MAY 10, 2024

PERFORMANCE ASSESSMENT

D213 TASK 1

ROBERT PATTON
WESTERN GOVERNORS' UNIVERSITY
Rpatt33@wgu.edu

Table of Contents

Part I: Research Question

A. Purpose of Data Analysis

1. Based on the previous two years of data, what should our revenue be in the first quarter of 2024?
2. The goals for this data analysis project are to use time series analysis, using data from the last two years, to help us forecast daily revenue predictions in the first 90 days of 2024.

Part II: Method Justification

B. Assumptions of Time Series Modeling

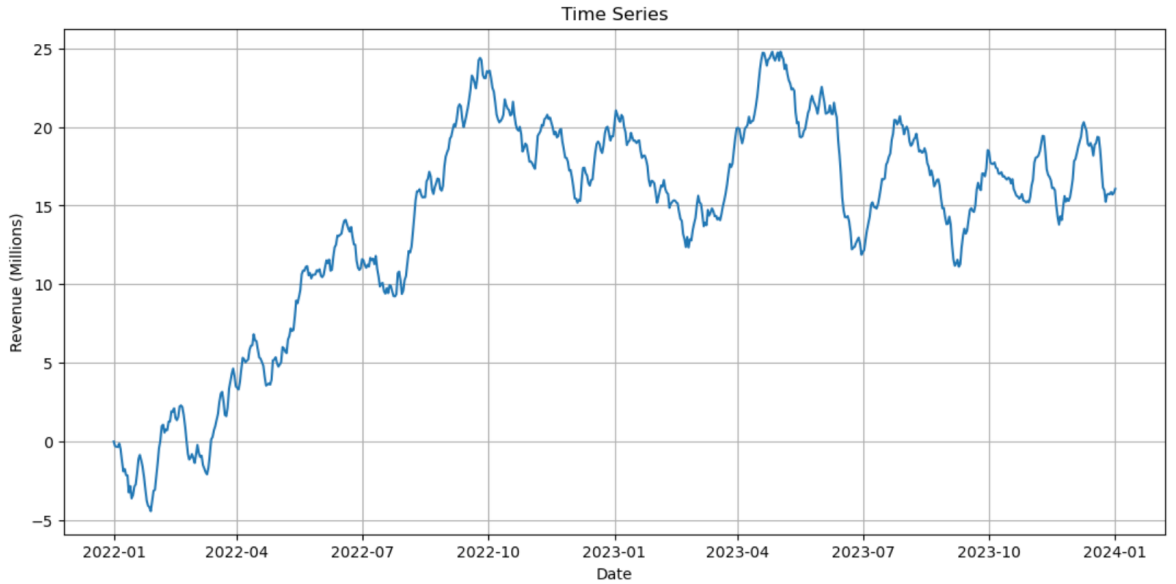
1. The assumptions of a time series model are:
    - **a.** The data should be stationary, meaning it is not showing any inconsistencies in mean, variance, or correlation. The data series should not show any trends. There should be no autocorrelation in the residuals of the data series and there should be no outliers present.
    - **b.** High autocorrelation evidence can help an analyst forecast future values, where data presenting with high autocorrelation is likely to show an analyst that previous values could be helpful in predicting current and future values (Schaffer et al. 2021).

Part III: Data Preparation

C. Summarize the Data Cleaning Process

1. Visualization of time series after converting it to date time format.

```
#Visualize the data
plt.figure(figsize=(13,6))
plt.plot(df.Revenue)
plt.title('Time Series')
plt.xlabel('Date')
plt.ylabel('Revenue (Millions)')
plt.grid(True)
plt.show()
```



2. The time step formatting for this analysis is by day. There are 731 days in the time series, or two years of data, and I converted the data from days to date in year/month/day format. The data does not present and gaps or missing days of data. The code for converting the data is seen below.

```
#Look at first 5 data entries
df.head(5)
```

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

```
#Create datetime index
df['Date']=(pd.date_range(start=datetime(2022,1,1),
                          periods=df.shape[0], freq='24H'))
#Set date as an index
df.set_index('Date', inplace=True)
df
```

Date	Revenue
2022-01-01	0.000000
2022-01-02	-0.292356
2022-01-03	-0.327772
2022-01-04	-0.339987
2022-01-05	-0.124888
...	...
2023-12-28	15.722056
2023-12-29	15.865822
2023-12-30	15.708988
2023-12-31	15.822867
2024-01-01	16.069429

731 rows × 1 columns

3. The data is obviously not stationary as seen in the visualization in C1. In order to convert the data to become stationary, I used the augmented dickey-fuller test to first check stationarity and then difference the data to transform the data. After transforming the data into a stationary form, I re-assessed the data for stationarity using the dickey-fuller test again. Based on the p-value of 0.19 during the initial ADF test, the data needs to be transformed. In the reassessed ADF, the p-value returns at 5.11320e-30, falling below the 0.05 threshold and therefore confirming stationarity. *See code below*

```
#Run Augmented Dickey-Fuller test to check for stationarity
df_adf=adfuller(df['Revenue'], autolag='AIC')
dfoutput=pd.Series(df_adf[0:4], index=['Test Statistic', 'p-value', '# of lags', 'No. Observations'])

for key, value in df_adf[4].items():
    dfoutput['Critical Value (%s) '%key]= value


print(dfoutput)
```

```
Test Statistic            -2.218319
p-value                    0.199664
# of lags                  1.000000
No. Observations         729.000000
Critical Value (1%)       -3.439352
Critical Value (5%)       -2.865513
Critical Value (10%)      -2.568886
dtype: float64
```

```
#Transform data into stationary form
df_trans=df.diff().dropna()
df_trans.head()
```

	Revenue
Date	
2022-01-02	-0.292356
2022-01-03	-0.035416
2022-01-04	-0.012215
2022-01-05	0.215100
2022-01-06	-0.366702

```
#Run Augmented Dickey-Fuller test to check for stationarity
dftrans_adf=adfuller(df_trans['Revenue'], autolag='AIC')
dfoutput=pd.Series(dftrans_adf[0:4], index=['Test Statistic', 'p-value', '# of lags', 'No. Observations'])

for key, value in dftrans_adf[4].items():
    dfoutput['Critical Value (%s) '%key]= value


print(dfoutput)
```

```
Test Statistic          -1.737477e+01
p-value                  5.113207e-30
# of lags                0.000000e+00
No. Observations         7.290000e+02
Critical Value (1%)     -3.439352e+00
Critical Value (5%)     -2.865513e+00
Critical Value (10%)    -2.568886e+00
dtype: float64
```

4. Steps used to prepare the data for analysis included:
    - **a.** Import all necessary python packages into Jupyter notebook.
    - **b.** Read in the CSV file using pandas.

c. Examine the data info, look at the shape of the data, and get data statistics.
d. Convert the data from a day to a datetime series using panadas.
e. Generate a visualization of the time series for revenue over two years.
f. Clean the data by checking for missing/null values and checking for any duplicate values.
g. Train/test/split the non-stationary data into an 80/20 split.
h. Check for stationarity in the data using the Augmented Dickey-Fuller test (ADF).
i. Difference the data to make it stationary.
j. Re-assess the data with the ADF test to confirm stationarity.
k. Perform train/test/split on the stationary data into an 80/20 split.
l. Export cleaned CSV file and train and test splits.

5. Cleaned copies of data and the test/train data are uploaded with the submission.

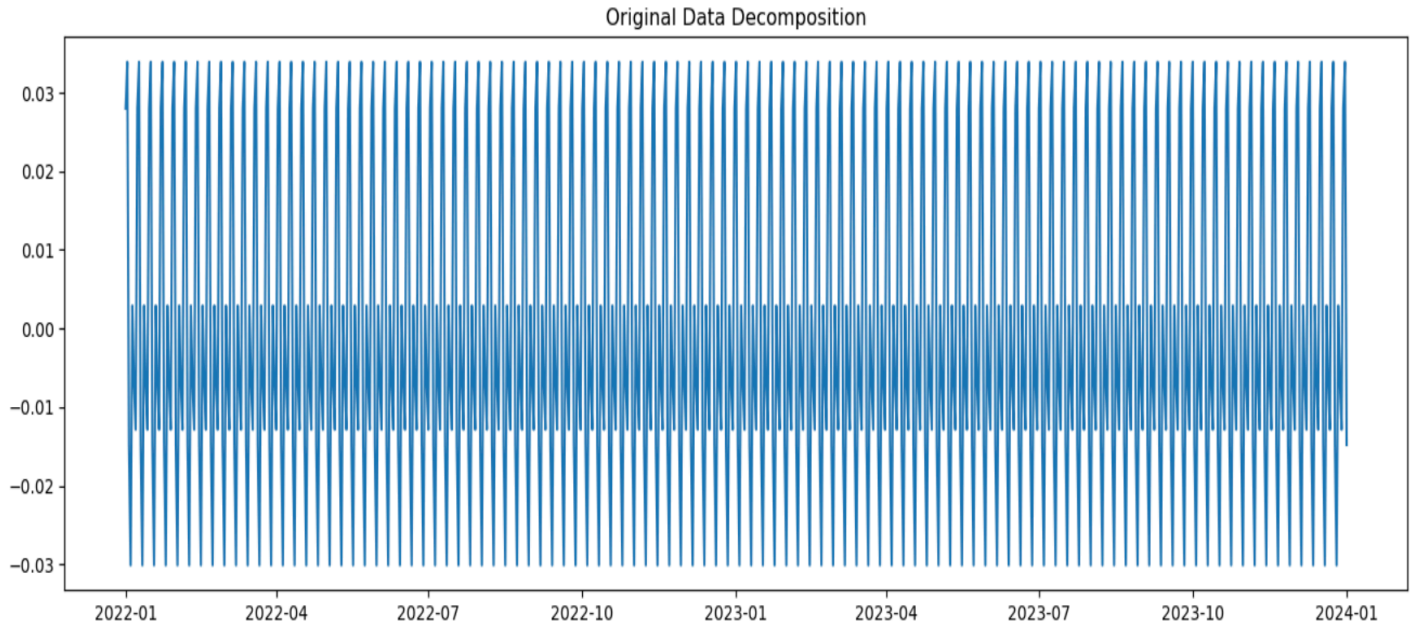
Part IV: Model Identification and Analysis

D. Time Series Analysis

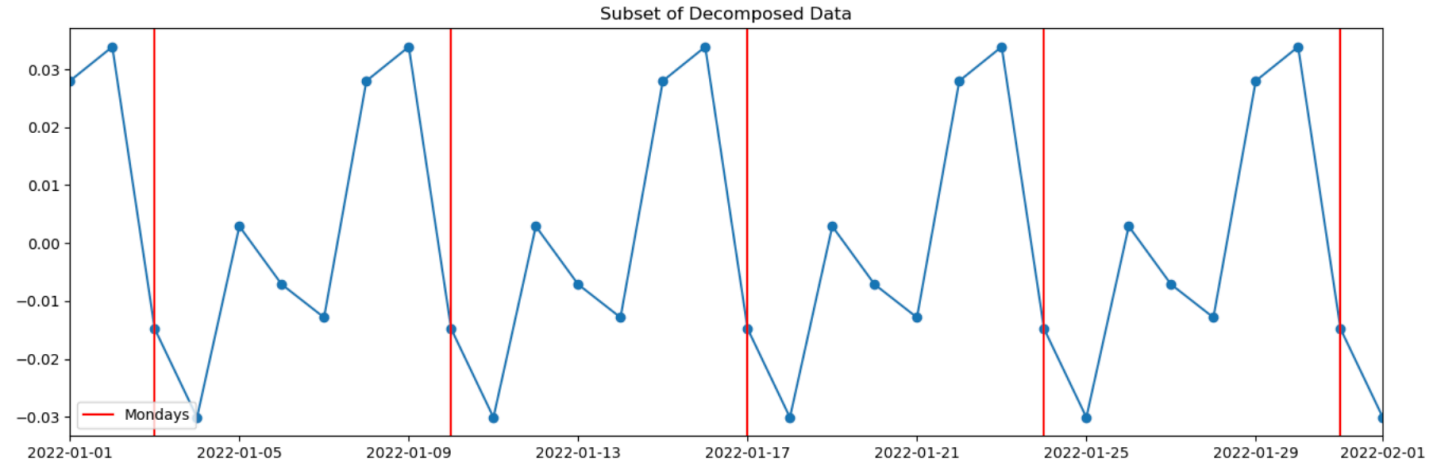
D1-a. Seasonality refers to a repetitive pattern observed over time in the data. By creating a chronological visualization we can see where seasonality takes place. I first decomposed the original dataset, using the stats model seasonal_decompose function, and then created the visual below looking at seasonality on the revenue data.

```
#Look for prescence of a seasonal component by decomposing data and plotting
decomposed_data= seasonal_decompose(df['Revenue'])
plt.figure(figsize = [16,5])
plt.title('Original Data Decomposition')
plt.plot(decomposed_data.seasonal)
```

```
[<matplotlib.lines.Line2D at 0x3cb9b9210>]
```



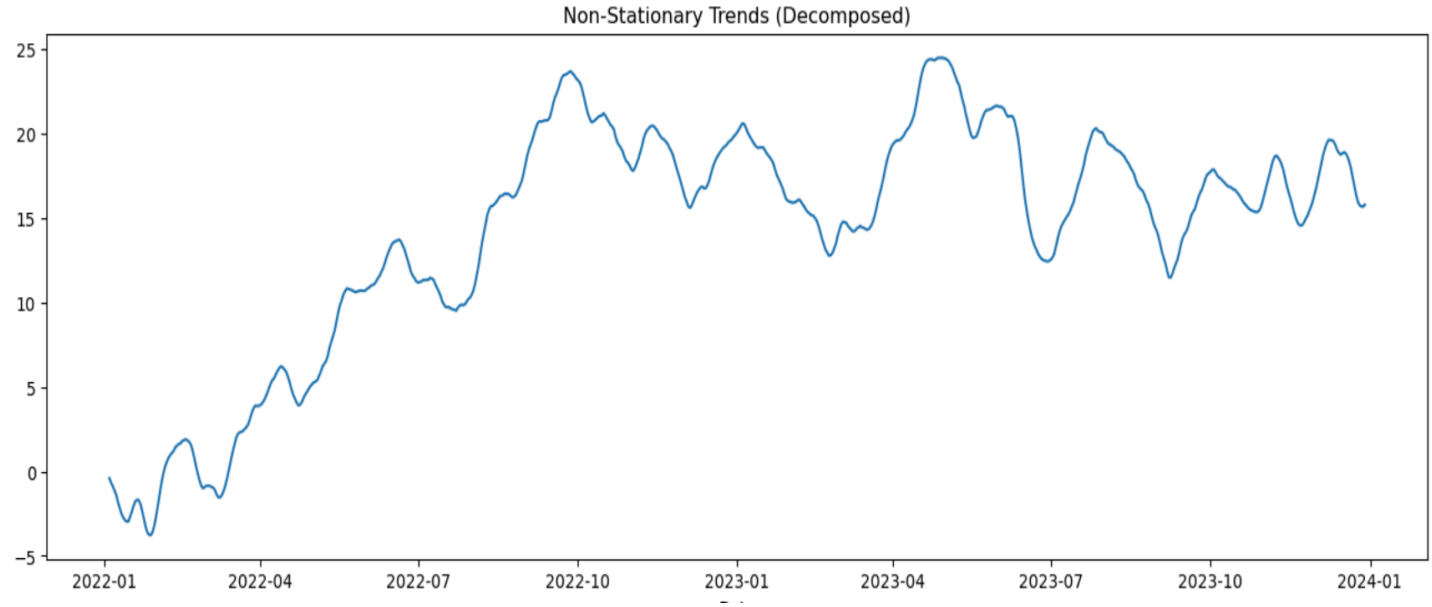
```
#View a subset of above graph to identify seasonality
plt.figure(figsize = [16,5])
# Plot seasonal component of the data
plt.plot(decomposed_data.seasonal, marker='o')
plt.title('Subset of Decomposed Data')
plt.xlim(pd.to_datetime('2022-01-01'), pd.to_datetime('2022-02-01'))
#Red lines = Mondays
plt.axvline(x=pd.to_datetime('2022-01-03'), color='red', label='Mondays')
plt.axvline(x=pd.to_datetime('2022-01-10'), color='red')
plt.axvline(x=pd.to_datetime('2022-01-17'), color='red')
plt.axvline(x=pd.to_datetime('2022-01-24'), color='red')
plt.axvline(x=pd.to_datetime('2022-01-31'), color='red')
plt.legend(loc="best");
```



In addition, I went further and examined a subset of the decomposed data to confirm seasonality. We can see in the subset above that from 1-1-2022 to 2-1-2022, there are high and low points within the data. The red lines represent the Mondays throughout the month. Tuesdays, one point to the right are the lows in the month, with Sundays being the high points. The consistency seen in this subset indicates seasonality.

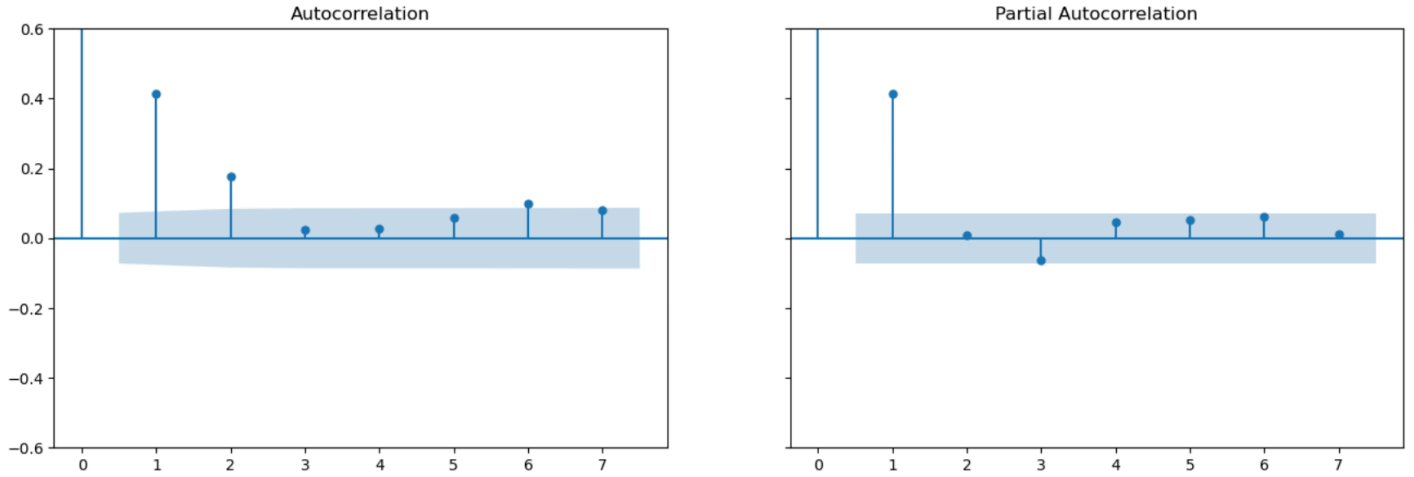
D1-b. There appears to be an upward trend in the original dataset. The visualization is shown below.

```
#Look for trends in data
plt.figure(figsize = [16,5])
plt.title('Non-Stationary Trends (Decomposed)')
plt.xlabel('Date')
plt.plot(decomposed_data.trend);
```



D1-c. Autocorrelation functions:

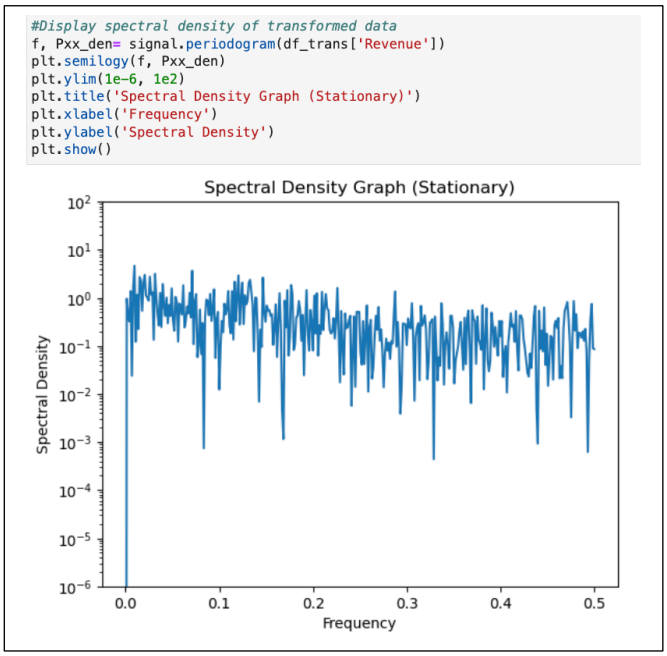
```
#Create an acf and pacf plot
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=[16,5], sharey=True)
plot_acf(df_trans, lags=7, ax=ax1)
plot_pacf(df_trans, lags=7, ax=ax2)
plt.ylim(-0.6, 0.6);
```



Using an ACF and PACF test, an analyst can determine whether the data is an autoregressive or moving average model. The shaded areas in the above image represent insignificant results, while the datapoints above the shaded area can be considered significant. The plot on the left (ACF) tails off after two and the plot on the right (PACF) tails off after one (Monigatti 2022). An ACF that tails off like the one above and a PACF that cuts off after lag p represents and AR model. Thus, based on our graphs above, we have an AR(1) model for this data.

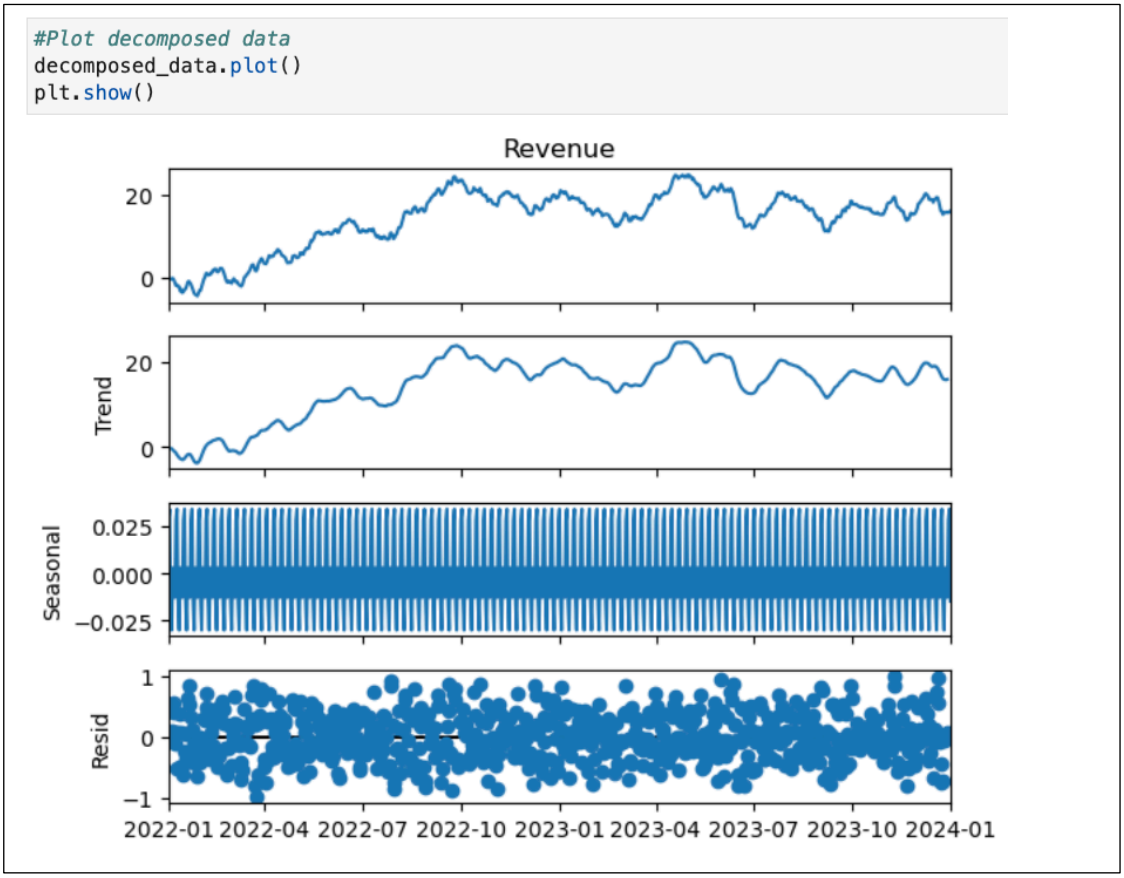
D1-d. Below is a graph of the power spectral density of the transformed data. It shows the number of observations as a frequency before the pattern repeats.

```
#Display spectral density of transformed data
f, Pxx_den= signal.periodogram(df_trans['Revenue'])
plt.semilogy(f, Pxx_den)
plt.ylim(1e-6, 1e2)
plt.title('Spectral Density Graph (Stationary)')
plt.xlabel('Frequency')
plt.ylabel('Spectral Density')
plt.show()
```



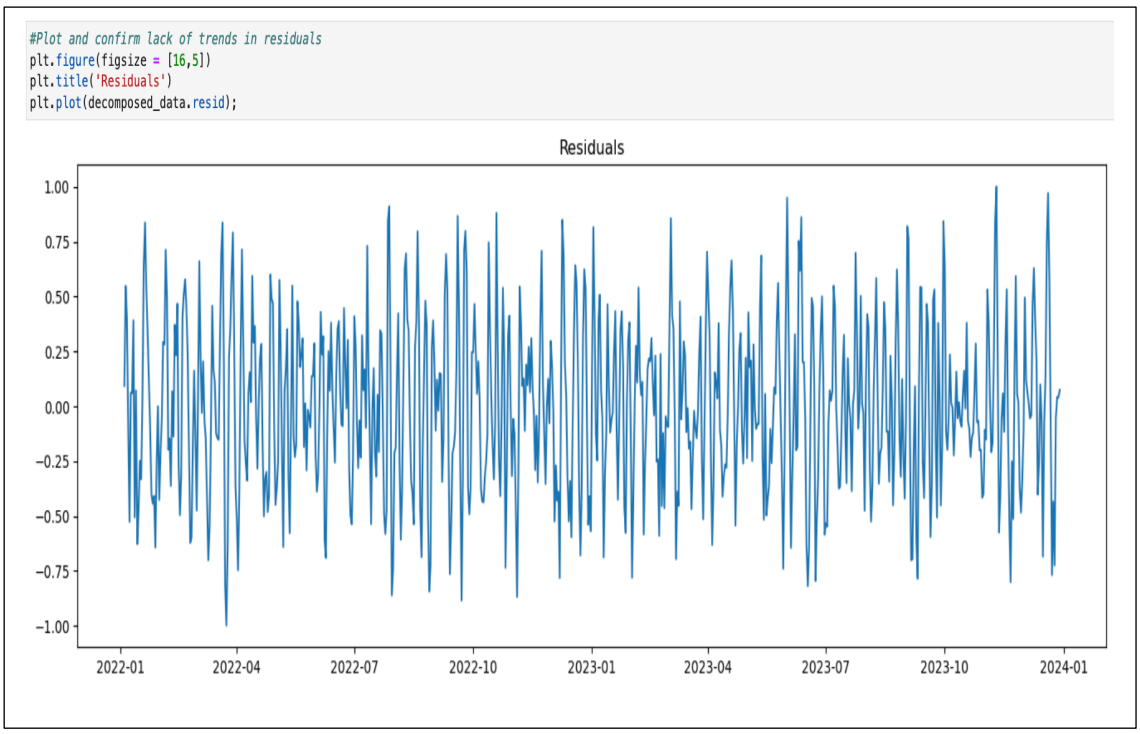
D1-e. The decomposed time series is seen below. It is split into three components, trend, seasonality, and residuals. It is another way to see if the data possesses or lacks these features.

```
#Plot decomposed data
decomposed_data.plot()
plt.show()
```



D1-f. This part seems to be repetitive of the above image, but the rubric wants me to show a lack of trends in the decomposed residuals of the data. This is seen below.

```
#Plot and confirm lack of trends in residuals
plt.figure(figsize = [16,5])
plt.title('Residuals')
plt.plot(decomposed_data.resid);
```



D2. After determining that we have an AR(1) model from running the ACF and PACF plots above, I decided to still run an auto ARIMA to check what the best model order should be. We can use the non-stationary data for this because the "I" in ARIMA is the procedure part of running the model, where it makes a stationary time series out of the non-stationary data, which would explain why it takes so long to run that line of code. Based on the results from the auto ARIMA, we can see that it suggests a model with a p,d,q of 1,1,0. In addition, it recommends a seasonal order of 0,1,0,90. The seasonal order of 90 means 90 days because each quarter is represented in 3-month intervals.

```
#Run auto ARIMA
model=pm.auto_arima(df,
                    seasonal=True, m=90,
                    d=1, D=1,
                    start_p=1, start_q=1,
                    max_p=2, max_q=2,
                    max_P=2, max_Q=2,
                    trace=True,
                    error_action='ignore',
                    supress_warnings=True)
print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(1,1,1)[90]             : AIC=inf, Time=192.03 sec
 ARIMA(0,1,0)(0,1,0)[90]             : AIC=1344.172, Time=4.78 sec
 ARIMA(1,1,0)(1,1,0)[90]             : AIC=inf, Time=16.97 sec
 ARIMA(0,1,1)(0,1,1)[90]             : AIC=inf, Time=115.90 sec
 ARIMA(0,1,0)(1,1,0)[90]             : AIC=inf, Time=10.83 sec
 ARIMA(0,1,0)(0,1,1)[90]             : AIC=inf, Time=53.69 sec
 ARIMA(0,1,0)(1,1,1)[90]             : AIC=inf, Time=117.75 sec
 ARIMA(1,1,0)(0,1,0)[90]             : AIC=1228.526, Time=5.16 sec
 ARIMA(1,1,0)(0,1,1)[90]             : AIC=inf, Time=114.99 sec
 ARIMA(1,1,0)(1,1,1)[90]             : AIC=inf, Time=107.23 sec
 ARIMA(2,1,0)(0,1,0)[90]             : AIC=1230.491, Time=5.18 sec
 ARIMA(1,1,1)(0,1,0)[90]             : AIC=1230.496, Time=5.57 sec
 ARIMA(0,1,1)(0,1,0)[90]             : AIC=1247.950, Time=4.91 sec
 ARIMA(2,1,1)(0,1,0)[90]             : AIC=1231.394, Time=11.37 sec
 ARIMA(1,1,0)(0,1,0)[90] intercept   : AIC=1230.497, Time=6.84 sec

Best model:  ARIMA(1,1,0)(0,1,0)[90]
Total fit time: 773.286 seconds
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                                 y   No. Observations:                  731
Model:             SARIMAX(1, 1, 0)x(0, 1, 0, 90)   Log Likelihood                -612.263
Date:                            Wed, 08 May 2024   AIC                           1228.526
Time:                                    08:45:40   BIC                           1237.449
Sample:                                01-01-2022   HQIC                          1231.990
                                     - 01-01-2024
Covariance Type:                              opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1          0.4095      0.037     11.080      0.000       0.337       0.482
sigma2         0.3966      0.023     17.137      0.000       0.351       0.442
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 0.66
Prob(Q):                              0.95   Prob(JB):                         0.72
Heteroskedasticity (H):               1.15   Skew:                            -0.00
Prob(H) (two-sided):                  0.32   Kurtosis:                         2.84
===================================================================================

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

After running the auto ARIMA, we can then run a SARIMAX using the derived recommendations from the auto ARIMA. We run the SARIMAX with the training data set on the revenue data. In determing if we have run a good model, we can look at the Ljung-box metric. An Ljung-box value above 0.05 indicates a good model.

SARIMAX Results

Dep. Variable:	Revenue	**No. Observations:**	585
Model:	SARIMAX(1, 1, 0)x(1, 1, 0, 90)	**Log Likelihood**	-409.291
Date:	Wed, 08 May 2024	**AIC**	824.582
Time:	10:31:26	**BIC**	837.190
Sample:	01-01-2022	**HQIC**	829.532
	- 08-08-2023		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.3818	0.040	9.642	0.000	0.304	0.459
ar.S.L90	-0.4655	0.043	-10.942	0.000	-0.549	-0.382
sigma2	0.2936	0.020	14.400	0.000	0.254	0.334

Ljung-Box (L1) (Q):	0.16	**Jarque-Bera (JB):**	2.10
Prob(Q):	0.69	**Prob(JB):**	0.35
Heteroskedasticity (H):	1.06	**Skew:**	-0.05
Prob(H) (two-sided):	0.73	**Kurtosis:**	2.69

Warnings:

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

```
#Run ARIMA model on best fit model
model=SARIMAX(X_train['Revenue'], order=(1,1,0), seasonal_order=(1,1,0,90))
results=model.fit()
results.summary()
```

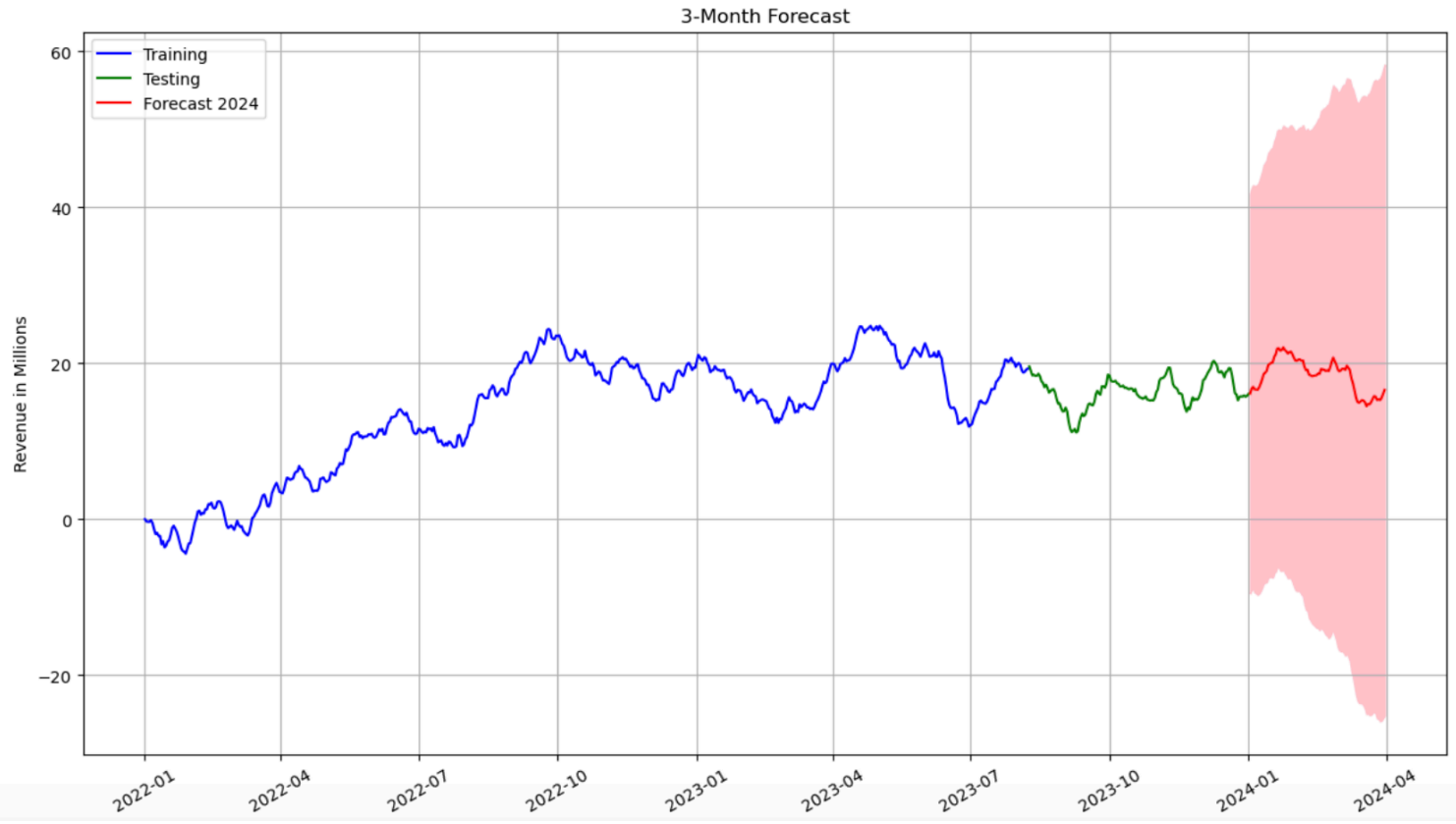
D3. The rubric asks to generate revenue predictions based on the ARIMA model results above, showing training, test, and forecasted data. The predictions and visualization are show below. The forecast projections are set for 90 days into year 2024.

```
#Generate forecast
diff_forecast=results.get_forecast(steps=30)
mean_forecast= diff_forecast.predicted_mean
confidence_intervals=diff_forecast.conf_int()
lower_limits=confidence_intervals.loc[:,'lower Revenue']
upper_limits=confidence_intervals.loc[:, 'upper Revenue']
#Generate prediction
prediction=results.get_prediction(start=len(df), end=len(df)+89)
mean_prediction=prediction.predicted_mean
confidence_intervals=prediction.conf_int()
lower_limits=confidence_intervals.loc[:,'lower Revenue']
upper_limits=confidence_intervals.loc[:,'upper Revenue']
#Display tail of mean prediction
mean_prediction.tail(30)
```

```
2024-03-02    19.077558
2024-03-03    19.293721
2024-03-04    19.296007
2024-03-05    19.152047
2024-03-06    19.688508
2024-03-07    19.407824
2024-03-08    19.118870
2024-03-09    18.071761
2024-03-10    17.463475
2024-03-11    16.584951
2024-03-12    15.657092
2024-03-13    15.057474
2024-03-14    14.927157
2024-03-15    15.096296
2024-03-16    15.270609
2024-03-17    15.209481
2024-03-18    14.975928
2024-03-19    14.490436
2024-03-20    14.778034
2024-03-21    14.738781
2024-03-22    14.978323
2024-03-23    15.401493
2024-03-24    15.767598
2024-03-25    15.724819
2024-03-26    15.286470
2024-03-27    15.381765
2024-03-28    15.282555
2024-03-29    15.576415
2024-03-30    16.074651
2024-03-31    16.573085
Freq: D, Name: predicted_mean, dtype: float64
```

```
#Create graph of test, train, and forecast
plt.figure(figsize=(15,8))
plt.plot(X_train, label='Training', color='b')
plt.plot(X_test, label='Testing', color='g')
plt.plot(mean_prediction, label='Forecast 2024', color='r')
plt.fill_between(lower_limits.index, lower_limits, upper_limits, color='pink')
plt.title('3-Month Forecast')
plt.xlabel('Date')
plt.ylabel('Revenue in Millions')
plt.grid()
plt.xticks(rotation=30, fontsize=10)
plt.legend(loc='upper left')
```

```
<matplotlib.legend.Legend at 0x3df87f150>
```

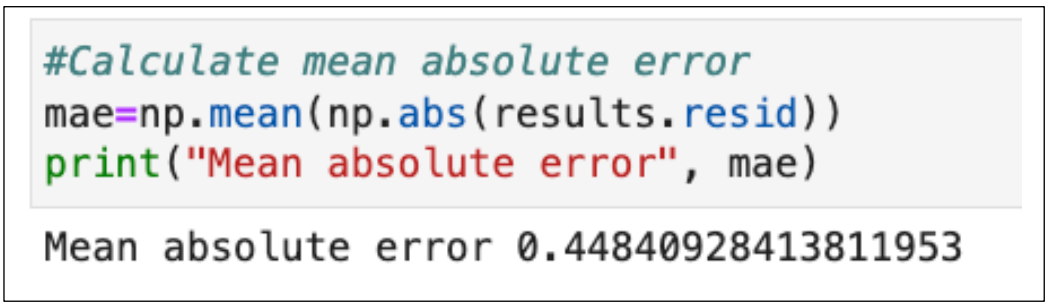


D4/5. All of the codes, calculations, and model outputs have been recorded above in D1-D3. I presented screen shots of the codes, calculations, and time series model. The Jupyter notebook will be included in the task one submission.

Part V: Data Summary and Implications

E. Summarize Findings and Assumptions

1. As discussed in D-1c, we used the ACF and PACF graphs to help determine if we were dealing with an AR or MA model. Based on those results, we determined that we had an AR(1) model. We then used an auto ARIMA to look for the best model based on the lowest AIC value. The code for the ACF and PACF graphs can be seen in part D1-c. The code for the ARIMA models can be seen in part D2 above.
2. The prediction interval for the forecast of future revenue is at 1-day intervals. The time series data that we were given for this task equated to two years of data, or 731 days. We converted the data from a day format to a datetime format. The SARIMAX model runs through the data and identifies correlation and seasonality to predict the revenue on a daily basis for the first quarter of 2024.
3. I chose to run a forecast for 90 days because I felt that looking at forecasted revenue in quarters would provide sufficient insight at a more accurate interval. Time series models can only forecast up to a year and it seems logical that forecasting that far out could present some accuracy issues.
4. Evaluating the model and error metrics is required in the rubric. This step allows to determine how accurate our model is in predicting future revenue. The first method of ensuring I would end up with an accurate model began with running an auto ARIMA and selecting the model with the lowest AIC score. The auto ARIMA also gave me the best seasonal order to use as well. After inputting those outputs into my SARMIAX model, I also looked the Ljung-box score to determine quality of the model, the score should be over 0.05 and came in at 0.16. I also ran a mean absolute error metric that came out to 0.44, indicating that error rates are low based on the number of observations in the data. A mean absolute error closer to 0 indicate a more accurate model with low chance of error. *See section D1-c and D2 for ACF/PACF graphs and ARIMA models*

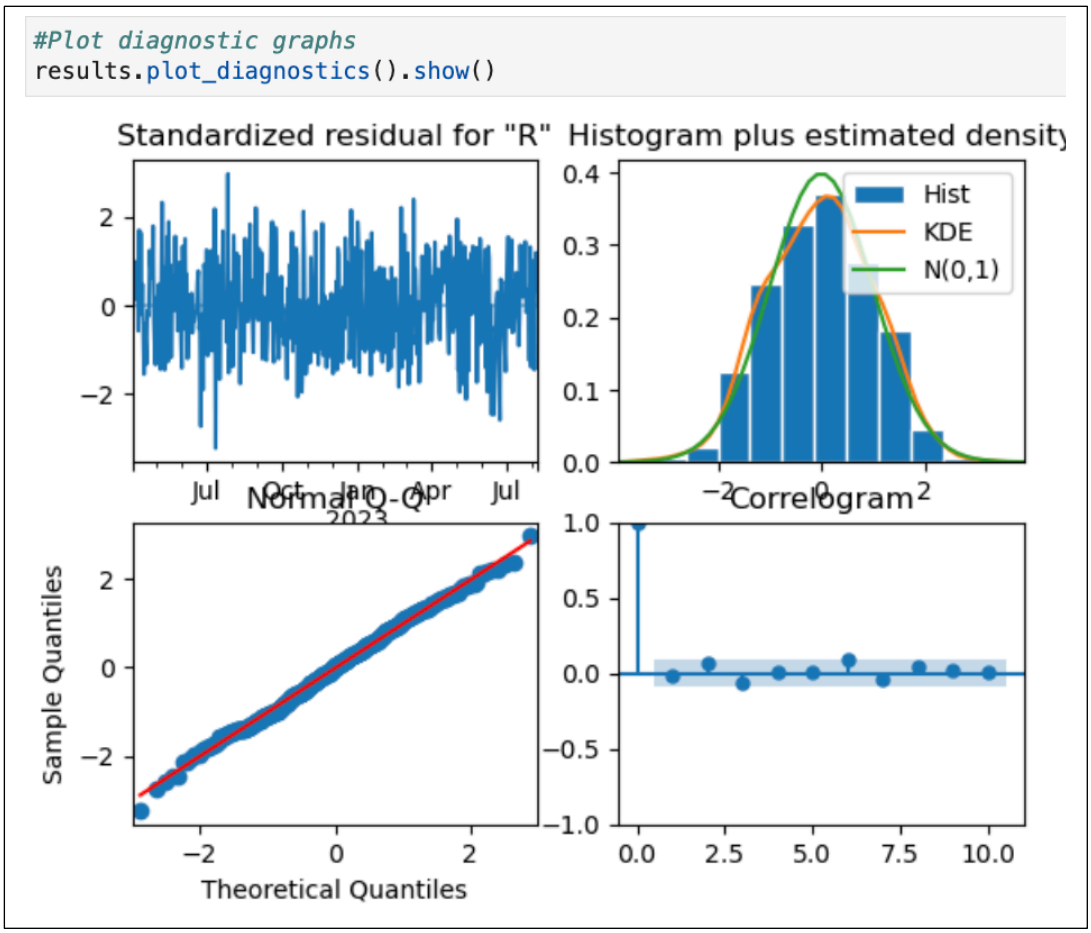


```
#Calculate mean absolute error
mae=np.mean(np.abs(results.resid))
print("Mean absolute error", mae)
```

```
Mean absolute error 0.44840928413811953
```

Furthermore, I also used the plot_diagnostics() function to produce visualizations about the model's performance. These can be seen below. The top left plot shows the standardized residuals, there should be no sign of obvious trends, and this plot does not show obvious patterns. The top right plot shows a histogram with a KDE metric, where the closer a KDE is to the normal distribution the better. The q-q plot in the bottom left should ideally have all datapoints on the red line, in this case they are, and we have a good model. For the correlogram in the bottom right, all correlations after some lag p should be within the shaded area and are to be considered insignificant.

```
#Plot diagnostic graphs
results.plot_diagnostics().show()
```



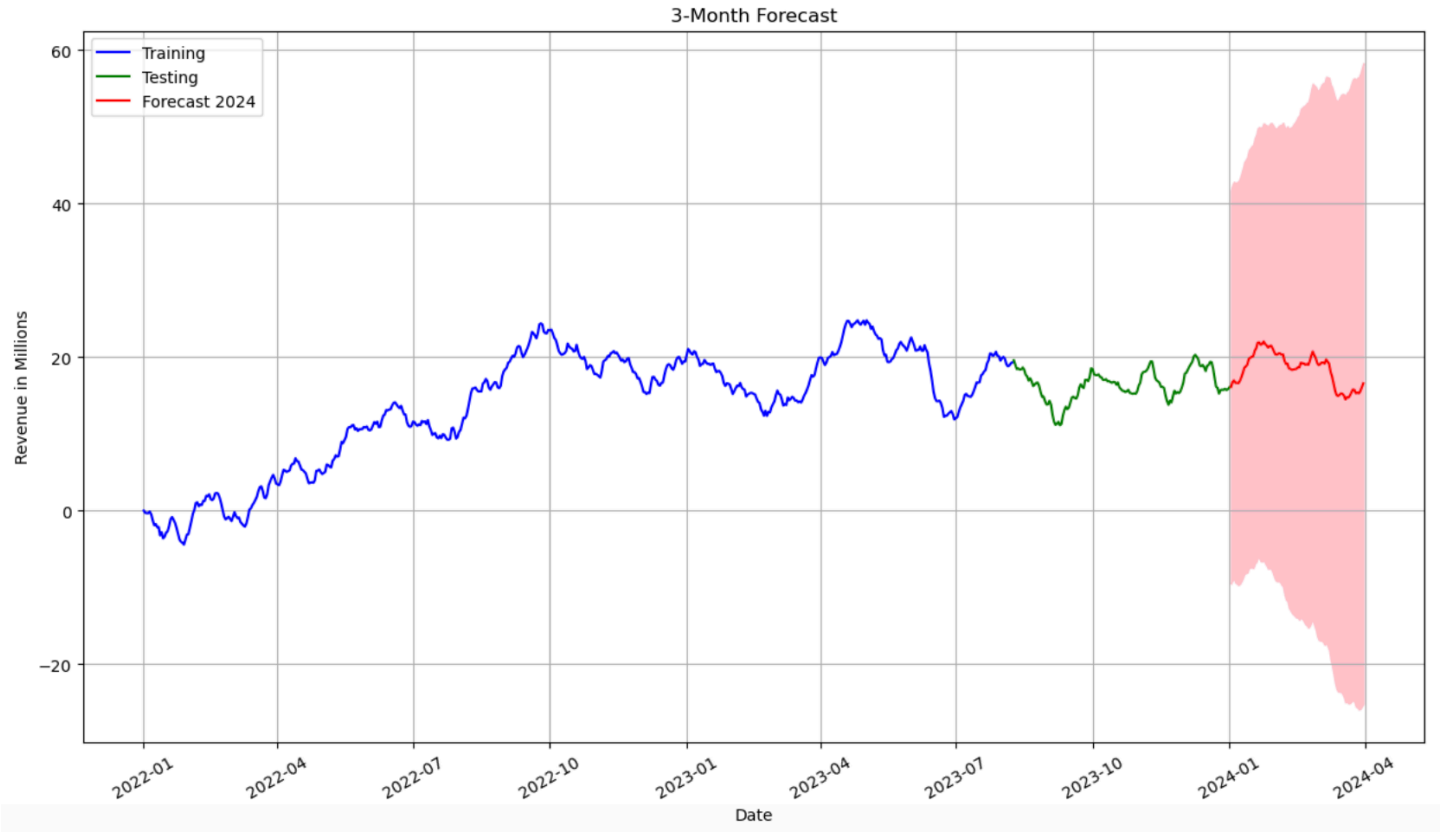
E2. Below is the final visualization as required by the rubric. It shows our training set compared to our test set which is then compared to our forecasted predictions. I have also included the code for generating the forecast/predictions.

```
#Generate forecast
diff_forecast=results.get_forecast(steps=30)
mean_forecast= diff_forecast.predicted_mean
confidence_intervals=diff_forecast.conf_int()
lower_limits=confidence_intervals.loc[:,'lower Revenue']
upper_limits=confidence_intervals.loc[:, 'upper Revenue']
#Generate prediction
prediction=results.get_prediction(start=len(df), end=len(df)+89)
mean_prediction=prediction.predicted_mean
confidence_intervals=prediction.conf_int()
lower_limits=confidence_intervals.loc[:,'lower Revenue']
upper_limits=confidence_intervals.loc[:,'upper Revenue']
#Display tail of mean prediction
mean_prediction.tail(30)
```

```
2024-03-02    19.077558
2024-03-03    19.293721
2024-03-04    19.296007
2024-03-05    19.152047
2024-03-06    19.688508
2024-03-07    19.407824
2024-03-08    19.118870
2024-03-09    18.071761
2024-03-10    17.463475
2024-03-11    16.584951
2024-03-12    15.657092
2024-03-13    15.057474
2024-03-14    14.927157
2024-03-15    15.096296
2024-03-16    15.270609
2024-03-17    15.209481
2024-03-18    14.975928
2024-03-19    14.490436
2024-03-20    14.778034
2024-03-21    14.738781
2024-03-22    14.978323
2024-03-23    15.401493
2024-03-24    15.767598
2024-03-25    15.724819
2024-03-26    15.286470
2024-03-27    15.381765
2024-03-28    15.282555
2024-03-29    15.576415
2024-03-30    16.074651
2024-03-31    16.573085
Freq: D, Name: predicted_mean, dtype: float64
```

```
#Create graph of test, train, and forecast
plt.figure(figsize=(15,8))
plt.plot(X_train, label='Training', color='b')
plt.plot(X_test, label='Testing', color='g')
plt.plot(mean_prediction, label='Forecast 2024', color='r')
plt.fill_between(lower_limits.index, lower_limits, upper_limits, color='pink')
plt.title('3-Month Forecast')
plt.xlabel('Date')
plt.ylabel('Revenue in Millions')
plt.grid()
plt.xticks(rotation=30, fontsize=10)
plt.legend(loc='upper left')
```

```
<matplotlib.legend.Legend at 0x3df87f150>
```



E3. It looks like future revenue forecasts stay consistent compared to July of 2023. I would recommend that executives still continue to focus on reducing readmission rates as an effort to also prevent fines from Medicare and Medicaid services. ARIMA models can run forecasts as far as one year out, however. It would be my recommendations to not forecast farther than two quarters, or 180 days. The shorter the forecast the more accurate the results for predictions. Shorter forecasts are also easier to perform, and the overall focus should be on reducing readmission rates to increase revenue by preventing fines. Reducing readmission rates also improves patient satisfaction and the way the community looks at the hospital.

Part VI: Reporting

F. The Jupyter notebook has been included with the task 1 submission, along with the split data and cleaned data csv files.

G. Sources

1. Schaffer, A. L., Dobbins, T. A., & Pearson, S.-A. (2021, March 22). *Interrupted time series analysis using autoregressive integrated moving average (ARIMA) models: A guide for evaluating large-scale health interventions - BMC Medical Research methodology*. BioMed Central. https://bmcmedresmethodol.biomedcentral.com/articles/10.1186/s12874-021-01235-8
2. Monigatti, L. (2022, March 15). *Time Series: Interpreting ACF and PACF*. Kaggle. https://www.kaggle.com/code/iamleonie/time-series-interpreting-acf-and-pacf/notebook