Performance Assessment D213 – Advanced Data Analytics  
Task I

Doug Haunsperger

College of IT, Western Governors University

April 22, 2024

# Part I. Rese**arch Questi**on

## A1. Question Proposal

I propose to research the question, “Can we successfully predict future hospital revenue based on historical revenue using an ARIMA model?”

## A2. Goal

The goal of this analysis would be to develop an ARIMA model (Auto-Regressive Integrated Moving Average) that best fits our historical revenue data and test its skill by back-predicting the two most recent quarters’ revenue. If the model shows skill in back-prediction, the hospital can use it to forecast future revenue.

# Part II. Technique Justification

## B. Explanation of ARIMA Models & Assumptions

I am using a model called ARIMA, or Auto-Regressive Integrated Moving Average, which is a class of model used for **univariate** time-series forecasting. According to Prabhakaran (n.d.), univariate time-series forecasting attempts to predict future values of data in a time series using only previous values of the series as input, with no exogenous predictor variables. Therefore, for such a model to have predictive skill, future results must be based primarily on past results – that is, it must be **autocorrelated**; the data points in the time series must be correlated with a “lagged copy of itself” (Reider, n.d.)

Simple ARMA models (auto-regressive, moving average) require stationarity in the data. “Achieving stationarity is a crucial step in time series analysis. A stationary time series has constant statistical properties over time, which simplifies modeling” (Kusawa, 2023). The “I” in ARIMA, or integrated component, “represents the number of differences [of the data with itself] needed to make the time series data stationary” (Kusawa, 2023). This means that ARIMA models can deal with time series data that has an overall trend to it.

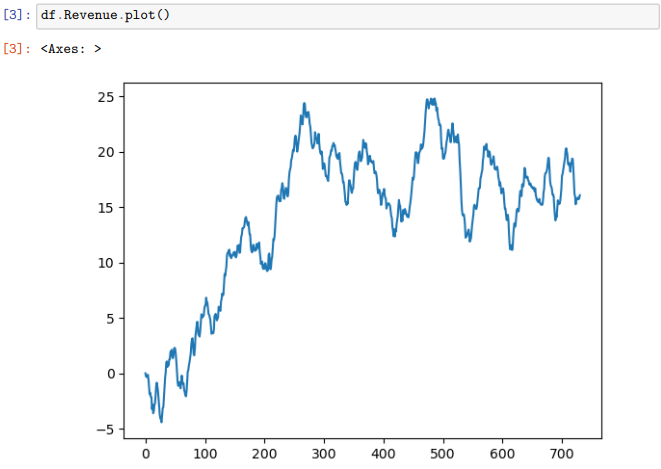
If data has seasonal variation, the model must be extended further to a SARIMA model, but I will show that our revenue data does not exhibit seasonality and therefore an ARIMA model is sufficient.

# Part III. Data Preparation

## C1. Line Graph

Figure 1 is a simple line chart of the data points in the Revenue series in the provided ‘medical\_time\_series.csv’ input file.

Figure 1



## C2. Time Step Formatting

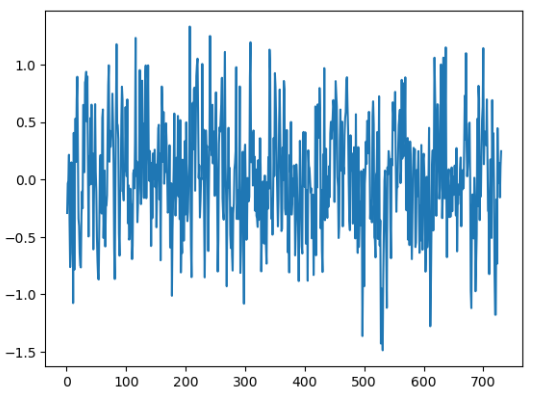
As seen in the attached Jupyter notebook ‘D213\_PA1.ipynb’, section C2, the series is given as daily revenue. There are 731 entries with no gaps or missing data. This represents 2 years’ worth of data.

## C3. Stationarity

Stationarity - or lack of an overall trend in the data - is evaluated through the Augmented Dickey-Fuller (ADF) test. Per Prabhakaran (n.d.), “[t]he null hypothesis of the ADF test is that the time series is non-stationary. So, if the p-value of the test is less than the significance level (0.05) then you reject the null hypothesis and infer that the time series is indeed stationary.”

See the attached Jupyter notebook, section C3 for code and calculations. For this series, the ADF test failed (was not statistically significant at p<0.05). After a single iteration of differencing, the ADF test passed. The differenced series is plotted in Figure 2.

Figure 2



## C4. Data Preparation

To prepare the data for analysis, I performed these steps:

1. Import input CSV file to a dataframe df
2. Set the ‘Day’ column to be the index of the dataframe
3. Difference the data series using df.Revenue.diff().dropna() and run the ADF test using adfuller() to show stationarity. [dropna() is necessary to remove the first term of the differenced series since the first term has no predecessor term to differ with.]
4. Split the data into training and test sets. I chose to use the first 18 months (547 days) as training data and the final 6 months as the test set.
5. Output training set, test set and differenced set to CSV files.

## C5. Cleaned Data Set

The files referenced in step 5 above, ‘train.csv’, ‘test.csv’, ‘difference\_1.csv’ are attached to the submission.

# Part IV. Model Identification & Analysis

## D1. Report Findings & Visualizations

See attached Jupyter notebook, section D1.

### Presence or Absence of Seasonality

I used the seasonal\_decompose() function from the statsmodels.tsa.seasonal package to look for seasonality in the data. I looped through all potential periods from weekly (7 days) to quarterly (91 days). There is a small potential seasonal signal at an 80 day period (approximately +/- $1.5 M), but the residuals (approx. +/- $5M) swamp that signal, indicating that the series can be considered not to be seasonal in nature.

### Trend

seasonal\_decompose() also provides a view of the trend. This is shown in Figure 3.

Figure 3

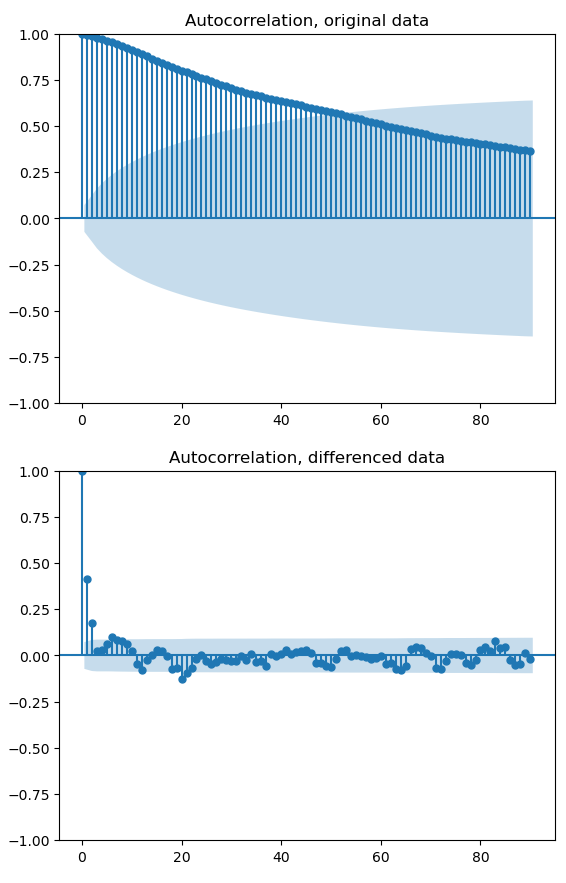
A graph with a line

Description automatically generated

### Autocorrelation Function

I used plot\_acf() from statsmodels to plot the autocorrelation function of both the original time series and the differenced time series, out to a lag period of 90 days. This is shown in Figure 4.

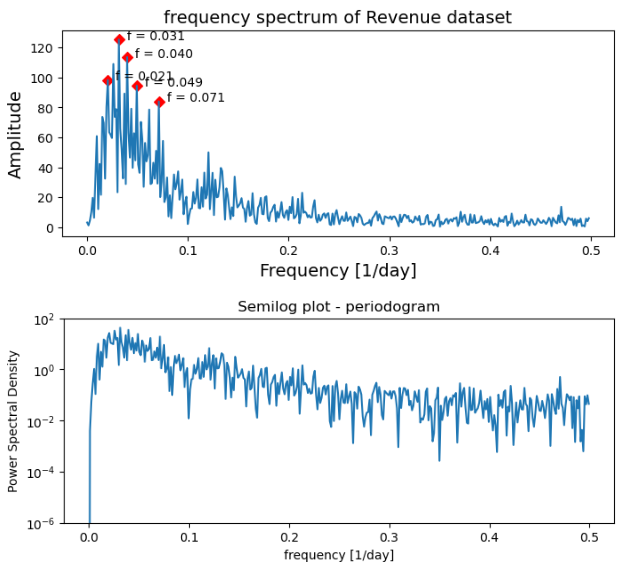
Figure 4



### Spectral Density

I did a Fourier transform on the original time series data. It shows a large amplitude at low frequency due to the overall trend. Therefore, I followed the guidance given by Taspinar (2022) and ran the Fourier transform on the detrended data. It showed some small peaks at low frequency, but when I plot the periodogram on a semilog scale as given in the Scipy API reference (2024), those peaks disappear into the noise floor. See Figure 5. This is a further indication that there is no seasonal or periodic component to the time series data.

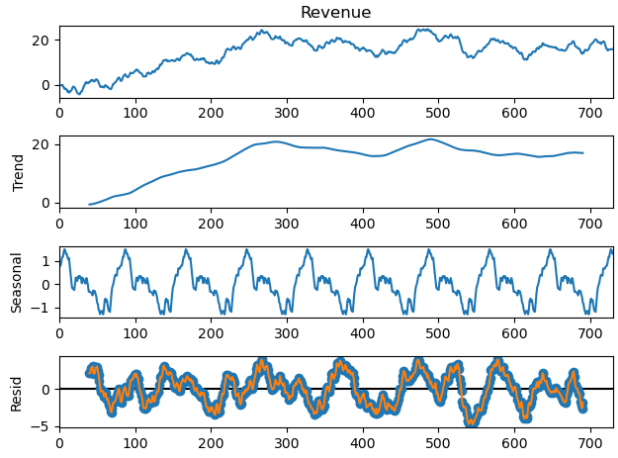
Figure 5



### Decomposed Time Series

The seasonal\_decompose() function provides the decomposition into trend, seasonal component, and residuals. See Figure 6.

Figure 6



### Lack of Trend in the Residuals

The residuals in Figure 6 bounce around the x-axis with no apparent trend.

## D2. ARIMA Model

Given the minimal seasonal signal in the time series, I chose to pursue an ARIMA model rather than the more general SARIMA.

See section D2 of the attached Jupyter notebook for how I manually found the (p, d, q) parameters of the ARIMA model. The parameters are the orders of the autoregressive, differencing, and moving average of the ARIMA model (Prabhakaran, n.d.). This turned out to be an ARIMA(1,1,2) model. Following Prabhakaran, I then used the pmdarima package to automatically search through the parameter space for the best model based on AIC criteria. This search found that ARIMA(0,1,2) performed slightly better.

## D3. Forecast

Using the ARIMA(0,1,2) model trained on the training data set, the model reasonable successfully forecasted the test data with a 15.4% mean absolute percentage error. Nearly all of the test data points were within the 95% confidence bounds. See Figure 7.

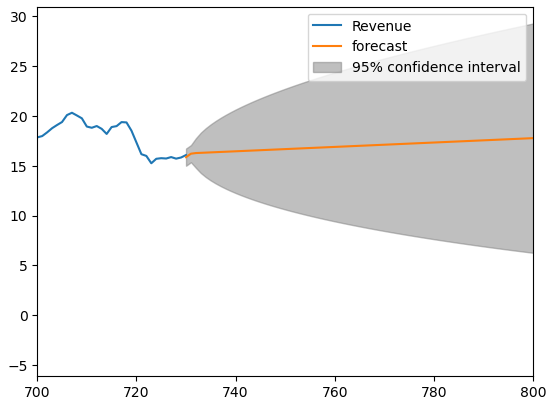
Figure 7

A graph showing the growth of a company

Description automatically generated with medium confidence

I then refit the model on the full 731-record data set to provide a forecast into the near future. This is plotted in Figure 8. I stopped after 2 months since the 95% confidence interval starts to get very wide, making the forecast of little use beyond that point.

Figure 8



## D4/D5. Output Calculations and Code

See attached Jupyter notebook.

# Part V. Data Summary & Implications

## E1. Analysis Results

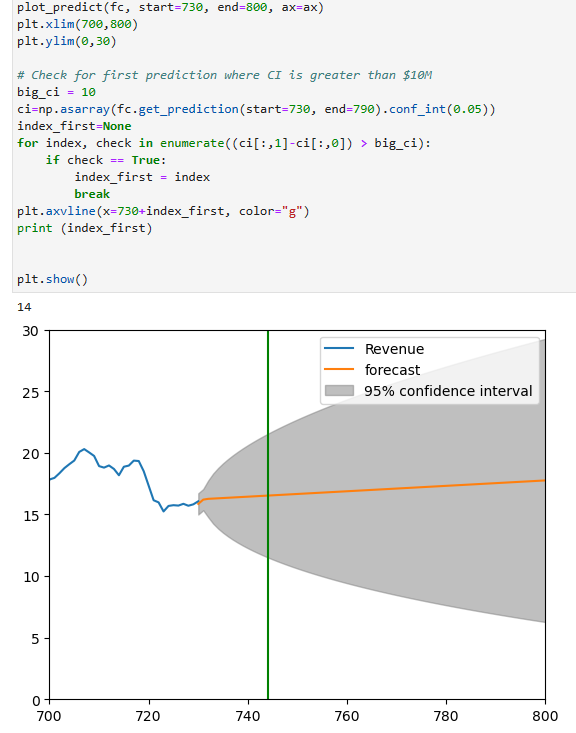
### Selection of ARIMA Model

This is largely addressed above. Figure 4 shows a strong autocorrelation in the time series, indicating that a univariate time-series forecast model such as ARIMA could work. In section D1, I showed that the time series did not display a strong seasonality signal, making the choice of ARIMA over SARIMA appropriate. To find the (p,d,q) parameters of the model, I showed in section C3 that the singly-differenced data was stationary, indicating a d order of 1. In section D2 of the Jupyter notebook, I used the partial autocorrelation function to show that only the lag-1 data has a significant partial autocorrelation. This indicates that the p order should be 1. Using the differenced data, I showed in Figure 4 that there was significant autocorrelation at lag-1 and lag-2, indicating a q order of 2. Ultimately, using the pmdarima package to search the parameter space, I found that a (0,1,2) model performed slightly better than the (1,1,2) model using the AIC criterion.

### Prediction Interval of the Forecast & Justification of Forecast Length

The forecast provides daily predictions of future revenue. I stopped plotting the forecast after ~60 days as the confidence interval grows without bound. Taking an arbitrary maximum 95% confidence interval width of $10M, I find that the forecast should stop after 14 days. See Figure 9.

Figure 9



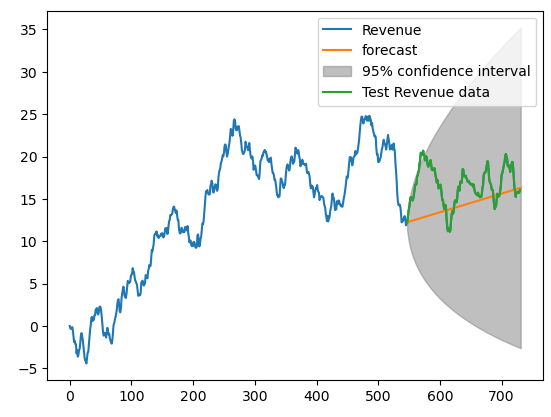
### Model Evaluation Procedure & Error Metric

When evaluating the candidate models based on the training set against the test set, I used the Akaike information criterion (AIC) following the method shown by Prabhakaran (n.d.). I also calculated the Mean Absolute Percentage Error of the final model when comparing to the test set, as this is an easily understood metric.

## E2. Annotated Visualization of Test Data Compared With Forecast

See Figure 10.

Figure 10



## E3. Recommendation

This ARIMA model shows a fair level of success in back-casting the last 6 months of revenue data, with only a 15.4% mean absolute percentage error over the 182-day forecast period. However, the confidence intervals are quite large by the end of that period. It would be better to only rely on this model for short-term forecasting of 14 days or less.

# Part VI. Reporting

## F. Jupyter Notebook Report

## The PDF output of my Jupyter notebook is attached as ‘D213\_PA1\_DSH.pdf’.

## G. Third-party Code Sources

Hayes, S. (June 7, 2021). *Finding Seasonal Trends in Time-Series Data with Python*. Towards Data Science. <https://towardsdatascience.com/finding-seasonal-trends-in-time-series-data-with-python-ce10c37aa861>

Scipy.org 1.13.0 API Reference. (2024). *scipy.signal.periodogram.* <https://docs.scipy.org/doc/scipy-1.13.0/reference/generated/scipy.signal.periodogram.html>

Taspinar, A. (Dec. 22, 2020). *Time-Series Forecasting with Stochastic Signal Analysis Techniques.* ML Fundamentals Blog. <https://ataspinar.com/2020/12/22/time-series-forecasting-with-stochastic-signal-analysis-techniques/>

## H. References

Kusawa, S. (October 3, 2023). *Demystifying ARIMA Model Parameters: A Step-by-Step Guide*. Data Magic. <https://datamagiclab.com/demystifying-arima-model-parameters-a-step-by-step-guide/>

Prabhakaran, S. (n.d.). *ARIMA Model – Complete Guide to Time Series Forecasting in Python*. Machine Learning Plus. <https://www.machinelearningplus.com/time-series/arima-model-time-series-forecasting-python/>

Reider, R. (n.d.) Lesson 1: Correlation and Autocorrelation. *Time Series Analysis in Python* [MOOC]. DataCamp. <https://campus.datacamp.com/courses/time-series-analysis-in-python/correlation-and-autocorrelation>