D603 Task 3
By Eric Williams

B1:RESEARCH QUESTION

When considering a time analysis involving revenue over time, our key question is this: How does the revenue change over time and what patterns can we identify to predict future revenue? Can we predict the change in revenue over time and if so, with how much confidence and accuracy?

B2:OBJECTIVES AND GOALS

The goal of this analysis is to provide actionable insights on how revenue changes over time in order to assist with decreasing readmission. In this analysis, I will not only analyze past revenue changes, but forecast future revenue changes.

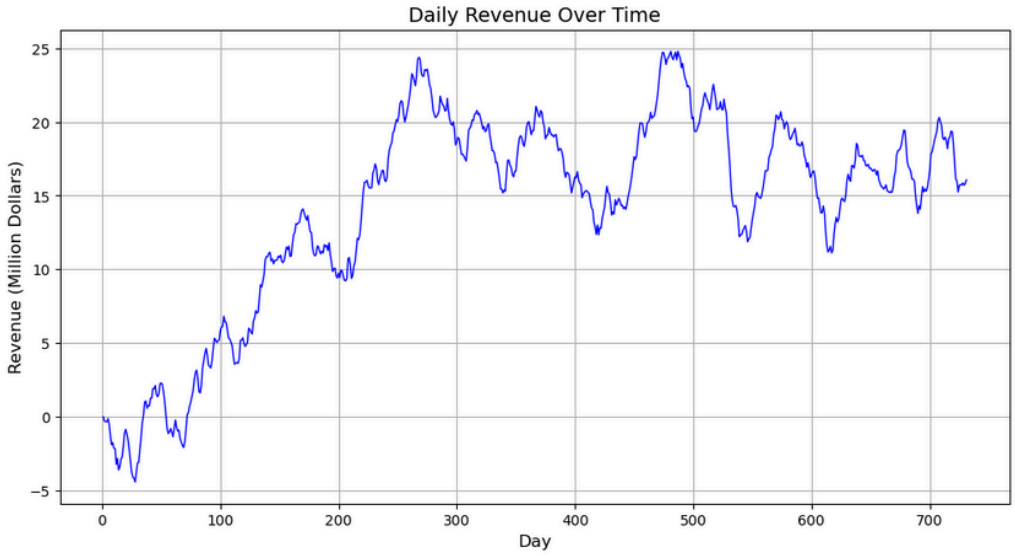
C:SUMMARY OF ASSUMPTIONS

Two assumptions we need to make is that the data is stationary and the data is autocorrelated.

When addressing stationarity, we are assuming statistical properties like mean, variance, and covariance remain constant over time. In the real world, hospital admissions may depend on holidays, weather, season, and a variety of variables that are not controlled for in the data. For example, if this hospital was located in Florida, we would expect anomalies during hurricane season. If this data was collected in a region with cold weather in the winter, we can assume readmissions could be very different in winter than in the summer. Because the data is sequential and unlabeled, we can't control for external factors like time of year and must assume they do not affect this model.

Similarly, in this analysis, we must assume autocorrelation. That is, we must assume that using past data, we can predict future data. Data that is positively autocorrelated suggests that high values are likely to reoccur at the same time in the future. Data that is negatively autocorrelated assumes that low values follow high values (and vice versa). In this analysis, I will predict future revenue based on past revenue.

D1:LINE GRAPH VISUALIZATION



D2:TIME STEP FORMATTING

There are no gaps in the revenue data and there are 731 entries for 731 days. There are no duplicate data entries.

D3:STATIONARITY

To assess the stationarity of the data, I started by looking for visual indications of trends and patterns in the line graph. The data trends upwards and it appears to have a few instances of repeating trends, especially the cycles of spikes and crashes after the two local maxima in the above graph. But a visual inspection can't quantify stationarity; I chose to run an Augmented Dickey-Fuller (ADF) Test to quantify the likelihood that the series is stationary:

```
from statsmodels.tsa.stattools import adfuller

#Augmented Dickey-Fuller (ADF) test
result = adfuller(df['Revenue'])

print("ADF Statistic:", result[0])
print("p-value:", result[1])
print("Critical Values:", result[4])
```

```
ADF Statistic: -2.2183190476089436
p-value: 0.199664006150644
Critical Values: {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
```

As you can see, with a p-value of 0.199, we fail to reject the null hypothesis that the data is stationary. Thus, we must address the fact that the data is not stationary (which will be explained below in D4).

D4:STEPS TO PREPARE THE DATA

Because our data is currently non-stationary, we can apply first order differencing to create a set of stationary data.

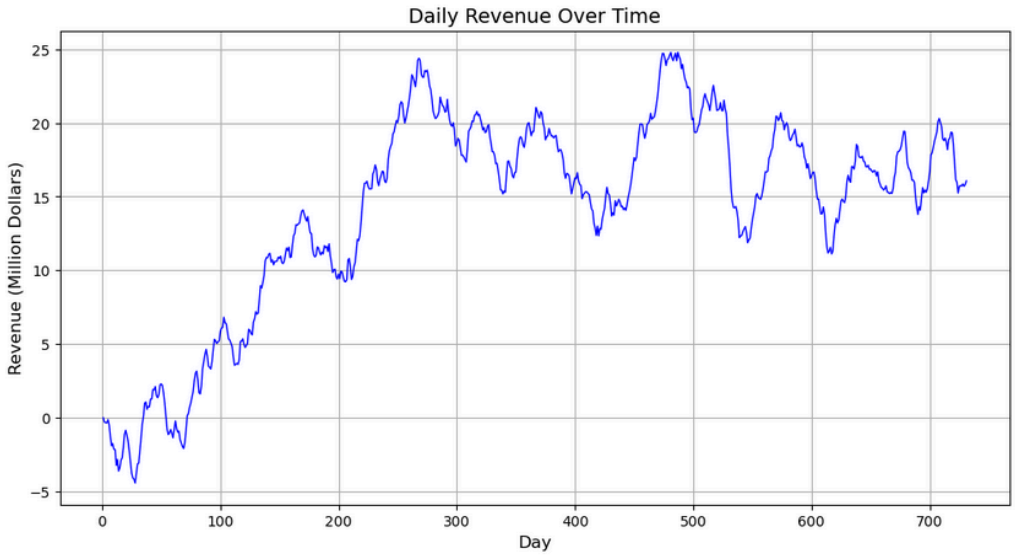
```
#Applying first-order differencing
df['Revenue_diff'] = df['Revenue'].diff().dropna()

result_diff = adfuller(df['Revenue_diff'].dropna())
print("ADF Statistic (Differenced):", result_diff[0])
print("p-value (Differenced):", result_diff[1])
print("Critical Values (Differenced):", result_diff[4])
```

```
ADF Statistic (Differenced): -17.374772303557066
p-value (Differenced): 5.113206978840171e-30
Critical Values (Differenced): {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
```

Testing again for stationarity after first-order differencing, we find a p-value near 0. Thus, we can conclude that the first order differencing adjusted the dataset so it is no longer non-stationary. I then split the data into training (80%) and test (20%) sets based on time order.

E1:REPORT FINDINGS AND VISUALIZATIONS

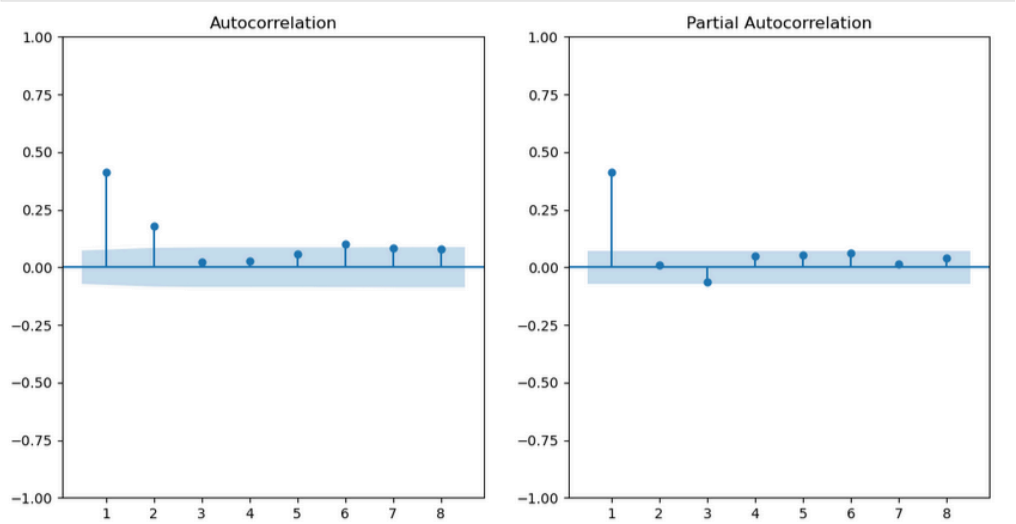
<u>Trends</u>



As we can see in the graph above, there is an upward trend in the graph and there also appears to be some autocorrelation with patterns between spikes and drops, particularly after the local maxima. There does appear to be a mild seasonal component, but this seems more likely to be attributed to the days of the week than to the seasons of the year.

The Autocorrelation Function

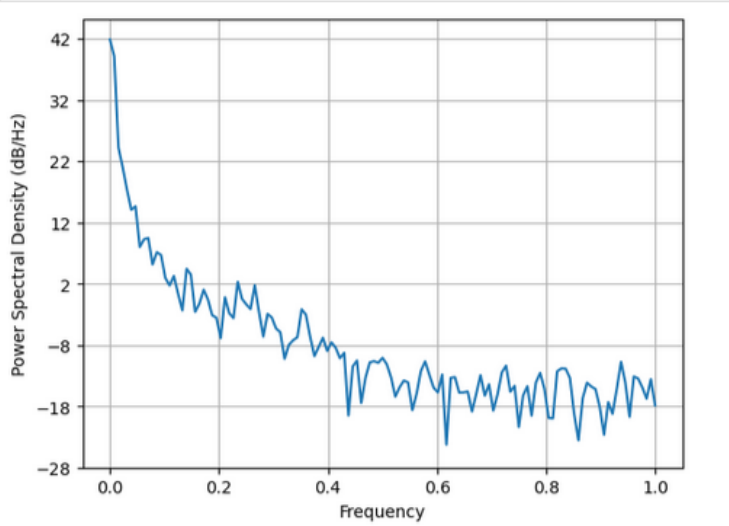
Here are the autocorrelation plots:



As you can see, after the initial adjustments of the lag in the plot at the start of the model, the rest of the datapoints are within the shaded region and are thus not statistically significant. Since the lag decreases in both the Autocorrelation and Partial Autocorrelation, we can determine p=1 and q=0 for our model later.

Spectral Density

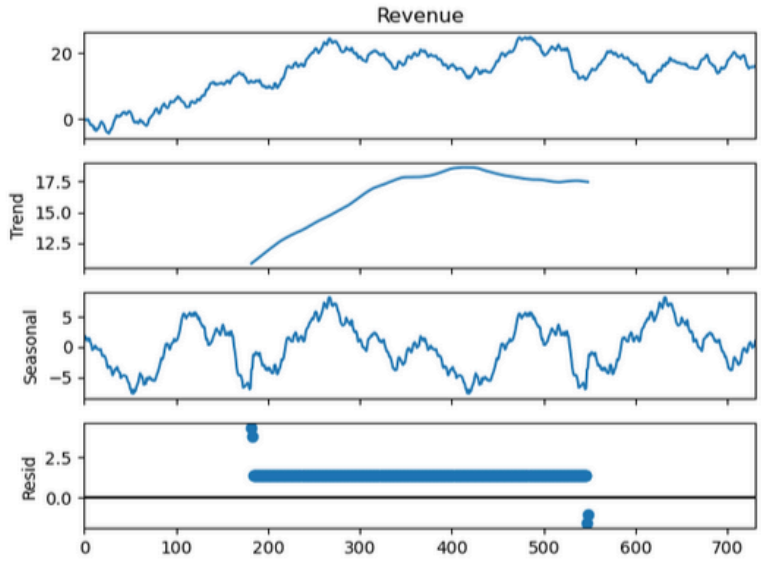
Here are the plots for analyzing Spectral Density:



The Spectral Density is consistent with our other data--after the initial adjustment period, we can look for periodic behavior and seasonality. The seasonality appears to be unpredictable and insignificant.

Decomposed Time Series

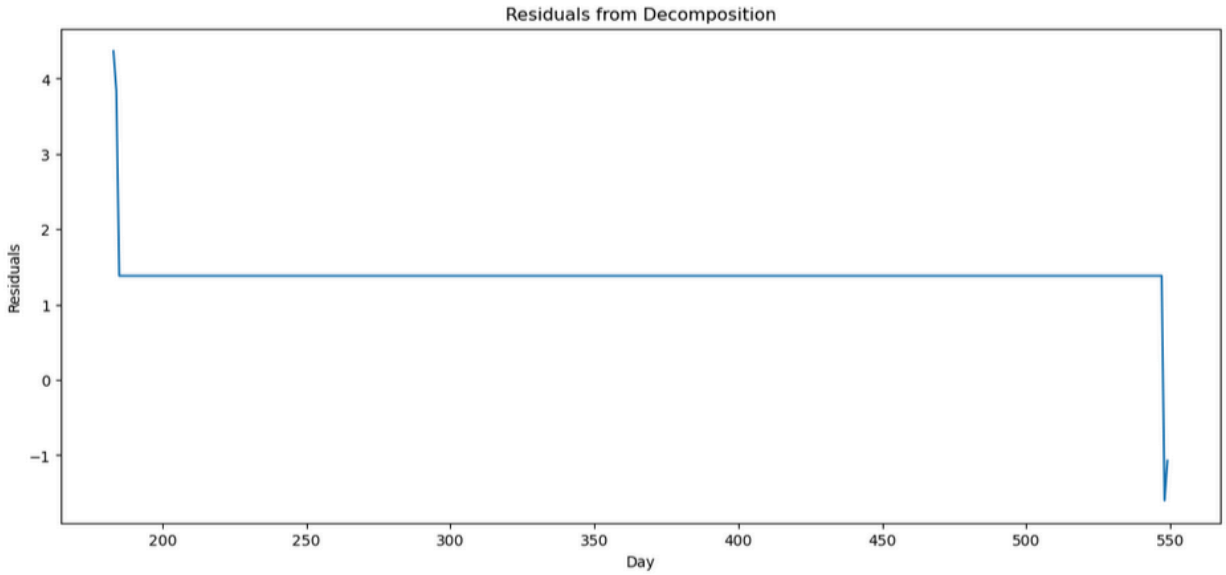
Here are the plots of the decomposed time series:



There is nothing new here--we have the revenue plot, the trend we have already analyzed, the seasonal findings we have discussed previously, and the residual I will discuss next.

Residuals

Here is the plot of my residuals of decomposition



We can confirm from this plot that there is no trend in the residuals of the decomposed series.

E2:ARIMA MODEL

I have chosen the ARIMA model based on the analysis and visualizations above. Although there appears to be a small seasonal component to the data based on the days of the week, these trends are not truly seasonal so I have chosen not to use the SARIMA. As we could see above in the autocorrelation plots above, the ARIMA model is optimal for this analysis.

E3:FORECASTING USING ARIMA MODEL

Here is the code I used to forecast using the ARIMA model

```
#Using the train dataset for the ARIMA model with p=1, d=0, q=0
model = ARIMA(train, order=(1, 0, 0))
results = model.fit()
print(results.summary())
```

For the code I used to prepare the model, see the code attached in the Jupyter notebook file. Output, calculations, and plots will be provided below.

E4:OUTPUT AND CALCULATIONS

Here are the results of the model:

SARIMAX Results

Dep. Variable:	Revenue_diff	No. Observations:	584			
Model:	ARIMA(1, 0, 0)	Log Likelihood	-350.349			
Date:	Thu, 05 Dec 2024	AIC	706.698			
Time:	23:46:16	BIC	719.808			
Sample:	0	HQIC	711.808			
	- 584					
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

Below I will provide further calculations, figures, and analysis.

F1:RESULTS

<u>The selection of an ARIMA model</u>

I chose the ARIMA model because of its time series capabilities. After pre-processing, the data was stationary and could be analyzed to predict future revenue. The SARIMA was an option, as listed above, but I did not find the model to be seasonal enough to justify choosing it instead of ARIMA. I used (1,0,0) as my model because of the observations I made in the autocorrelation and partial autocorrelation graphs, as well as the first order differencing required to create a stationary dataset.

The prediction interval of the forecast AND a justification of the forecast length

I did a standard 80/20 test split, so the forecast was the 146 dates in the test set. Essentially I used 584 days of training to predict 146 datapoints. As mentioned, it is standard to use 80% of data for training in order to produce an accurate model.

The model evaluation procedure and error metric

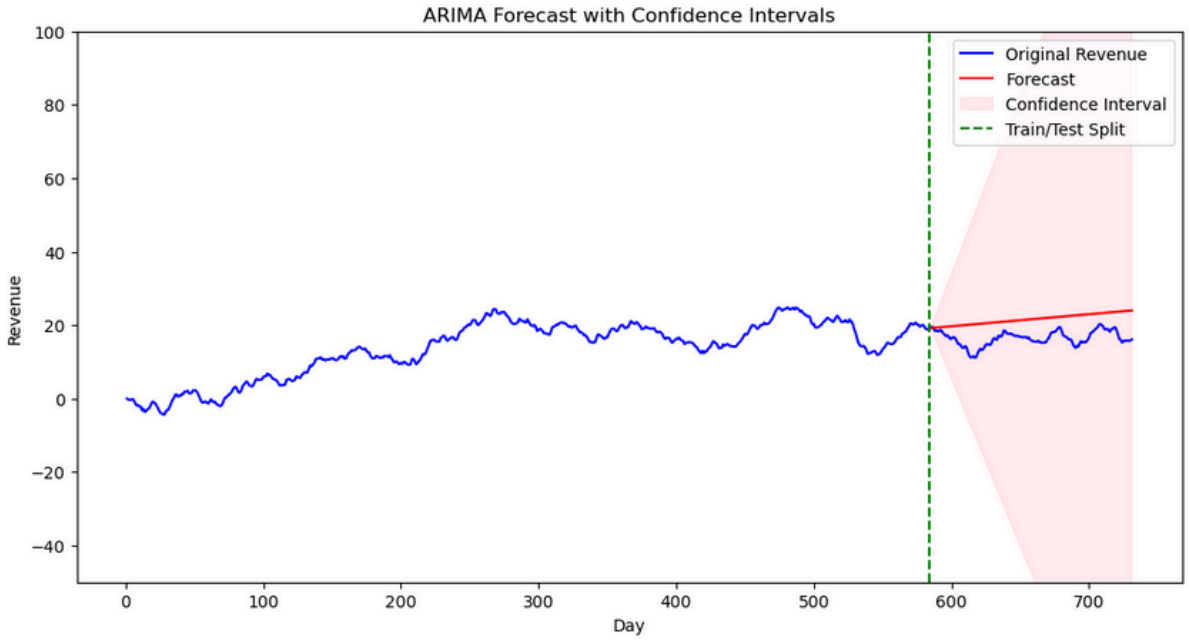
The evaluate the model and produce an error metric, I created a set of predicted values, compared them to the actual test values, and calculated the mean squared error. I then calculated the root mean squared error to evaluate the average distance from the predicted values to the actual values:

```
from sklearn.metrics import mean_squared_error
rmse = np.sqrt(mean_squared_error(test, forecast_mean))
print("Root Mean Squared Error (RMSE):", rmse)
```

```
Root Mean Squared Error (RMSE): 0.4886573053785873
```

F2:ANNOTATED VISUALIZATION

Here is the forecast of the model compared to the test set. Note that it includes the original output, a new prediction line, and a confidence cone. I have also labeled the point where the data switches from training to testing:



As you can see, the forecast is relatively close to the original data. It makes sense that the forecast generally trends slightly upward because our dataset as a whole trends upwards.

F3:RECOMMENDATIONS

My recommendation is that this model should be used to predict revenue in the short term. No model can accurately predict the revenue changes each day because there will always be some unpredictability in a population's hospital needs and costs. However, this model can project short term revenue changes and it can identify the general trend of revenue. In the long term, however, the model must be maintained and updated because the error of the model will continue to grow as time goes on. I also recommend trying a SARIMA model to account for the slight presence of seasonal fluxations in hospital needs and revenue to compare it to this model.

Acknowledgements of Code Sources

Datacamp: ARIMA Models in Python by James Fulton was used for the Augmented Dicky-Fuller to test for stationarity, fitting the ARIMA model, creating forecasts, and confidence intervals.

How to Decompose Time Series Data into Trend and Seasonality by Jason Brownlee was used for generating Additive Decomposition plots.

Sources

No sources were used except for WGU official course materials and the code sources listed above.