# TIME SERIES MODELING

**Part I: Research Question**

A. Describe the purpose of this data analysis by doing the following:

1. Summarize **one** research question that is relevant to a real-world organizational situation captured in the selected data set and that you will answer using time series modeling techniques.

What are the meaningful patterns we can find in the provided time series data? Once we find these patterns, we can extrapolate them to find predictions and the relationship with churn rate.

2. Define the objectives or goals of the data analysis. Ensure that your objectives or goals are reasonable within the scope of the scenario and are represented in the available data.

The objective is to identify patterns in revenue that will offer us a better understanding of the churn data. This understanding will enable us to correlate churn over time, and make data-backed recommendations to company stakeholders to ultimately reduce churn.

**Part II: Method Justification**

B. Summarize the assumptions of a time series model including stationarity and autocorrelated data.

One common assumption is that the data is stationary. This means that the mean, variance and autocorrelation structure don't change over time[1].

Autocorrelation in the data means that there is a degree of similarity between a given time series and the time-lagged version[2].

**Part III: Data Preparation**

C. Summarize the data cleaning process by doing the following:

1. Provide a line graph visualizing the realization of the time series.

```
#Data Visualization
def plot_ts(df, x, y, title = '', xlabel = 'Day', ylabel = 'Rev', dpi=100):
    plt.figure(figsize = (10,10), dpi=dpi)
    plt.plot(x, y, color = 'tab:blue')
    plt.gca().set(title = title, xlabel = xlabel, ylabel = ylabel)
    plt.show()

plot_ts(time_series_df, x = time_series_df['Day'], y =
time_series_df['Revenue'], title = '2 Years of Revenue (MM)')
```

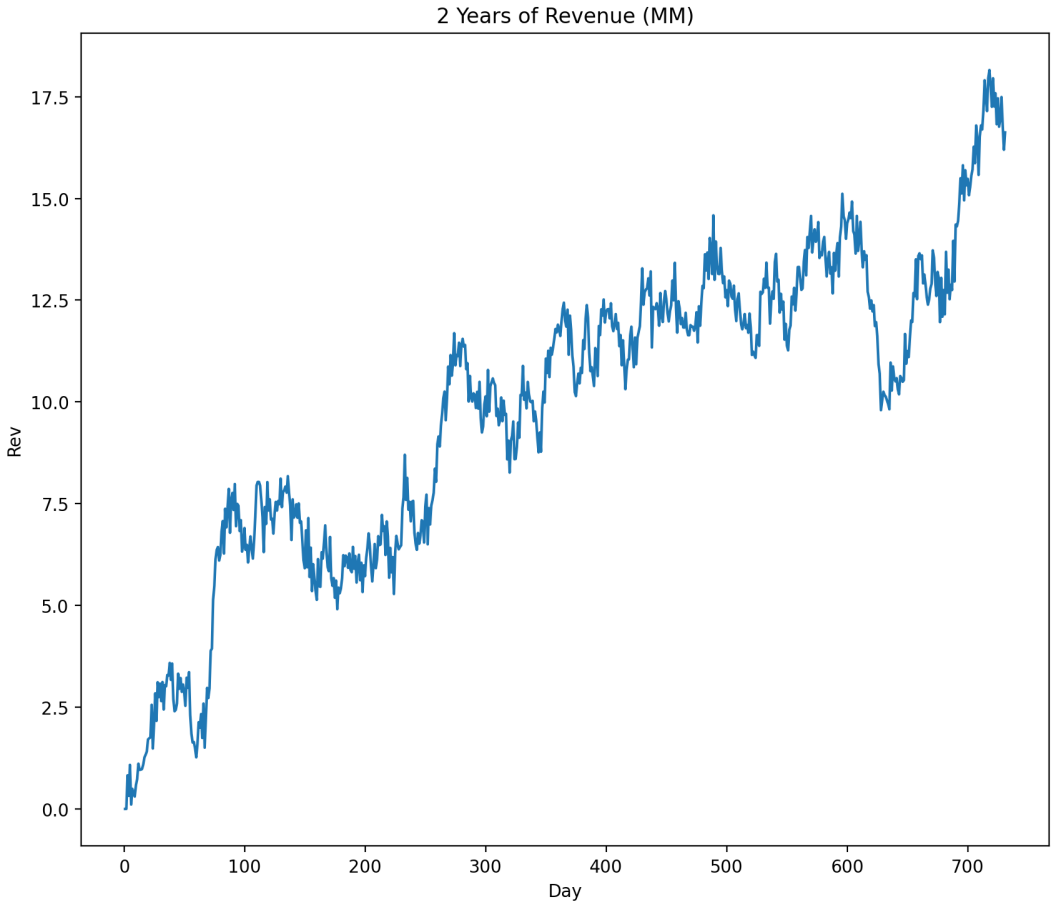



**Figure 1: Two Years of Revenue in MM**

2. Describe the time step formatting of the realization, including any gaps in measurement and the length of the sequence.

```
print(time_series_df.info)
print(time_series_df.dtypes)
```

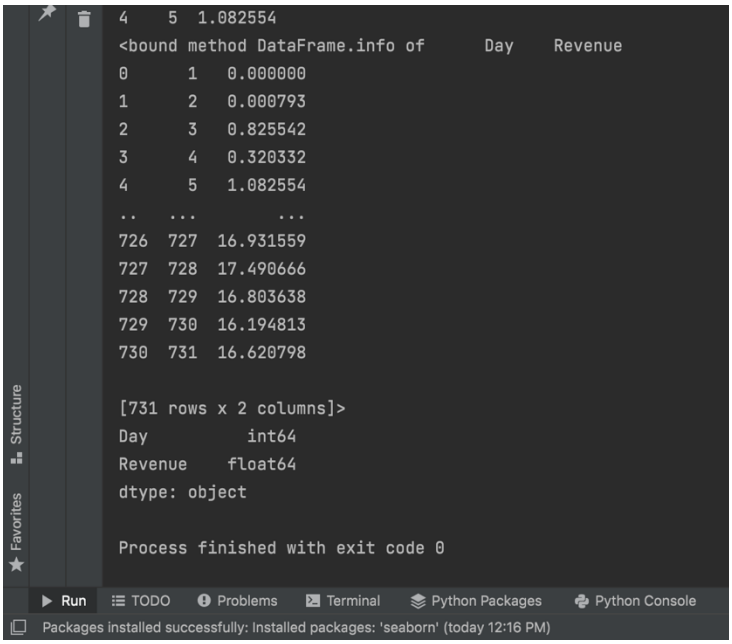



**Figure 2: Results for info and dtypes**

```
#Checking if any value is missing or if we have NAs
print(time_series_df.isnull().values.any())
print(time_series_df.isna().values.any())
```

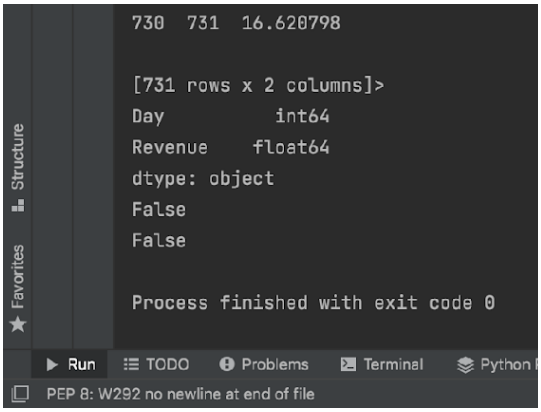



**Figure 3: Results for Missing Data (False) and NAs (False)**

Notice that there are no missing data points, or NAs in the time step formatting. The next step is to check if there are any duplicates in the “Day” column.

```
#Checking for duplicates in "Day"
print(time_series_df.Day.duplicated().sum())
```

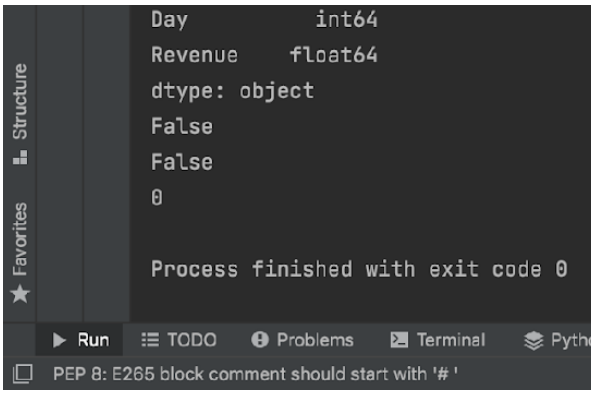



**Figure 4: Results for Duplicate Values on the "Day" column**

There are no duplicates. Analyzing the basic stats of the data will validate this.

```
#Basic Stats of the data
print(time_series_df.describe())
```

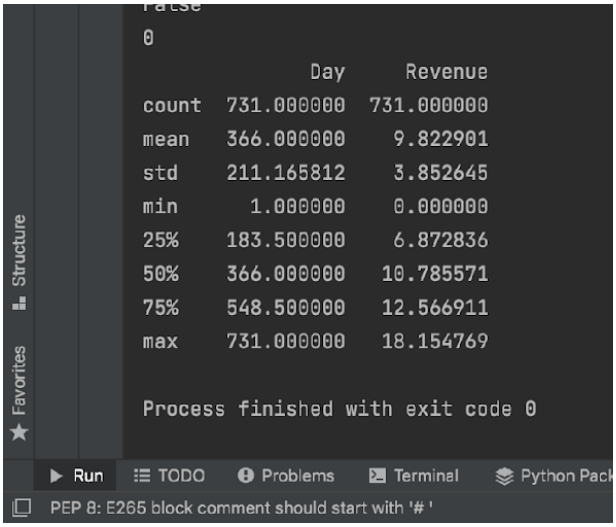



**Figure 5: Basic Stats of the provided data**

It is possible to check for outliers by analyzing the percentiles. If they keep increasing, like the examples above, there are no outliers.

3. Evaluate the stationarity of the time series.

The most famous method of statistical analysis to determine if a time series is stationary or not is the Augmented Dickey-Fuller test (or unit root test). This test will analyze the p-value. If p-value is <= 0.05 the null hypothesis is rejected, since the data doesn't have a unit root and is stationary[2].

```
#Augumented Dickey-Fuller
df =
```

```
pd.read_csv('/Users/bia/Desktop/Datasets_for_D213/teleco_time_series.csv',
header=0, index_col=0, squeeze=True)
X = df.values
result = adfuller(X)
print('ADF Statistic: ', result[0])
print('p-value: ', result[1])
print('Critical Values')
for key , value in result[4].items():
    print('\t%s: %.3f' % (key,value))
```

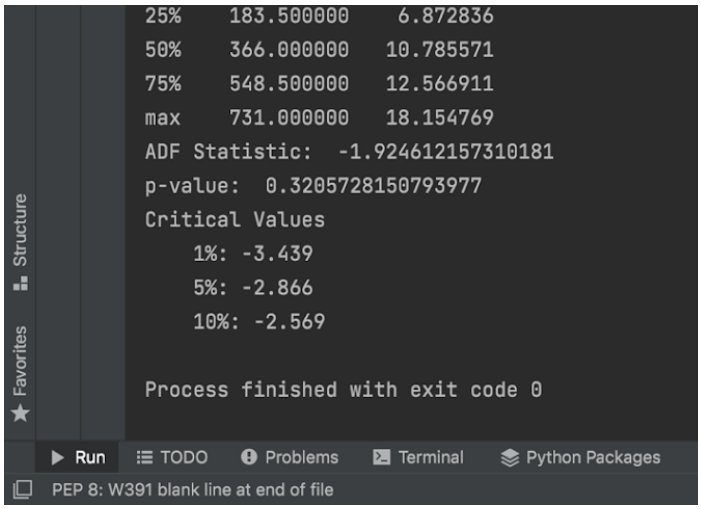



**Figure 6: Augment Dickey-Fueller (ADF) Analysis**

The ADF Statistical value is a high negative number, meaning the null hypothesis can't be rejected, and the time series is non-stationary.

The p-value for this data is 0.32. Since it is greater than 0.05 the null hypothesis cannot be rejected.

The critical value shows the ADF value at -1.92 is higher than -3.439 at 1%. This means the null hypothesis cannot be rejected.

Therefore, the conclusion is that this time series is **non-stationary**.

4. Explain the steps used to prepare the data for analysis, including the training and test set split.

1-) Read the data using Pandas' read_csv command
2-)Use *info*, *dtypes*, and *describe* to better understand the data
3-) Check for missing data/NAs
4-) Check for duplicate values
5-) Split the data in two: *training* and *test* data
6-) Extract the cleaned dataset

The provided data is a two-year time series (730 points), so when data is split, there are 365 points in each of the *training* and *test* data segments.

```
#Splitting the data in test and training
time_series_df['Day'] = time_series_df.index
time_series_df_train = time_series_df.iloc[:len(time_series_df) - 365]

#Dataset for Train
time_series_df_train['time_series_df_train'] =
time_series_df_train['Revenue']
del time_series_df_train['Day']
del time_series_df_train['Revenue']

#Dataset for Test
time_series_df_test = time_series_df.iloc[len(time_series_df) - 365:]

time_series_df_test['time_series_df_test'] = time_series_df_test['Revenue']
del time_series_df_test['Day']
del time_series_df_test['Revenue']
```

5. Provide a copy of the cleaned dataset. The copy will be uploaded with the code and this document. The name of the clean dataset is: "**time_series_clean.xls**"

```
#Extract the Clean time series dataset
time_series_df.to_csv('/Users/bia/Desktop/Datasets_for_D213/time_series_clean
.csv')
```

**Part IV: Model Identification and Analysis**

D. Analyze the time series dataset by doing the following:

1. Report the annotated findings with visualizations of your data analysis, including the following elements:
   - the presence or lack of a seasonal component - presence
   - trends – increase in revenue
   - auto correlation function – high autocorrelation
   - spectral density
   - the decomposed time series
   - confirmation of the lack of trends in the residuals of the decomposed series - yes

**Seasonality**: in data series it means periodic fluctuations. When a graph is analyzed, it usually shows the presence of a sinusoidal wave. The training dataset was decomposed so the seasonality could be analyzed:

```
#Let's decompose our training set to check for seasonality
decomposition =
seasonal_decompose(time_series_df_train['time_series_df_train'],
model='additive', period=7)
decomposition.plot()
plt.show()
```

According to the seasonal portion of the decomposition, seasonality is clearly visible.

**Trends**: Trend is the general direction of the time series over a long period of time. Analyzing the "trend" graph shows an obvious increase of revenue over time.

**Residual**: The residual is the difference between the observed and predicted values. The "Resid" portion of the decomposition shows no clear trend.

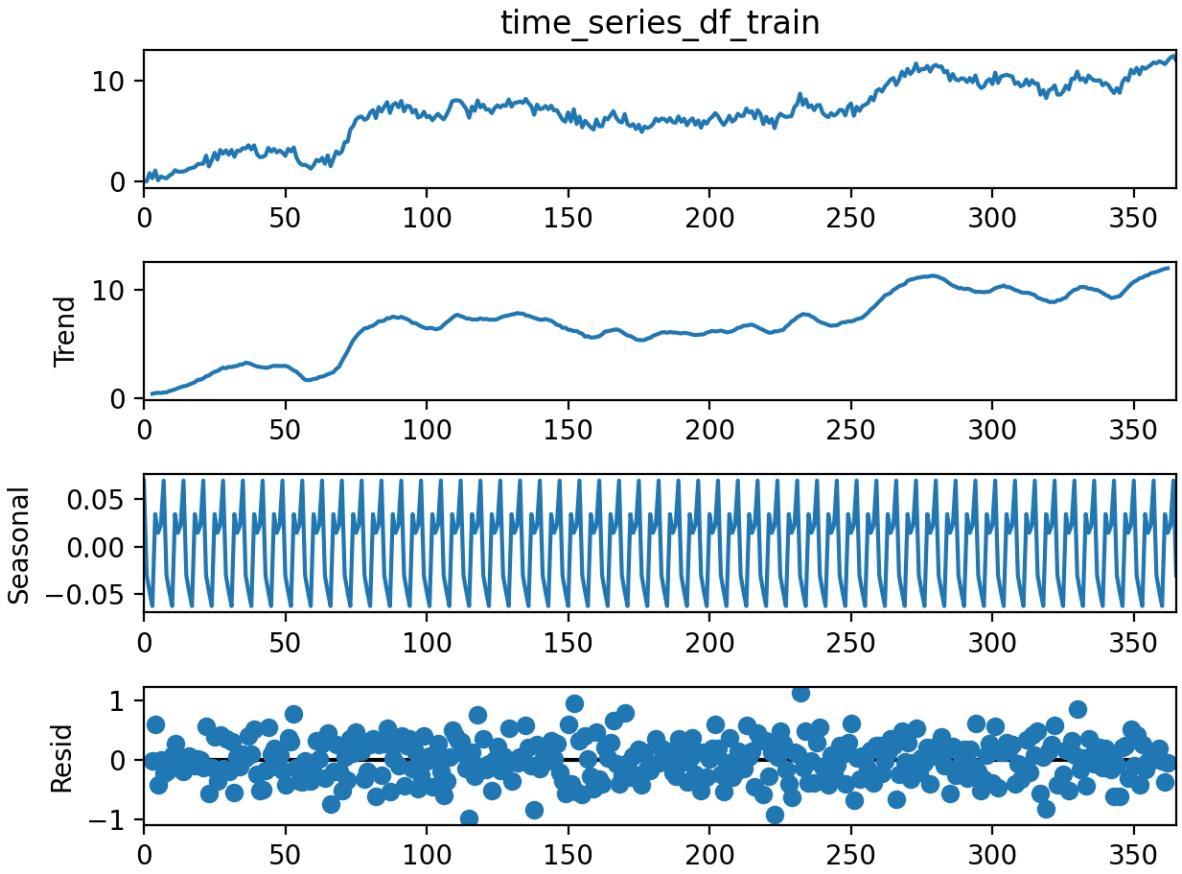



**Figure 7: Training Dataset Decomposition**

**Autocorrelation**: Just as correlation measures the extent of a linear relationship between two variables, autocorrelation measures the linear relationship between *lagged values* of a time series[3]. I am using the autocorrelation function ACF_PLOT to plot the autocorrelations. In this plot, each bar represents the size and direction of the correlation[4].

```
#Autocorrelation coefficients:
#31 days
autocorrelation_lag_1month = time_series_df['Revenue'].autocorr(lag=31)
print("One Month Lag: ", autocorrelation_lag_1month)
```

```
#62 days
autocorrelation_lag_2months = time_series_df['Revenue'].autocorr(lag=62)
print("Two Month Lag: ", autocorrelation_lag_2months)

#182 days (365/2)
autocorrelation_lag_6months = time_series_df['Revenue'].autocorr(lag=182)
print("Six Month Lag: ", autocorrelation_lag_6months)

#Plot Autocorrelation of Training set
plot_acf(time_series_df_train)
plt.show()
```

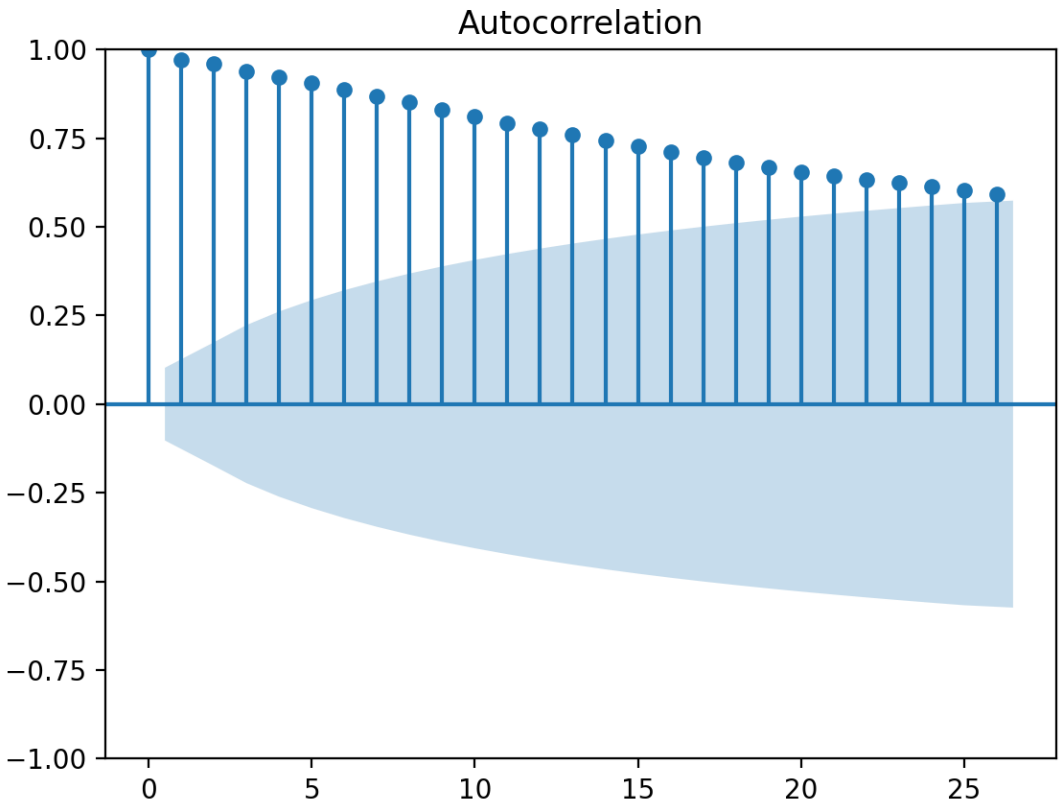



**Figure 8: Autocorrelation Plot**

From the above plot, it is reasonable to say that the autocorrelation is high. Here are results for the 1st, 2nd and 6th month.

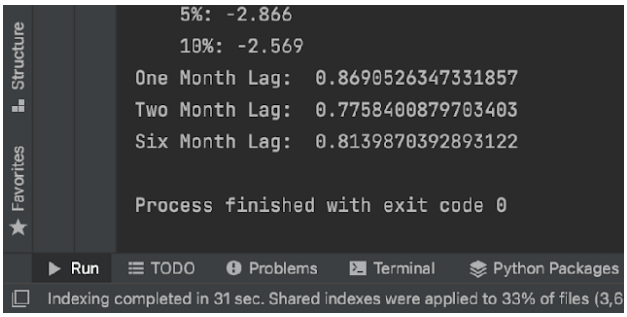



**Figure 9: Autocorrelation**

**Spectral Density**: is a frequency domain representation of a time series that is directly related to the autocovariance time domain representation[5].

```
#Spectral Density
plt.psd(time_series_df['Revenue'])
plt.show()
```

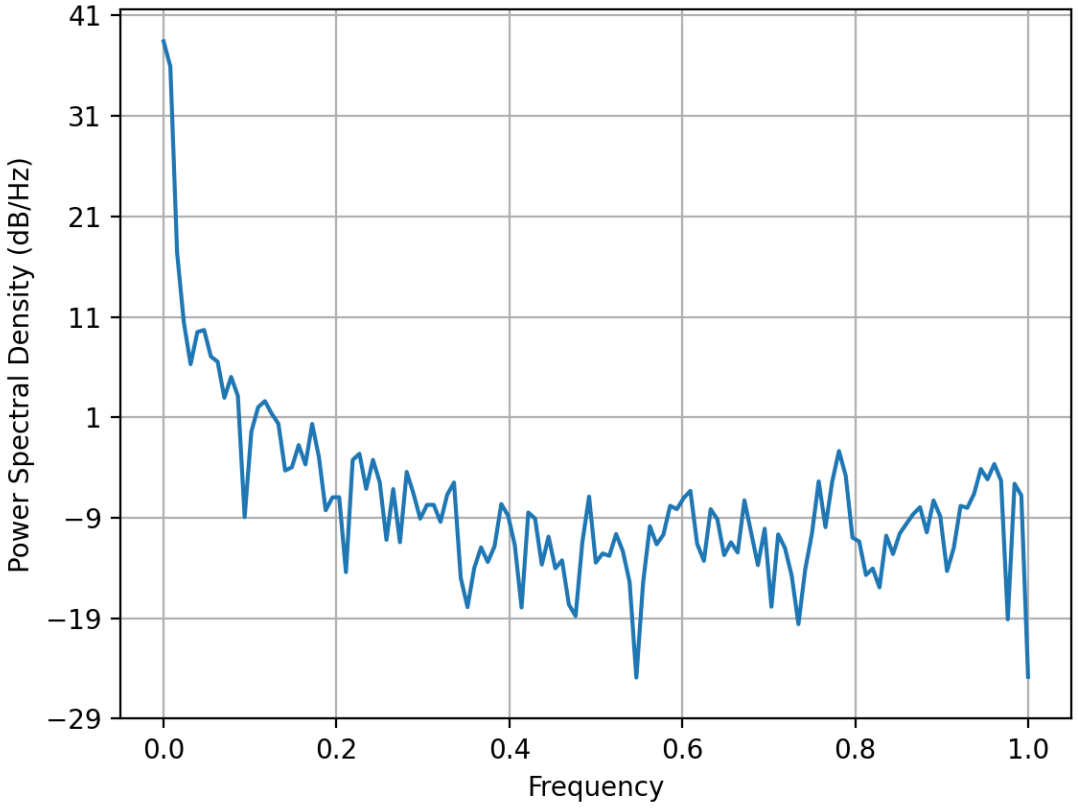



**Figure 10: Spectral Density**

2. Identify an autoregressive integrated moving average (ARIMA) model that takes into account the observed trend and seasonality of the time series data.

It was previously concluded that the data is non-stationary, therefore it can not be forecasted using traditional time series models. The data will need to be transformed in order to flatten the variance[6]. I used the difference transform to make my data stationary.

```
#ARIMA model

#Diff data to transform it to stationary
time_series_diff = time_series_df.diff().dropna()
result_diff = adfuller(time_series_diff['Revenue'])

fig, ax = plt.subplots()
time_series_diff.plot(ax=ax)
plt.show()
```

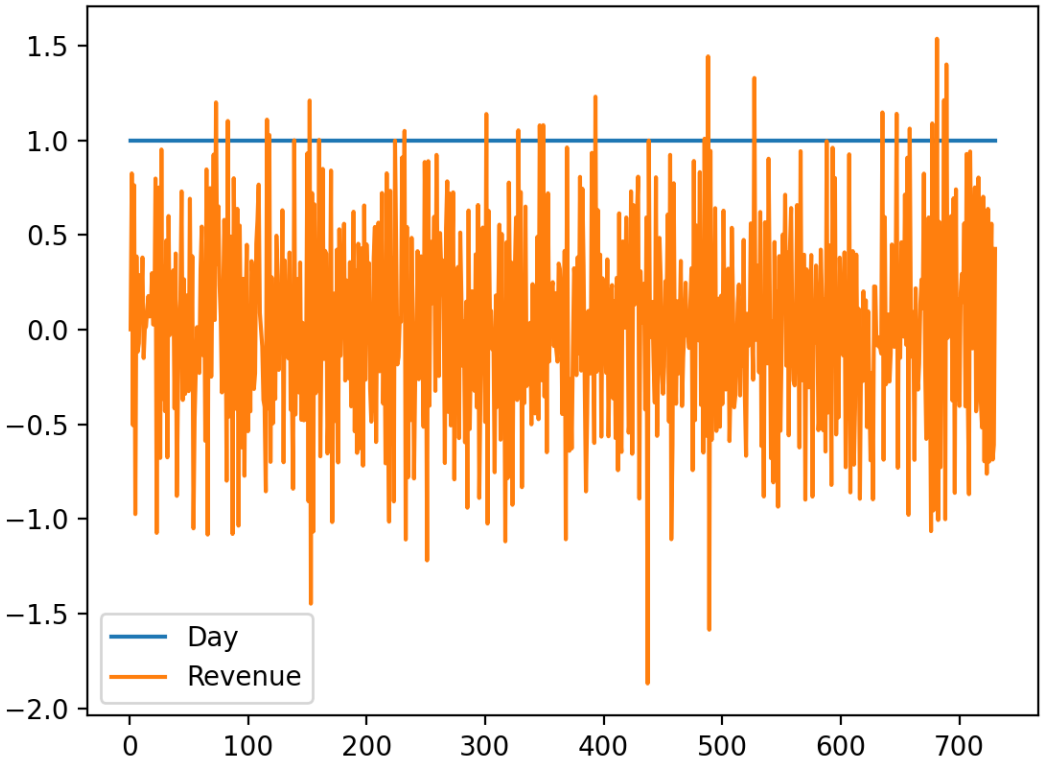



**Figure 11: Difference Transform**

```
#ADF for difference tranformation
print('Test Statistic:', result_diff[0])
# print('p-value: %s' % result_diff[1])
print(f'p-value: {result_diff[1]}')
print('Critical Values')
for key , value in result_diff[4].items():
    print('\t%s: %.3f' % (key,value))
```

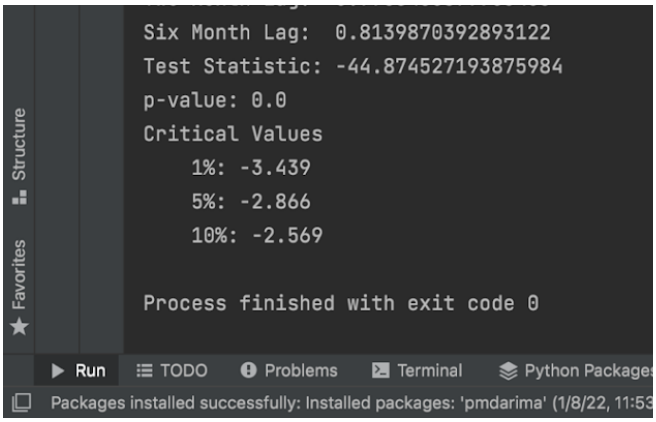



**Figure 12: ADF results for the difference transform**

Since the ADF value is much lower than the critical value for 1%, the p-value is very small. We have reached stationarity.

The objective of the library **auto_arima** is to identify the most optimal parameters for an ARIMA/SARIMA and return a fitted model[7].

```
#Fit auto_arima to dataset
auto_arima_fit = auto_arima(time_series_df['Revenue'], start_P=1, start_q=1,
max_p=3, max_q=3, m=12, seasonal=True,
                            d=None, D=1, trace=True, error_action='ignore',
suppress_warnings=True, stepwise=True)

auto_arima_fit.summary()
```

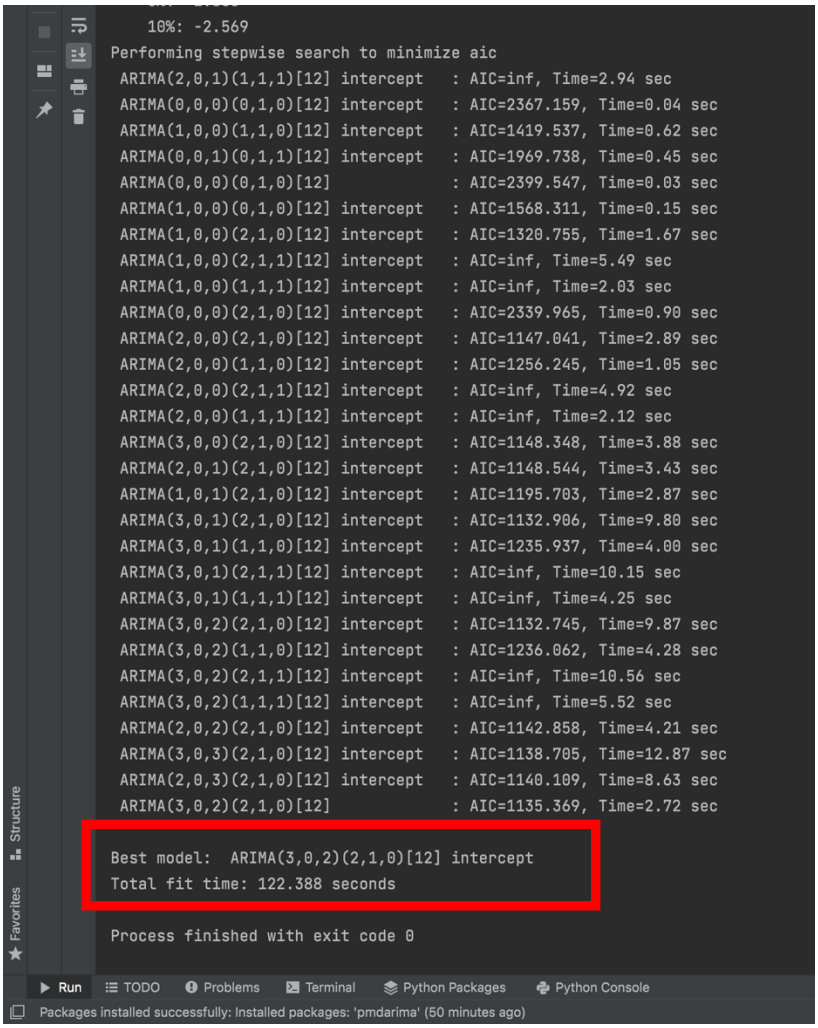
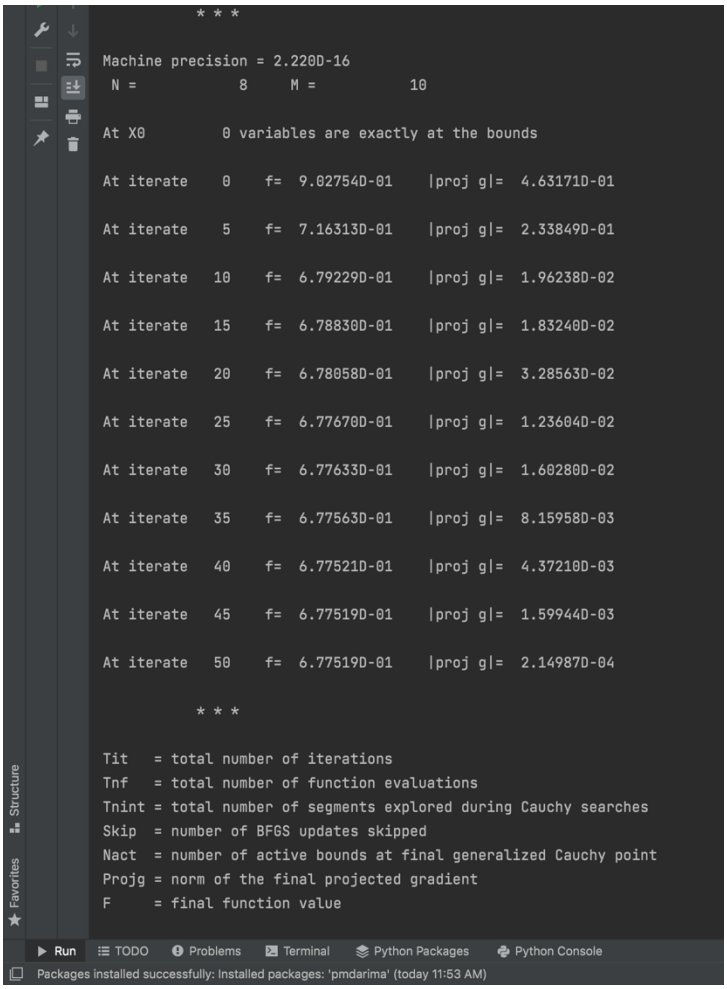



**Figure 13: Best parameters for the model**

Using the auto_arima library showed that (3,0,2)(2,1,0)[12] are the best parameters for the model. The ARIMA model shown above has an AIC of 1132.




The previously found ARIMA model will now be compared with the SARIMAX model. SARIMAX is a seasonal equivalent model to both SARIMA and AUTO ARIMA. It is considered to be an updated version of the ARIMA model.

```
#Build SARIMAX model with the parameters found by auto_arima
model = sm.tsa.SARIMAX(time_series_df_train, order=(3, 0, 2),
seasonal_order=(2, 1, 0, 12), enforce_stationarity=False,
                       enforce_invertibility=False)
SARIMAX_results = model.fit()

#Print results tables
print(SARIMAX_results.summary())
```

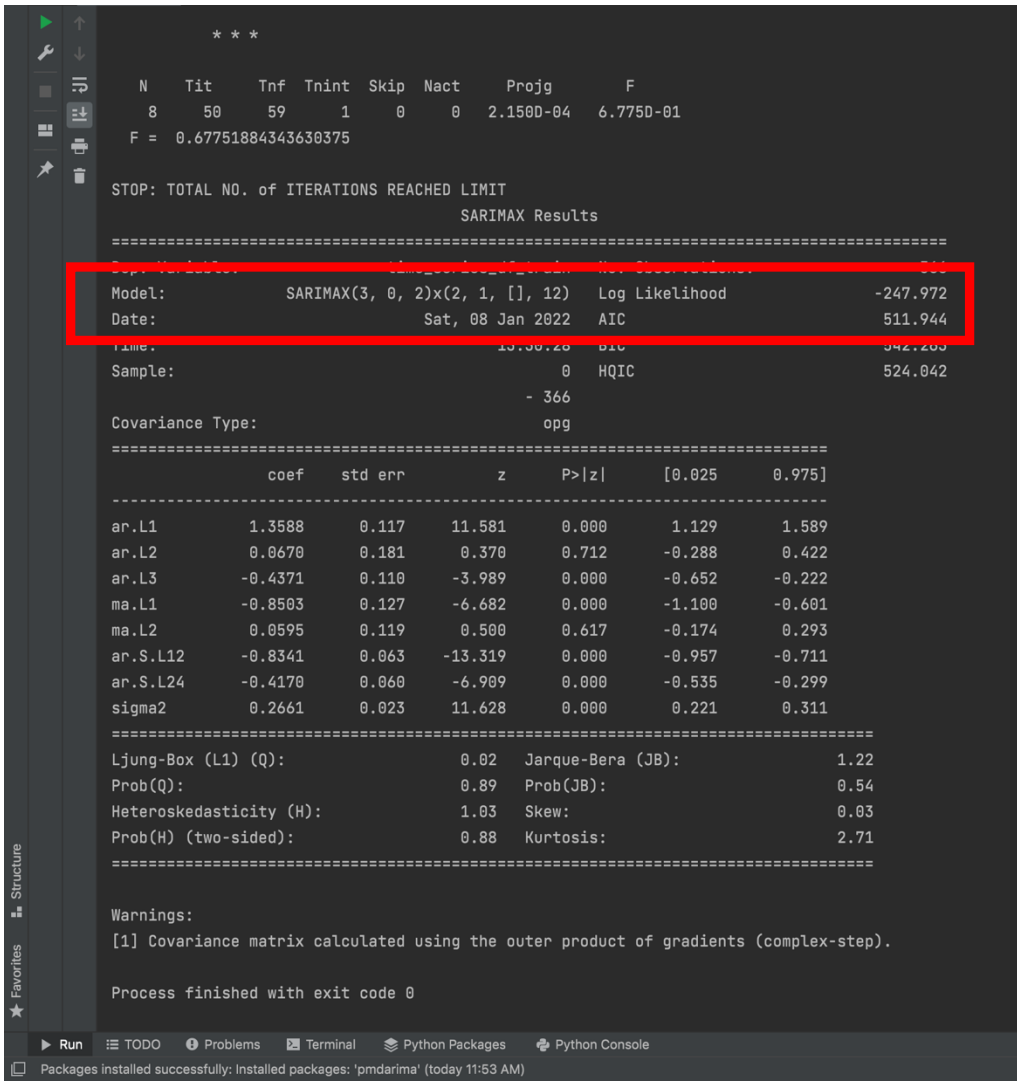



**Figure 14: SARIMAX Results (AIC = 511)**

The library "auto_ARIMA" was used to generate the optimal values for: **p** (the number of lag observations included in the model), **d** (the number of times that the raw observations are differenced) and **q** (the order of the moving average window). With that in hand, the model SARIMAX was selected with values of (3, 0, 2)(2, 1, 0) [12], as seen in the above figure. The AIC (Akaike Information Criterion) value for this model is **511.944**, which is lower than the AIC found for the ARIMA model (1132). According to Bevans[8], the model with the lowest AIC is best - SARIMAX in this case.

Now that a model has been built, forecasts can be made!

3. Perform a forecast using the derived ARIMA model.

```
#SARIMAX forecast
result = SARIMAX_results.get_forecast()
```

```
#Results of forecast and expected and error
test = time_series_df_test['time_series_df_test'].values.astype('float32')
forecast = result.predicted_mean
print('Expected: %.2f' % forecast)
print('Forecast: %.2f' % test[0])
print('Standard Error: %.2f' % result.se_mean)
```

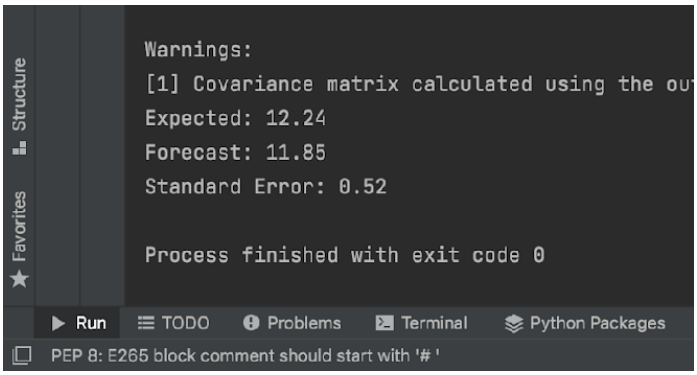



**Figure 15: Forecast of derived ARIMA model**

```
# Intervals
intervals = [0.2, 0.1, 0.05, 0.01]
for i in intervals:
    ci = result.conf_int(alpha=i)
    print('%.1f%% Confidence Interval: %.2f between %.2f and  %.2f' % (
    (1 - i) * 100, forecast, ci['lower time_series_df_train'], ci['upper
time_series_df_train']))

ci
```

```
80.0% Confidence Interval: 12.24 between 11.58 and  12.90
90.0% Confidence Interval: 12.24 between 11.39 and  13.09
95.0% Confidence Interval: 12.24 between 11.23 and  13.25
99.0% Confidence Interval: 12.24 between 10.91 and  13.57
```

| | lower time_series_df_train | upper time_series_df_train |
|---|---|---|
| 366 | 10.909486 | 13.567192 |

**Figure 16: Confidence Intervals**

```
#Test dataset - predictions
start = len(time_series_df_train)
end = len(time_series_df_train) + len(time_series_df_test) - 1

#Predict with respect to test set
predictions = SARIMAX_results.predict(start, end, typ =
'levels').rename('Predictions')
plt.plot(time_series_df_train, label = 'Set Train')
plt.plot(time_series_df_test, label = 'Set Test')
plt.plot(predictions, label = 'Predictions')
```

```
plt.title('Dataset Revenue Predictions')
plt.xlabel('Day')
plt.ylabel('Actual Revenue')
plt.legend(loc='upper left', fontsize = 8)
plt.show(
```

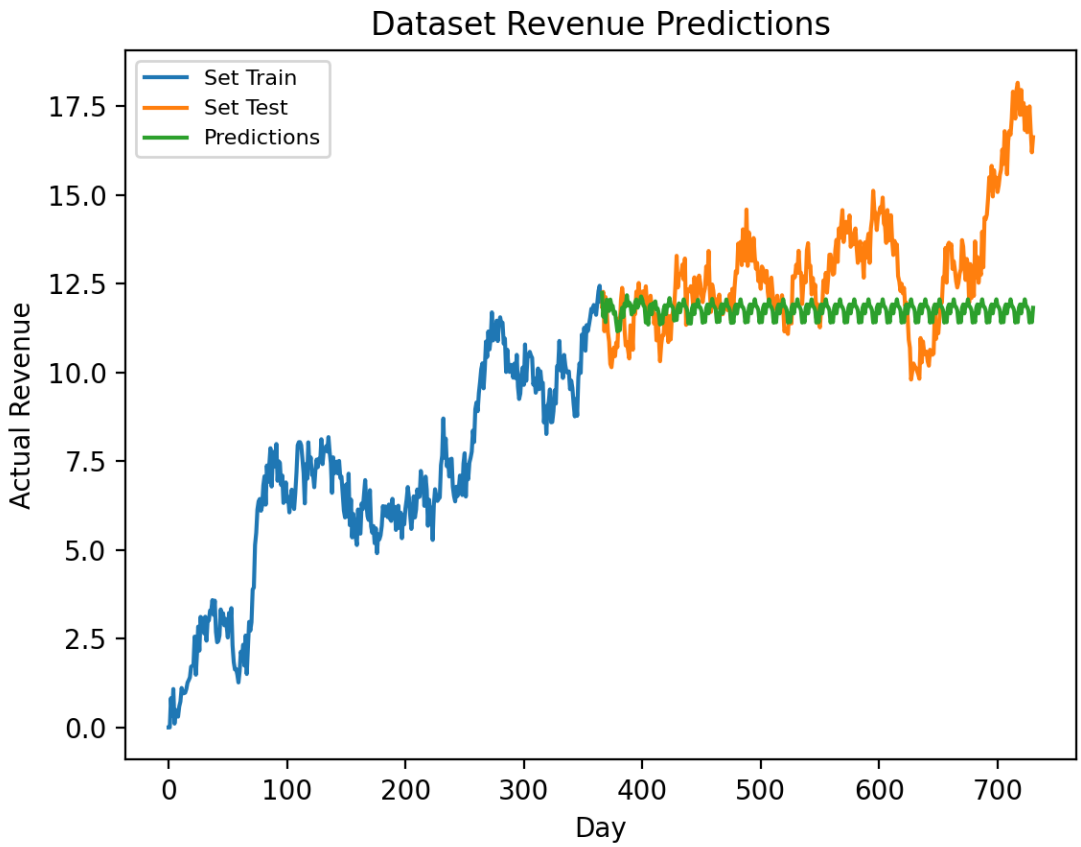



**Figure 17: Revenue Predictions on Test Set**

Making predictions for the entire dataset:

```
#1 year forecast
forecast = results.predict(start = len(time_series_df['Revenue']),
                           end = (len(time_series_df['Revenue']) - 1) + 365,
                           typ = 'level').rename('1 Year Forecast')

#Visualize predicted values
time_series_df['Revenue'].plot(figsize = (12, 5), legend = True)
forecast.plot(legend = True)
plt.show()
```

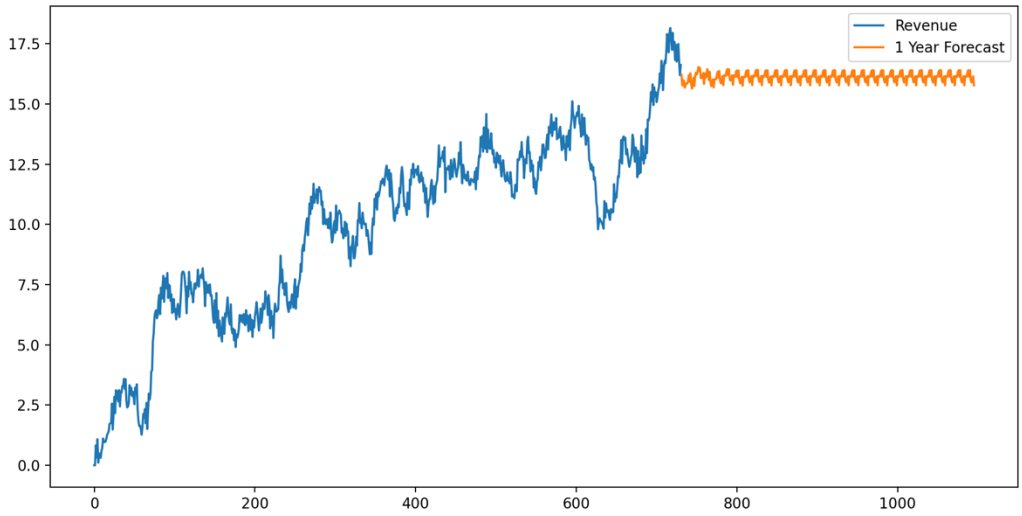



**Figure 18: One Year Prediction**

4. Provide the output and calculations of the analysis you performed.

The mean squared error, or MSE, is calculated as the average of the squared forecast error values (which is the expected value minus the forecast value). Squaring the forecast error values forces them to be positive; it also has the effect of putting more weight on large errors. The mean squared error described here is in the squared units of the predictions[9].

The root mean squared error, or RMSE can be transformed back into the original units of the predictions by taking the square root of the mean squared error score[9].

```
#Calculating MSE
MSE = mean_squared_error(time_series_df_test['time_series_df_test'],
predictions)
print('MSE: ', round(MSE, 4))

#Calculating RMSE
RMSE = rmse(time_series_df_test['time_series_df_test'], predictions)
print('RMSE: ', round(RMSE, 4))
```

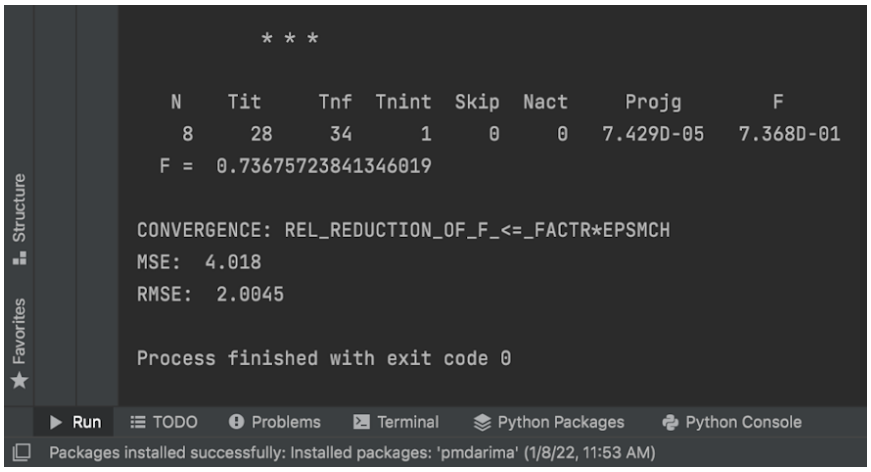



**Figure 19: MSE and RMSE**

The results are not great. The MSE/RMSE are a little high, which means that any predictions will be unreliable.

5. Provide the code used to support the implementation of the time series model.

**Part V: Data Summary and Implications**

E. Summarize your findings and assumptions, including the following points:

1. Discuss the results of your data analysis, including the following:
   - the selection of an ARIMA model

The "auto_ARIMA" library was used to generate the optimal p (the number of lag observations included in the model), d (the number of times that the raw observations are differenced) and q (the order of the moving average window) values. With that in hand, the model SARIMAX was selected with values of (3, 0, 2)(2, 1, 0) [12], as seen in Figure 13. The AIC (Akaike Information Criterion) value is **511.944** and it is the lowest compared to others.

Once several possible models have been created, AIC can be used to compare them. Lower AIC scores are better, and AIC penalizes models that use more parameters. So if two models explain the same amount of variation, the one with fewer parameters will have a lower AIC score and will be a better fit[8].

   - the prediction interval of the forecast: As shown in **Figure 16**

```
#5 Intervals
intervals = [0.2, 0.15, 0.1, 0.05, 0.01]
for i in intervals:
    ci = result.conf_int(alpha=i)
    print('%.1f%% Confidence Interval: %.2f between %.2f and  %.2f' % (
    (1 - i) * 100, forecast, ci['lower time_series_df_train'], ci['upper
time_series_df_train']))

ci
```

Five confidence intervals were tested for this dataset:

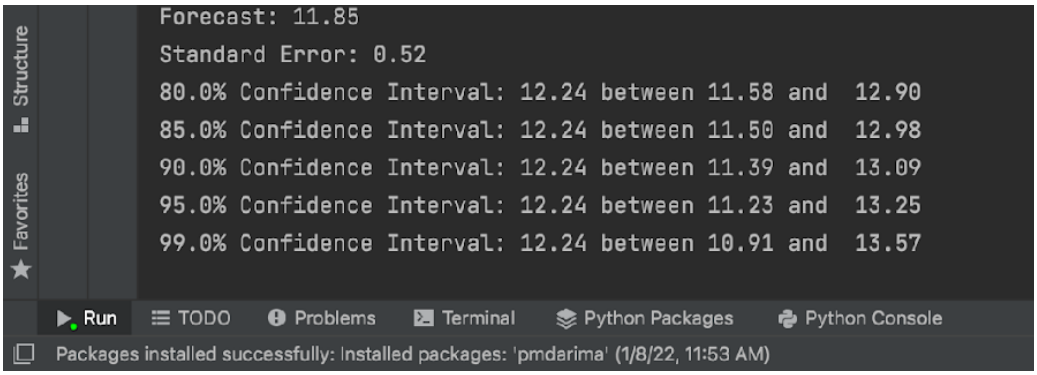



**Figure 20: Confidence Intervals**

1. 80.0% Confidence Interval: 12.24 between 11.58 and 12.90
2. 85.0% Confidence Interval: 12.24 between 11.50 and 12.98
3. 90.0% Confidence Interval: 12.24 between 11.39 and 13.09
4. **95.0% Confidence Interval: 12.24 between 11.23 and 13.25**
5. 99.0% Confidence Interval: 12.24 between 10.91 and 13.57

When 95% is used for the confidence interval, the result is a 5% likelihood that the real observation will fall outside the range of 11.23 and 13.25.

- a justification of the forecast length

In order to give stakeholders a short term forecast, I attempted to predict the revenue over a one-year period.

**Figure 18** shows that the forecasted revenue also presents some ups and downs, just like the previous year, but it fails to predict that the revenue will continue to increase.

Further analysis and more data would improve the ability to forecast next year's profits.

- the model evaluation procedure and error metric

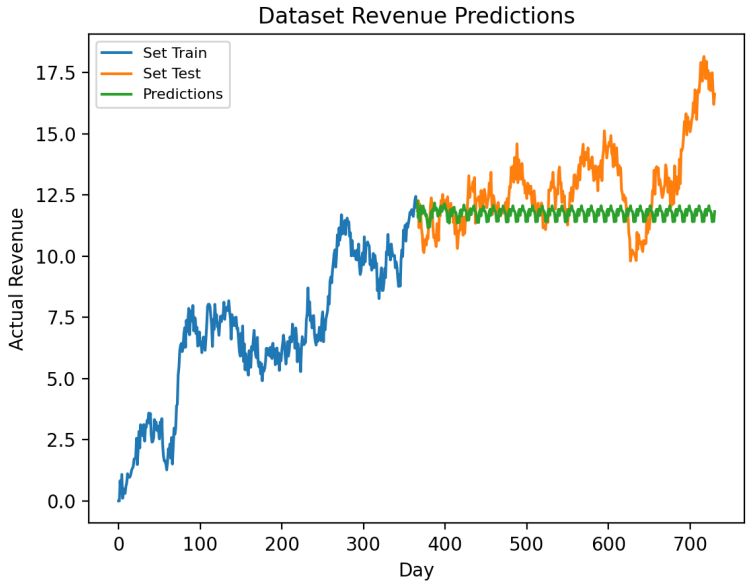
I have used MSE and RMSE to evaluate the model SARIMAX.

**MSE: 4.018**
**RMSE: 2.0045**

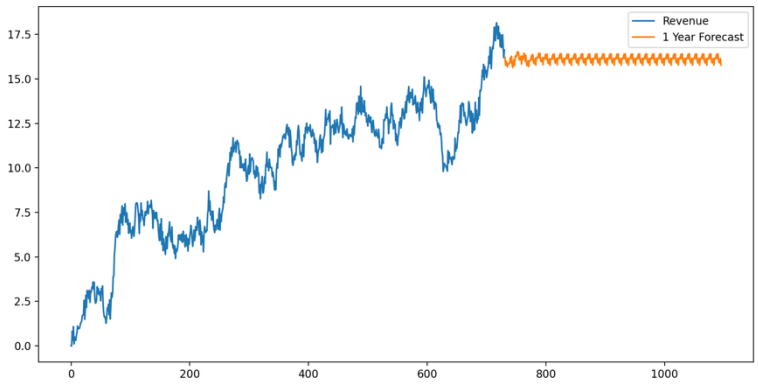
These values are not great, so the predictions are not very accurate.

2. Provide an annotated visualization of the forecast of the final model compared to the test set

Test set:




Final Model:




3. Recommend a course of action based on your results.

Since the prediction didn't offer a lot of new information, it is very hard to make any new recommendations. Previous analysis shows that the more services a customer signs up for, the more likely they are to stay with the company. The churn rate is known to be 26.5%, which is not bad compared to other companies. Therefore it is not unreasonable to assume a continuous increase in revenue (as seen in the 2yr data), with some seasonal fluctuation.

Analysis suggests periodicity and an increase in revenue commensurate with last year's performance.

**Part VI: Reporting**

F. Create your report from part E using an industry-relevant interactive development environment (e.g., an R Markdown document, a Jupyter Notebook, etc.). Include a PDF or HTML document of your executed notebook presentation.

The HTML document will be uploaded with this document.

H. Acknowledge sources, using in-text citations and references, for content that is quoted, paraphrased, or summarized.

[1] (2003, June 1st) NIST/SEMATECH, e-Handbook of Statistical Methods, 6.4.4.2 Stationarity, https://www.itl.nist.gov/div898/handbook/pmc/section4/pmc442.htm

[2] (2020, October, 19th) CHHUGANI, Rithwik, An Overview of Autocorrelation, Seasonality and Stationarity in Time Series Data https://analyticsindiamag.com/an-overview-of-autocorrelation-seasonality-and-stationarity-in-time-series-data/

[3] (2018) HYNDMAN, R.J., ATHANASOPOULOS, G., Forecasting: Principles and Practice https://otexts.com/fpp2/autocorrelation.html

[4] (2021) FROST, Jim, Autocorrelation and Partial Autocorrelation in Time Series Data, https://statisticsbyjim.com/time-series/autocorrelation-partial-autocorrelation/

[5] STAT 510: Applied Time Series Analysis, Chapter 12: Spectral Analysis, Penn State – Eberly College of Science https://online.stat.psu.edu/stat510/lesson/12/12.1

[6] (2020, July 11th) RASHEED, Rayhaan. Why does Stationarity Matter in Time Series Analysis? https://towardsdatascience.com/why-does-stationarity-matter-in-time-series-analysis-e2fb7be74454

[7] (2020, July 15th) CHATTERJEE, Sharmistha. Basic Understanding of ARIMA/SARIMA vs Auto ARIMA/SARIMA using Covid-19 Data Predictions https://hackernoon.com/basic-understanding-of-arimasarima-vs-auto-arimasarima-using-covid-19-data-predicions-8o1v3x4t

[8] (2020, March 26th) BEVANS, Rebecca. Akaike Information Criterion | When & How to Use It, https://www.scribbr.com/statistics/akaike-information-criterion/

[9] (2017, Feb 1st) BROWNIEE, Jason. Time Series Forecasting Performance Measures With Python https://machinelearningmastery.com/time-series-forecasting-performance-measures-with-python/