# D213 - Advanced Data Analytics

### Task 1: Time Series Modeling

## Part I: Research Question

### A1: Proposal of Question

One research question that is relevant to the organizational situation that we hope to address in the following analysis is:

* "With the first 2 years of daily revenue data, can a create a relatively reliable model for forecasting future revenues?"

### A2: Objectives and Goals

The primary objective of this analysis is to study the daily revenue data and discover any underlying trends, seasonality, or other patterns are present. Following on from that, an additional goal is to create an ARIMA model that is reasonably accurate in the prediction of future daily revenues.

## Part II: Method Justification

### B1: Summary of Assumptions

Time series modeling relies on a few assumptions:

* **Stationarity** - The time series summary statistics such as ***mean*** and ***standard deviation*** as well as ***autocorrelation*** remain constant over time, there is ***no observable seasonality*** to the data, and if these properties exist, the data has been transformed so as to extract those elements prior to analysis
* **Homoskedasticity** - The time series distribution and variance of errors remains relatively constant over time
* **Univariate Data** - The time series consists of only one variable

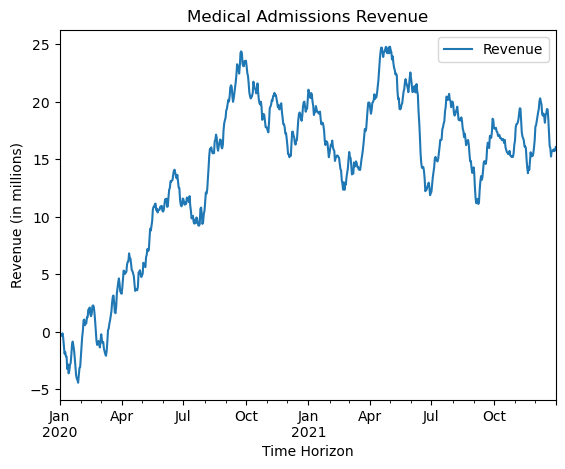
## Part III: Data Preparation

### C1: Line Graph Visualization

# Import main packages and read in data  
import pandas as pd  
import numpy as np  
from statsmodels.tsa.stattools import adfuller   
  
df = pd.read\_csv('./data/medical\_time\_series.csv')  
df.head()

Day Revenue  
0 1 0.000000  
1 2 -0.292356  
2 3 -0.327772  
3 4 -0.339987  
4 5 -0.124888

# Set datetime index and plot to line graph  
df = df.set\_index(pd.date\_range(start='2020-1-1', periods=df.shape[0], freq='D'))  
df.drop('Day', axis=1, inplace=True)  
df.plot(title='Medical Admissions Revenue', ylabel='Revenue (in millions)', xlabel='Time Horizon')



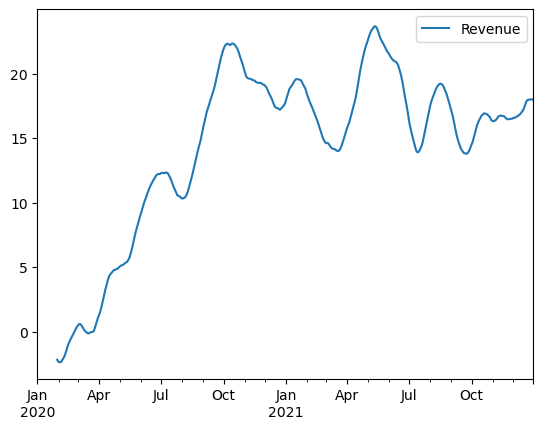
### C2: Time Step Formatting

The time step formatting of the above time series is in datetime format. The frequency of the time series is **daily** and the total length is **731 days**.

### C3: Stationarity

We will begin by plotting the **rolling mean** of the raw dataset as an initial glimpse at stationarity.

# Plot the rolling mean to check for stationarity  
df.rolling(window=30).mean().plot()



It seems visually evident from the plot above that the data are not stationary. As further confirmation, we will next utilize the Augmented Dickey-Fuller test to evaluate the stationarity of the data.

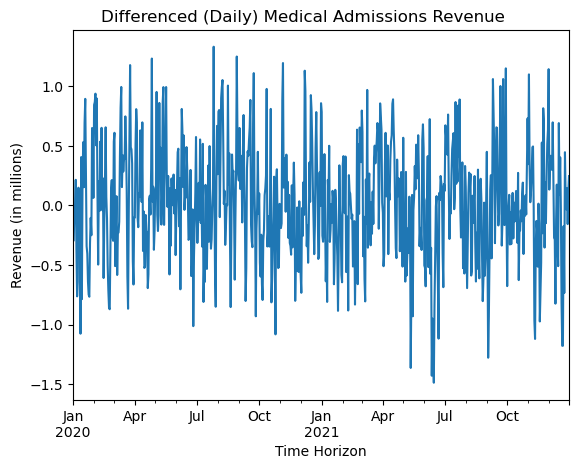
# Evaluate the stationarity of time series data using Augmented Dickey-Fuller and print results  
def adf\_test(ts):  
 df = adfuller(ts, autolag='AIC')  
 results = pd.DataFrame(df[:4], columns=['Results '], dtype=object)  
 results.index=['Test Statistic', 'p-value', 'Num Lags', 'Num Observations']  
 print('Results of Augmented Dickey-Fuller Test:\n\n', results)  
 if df[1] <= 0.05:  
 print('\nStrong evidence against the null hypothesis')  
 print('Reject the null hypothesis')  
 print('Data may have no unit root and is stationary')  
 else:  
 print('\nWeak evidence against the null hypothesis')  
 print('Fail to reject the null hypothesis')  
 print('Data may have a unit root and is non-stationary')  
  
# Run the ADF test on the raw time series data  
adf\_test(df['Revenue'])

Results of Augmented Dickey-Fuller Test:  
  
 Results   
Test Statistic -2.218319  
p-value 0.199664  
Num Lags 1.0  
Num Observations 729.0  
  
Weak evidence against the null hypothesis  
Fail to reject the null hypothesis  
Data may have a unit root and is non-stationary

With a non-statistically significant p-value at ~0.20, this is further evidence that the raw data are not stationary. We will now attempt to transform our time series by taking the first differences of the Revenue series so as to achieve stationarity.

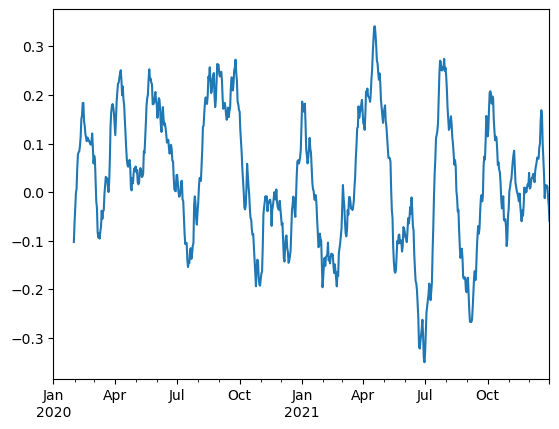
# Difference the time series and plot to line graph  
df['Revenue\_diff'] = df['Revenue'].diff()  
df['Revenue\_diff'].plot(title='Differenced (Daily) Medical Admissions Revenue ', ylabel='Revenue (in millions)', xlabel='Time Horizon')

<AxesSubplot:title={'center':'Differenced (Daily) Medical Admissions Revenue '}, xlabel='Time Horizon', ylabel='Revenue (in millions)'>



Visually, this appears to be more stationary. We'll now plot the rolling mean again to see if it remains relatively constant as opposed to the raw data.

# Plot the differenced rolling mean to check for stationarity  
df['Revenue\_diff'].rolling(window=30).mean().plot()



# Test stationarity of the differenced time series using ADF  
adf\_test(df['Revenue\_diff'].dropna())

Results of Augmented Dickey-Fuller Test:  
  
 Results   
Test Statistic -17.374772  
p-value 0.0  
Num Lags 0.0  
Num Observations 729.0  
  
Strong evidence against the null hypothesis  
Reject the null hypothesis  
Data may have no unit root and is stationary

It now appears we have achieved stationarity in our dataset. Next, we'll look at seasonality and utilize the auto\_arima tool to help us with that.

### C4: Steps to Prepare the Data

The following steps were used to prepare the data for training and testing:

* **Step 1:** Import necessary Python libraries (pandas, numpy, and statsmodels)
* **Step 2:** Load the medical\_time\_series.csv file using pandas
* **Step 3:** Set a datetime index using pandas' date\_range function with a daily frequency, starting from '2020-1-1' for a total of 731 days
* **Step 4:** Transform the time series to achieve stationarity by taking the first difference of the 'Revenue' series as Revenue\_diff
* **Step 5:** Split the data into train and test subsets by allocating the first 80% of days to the train set and the remaining 20% of days to the test set (*see section D1*)

### C5: Prepared Dataset

# Please see attached `medical\_time\_series\_clean.csv`  
df['Revenue\_diff'].dropna().to\_csv('./data/medical\_time\_series\_clean.csv')

## Part IV: Model Identification and Analysis

### D1: Report Findings and Visualizations

Next, we will use the SARIMAX model to programmatically determine the optimal ARIMA parameters as well as identify any seasonal component of the cleaned dataset.

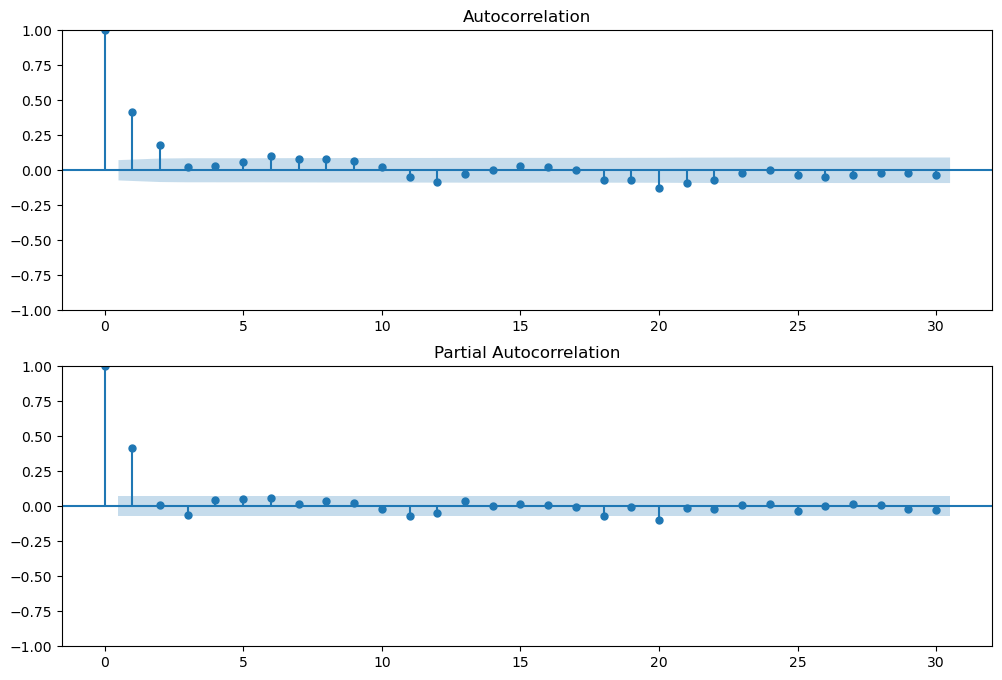
# Import packages for ARIMA modeling using auto\_arima  
from pmdarima import auto\_arima  
  
arimafit = auto\_arima(df['Revenue\_diff'].dropna(), trace=True)  
arimafit.summary()

Performing stepwise search to minimize aic  
 ARIMA(2,0,2)(0,0,0)[0] intercept : AIC=883.277, Time=0.46 sec  
 ARIMA(0,0,0)(0,0,0)[0] intercept : AIC=1015.972, Time=0.08 sec  
 ARIMA(1,0,0)(0,0,0)[0] intercept : AIC=881.359, Time=0.07 sec  
 ARIMA(0,0,1)(0,0,0)[0] intercept : AIC=906.199, Time=0.07 sec  
 ARIMA(0,0,0)(0,0,0)[0] : AIC=1015.481, Time=0.05 sec  
 ARIMA(2,0,0)(0,0,0)[0] intercept : AIC=883.300, Time=0.09 sec  
 ARIMA(1,0,1)(0,0,0)[0] intercept : AIC=883.314, Time=0.12 sec  
 ARIMA(2,0,1)(0,0,0)[0] intercept : AIC=883.348, Time=0.29 sec  
 ARIMA(1,0,0)(0,0,0)[0] : AIC=879.982, Time=0.04 sec  
 ARIMA(2,0,0)(0,0,0)[0] : AIC=881.911, Time=0.06 sec  
 ARIMA(1,0,1)(0,0,0)[0] : AIC=881.927, Time=0.06 sec  
 ARIMA(0,0,1)(0,0,0)[0] : AIC=905.166, Time=0.03 sec  
 ARIMA(2,0,1)(0,0,0)[0] : AIC=881.947, Time=0.18 sec  
  
Best model: ARIMA(1,0,0)(0,0,0)[0]   
Total fit time: 1.608 seconds

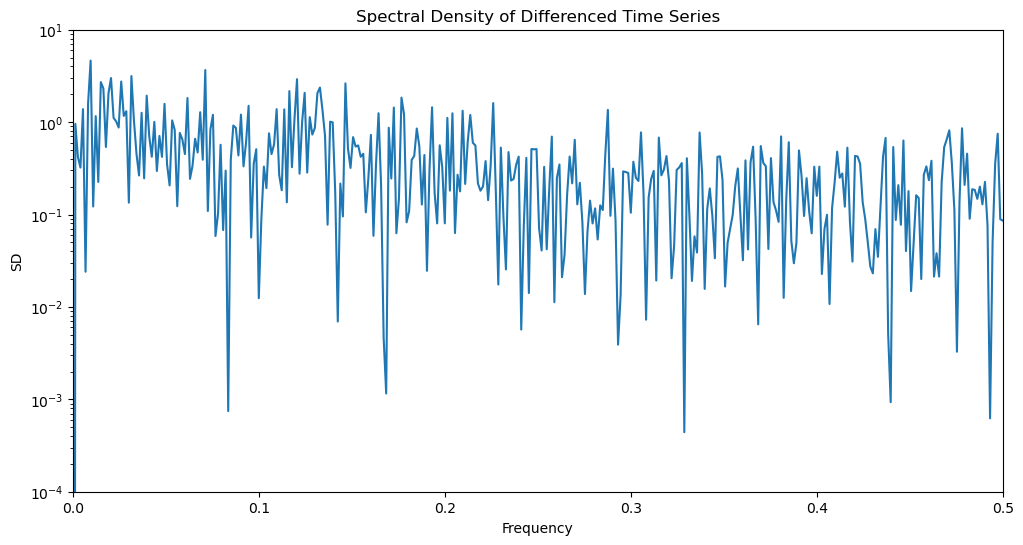
SARIMAX Results   
==============================================================================  
Dep. Variable: y No. Observations: 730  
Model: SARIMAX(1, 0, 0) Log Likelihood -437.991  
Date: Mon, 29 May 2023 AIC 879.982  
Time: 22:39:58 BIC 889.168  
Sample: 01-02-2020 HQIC 883.526  
 - 12-31-2021   
Covariance Type: opg   
==============================================================================  
 coef std err z P>|z| [0.025 0.975]  
------------------------------------------------------------------------------  
ar.L1 0.4142 0.034 12.258 0.000 0.348 0.480  
sigma2 0.1943 0.011 17.842 0.000 0.173 0.216  
===================================================================================  
Ljung-Box (L1) (Q): 0.02 Jarque-Bera (JB): 1.92  
Prob(Q): 0.90 Prob(JB): 0.38  
Heteroskedasticity (H): 1.00 Skew: -0.02  
Prob(H) (two-sided): 0.97 Kurtosis: 2.75  
===================================================================================  
  
Warnings:  
[1] Covariance matrix calculated using the outer product of gradients (complex-step).

It appears that the **optimal (p, d, q) parameters for our dataset are (1, 0, 0)** which represents the best Akaike Information Criteria. Also, the above results **suggest no seasonality** in our dataset. Next, we will look at the autocorrelation function and partial autocorrelation function to further examine trends/seasonality.

# Plot stationarity of the raw time series  
from statsmodels.graphics.tsaplots import plot\_acf, plot\_pacf  
import matplotlib.pyplot as plt  
  
# Plot stationarity of the diffed time series  
diff\_fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(12, 8))  
plot\_acf(df['Revenue\_diff'].dropna(), ax=ax1, lags=30)  
plot\_pacf(df['Revenue\_diff'].dropna(), ax=ax2, lags=30, method='ywm')  
plt.show()

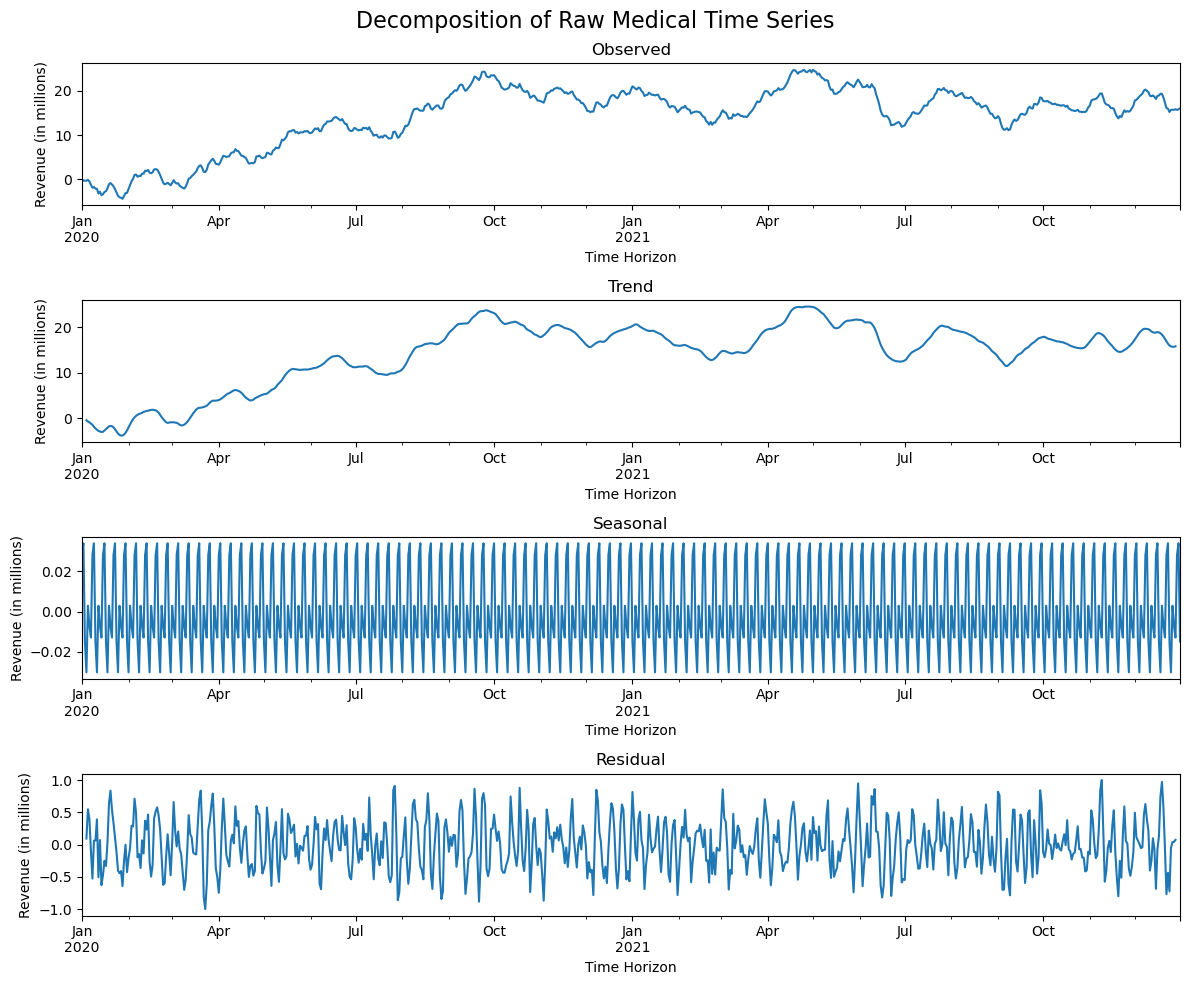


# Evaluate the spectral density of the time series  
from scipy import signal  
  
# Plot spectral density  
fig, ax = plt.subplots(1, 1, figsize=(12, 6))  
f, Pxx\_den = signal.periodogram(df['Revenue\_diff'].dropna())  
ax.semilogy(f, Pxx\_den)  
ax.set\_xlabel('Frequency')  
ax.set\_ylabel('SD')  
plt.ylim([1e-4, 1e1])  
ax.set\_xlim([0, 0.5])  
ax.set\_title('Spectral Density of Differenced Time Series')  
plt.show()

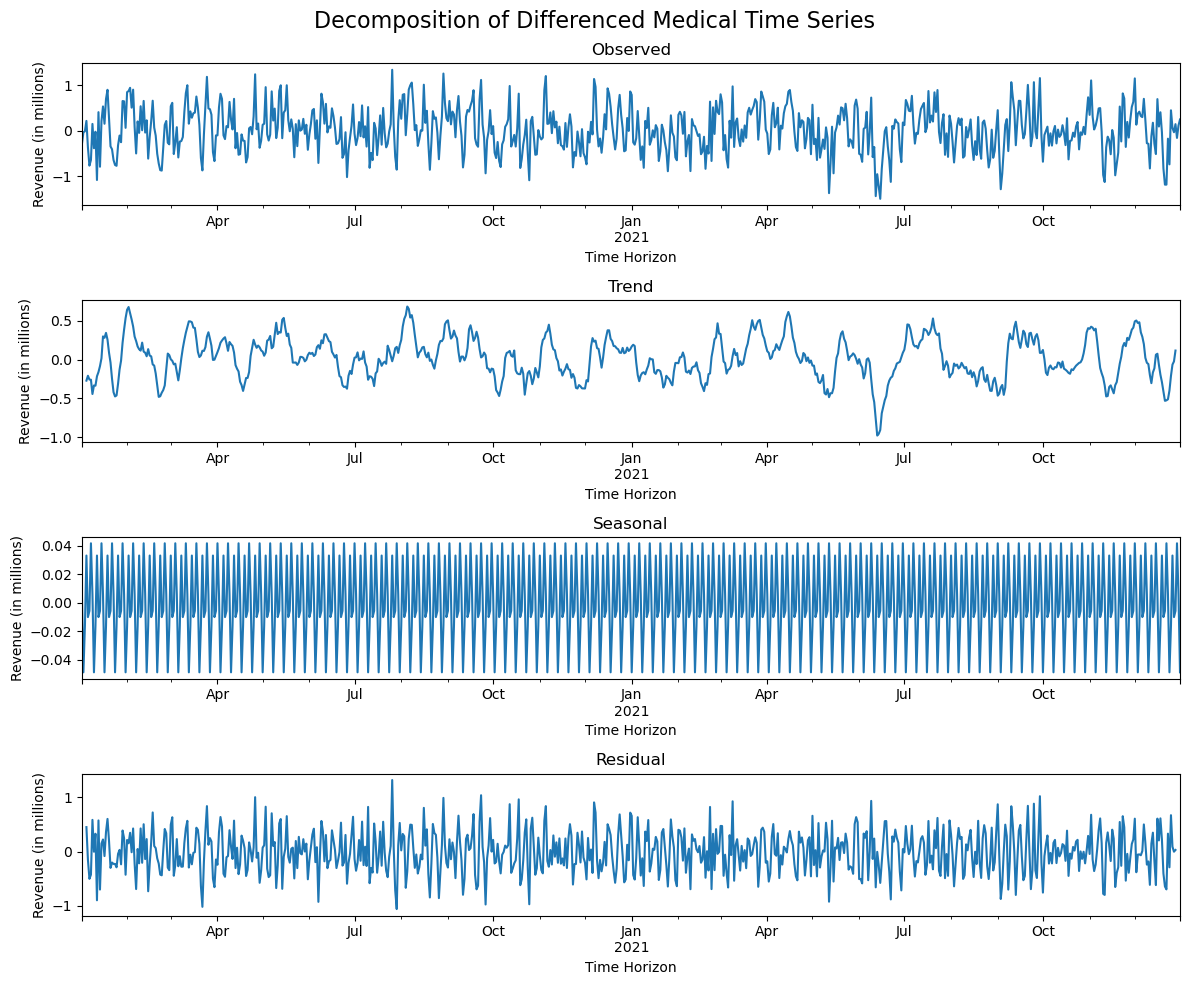


# Decompose the raw time series into trend, seasonal, and residual components and plot  
from statsmodels.tsa.seasonal import seasonal\_decompose  
  
# Define function to generate decomposition plots  
def plot\_decomposition(decomp, title, ylabel=None, xlabel=None, figsize=(12, 10)):  
 decomp = seasonal\_decompose(decomp)  
 fig, (ax1, ax2, ax3, ax4) = plt.subplots(4, 1, figsize=(12, 10))  
 fig.suptitle(title, fontsize=16)  
 decomp.observed.plot(ax=ax1, title='Observed', ylabel=ylabel, xlabel=xlabel)  
 decomp.trend.plot(ax=ax2, title='Trend', ylabel=ylabel, xlabel=xlabel)  
 decomp.seasonal.plot(ax=ax3, title='Seasonal', ylabel=ylabel, xlabel=xlabel)  
 decomp.resid.plot(ax=ax4, title='Residual', ylabel=ylabel, xlabel=xlabel)  
 plt.tight\_layout()  
 plt.show()

# Plot decomposition of raw time series  
plot\_decomposition(df['Revenue'], ylabel='Revenue (in millions)', xlabel='Time Horizon', title='Decomposition of Raw Medical Time Series')



# Decomposition of the differenced time series  
plot\_decomposition(df['Revenue\_diff'].dropna(), ylabel='Revenue (in millions)', xlabel='Time Horizon', title='Decomposition of Differenced Medical Time Series')



Once again, it appears clear that our cleaned and transformed dataset does not feature seasonality or other obvious trends.

Now that we have prepared the data for training our ARIMA model, one final step prior to training is to split our dataset for testing. We will allocate 80% of the data for training and 20% for testing.

# Split the dataset into train and test sets at 80/20  
train = df['Revenue\_diff'].dropna().iloc[:int(df.\_\_len\_\_() \* 0.8)]  
test = df['Revenue\_diff'].dropna().iloc[int(df.\_\_len\_\_() \*0.8):]

### D2: ARIMA Model

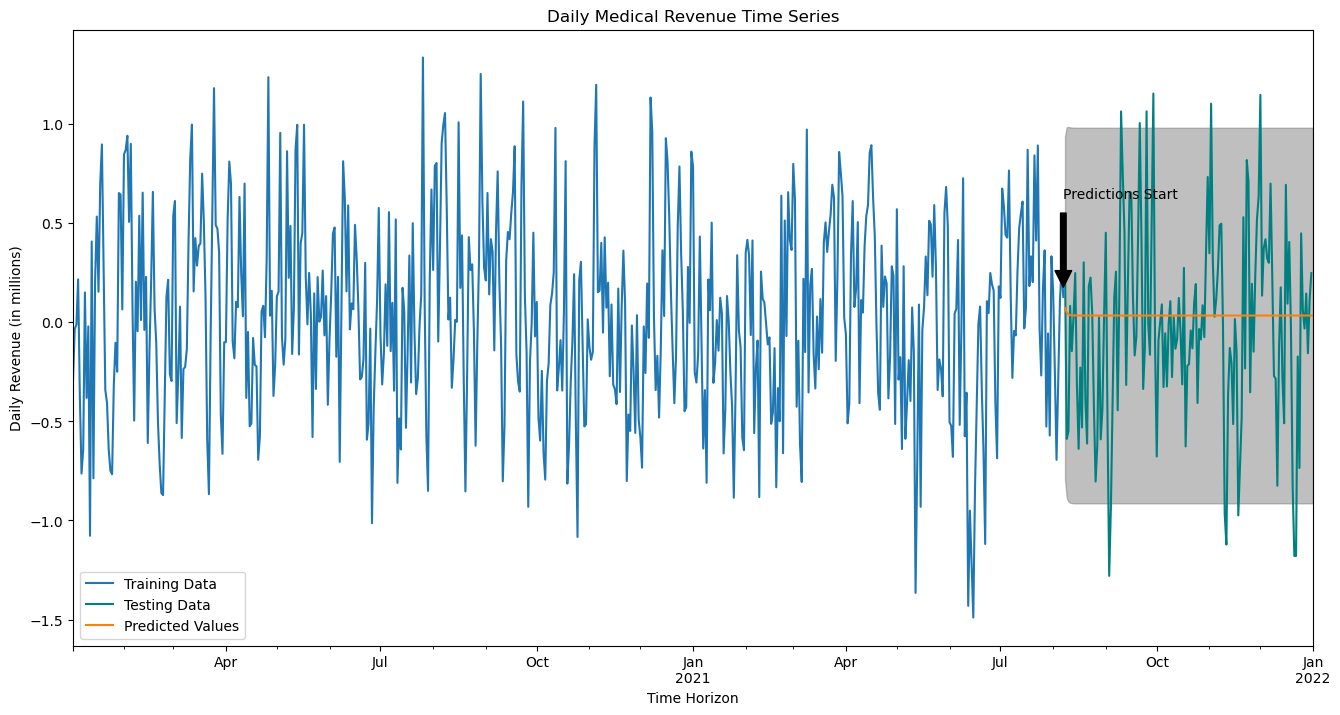
As identified above in section D1, we will fit our training data to an **ARIMA(1, 0, 0)** model, which we determined was the optimal model for our dataset.

# Train the ARIMA model on the training set and get summary  
from statsmodels.tsa.arima.model import ARIMA  
mod = ARIMA(train, order=(1, 0, 0))  
res = mod.fit()  
res.summary()

SARIMAX Results   
==============================================================================  
Dep. Variable: Revenue\_diff No. Observations: 584  
Model: ARIMA(1, 0, 0) Log Likelihood -350.349  
Date: Mon, 29 May 2023 AIC 706.698  
Time: 23:16:03 BIC 719.808  
Sample: 01-02-2020 HQIC 711.808  
 - 08-07-2021   
Covariance Type: opg   
==============================================================================  
 coef std err z P>|z| [0.025 0.975]  
------------------------------------------------------------------------------  
const 0.0328 0.031 1.063 0.288 -0.028 0.093  
ar.L1 0.4079 0.038 10.748 0.000 0.333 0.482  
sigma2 0.1943 0.012 15.948 0.000 0.170 0.218  
===================================================================================  
Ljung-Box (L1) (Q): 0.10 Jarque-Bera (JB): 1.80  
Prob(Q): 0.75 Prob(JB): 0.41  
Heteroskedasticity (H): 1.04 Skew: -0.05  
Prob(H) (two-sided): 0.78 Kurtosis: 2.75  
===================================================================================  
  
Warnings:  
[1] Covariance matrix calculated using the outer product of gradients (complex-step).  
"""

### D3: Forecasting Using ARIMA Model

# Generate predictions and plot to line graph with test and training data  
from statsmodels.graphics.tsaplots import plot\_predict  
  
def arima\_forecast\_plot(model, train, test, title, ylabel, xlabel, start=len(train), end=len(train)+len(test), figsize=(16, 8)):  
 fig, ax = plt.subplots(figsize=figsize)  
 ax.set(title=title, xlabel=xlabel, ylabel=ylabel)  
 train.plot(ax=ax)  
 test.plot(ax=ax, color='teal')  
 plot\_predict(model, start=start, end=end, ax=ax, alpha=0.05)  
 ax.annotate('Predictions Start', xy=(train.index[-1], train.iloc[-1]), xytext=(train.index[-1], train.iloc[-1] + 0.5), arrowprops=dict(facecolor='black', shrink=0.1))  
 ax.legend(['Training Data', 'Testing Data', 'Predicted Values'])  
 plt.show()  
   
arima\_forecast\_plot(res, train, test, title='Daily Medical Revenue Time Series', ylabel='Daily Revenue (in millions)', xlabel='Time Horizon')



### D4: Output and Calculations

In addition to the visualizations above, here are some of the forecasted values for our time series data:

# Show predicted values  
res.predict(start=len(train), end=len(train)+len(test))

2021-08-08 0.071071  
2021-08-09 0.048405  
2021-08-10 0.039159  
2021-08-11 0.035388  
2021-08-12 0.033850  
 ...   
2021-12-28 0.032791  
2021-12-29 0.032791  
2021-12-30 0.032791  
2021-12-31 0.032791  
2022-01-01 0.032791  
Freq: D, Name: predicted\_mean, Length: 147, dtype: float64

# Show mean of predicted values  
pred\_mean = res.predict(start=len(train), end=len(train)+len(test)).mean()  
print(f'The mean of the predicted values is {pred\_mean:.4f} or ${pred\_mean \* 1000000:,.0f}')

The mean of the predicted values is 0.0332 or $33,231

# Show the RMSE of the model  
from sklearn.metrics import mean\_squared\_error  
print('RMSE:', np.sqrt(mean\_squared\_error(test, res.predict(start=len(train), end=len(train)+len(test)-1))))

RMSE: 0.4887235607100212

### D5: Code

*Please see above code and attached notebook file.*

## Part V: Data Summary and Implications

### E1: Results

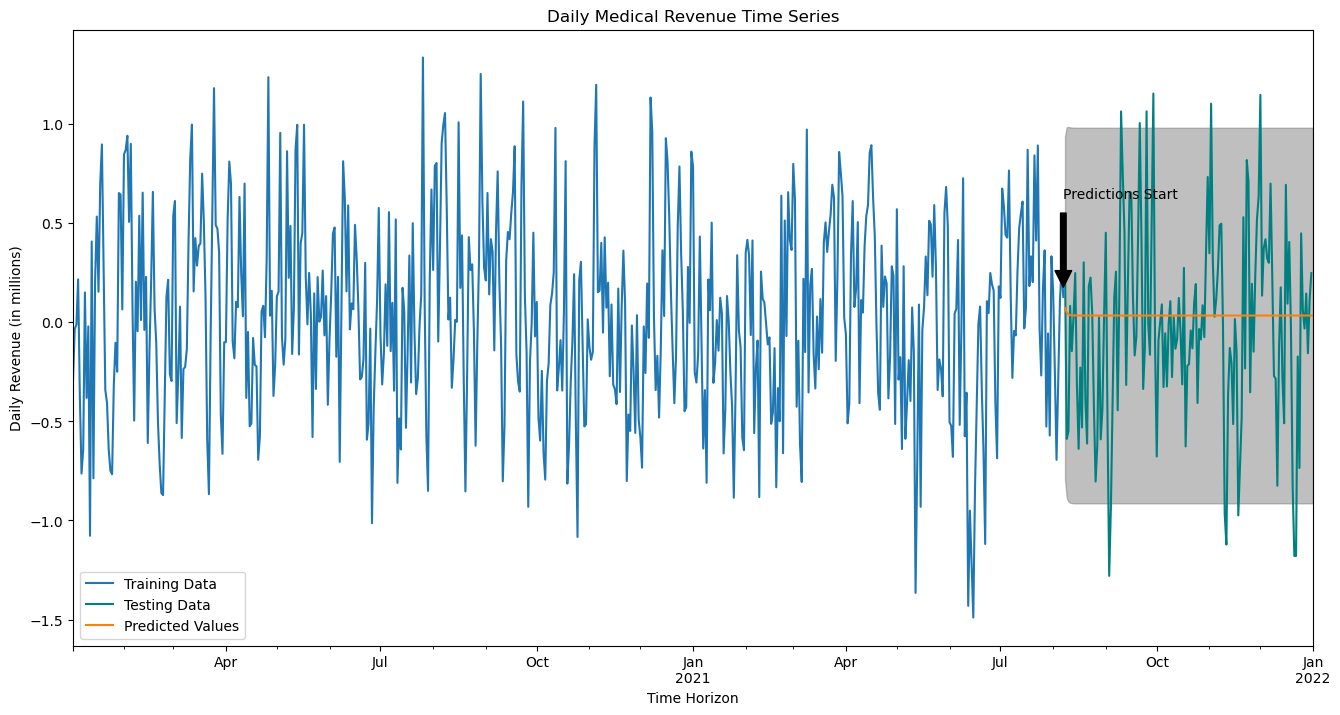
In summary, I will discuss the findings of the above analysis with regard to...

* The selection of the ARIMA model:
  + The AutoRegressive Integrated Moving Average (ARIMA) model was selected for this analysis based on the nature of the data itself. ARIMA is particularly well-suited for time series data, especially time series data that exhibit trends and/or seasonality. After cleaning and transforming the Medical Revenue dataset, the data were determined to be stationary. Due to that fact as well as the process of selection by minimization of the Akaike Information Criteria (AIC), which incorporates a penalty for each additional term, ARIMA (1, 0, 0) was selected as the best and simplest fit for the data. (Bruce, Bruce, & Gedeck, 2020)
* The prediction interval of the forecast
  + In the analysis, the prediction interval spanned the length of the test dataset, which constituted 20% of the total data or 146 days. This forecast period was chosen to ensure a sufficient sample size for evaluating the model's predictive accuracy and generalizability to unseen data.
* Justification of the forecast length
  + Our forecast length corresponds to the size of the test dataset, facilitating a thorough evaluation of our ARIMA model. The length provides us enough data for validation while ensuring predictions remain within a reliable time frame, where the patterns we have identified are more likely to hold.
* The model evaluation procedure and error metric
  + The ARIMA model was evaluated using the Root Mean Squared Error (RMSE), which measures the average magnitude of the prediction error. (Larose & Larose, 2019) An RMSE of 0.49 shows that the model has done a reasonably good job at fitting the data, with predictions typically off by about 0.49 on average.

### E2: Annotated Visualization

Below I will show again the above annotated forecast plot.

# Show annotated plot of the predicted values  
arima\_forecast\_plot(res, train, test, title='Daily Medical Revenue Time Series', ylabel='Daily Revenue (in millions)', xlabel='Time Horizon')



### E3: Recommendations

The analysis performed with the ARIMA model provides a compelling answer to the research question posed: using the first two years of daily revenue data, we indeed created a reliable model for forecasting future revenues. Therefore, one proposed course of action based on these results is to integrate this model into the organization's operational procedures for revenue prediction and planning.

However, it is important to ensure the model continues to provide accurate predictions as the business environment and conditions change over time. Hence, it is recommended to implement a procedure for regular model updates and validations. As the organization continues to generate revenue data, this data should be used to retrain the ARIMA model periodically.

Additionally, continually re-evaluating the model's forecasting performance will be critical. The use of metrics like RMSE in evaluating the model’s performance has been effective in assessing the quality of forecasts. Therefore, maintaining a regular evaluation schedule using these metrics will be crucial in ensuring the model's ongoing reliability.

While the ARIMA model provided valuable insights, considering other modeling techniques that build on or are in tandem with the current model can provide a more comprehensive and responsive view of future revenues. It will be important to consider these models including models that utilize machine learning and neural networks which are becoming more and more useful.

In conclusion, ARIMA has proved a useful model in modeling the Medical Revenue dataset and forecasting future cash flows. The recommendations provided here are just a small sampling of the potential options available for time series analysis, so truly there are many possible solutions.

# H: References

Bruce, P., Bruce, A., & Gedeck, P. (2020). *Practical Statistics for Data Scientists.* Sebastopol: O'Reilly Media, Inc.

Larose, C. D., & Larose, D. T. (2019). *Data Science Using Python and R.* Hoboken: John Wiley & Sons, Inc.