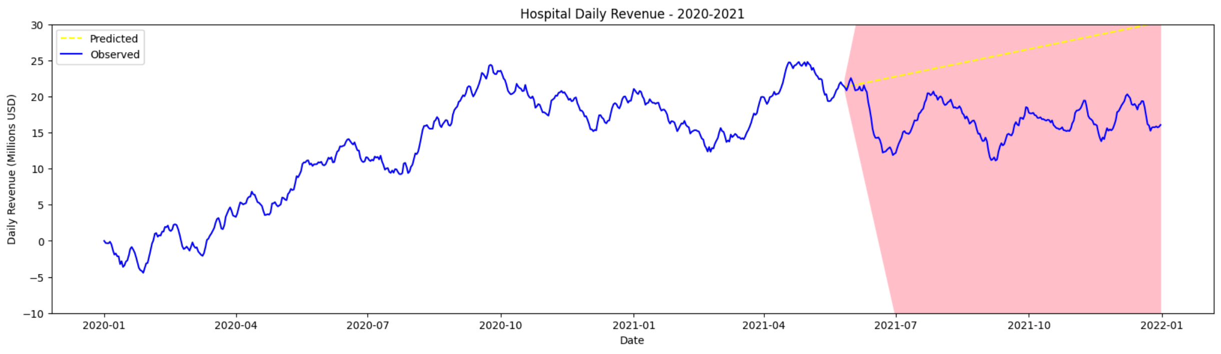
Third, the forecast length was set to the same amount of time as the testing data (May 27th, 2021 to December 31, 2021). This allowed me to compare the predicted data to the known data that was used to train the model.

Finally, I evaluated the model using the AIC (Akaike Information Criterion) and the RMSE. The AIC of the model is 613.567. Compared to the AR(2) model with an AIC of 615.318, the AR(1) model is a slightly better fit. The RMSE of the AR(1) model is 9.80814, meaning that there is an average error in the caluclation of revenue of 9.81 (rounded up). While this could be perceived as not a large error, a change of 10 million dollars is astronomical when it comes to revenue.

E2: Annotated Visualizatiom

Below is the annotated visualization of the forecast compared to the observed values.

In [28]:
```
plt.figure(figsize=[20,5])
plt.title("Hospital Daily Revenue - 2020-2021")
plt.xlabel("Date")
plt.ylabel("Daily Revenue (Millions USD)")
plt.plot(forecast_df, color='yellow', linestyle='dashed')
plt.plot(med_df, color='blue')
plt.fill_between(conf_int.index, conf_int['lower Revenue'], conf_int['upper
plt.ylim(-10,30)
plt.legend(['Predicted','Observed'])
plt.show();
```



E3: Recommendation

Sadly, due to the model not accurately predicting future revenue, the only recommendation that I feel comfortable giving is the need for further study. Two years worth of data seems to be insufficient in predicting future revenue, so more data will need to be collected before another analysis is given. Also, since the data includes the first two years of operation, the first few months of data may be hindering training set as these values could be seen as outliers. This can all be confirmed with further data acquisition and analysis.

F: Reporting

A PDF will be provided with the submission.

G: Sources for Third-Party Code

William Townsend provided code used in sections C1 and D3.

H: Sources

IBM provided the general practice for choosing an ARIMA model.