D213 – Advanced Data Analytics

WGU M.S. Data Analytics

Lyssa Kline

March 23, 2025

A, Part 1: Research Question

A1, Research Question and Technique

The research question in scope for this analysis is "How do hospital readmission penalties impact daily revenue trends over time, and can we forecast future revenue fluctuations using time series modeling?" This question is crucial in understanding the financial consequences of readmission penalties imposed by CMS. By analyzing revenue trends, hospitals can anticipate potential revenue losses due to excessive readmissions and develop strategies to mitigate financial risks.

A2, Analysis Goal and Objectives

The objective of this time series analysis includes identifying revenue patterns by examining trends and seasonality in the daily revenue data over time. This allows us to detect anomalies by identifying significant drops or fluctuations in revenue that may be linked to policy changes, penalties, or operational inefficiencies.

An additional objective of the analysis would be to forecast future revenue using the ARIMA model, allowing us to predict revenue trends. This would help executives make data-driven financial and operational decisions. Lastly, this analysis would assess the impact of readmission penalties. In doing so, the hospitals could correlate revenue trends with known CMS policy changes to understand how penalties affect hospital finances.

The insights from this analysis will help hospital executives optimize their strategies for improving financial stability and reducing readmission rates.

B, Part 2: Method Justification

B1, Assumptions

To apply the ARIMA model effectively, the data must meet certain key assumptions most notably, stationarity and the presence of autocorrelation.

To test for stationarity, we can conduct a Dickey-Fuller test. A time series is stationary if its statistical properties, such as mean and variance, do not change over time. ARIMA requires stationarity for reliable predictions. If the dataset exhibits trends or seasonality, differencing may be applied to achieve stationarity.

Autocorrelation is the correlation of a time series with a lagged version of itself, it measures how past values of a variable influence its future values. A time series model assumes that past values influence future values, meaning the data points are autocorrelated. In this analysis, we will use autocorrelation function (ACF) and partial autocorrelation function (PACF) plots to identify significant correlations in the dataset.

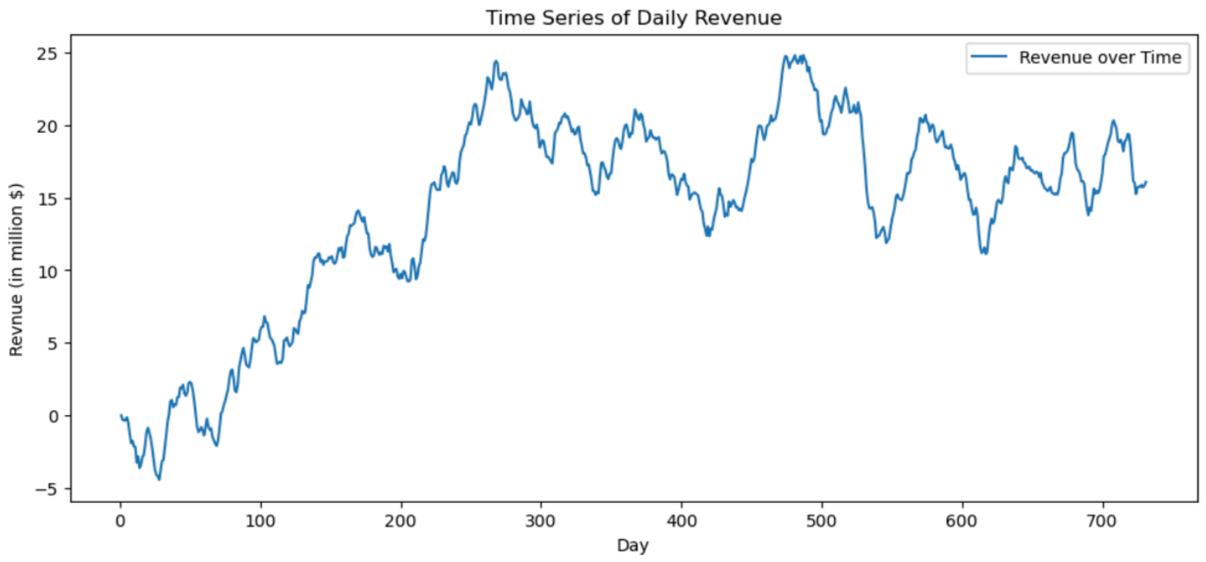
ARIMA works best for non-seasonal time series. By ensuring these assumptions hold, we can build a robust ARIMA model for forecasting hospital revenue trends.

C, Part 3: Data Preparation

C1, Time Series Line Graph

A line plot of the "Revenue" variable was created over 731 days to begin the time series analysis. The graph below visually shows the revenue fluctuations throughout the period, displaying both upward and downward trends. This visualization helps to detect patterns, anomalies, and the overall structure of the data before modeling. The revenue demonstrates periods of sharp growth and cyclical declines, suggesting possible seasonality or underlying patterns.

```
# Data preparation steps
# Plot the time series
plt.figure(figsize=(12,5))
plt.plot(df["Day"], df["Revenue"], label="Revenue over Time")
plt.xlabel("Day")
plt.ylabel("Revnue (in million $)")
plt.title("Time Series of Daily Revenue")
plt.legend()
plt.show()
```



C2, Time Step Formatting

The dataset originally used an integer "Day" column (1 to 731). This was converted into a proper datetime format using pd.to_datetime() with daily frequency (unit="D") and then set as the index of the DataFrame. This conversion allows for more accurate time series modeling and plotting. To ensure data quality, checks were made for missing values and gaps in the date range. No missing values were found, and 0 missing days in the date range were found meaning the time series is continuous. The length of the time series equals the total number of unique daily entries between the start and end date, since there are no missing days and this data is continuous the length is 731 unique days.

Due to the findings shown below, we can determine that the time series is continuous, evenly spaced, and fully complete, with 731 daily records and no missing observations.

```
# Time step formatting and checking for gaps
# Check for missing values
print("Missing values:\n", df.isnull().sum())
```

```
Missing values:
 Day        0
Revenue    0
dtype: int64
```

```
# Check for gaps in the sequence
df["Day"] = pd.to_datetime(df["Day"], unit="D") # Convert to datetime format
df.set_index("Day", inplace=True)

# Check if there are missing days
missing_days = pd.date_range(start=df.index.min(), end=df.index.max()).difference(df.index)
print(f"Number of missing days: {len(missing_days)}")
```

```
Number of missing days: 0
```

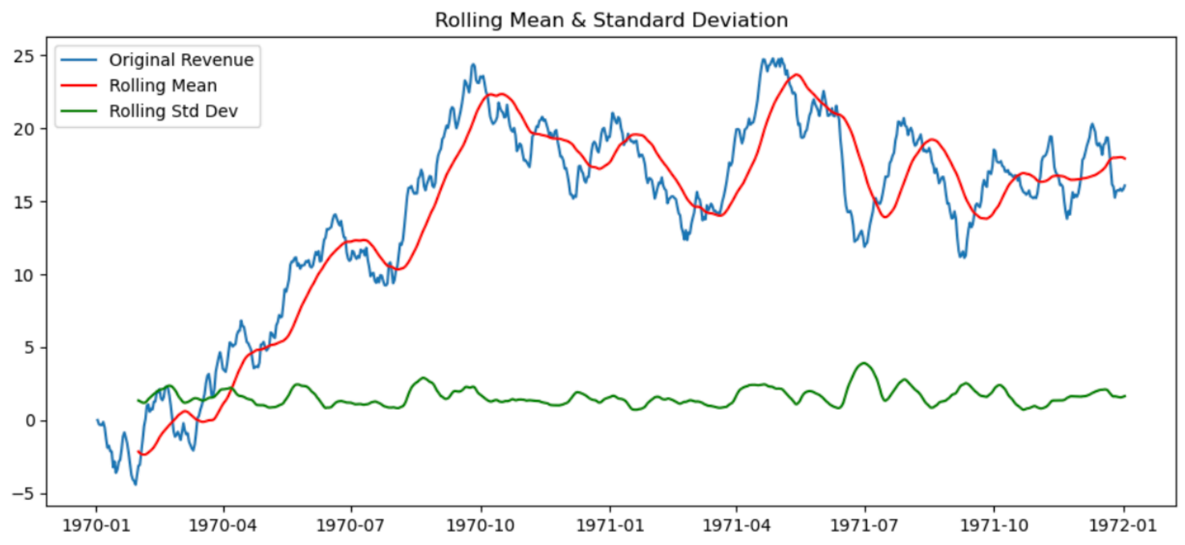
C3, Stationarity Evaluation

Stationarity was evaluated using the rolling mean and standard deviation, as well as an augmented Dickey-Fullers (ADF) test. A 30-day rolling window was plotted over the original time series for the rolling mean and standard deviation. The rolling mean and standard deviation visibly fluctuated over time, indicating the presence of a trend and non-stationarity.

For the Augmented Dickey-Fuller (ADF) Test, if the p-value > .05 then the time series is not stationary, if the p-value is <= .05 then the time series is stationary. From the results of the first output, we can observe that the original series (p-value=0.1997**)** meaning it is not stationary. To continue we need to apply differencing to make it stationary. After applying to difference and re-running the dickey-fuller test, the p-value reaches ~5.11e-30, indicating strong evidence of stationarity.

After applying first-order differencing, the revenue data became stationary, satisfying a key assumption for ARIMA modeling. See the output of stationarity and Dickey fuller test below.

```
# Evaluate Stationarity
# Rolling statistics
rolling_mean = df["Revenue"].rolling(window=30).mean()
rolling_std = df["Revenue"].rolling(window=30).std()

# Plot rolling statistics
plt.figure(figsize=(12,5))
plt.plot(df["Revenue"], label="Original Revenue")
plt.plot(rolling_mean, label="Rolling Mean", color="red")
plt.plot(rolling_std, label="Rolling Std Dev", color="green")
plt.legend()
plt.title("Rolling Mean & Standard Deviation")
plt.show()
```



Once stationarity and differencing have been applied, the data is now ready to be prepared. To prepare the data for modeling the first-order difference was computed to remove trends and stabilize the series. Next, the dataset was split into training and testing sets to evaluate model performance. The training set was used to fit the ARIMA model. The test set was reserved for forecasting evaluation. Then a plot was generated to visualize the split, clearly distinguishing the train and test periods.

The differenced and split data is ready for time series forecasting, with the structure needed to train and validate the model. See the steps taken to prepare and split the data below as well as the visualized plot shown.

```
# Perform Augmented Dickey-Fuller Test
adf_test = adfuller(df["Revenue"])
print("ADF Statistic:", adf_test[0])
print("p-value:", adf_test[1])
print("Critical Values:", adf_test[4])
```

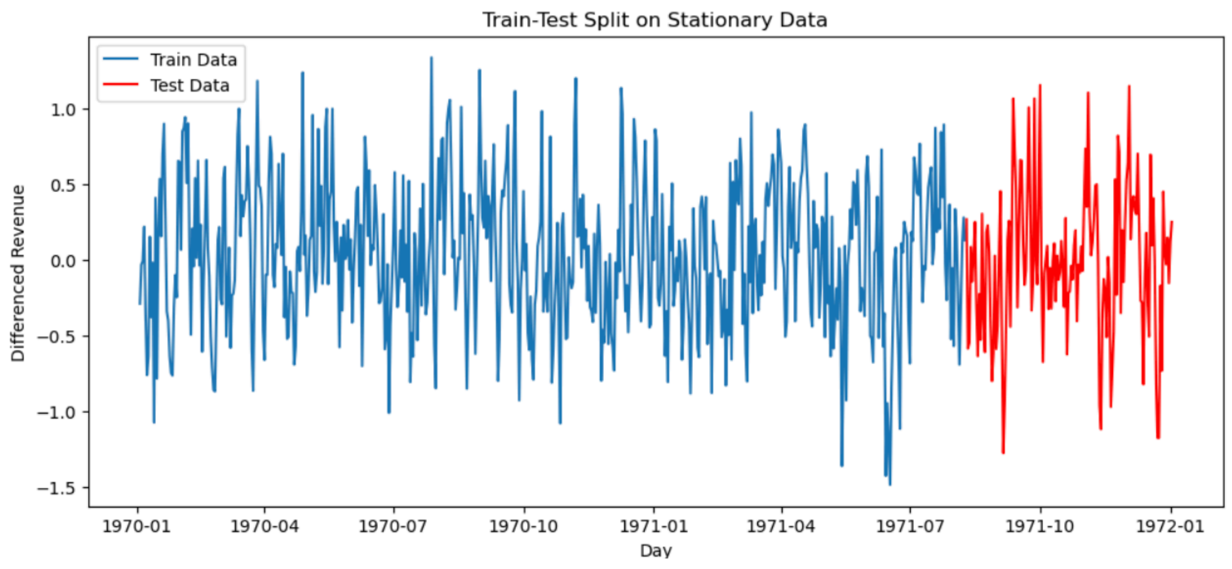
```
ADF Statistic: -2.2183190476089454
p-value: 0.19966400615064356
Critical Values: {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
```

```
# Apply differencing to make the data stationary
df["Revenue_diff"] = df["Revenue"].diff()  # First-order differencing
df.dropna(inplace=True)  # Remove NaN values from differencing
```

```
# Run Dickey-Fuller (ADF) test again
adf_test_diff = adfuller(df["Revenue_diff"])
print("ADF Statistic after differencing:", adf_test_diff[0])
print("p-value after differencing:", adf_test_diff[1])
print("Critical Values:", adf_test_diff[4])
```

```
ADF Statistic after differencing: -17.374772303557066
p-value after differencing: 5.113206978840172e-30
Critical Values: {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
```

```
# Define train-test split
train_size = int(len(df) * 0.8)  # 80% train, 20% test
train, test = df.iloc[:train_size], df.iloc[train_size:]

# Plot train-test split
plt.figure(figsize=(12, 5))
plt.plot(train.index, train["Revenue_diff"], label="Train Data")
plt.plot(test.index, test["Revenue_diff"], label="Test Data", color="red")
plt.xlabel("Day")
plt.ylabel("Differenced Revenue")
plt.title("Train-Test Split on Stationary Data")
plt.legend()
plt.show()
```



C4, Cleaned Data

A prepared dataset outputted in CSV format can be found within the attached 'prepared_timeseries_task1.csv' file.

D, Part 4: Model Identification and Analysis

D1, Annotated Findings

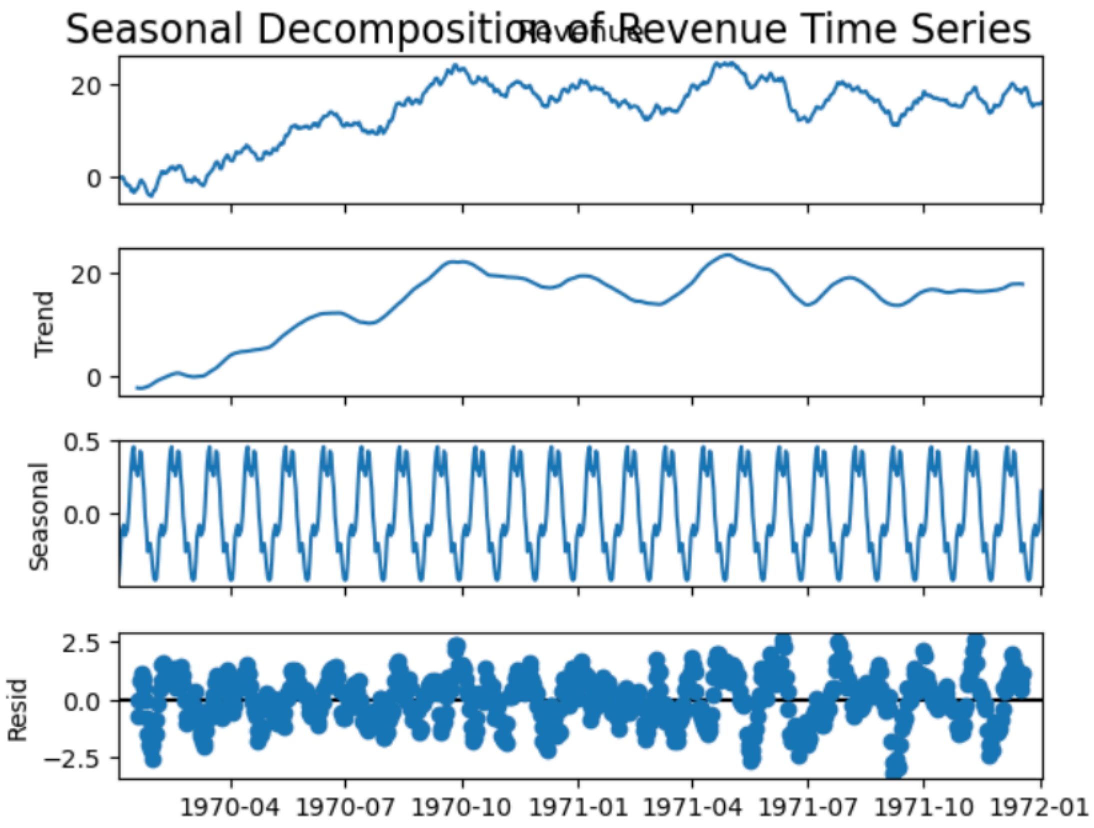
To analyze the time series data, we performed a series of functions that allowed us to report the annotated findings with visualizations of the data analysis. These visualizations showcased a variety of components, allowing us to conclude the lack of a seasonal component, trends, the autocorrelation function, spectral density, the decomposed time series, and a confirmation of the lack of trends in the residuals of the decomposed series. See the reported annotated findings with correlated visualizations below.

To observe the presence of a lack of a seasonal component. The analysis uses seasonal decomposition with an additive model and a 30-day period, a clear seasonal component was observed. The seasonal plot displayed recurring peaks and troughs in a repeating pattern, suggesting the presence of monthly seasonality in revenue trends.

To observe trends, the analysis used decomposition. The decomposition also revealed a long-term upward trend followed by stabilization. This indicates a period of early revenue growth followed by periodic fluctuations. This is important for model selection, as trend removal (via differencing) is necessary for ARIMA modeling.

See the data output below showing the seasonality and trend check performed.

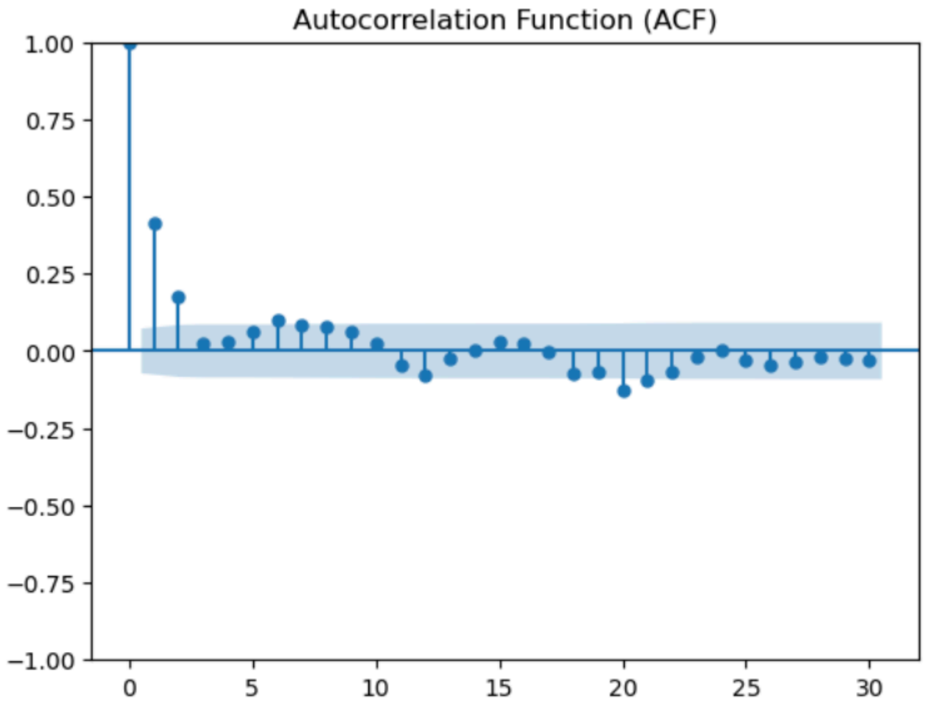
```python
# Annotated Findings and Visualizations
# Seasonality and trend check using seasonal decomposition and spectral density
decomposition = seasonal_decompose(df["Revenue"], model="additive", period=30)
decomposition.plot()
plt.suptitle("Seasonal Decomposition of Revenue Time Series", fontsize=16)
plt.show()
```

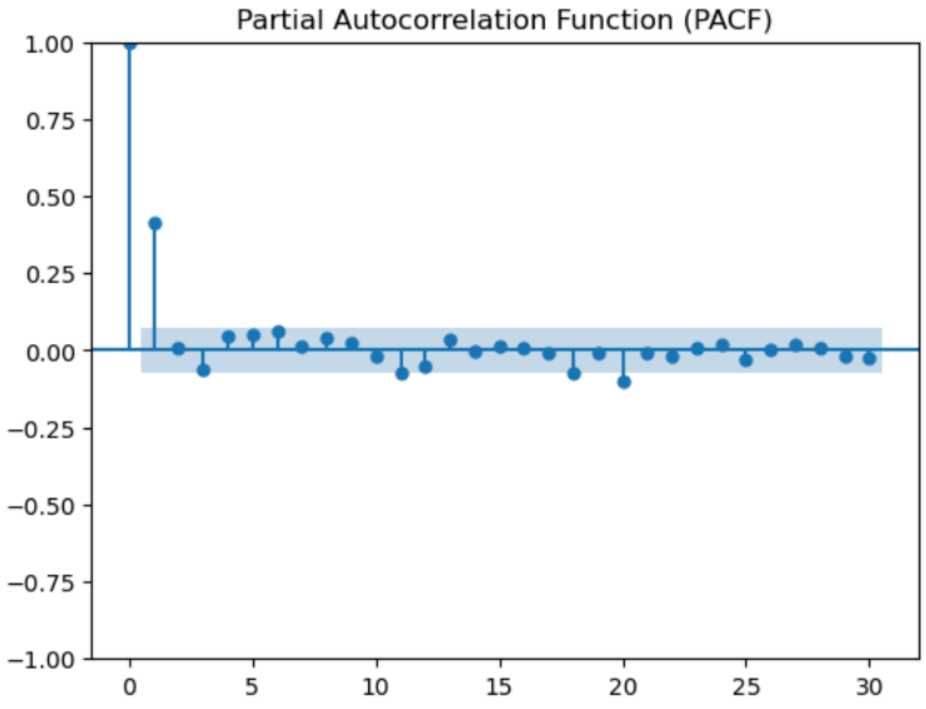


To identify ACF and PACF the plot_acf and plot_pacf functions were used. The ACF plot showed a slow decay, confirming the presence of autocorrelation in the series. The first lag was significant, suggesting that past values influence future ones. This is typical for data with trend or seasonality and justifies the inclusion of an autoregressive term in the model. The PACF plot had a strong spike at lag 1, dropping off afterward. This supports the inclusion of 1 autoregressive (AR) term in the ARIMA model.

See the data output below showing the ACF and PACF checks performed.

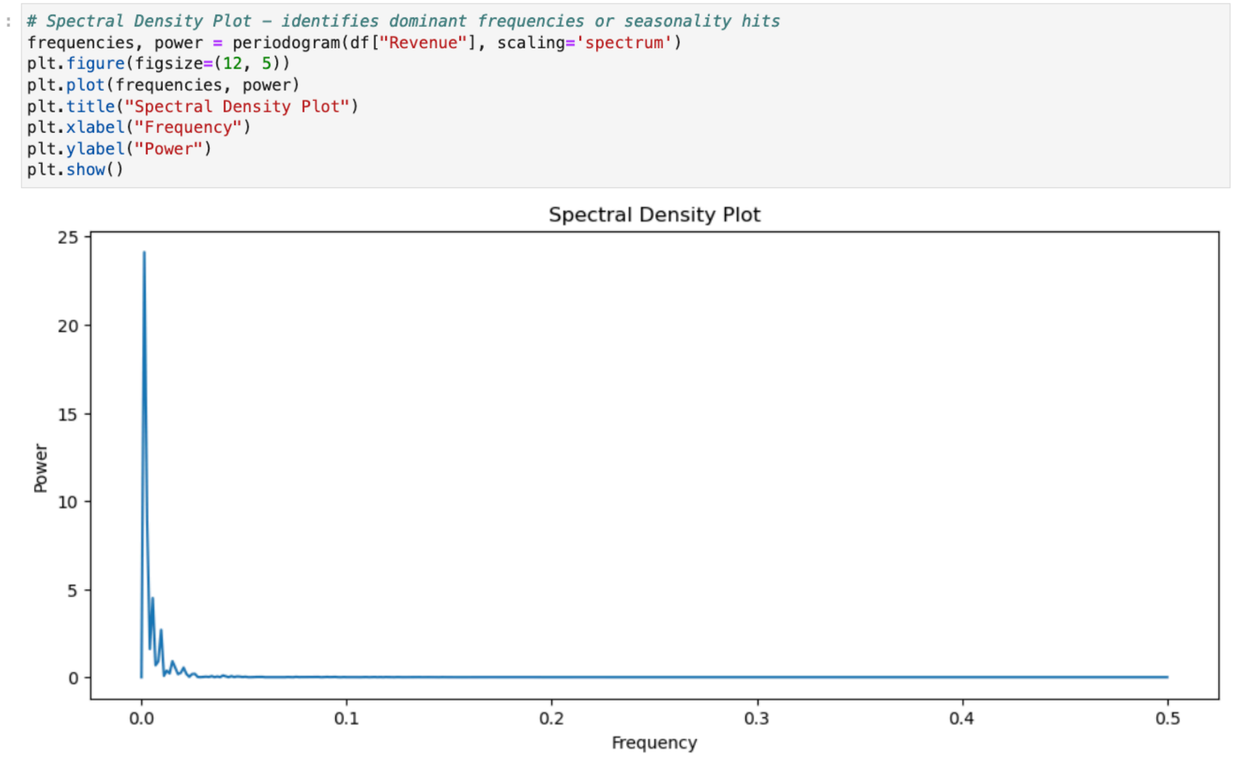
```
# Autocorrelation and partial autocorrelation identification (ACF & PACF)
plot_acf(df["Revenue_diff"], lags=30)
plt.title("Autocorrelation Function (ACF)")
plt.show()

plot_pacf(df["Revenue_diff"], lags=30)
plt.title("Partial Autocorrelation Function (PACF)")
plt.show()
```





To identify spectral density, a spectral density plot was used. The spectral density plot displayed a dominant low-frequency spike, further indicating the presence of seasonal cycles in the data. Peaks at specific frequencies correspond to periodicities in the time series.

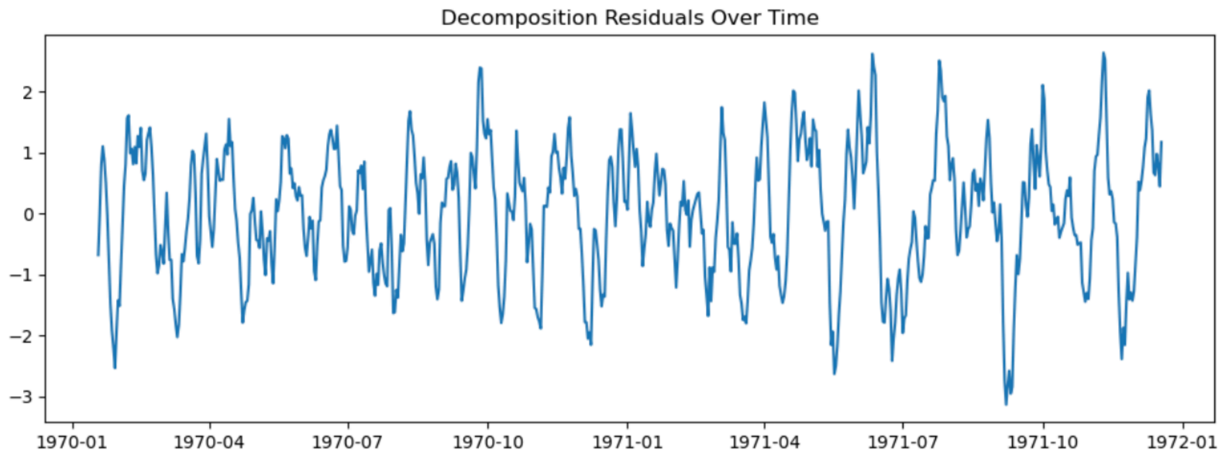
See the data output below showing the plotted spectral density.


```
# Spectral Density Plot - identifies dominant frequencies or seasonality hits
frequencies, power = periodogram(df["Revenue"], scaling='spectrum')
plt.figure(figsize=(12, 5))
plt.plot(frequencies, power)
plt.title("Spectral Density Plot")
plt.xlabel("Frequency")
plt.ylabel("Power")
plt.show()
```

Lastly, we can extract the residuals from decomposition to check for trends and randomness. The residuals extracted from the seasonal decomposition appear randomly distributed around zero with no visible trend, confirming that the model effectively captured trend and seasonality. This supports the use of ARIMA on the detrended/de-seasonalized series.

See the data output below showing the decomposition residuals over time plotted.

```
# Extract residuals from decomposition to check for trends and randomness
residual = decomposition.resid.dropna()

plt.figure(figsize=(12, 4))
plt.plot(residual)
plt.title("Decomposition Residuals Over Time")
plt.show()
```



D2, ARIMA Model

The ARIMA model accounts for the observed trend and seasonality of the time series data. To identify the (p,d,q) for the ARIMA model, the auto_arima() function was used. See the output below.

```
# Identify the ARIMA model - accounts for observed trend and seasonality of the time series data
model_auto = pm.auto_arima(df["Revenue"], seasonal=False, trace=True,
                           error_action='ignore', suppress_warnings=True,
                           stepwise=True)

print(model_auto.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=882.726, Time=0.21 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1015.163, Time=0.03 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=880.725, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=905.489, Time=0.02 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1014.728, Time=0.01 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=882.666, Time=0.03 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=882.679, Time=0.04 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=882.775, Time=0.09 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=879.385, Time=0.01 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=881.314, Time=0.01 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=881.330, Time=0.02 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=904.505, Time=0.01 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=881.407, Time=0.04 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 0.558 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  730
Model:               SARIMAX(1, 1, 0)   Log Likelihood                -437.693
Date:                Sat, 22 Mar 2025   AIC                            879.385
Time:                        10:05:24   BIC                            888.568
Sample:                    01-03-1970   HQIC                           882.928
                         - 01-02-1972
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4142      0.034     12.246      0.000       0.348       0.480
sigma2         0.1945      0.011     17.824      0.000       0.173       0.216
===================================================================================
Ljung-Box (L1) (Q):                   0.01   Jarque-Bera (JB):                 1.94
Prob(Q):                              0.90   Prob(JB):                         0.38
Heteroskedasticity (H):               1.00   Skew:                            -0.02
Prob(H) (two-sided):                  0.98   Kurtosis:                         2.75
===================================================================================

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

From the output shown, the best-fitting model was determined to be ARIMA(1,1,0). The first value in the model is the 'p' value, a p =1 value shows one autoregressive term. The second number is the 'd' value, d=1 value showing the first-order differencing applied to remove the trend. The last value is the 'q' value, q=0 means there is no moving average term. The model was selected based on the AIC (Akaike Information Criterion) minimization, with AIC = 879.385.

D3, Forecasted ARIMA Model

To perform a forecast using the ARIMA Model, the ARIMA(1, 1, 0) model was fit on the training set (80% of data), and a forecast was generated for the test set (remaining 20%).

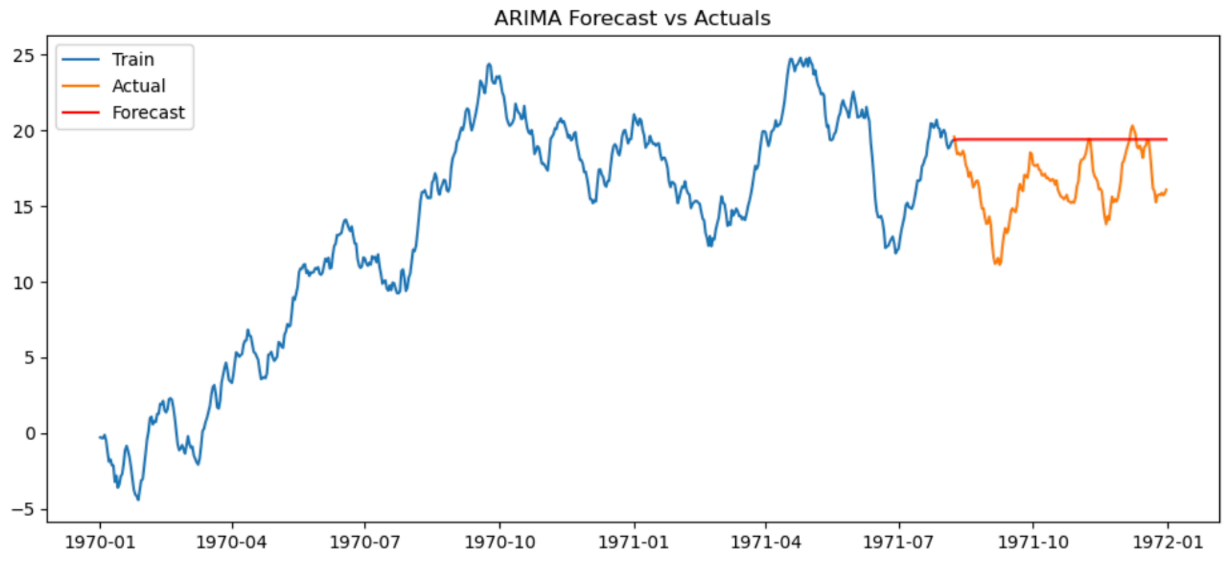
The forecasted revenue values were plotted alongside actual test values. While the forecast captured the overall level of the revenue, it underestimated the variance of the observed test values, a common limitation of simple ARIMA models without seasonal components.

See the code used to forecast the model below, with the plotted ARIMA forecast vs Actuals.

```
[ ]: # Fix issue with model below by setting datetimeindex with frequency
     df.index = pd.date_range(start='1970-01-01', periods=len(df), freq='D')

     # Reset train/test split after index
     train_size = int(len(df) * 0.8)
     train, test = df.iloc[:train_size], df.iloc[train_size:]
```

```
[ ]: # Perform the forecast using training set to fit the model and forecast the test set length
     # Fit model on train data
     model = ARIMA(train["Revenue"], order=(1, 1, 0))  # Using values identified above
     model_fit = model.fit()

     # Forecast the test set range
     forecast = model_fit.forecast(steps=len(test))
```

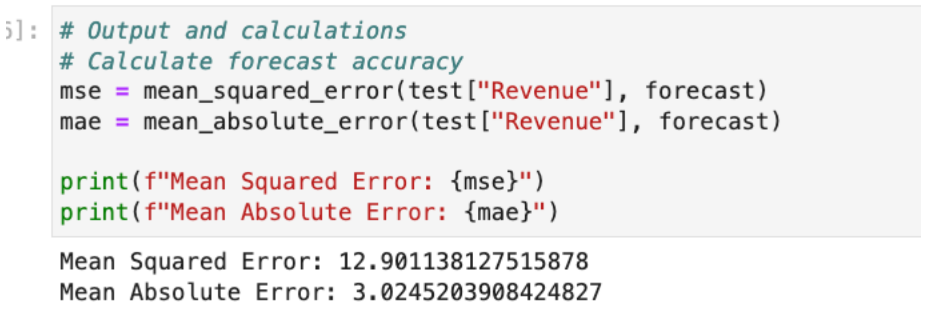
```
# Plot forecast vs actual
plt.figure(figsize=(12, 5))
plt.plot(train.index, train["Revenue"], label="Train")
plt.plot(test.index, test["Revenue"], label="Actual")
plt.plot(test.index, forecast, label="Forecast", color="red")
plt.title("ARIMA Forecast vs Actuals")
plt.legend()
plt.show()
```



D4, Output and Calculations

The output and calculations of the analysis performed included computing the mean squared error and the mean absolute error. The mean squared error (MSE) was 12.90, and the mean absolute error (MAE) was 3.02.

These metrics indicate a moderate forecasting error, acceptable for a basic ARIMA model without seasonal adjustments. If needed, a more complex SARIMA model can be used in future iterations to better capture seasonal dynamics.



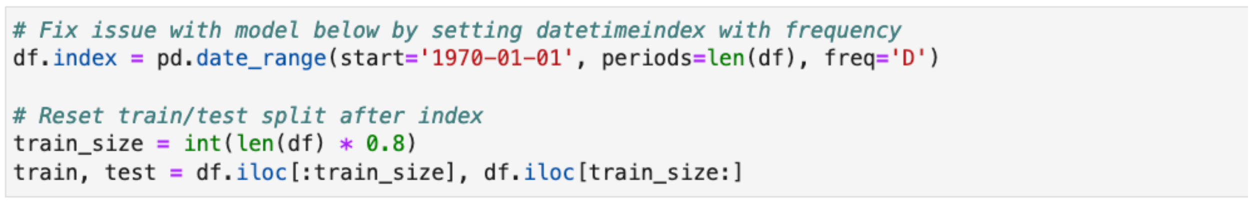
```
# Output and calculations
# Calculate forecast accuracy
mse = mean_squared_error(test["Revenue"], forecast)
mae = mean_absolute_error(test["Revenue"], forecast)

print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
```

```
Mean Squared Error: 12.901138127515878
Mean Absolute Error: 3.0245203908424827
```

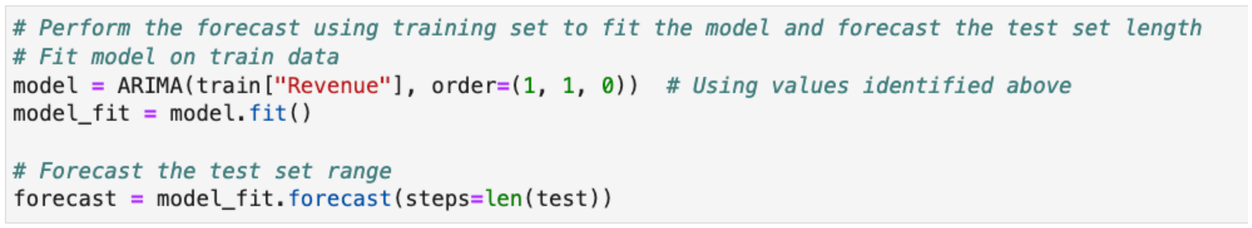
These metrics indicate a moderate forecasting error, acceptable for a basic ARIMA model without seasonal adjustments. If needed, a more complex SARIMA model can be used in future iterations to better capture seasonal dynamics.

D5, Time Series Model Code

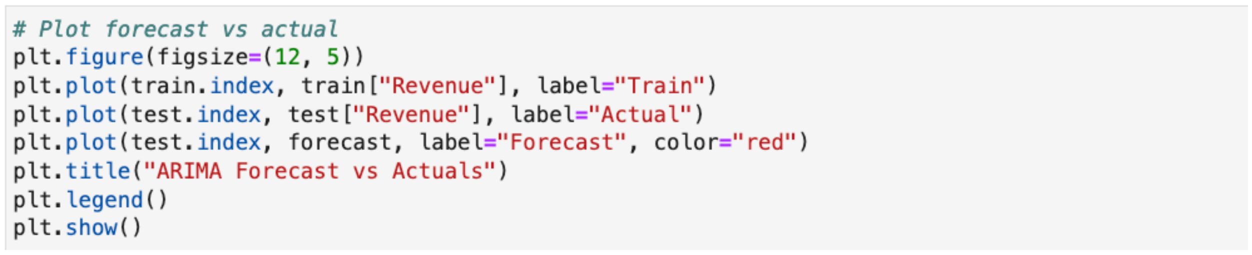
The code used to support the implementation of the time series model can be seen below. This included setting a date-time index with frequency, re-setting the train/test split, performing the forecast using the training set to fit the model and forecasting the test set length, then forecasting the test set range, and lastly, plotting the forecast vs actual.



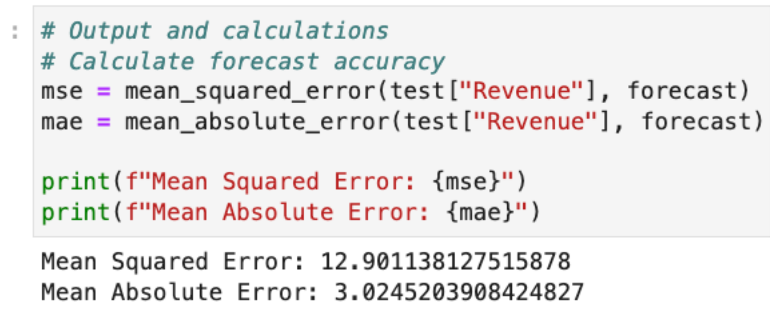
```
# Fix issue with model below by setting datetimeindex with frequency
df.index = pd.date_range(start='1970-01-01', periods=len(df), freq='D')

# Reset train/test split after index
train_size = int(len(df) * 0.8)
train, test = df.iloc[:train_size], df.iloc[train_size:]
```



```
# Perform the forecast using training set to fit the model and forecast the test set length
# Fit model on train data
model = ARIMA(train["Revenue"], order=(1, 1, 0))  # Using values identified above
model_fit = model.fit()

# Forecast the test set range
forecast = model_fit.forecast(steps=len(test))
```



```
# Plot forecast vs actual
plt.figure(figsize=(12, 5))
plt.plot(train.index, train["Revenue"], label="Train")
plt.plot(test.index, test["Revenue"], label="Actual")
plt.plot(test.index, forecast, label="Forecast", color="red")
plt.title("ARIMA Forecast vs Actuals")
plt.legend()
plt.show()
```



```
# Output and calculations
# Calculate forecast accuracy
mse = mean_squared_error(test["Revenue"], forecast)
mae = mean_absolute_error(test["Revenue"], forecast)

print(f"Mean Squared Error: {mse}")
print(f"Mean Absolute Error: {mae}")
```

```
Mean Squared Error: 12.901138127515878
Mean Absolute Error: 3.0245203908424827
```

E, Part 5: Data Summary and Implications

E1, Results

Using the auto_arima() function, the most appropriate model for the revenue time series was determined to be ARIMA(1, 1, 0). This configuration was chosen based on minimizing the Akaike Information Criterion (AIC), with a resulting AIC of 879.385.

- p = 1: Indicates that the model includes one lag of the autoregressive term, consistent with the PACF plot.
- d = 1: Indicates first-order differencing was necessary to remove the non-stationary trend, confirmed by the Augmented Dickey-Fuller (ADF) test.
- q = 0: No moving average term was included based on the ACF results.

While the plot displays point forecasts, the ARIMA model also allows for confidence intervals around each predicted value. In practice, a 95% prediction interval can be extracted using get_forecast().conf_int() from the model to show the range of likely values. This interval accounts for model uncertainty and highlights the range in which actual revenue is expected to fall.

The forecast was made for the length of the test set, which represented 20% of the original dataset — equivalent to 146 days out of 731. This is a standard choice in time series modeling and allows for meaningful model validation without overfitting to short-term fluctuations.

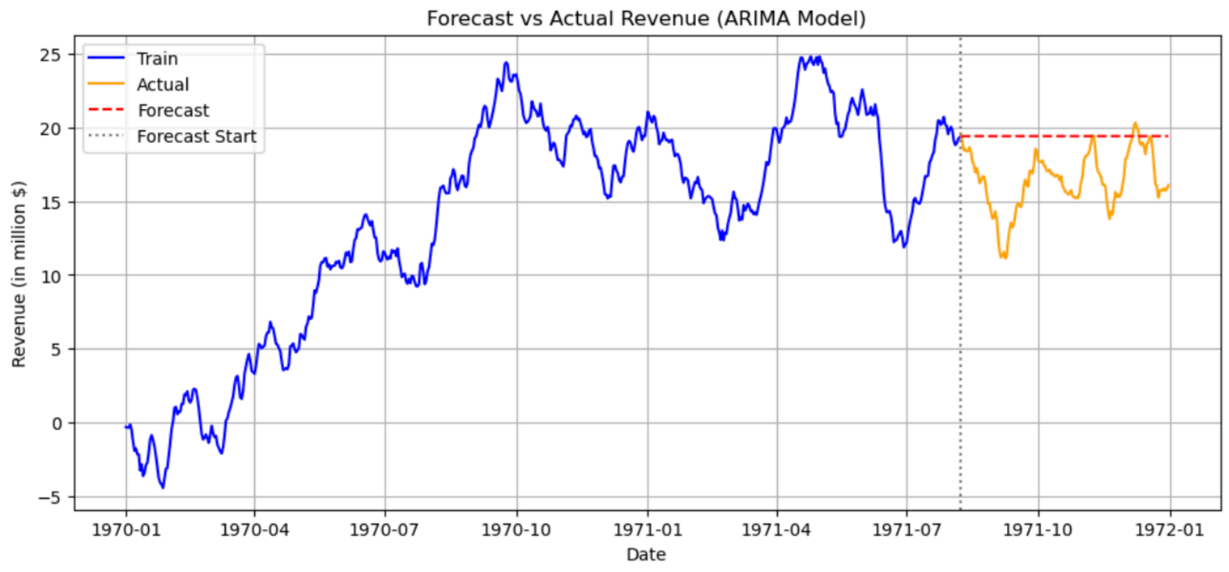
Model performance was evaluated using two commonly accepted error metrics. (MSE) = 12.90 and (MAE) = 3.02, These metrics reflect the average prediction error of the ARIMA model. While the model captured the general trend, the forecast underestimated the amplitude of fluctuations, likely due to the lack of a seasonal component in the basic ARIMA model.

Overall, this time series analysis of hospital revenue data explored the financial impact of patient readmissions over two years using ARIMA modeling. Through thorough data cleaning, stationarity testing, and model evaluation, the ARIMA(1,1,0) model was identified as the most suitable approach based on AIC optimization. The model provided reasonable short-term forecasts, capturing the general trend but slightly underestimating variability. Analysis of the decomposed series and spectral density confirmed the presence of seasonal patterns, suggesting that future models could benefit from incorporating seasonal or external factors.

E2, Final Forecast Model Visualization

The following plot shows the forecasted revenue compared to the actual test set values, with the training data included for context. The forecast line (dashed red) follows the trend of the test set but underestimates variability. This suggests the potential for improved performance with a SARIMA or exogenous factor model if external data (e.g., policy changes, staffing) were available.

```python
# Plot forecast vs actual with annotations
plt.figure(figsize=(12, 5))
plt.plot(train.index, train["Revenue"], label="Train", color="blue")
plt.plot(test.index, test["Revenue"], label="Actual", color="orange")
plt.plot(test.index, forecast, label="Forecast", color="red", linestyle='--')
plt.title("Forecast vs Actual Revenue (ARIMA Model)")
plt.xlabel("Date")
plt.ylabel("Revenue (in million $)")
plt.axvline(test.index[0], color='gray', linestyle=':', label='Forecast Start')
plt.legend()
plt.grid(True)
plt.show()
```



E3, Course of Action

Based on the results of this ARIMA analysis we can begin to draw several conclusions and identify a course of action.

One short-term recommendation would be for the hospital administration to monitor revenue trends closely, especially during months with high volatility. The current model can provide baseline forecasts, helping identify unexpected revenue drops early.

Another recommendation is to continue to enhance and improve the model. Future versions of the model should incorporate seasonality explicitly (e.g., SARIMA) and potentially include external features like policy changes or readmission rates to improve forecast accuracy.

Lastly, a strategic objective or insight is that the observed patterns of seasonality and trend suggest that administrative decisions and patient management strategies should be planned cyclically, matching expected dips and rises in revenue.

Overall, moving forward, it is recommended that hospital administrators use this model as a baseline forecasting tool while preparing for predictable fluctuations in revenue. Enhancing the model with seasonal adjustments or readmission-related indicators could further improve accuracy. This analysis supports the need for proactive, data-informed strategies to manage financial planning in the face of readmission penalties.

F, Part 6: Reporting

F1, Jupyter Notebook Presentation

A prepared report has been completed using Jupyter Notebook. The outputted notebook presentation has been saved into an HTML document format which can be found within the attached ‘D213-Task1_HTML.html’ file. A copy of it is also uploaded here.

G, Web Sources

No sources or segments of third-party code were used to acquire data or to support the report.

H, Acknowledge the Sources

I acknowledge that no segments of third-party sources were directly stated or copied from the web into this report.