Eric Yarger

Time Series Modeling

Research Question

Time series analysis is a statistical tool used to analyze past data within a certain timeframe with the goal of better forecasting the future. The Executive Team of our organization is interested in a time series on revenue from the first two years of operation, to be used to look for patterns in our dataset. Any insights or patterns discovered in the data have the potential to help our organization better control and manage readmission rates. The question for this analysis is:

"Using Time Series Analysis, can we create a model that can be used to predict future revenues, which can be used by our Executive Team to better address readmission?"

Objectives and Goals

The top priority of this analysis, our teams' top goal, is to create an ARIMA model that addresses all necessary conditions and can be used as a tool by our organization for future revenue prediction. This model can then be used to help our Executive Team have a more accurate depiction of future revenues, which in turn allows for better business decisions to be made. The secondary goal is to identify any trends or seasonality that our revenue stream exhibits. This allows our Executive Team to more accurately manage organizational needs, such as staffing levels for both medical and non-medical staff.

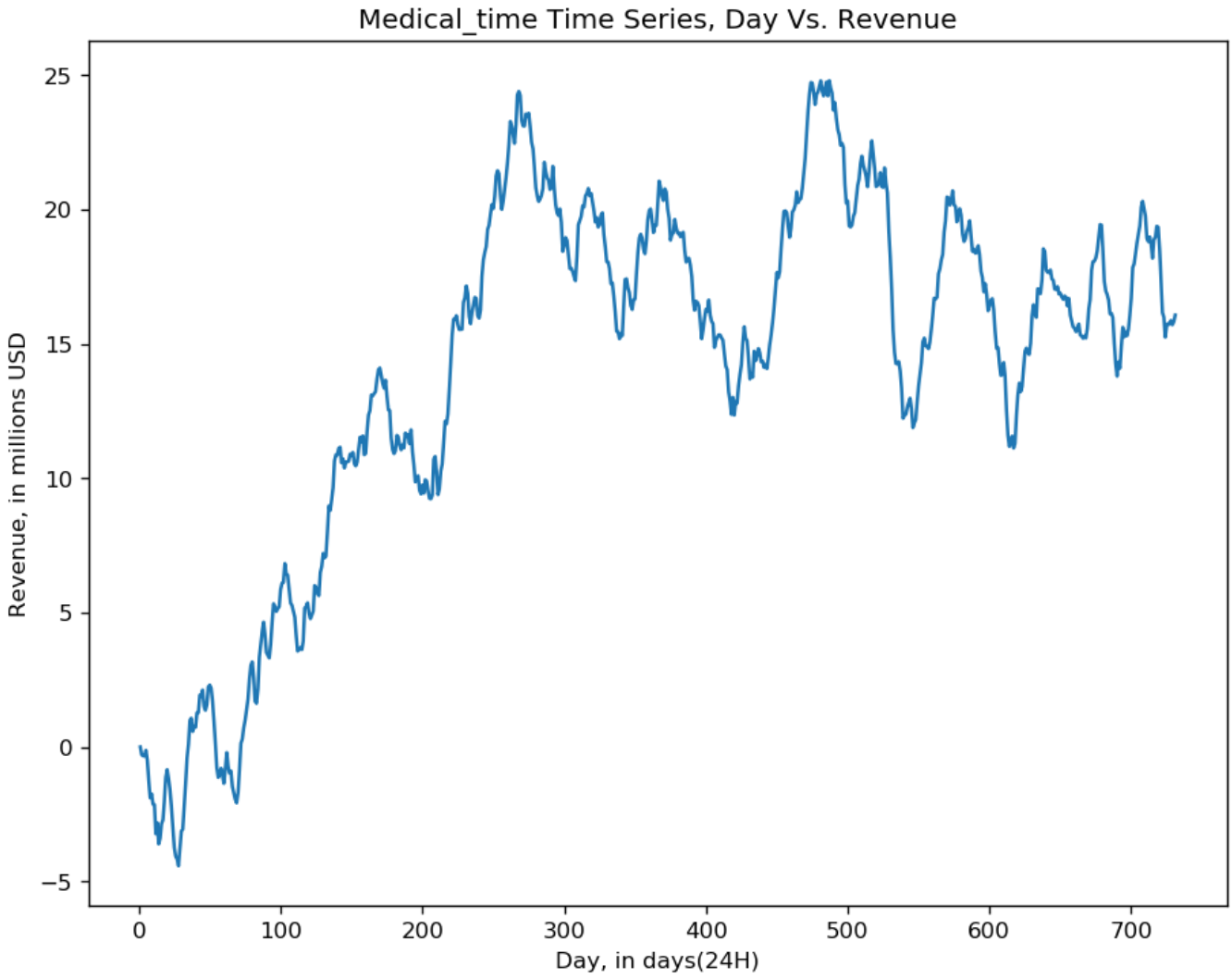
Summary of Assumptions

From WGU's Professor Elleh, the assumptions of a time series analysis are (Elleh, 2022):

1. The time series data should be stationary.
2. The error term is randomly distributed, and the mean and the variance are constant over the time period. Error in time series analysis is assumed to be uncorrelated.
3. No outliers in the time series. Outliers can lead to inaccurate results from the model.
4. The residuals are not autocorrelated.

Line Graph Visualization

Provided below is a line graph of the Medical dataset provided with the course, medical_time.csv. The graph is fully labeled and displays a complete realization of the time series.



Time Step Formatting

Time step formatting is undertaken in two steps.

- Step 1: convert df['Day'] into datetime object.
    - This is performed using pandas pd.to_datetime function. Where there is no specified beginning date for this dataset, pd.to_datetime's 'origin' Parameter is left as default, which defaults to 'unix' or 'POSIX' time. This sets the default origin for df['Day']'s first data point to datetime 1970-01-01. For the purpose of this analysis this is adequate for our analysis, as the beginning day, month, and year is not specified.
- Step 2: set df['Day'] to dataset index using df.set_index.
    - This sets the correct datetime object as the index for the dataset. This ensures that our data is optimized and in the proper format for time series modeling.

The time interval is measured in Days(1 interval is one 24 hour period/Day). The time sequence length is 731. The datatype is datetime64. There are no gaps in measurement, verified using df.isnull().any() in the accompanying code. The entirety of code used to perform this section of analysis can be found in the accompanying PDF of my JupyterNotebook under section C2: Time Step Formatting.

Stationarity

Stationarity of the time series is evaluated using the Augmented Dickey Fuller (ADF) Test. All code used for this analysis can be found under 'C3: Stationarity' in the accompanying PDF of my JupyterNotebook. For this analysis, statsmodels.tsa.stattools adfuller function is used to perform the ADF Test. The ADF test tests the null hypothesis that a unit root is present in a time series sample (Augmented Dickey–Fuller test, 2022). The ADF statistic resulting from the 'adfuller' function is a negative number. The more negative it is, the stronger the rejection of the hypothesis that there is a unit root at our necessary level of confidence of 5%. The 'adfuller' function also prints the P-value resulting from the test.

Provided below is a screenshot of the results of the ADF test.

C3: Stationarity

Augmented Dickey Fuller (ADF) Test

Assess stationarity of dataset

```
]: # Code Reference (Making time series stationary | Python, n.d.)

   dicky_fuller_test = adfuller(df)
```

```
]: # p = .19964
   # data does not reject null hypothesis at p < .05

   dicky_fuller_test
```

```
]: (-2.2183190476089485,
    0.19966400615064228,
    1,
    729,
    {'1%': -3.4393520240470554,
     '5%': -2.8655128165959236,
     '10%': -2.5688855736949163},
    842.4530276176408)
```

To summarize the results, rounded to the fifth decimal, the ADF test statistic is -2.21832, the P-value is .19966. In order to reject the null-hypothesis of stationarity for this dataset, the ADF test statistic would have to be -2.86551 at P=5%.

Therefore, we fail to reject the null hypothesis of stationarity for this dataset and assume the data does not exhibit stationarity.

To rectify the lack of stationarity in the dataset, differencing is implemented. The original series, the series 1st order differenced, and the season 2nd order differenced are plotted. Also, pmdarima's 'ndiffs' function was implemented to estimate the correct differencing term. Below is a screenshot of the results of this section of the analysis.

From this analysis it's determined that a differencing term of 1 is optimal for rectifying the non-stationarity of the dataset for Time Series analysis using ARIMA.

1st and 2nd order Differencing

finding 'd' for ARIMA model

[22]:
```
plt.rcParams.update({'figure.figsize':(9,7), 'figure.dpi':120})

# Original Series
fig, (ax1, ax2, ax3) = plt.subplots(3)
ax1.plot(df); ax1.set_title('Original Series'); ax1.axes.xaxis.set_visible(False)
# 1st Differencing
ax2.plot(df.diff()); ax2.set_title('1st Order Differencing'); ax2.axes.xaxis.set_visible(False)
# 2nd Differencing
ax3.plot(df.diff().diff()); ax3.set_title('2nd Order Differencing')
plt.show()
```

[31]:
```
# Using pmdarima's ndiffs to find differencing term
# Code reference (Verma, 2021)
from pmdarima.arima import ndiffs

kpss_diffs = ndiffs(train, alpha=0.05, test='kpss', max_d=6)
adf_diffs = ndiffs(train, alpha=0.05, test='adf', max_d=6)
n_diffs = max(adf_diffs, kpss_diffs)

print(f"Estimated differencing term: {n_diffs}")
```

```
Estimated differencing term: 1
```

Steps to Prepare the Data

Provided below is a table of the steps used to prepare the data for Time Series modeling.

STEP	DESCRIPTION	EXPLANATION
1	Load Dataset into JupyterNotebook	JupyterNotebook is the environment used for EDA and preparation of the dataset for this analysis. Python 3.7.13 is the language and version used.
2	EDA	In this step we are doing a brief exploration of the dataset. Identification of the dataset size, datatypes, number of features, and shape of dataset are identified. Examples of each feature are analyzed, and the count, mean, standard deviation, and distribution of the data is identified. It's also made sure there are no missing values in the dataset. This step provides us with a solid foundation for what needs to be altered to make this dataset optimized for Time Series modeling.
3	Visualization	The original dataset is visualized to provide our analyst team with a first glimpse into how revenue looks over the two year period of this dataset
4	Cleaning	Dataset appears to be clean. To ensure there are no null values df.dropna() is called as a preliminary measure.
5	Time Step Formatting	df['Day'] is set to datetime. df['Day'] is set to the dataset Index. This changes the dataset to be optimized for Time Series analysis
6	Stationarity Analysis	Stationarity is assessed using the Augmented Dickey Fuller Test. Time series is shown to not be stationary.
7	Differencing	The time series is plotted, as well as the first and second order differencing of it, to assess the correct 'd' value for addressing the non-stationarity of the time series. Pmdarima's 'ndiffs' is also used to estimate the best differencing term. Both methods point to the optimal differencing term value of 1.
8	Seasonality Analysis	Statsmodels seasonal_decompose is used to plot the Trend, Seasonal, and Residuals of the Time Series. This provides our analyst team with vital information for assessing and addressing the seasonality of the time series. It also provides us with validation that there is no autocorrelation in the error term, which is a necessary assumption of Time Series analysis.
9	Autocorrelation Function (ACF)	ACF is assessed using statsmodels plot_acf. The original Time Series, and the first and second order differencing are plotted to assess autocorrelation, and to find the optimal MA term 'q'.
10	Partial Autocorrelation	PACF is assessed using statsmodels plot_pacf. The original

	Function (PACF)	Time Series, and the first and second order differencing are plotted to assess autocorrelation, and to find the optimal AR term 'p'.
11	Spectral Density	Spectral Density is used to find the periodicity and the frequency before the seasonal pattern repeats in a Time Series.
12	Train/Test Split	The Time Series is split into 75% Training and 25% Test sets using pmdarima's train_test_split.
13	Read Prepared sets to Excel File	The Training and Test set variables are read out to Excel files for submission with this analysis for replicability.

Prepared Dataset

Provided with this submission as Excel files are copies of the Test and Training sets.

Report Findings and Visualizations

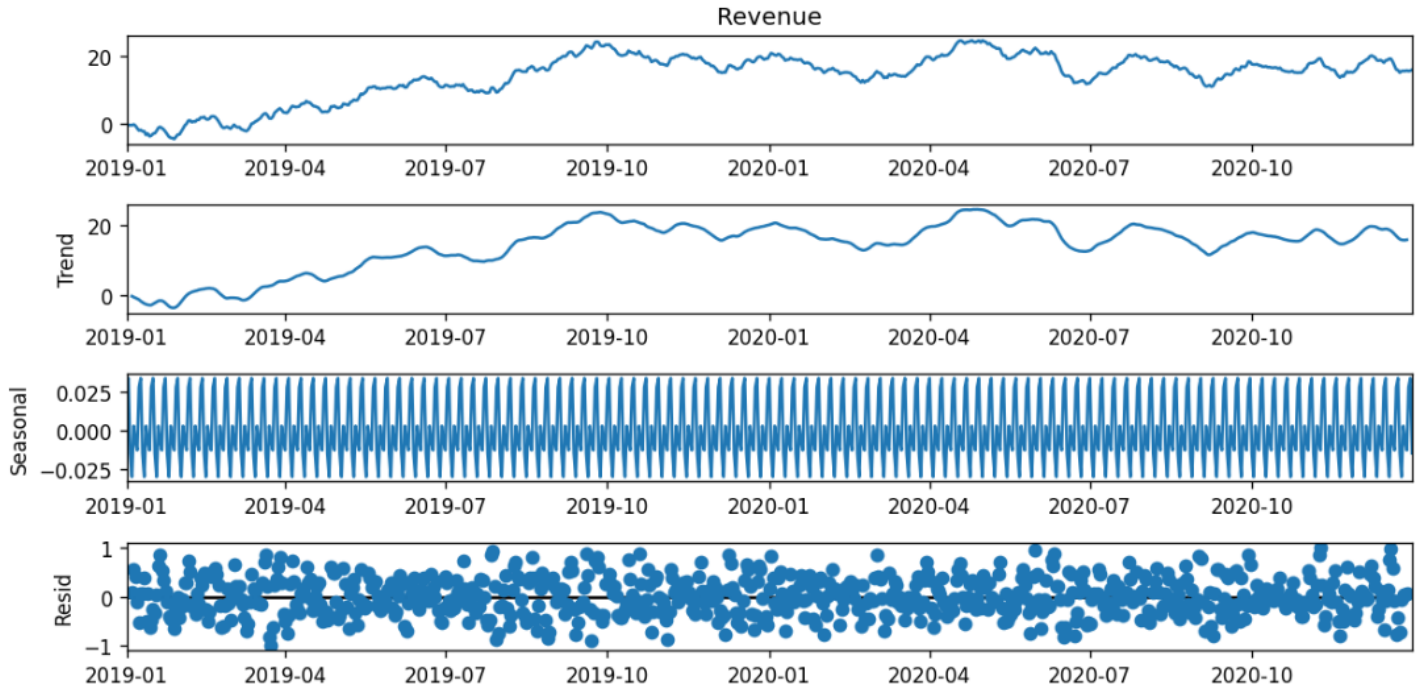
Provided below are the findings with annotated visualizations for each of the 6 listed elements.

Decomposed Time Series

```
# Code Reference (Boston, 2020)
from statsmodels.tsa.seasonal import seasonal_decompose
```

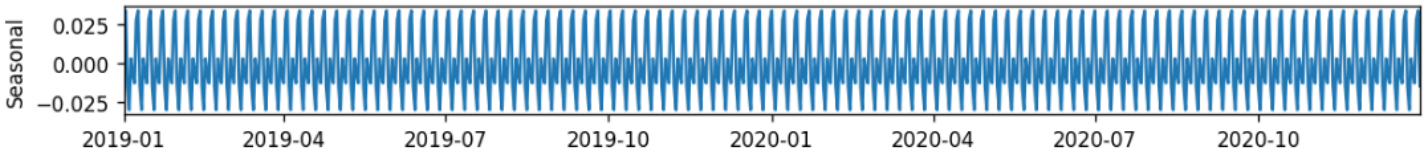
```
result = seasonal_decompose(df['Revenue'])
```

```
from pylab import rcParams
rcParams['figure.figsize'] = 10,5
result.plot();
```



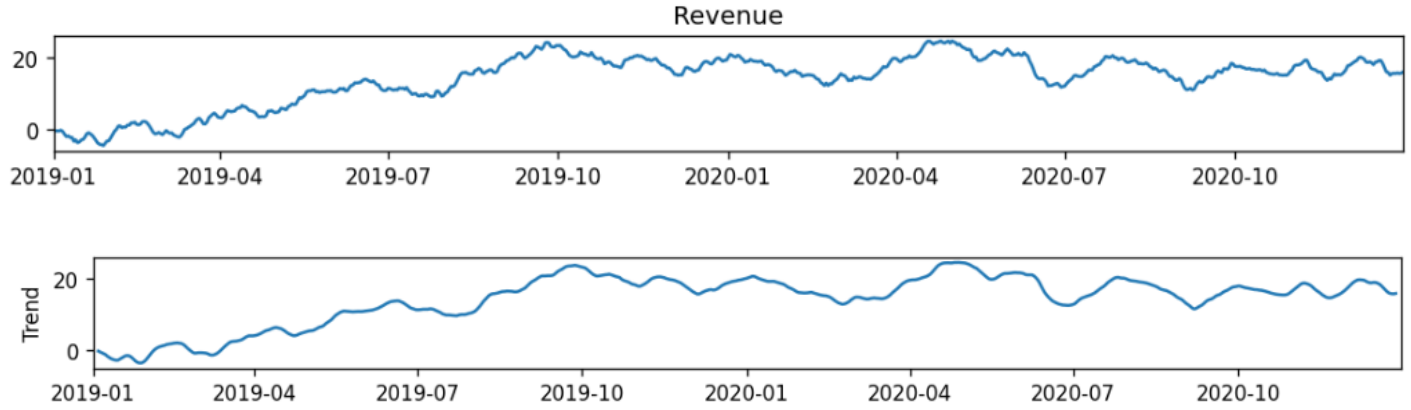
Statsmodels seasonal_decompose was used to decompose the time series. This breaks the series down into its trend, seasonality, and residuals. Each of these attributes provides clues and insights into what will be the best method for modeling the time series. Each of these attributes is discussed in depth further in this section.

Seasonality



Provided above is the plot of Seasonality from using statmodel's seasonal_decompose to analyze the time series. The time series appears to exhibit regular intervals of seasonality at nearly a weekly frequency. Therefore, we will need to use a Seasonal ARIMA model when performing our modeling to account for any seasonality present in the time series. Pmdarima's auto_arima will be used for ARIMA, and for this analysis auto_arima parameter 'seasonal' will be set to 'True', and parameter 'seasonal_test' will be set to 'ocsb'. This ensures our model takes into consideration the appropriate values for seasonality.

Trends

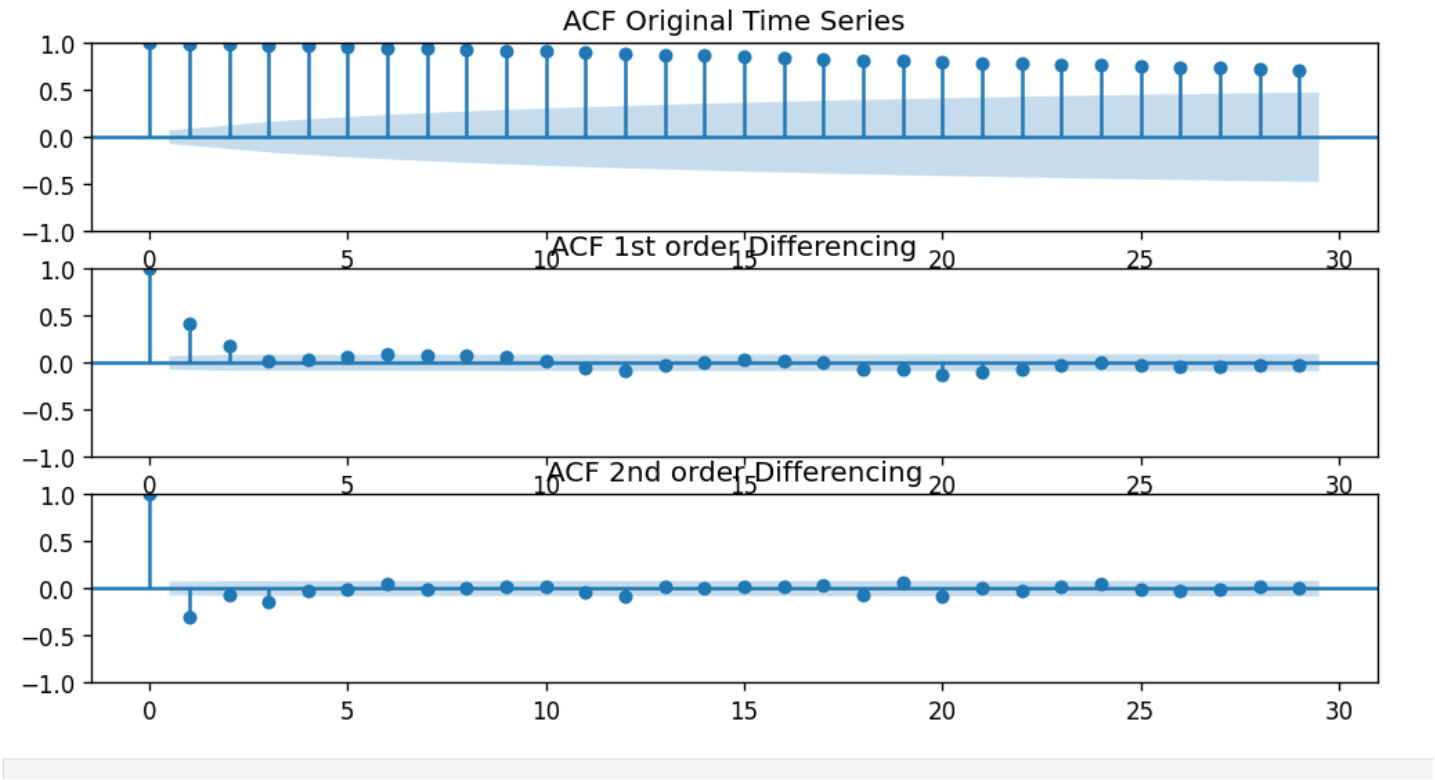


Provided above are the visualizations for both the Time Series, unmodified, as well as the Time Series' Trend graph. Plotting the Trend allows us to see the Time Series overall direction more easily. As seen from the Trend graph, Revenue increased more than it decreased steadily for the first 10 months, from 01/2019 to 10/2019. It then stops growing, and for the next 14 months of the Time Series stabilizes and oscillates between 20 million and 12 million dollars.

Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF)

Using Autocorrelation function (ACF)

```
from statsmodels.graphics.tsaplots import plot_acf

fig, (ax1, ax2, ax3) = plt.subplots(3)
plot_acf(df, ax=ax1, title='ACF Original Time Series');
plot_acf(df.diff().dropna(), ax=ax2, title='ACF 1st order Differencing');
plot_acf(df.diff().diff().dropna(), ax=ax3, title='ACF 2nd order Differencing');
```



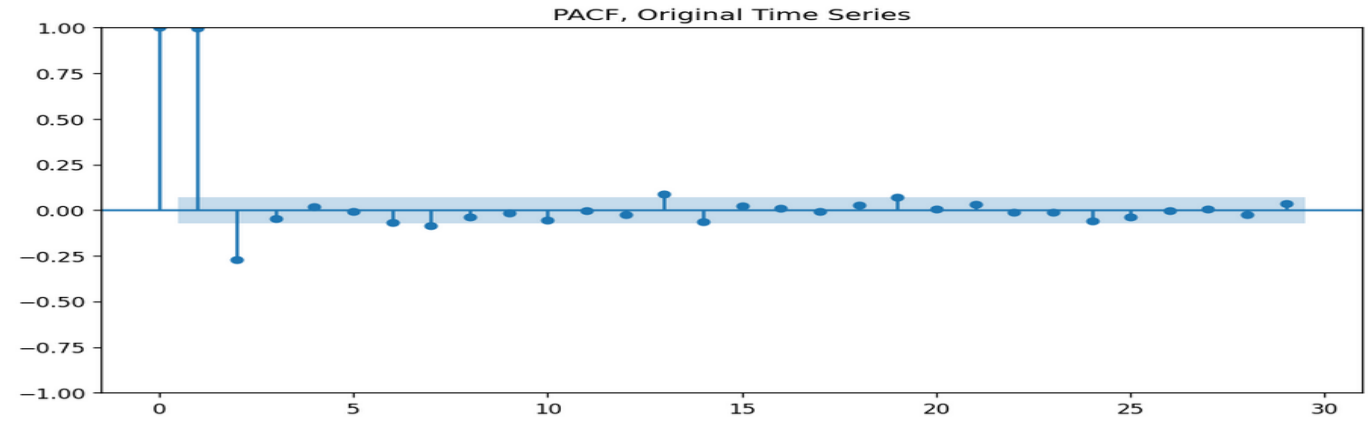
The ACF shows how the data is correlating across periods of time. The function at lag-1 is the correlation between a time series and the offset of itself by one step (Elleh, 2022). Values between 0 and 1 represent positive correlation, values between -1 and 0 represent data that is not correlated. This helps us find the optimal Moving Average (MA) 'q' term for our ARIMA model.

The ACF is also useful for identifying the stationarity of our time series. As seen from the visualization above for our Original Time Series, the ACF drops slowly over time, which points to a non-stationary time series. The first and second order differencing drop to 0 quickly and oscillate within the shaded region, which indicates stationarity in the time series at that level of differencing.

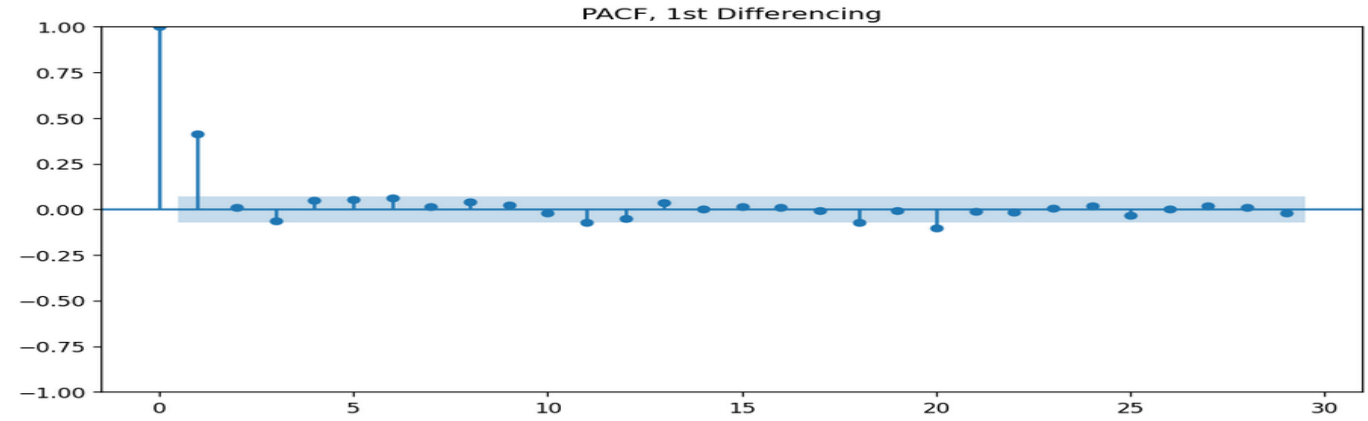
PACF

Using Partial autocorrelation (PACF)

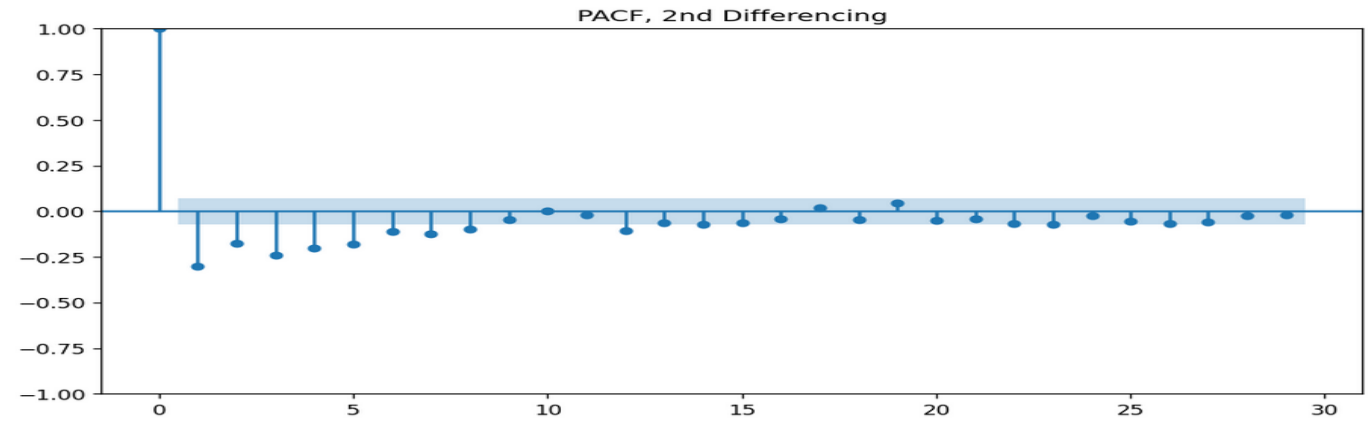
```
from statsmodels.graphics.tsaplots import plot_pacf
#Warnings ignored because this analysis uses deprecated ARIMA model from statsmodels
import warnings
warnings.filterwarnings("ignore")
plot_pacf(df.dropna(), title='PACF, Original Time Series');
```




```
plot_pacf(df.diff().dropna(), title='PACF, 1st Differencing');
```




```
plot_pacf(df.diff().diff().dropna(), title='PACF, 2nd Differencing');
```

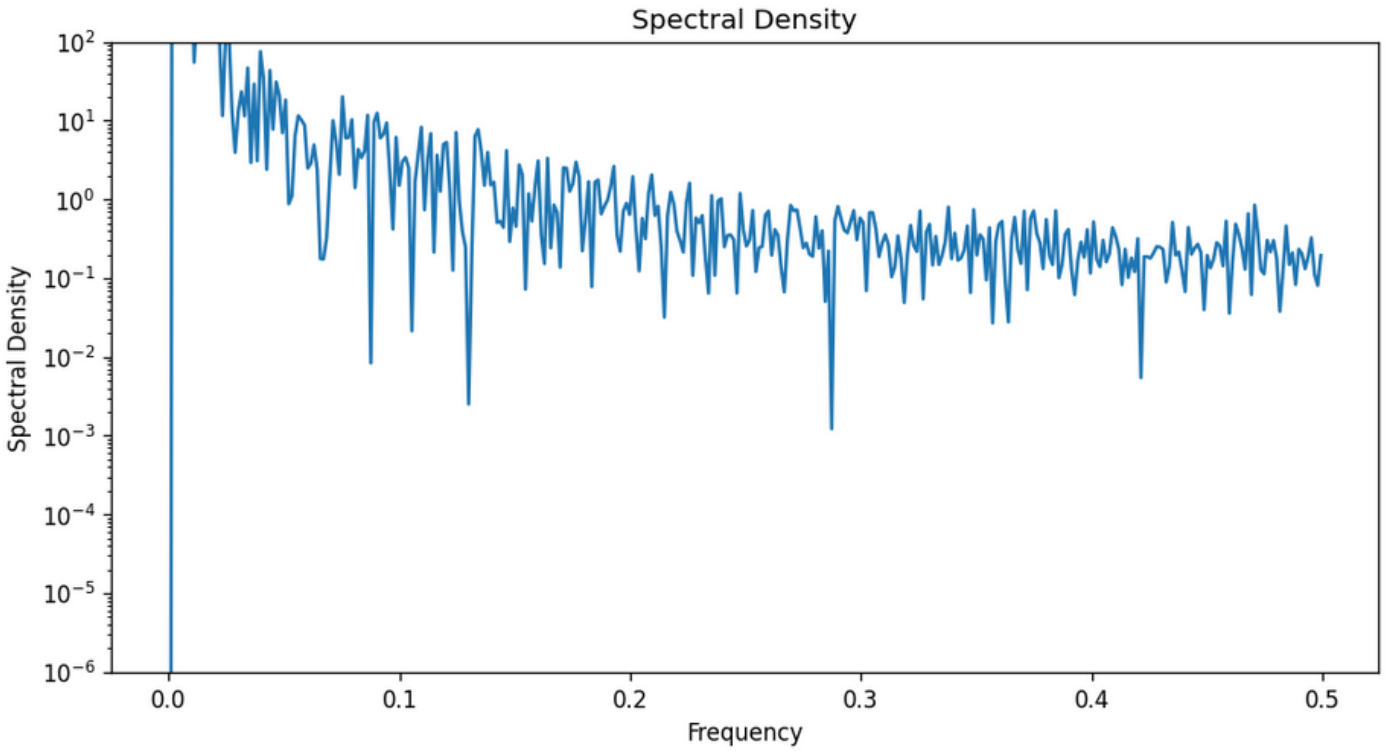


Partial Autocorrelation Function (PACF) helps us identify the Autoregressive (AR) term 'p' that is most viable for the ARIMA model. It shows additional correlation explained by each successive lagged term. This helps us identify the most viable figure for the AR 'p' component in our ARIMA model.

Spectral Density

Spectral Density

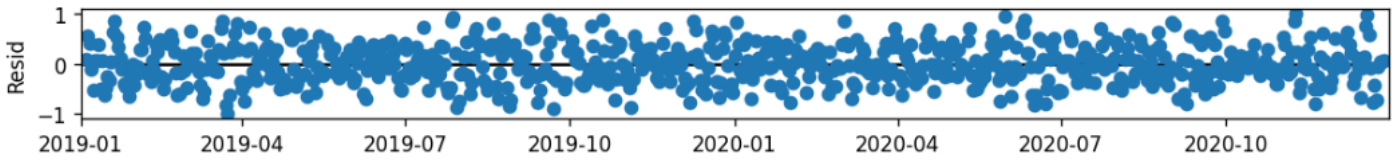
```
# Spectral Density
# Code Reference (Festus, 2022)
from scipy import signal

f, Pxx_den = signal.periodogram(df['Revenue'])
plt.semilogy(f, Pxx_den)
plt.ylim(1e-6, 1e2)
plt.title('Spectral Density')
plt.xlabel('Frequency')
plt.ylabel('Spectral Density')
plt.show()
```



The Spectral Density visualizes the frequencies related to autocovariance across our time series. It does this by transforming our dataset from time domain to frequency domain. Graphing spectral density can help us visualize the seasonality of the time series. This is because the 'frequency' aspect of spectral density represents the number of observations before our seasonal pattern repeats itself.

Analysis of the spectral density yields that while our Time Series exhibits many peaks and troughs, it is overall just very noisy. This indicates that what originally could be taken for seasonality in the time series may just be randomness, akin to white noise.

Residuals



We can see from the Residual plot above that there appears to be a large amount of randomness in the time series data. Because revenue is susceptible to ups and downs across our organization, this tells us our model will benefit from the inclusion of a Moving Average component (MA). From plotting the residuals we can also verify there is no autocorrelation of the residuals, which is a necessary assumption for time series modeling (Dail, 2011).

ARIMA Model

Pmdarima's auto_arima was used to identify the optimal ARIMA model. The model's parameters were set up to take into account the observed trend of the time series, as identified in section D1. The model's parameter 'seasonal' was set to 'True' to ensure auto_arima takes into account any seasonality found in the dataset. This means that auto_arima will optimize the ARIMA model and take into consideration seasonality with a SARIMAX model.

This enables auto_arima to fit a model that takes into account any found seasonality in the time series. If no seasonal P, D, and Q are identified by the user (which in this case, there wasn't, to allow auto_arima to find and adjust for seasonality as it iterates), then auto_arima defaults to start_P=1, start_Q=1, and start_D=the results of auto_arima's seasonal_test.

The seasonal_test selected in auto_arima's parameters was the 'ocsb' which is the functions' default for this parameter. The OCSB test stands for the Osborn, Chui, Smith, and Birchenhall Test for Seasonal Unit Roots and tests to see if the time series contains a seasonal unit root. This test informs auto_arima if adjusting the model for seasonality is necessary to yield the best result (Pmdarima.arima.auto_arima — pmdarima 2.0.1 documentation, n.d.). From the model's results of (0,0,0)[0] for P,D,Q,[#Periods], auto_arima's seasonal_test result indicates that there is no viable seasonality that will enhance the SARIMAX model results.

The best model is SARIMAX(1, 1, 0), or described as an ARIMA model: ARIMA(1, 1, 0)(0, 0, 0)[0], with a minimized AIC of 671.106. This model is the outcome of auto_arima's stepwise search to minimize AIC. It

takes into account the trend and seasonality (or in this instance, the absence of seasonality) of the data to output the optimal ARIMA model. The model summary results are provided as a screenshot below.

Auto-arima

Using pmdarima's auto_arima

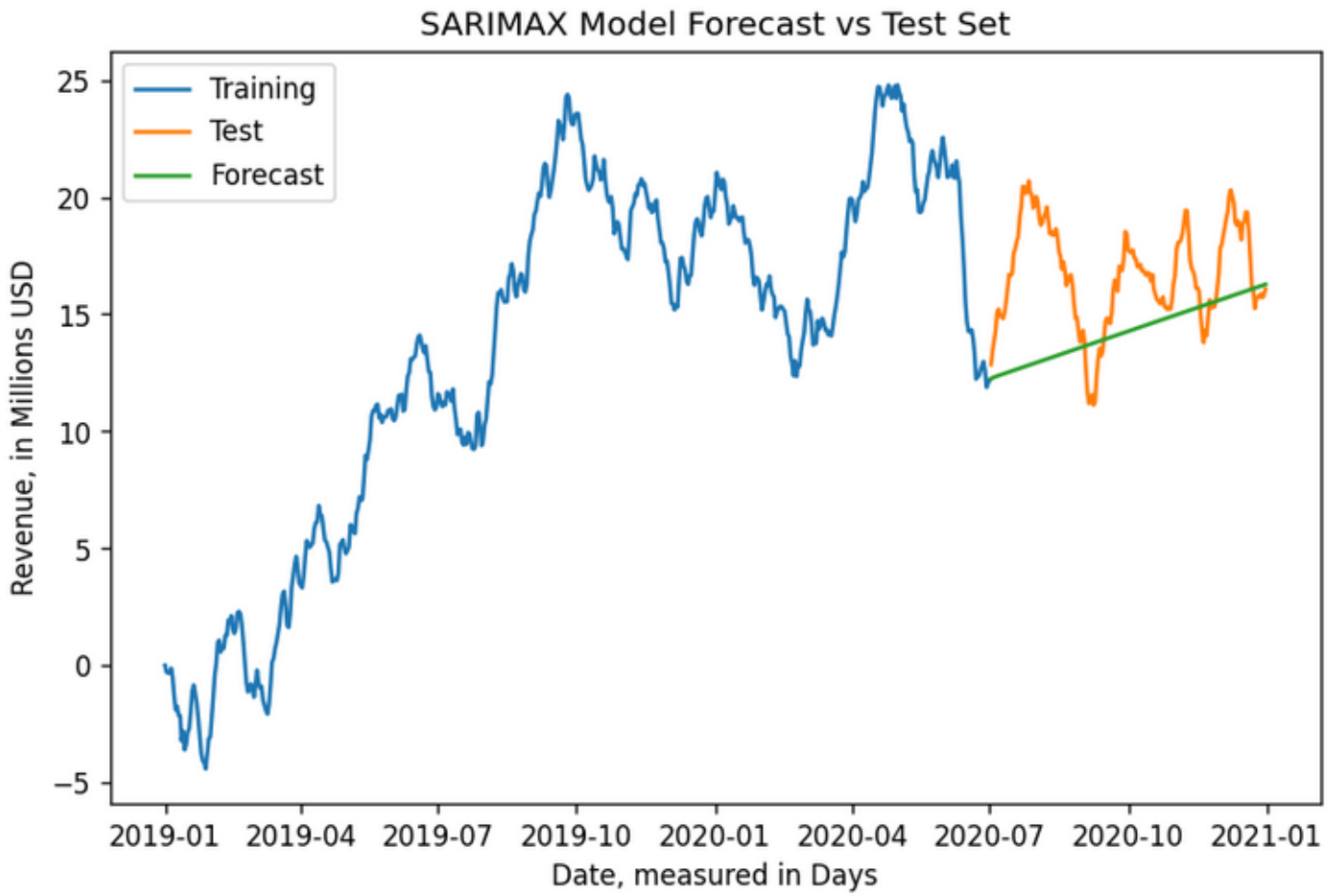
```
]: # Fit the model using auto_arima
   # Auto-arima code reference (6. Tips to using auto_arima — pmdarima 2.0.1 documentation, n.d.)
   # Additional code reference (Pmdarima.arima.AutoARIMA — pmdarima 2.0.1 documentation, n.d.)
   # Auto-arima, initial parameter attempt
   # Code Reference (Kosaka, 2021)
   from pmdarima.arima import StepwiseContext
   from pmdarima.arima import auto_arima


   model = auto_arima(train, start_p=1, start_q=1,
                      test='adf',
                      max_p=4,
                      max_q=4,
                      m=1,
                      d=1,
                      seasonal=True,
                      stationarity=False,
                      seasonal_test='ocsb',
                      start_P=0,
                      D=0,
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True,
                      stepwise=True,
                     trend='c')
   print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=672.789, Time=0.21 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=767.938, Time=0.18 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=671.106, Time=0.12 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=691.699, Time=0.11 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=767.938, Time=0.15 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=672.640, Time=0.14 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=671.553, Time=0.44 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=671.106, Time=0.07 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 1.442 seconds
                               SARIMAX Results
==========================================================================================
Dep. Variable:                          y   No. Observations:                  548
Model:               SARIMAX(1, 1, 0)   Log Likelihood                -332.553
Date:                Mon, 26 Sep 2022   AIC                           671.106
Time:                        14:36:40   BIC                           684.019
Sample:                    01-01-2019   HQIC                          676.154
                         - 07-01-2020
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0131      0.019      0.684      0.494      -0.024       0.050
ar.L1          0.4063      0.039     10.399      0.000       0.330       0.483
sigma2         0.1974      0.013     15.436      0.000       0.172       0.223
===================================================================================
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):                 1.58
Prob(Q):                              0.79   Prob(JB):                         0.45
Heteroskedasticity (H):               1.06   Skew:                            -0.03
Prob(H) (two-sided):                  0.69   Kurtosis:                         2.74
===================================================================================
```

Forecasting using ARIMA Model

Provided below is a screenshot of the forecast from the SARIMAX model from section D2. The predicted values for the model are stored in variable 'forecast'. There are 183 predictions, each representing one Day (24 hour period).

```
# Prediction assignment, predicted revenue column named
# Training, Test, and Predicted data plotted together
# Code Reference (Matplotlib.pyplot.plot — Matplotlib 3.6.0 documentation, n.d.)

forecast = pd.DataFrame(model.predict(n_periods = 183),index=test.index)
forecast.columns = ['predicted revenue']

plt.figure(figsize=(8,5))
plt.plot(train,label="Training")
plt.xlabel('Date, measured in Days')
plt.ylabel('Revenue, in Millions USD')
plt.title('SARIMAX Model Forecast vs Test Set')
plt.plot(test,label="Test")
plt.plot(prediction,label="Forecast")
plt.legend(loc = 'upper left')
#plt.savefig('SecondPrection.jpg')
plt.show()
```



Output and Calculations

All inputs, outputs, and calculations used to perform this analysis are included in the accompanying PDF file containing a copy of my Jupyter Notebook.

Code

A PDF copy of my programming environment is included with this submission. The PDF contains a copy of my JupyterNotebook with all inputs, outputs, and calculations used to support the implementation of this time series model and analysis.

Results

The overall results of the analysis shows that the best model is a SARIMAX model where ARIMA(p, d, q)(P, D, Q)[# Periods] is ARIMA (1, 1, 0)(0, 0, 0)[0]. This model returns an AIC of 671.106. The model is evaluated using Root Mean Square Error (RMSE) and Mean Absolute Error (MAE). The RMSE of the model is 3.37525 and the MAE is 2.71661. The prediction interval of the model is 1 day, with a total 183 forecasted days. This number of forecasted days was chosen to match the test dataset. This allows for direct and equal analysis of the forecasted figures to the actual figures in the test dataset.

Discussed below in detail are: (1) The reasons for selecting this model, (2) The prediction interval of the forecast, (3) The justification of the forecast length, and (4) The model evaluation procedure and error metrics.

Selection of the ARIMA model

The process of selecting the appropriate ARIMA model begins in the research stage of this analysis. The 'd' term of the ARIMA model represents the number of nonseasonal differences. It was evaluated by differencing the time series and plotting the series for first and second order differencing. The 'd' term is also selected using pmdarima's 'ndiffs' function. Both of these research techniques point to the appropriate value for 'd' in this analysis to be 1.

The time series is decomposed using statsmodel's seasonal_decompose to see the trend, seasonality, and residual distribution. This helps us identify the seasonality of the time series, and provides a visual identification that the residuals are not autocorrelated.

The 'q' term of the ARIMA model is the number of lagged forecast errors in the prediction equation. Autocorrelation function (ACF) is used to find the order of MA term 'q' in the ARIMA model.

The 'p' term of the ARIMA model is the number of autoregressive terms. Partial Autocorrelation Function (PACF) is used to find the 'p' term.

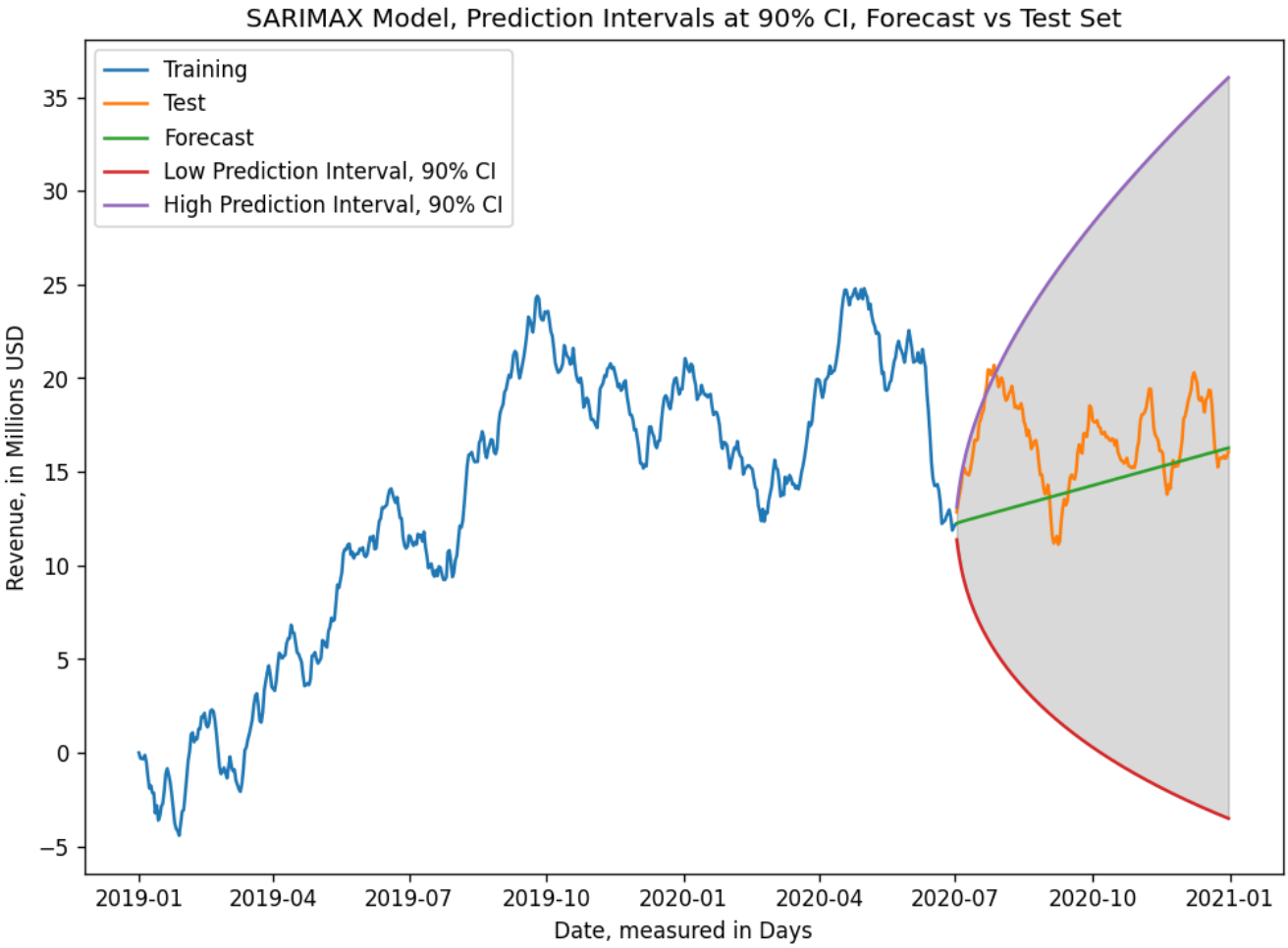
Prediction interval of the forecast

Prediction intervals are a very important aspect to take into consideration when predicting future values. No matter how much statistical data is recorded and trained on, future observations always come with variation and uncertainty. A prediction interval is a range of possible values for the next single observation that is most likely as forecasted by the history of the previous observations (Lewis, 2022).

Every prediction interval comes attached with a certain level of confidence. This confidence interval (CI) is a percentage that identifies the degree of likelihood that the forecasted observation will be within the predicted range. The higher the selected CI, the larger the potential range of predicted values, or said another way the larger the prediction intervals are. For this analysis the CI selected was 90%. This means that with 90% certainty the forecasted daily revenue value will lie within the prediction intervals for each data forecast.

For the application of the prediction intervals, the range (low prediction and high prediction @ 90% CI) was calculated using the model.predict function. This returns an array of the low-and-high predictions for each daily forecast from our model. The prediction intervals are converted to a pandas dataframe and assigned to individual variables: low_prediction and high_prediction. Next, the variables were read out to a .csv file and a day column is added that corresponds to the 'Day' index from our test set. This ensures that when the prediction interval is graphed the data will correspond to the correct time frame from our dataset. The variables are read back into our Jupyter Notebook, and column 'Day' is Indexed and converted to datetime.

The predicted intervals are now prepared to be accurately visualized alongside both our forecast and actual values. Provided below is a screenshot of the plotted prediction intervals corresponding to our forecasts and the actual revenue value. The shaded area represents the Prediction Interval at 90% confidence that the forecast value will fall within the shaded area.



Justification of the forecast length

The forecast length of a model is guided by the size of the time series provided for the model build. According to Professor Elleh of WGU, for a 2-year daily revenue data, we can only forecast up to 1 year of future revenue: but predictions shorter than this will be more accurate (Elleh, 2022). For our model, where the training data size was 548 days, it was decided to forecast the model for 183 days. This forecast aligns with the test data size, as well as with Professor Elleh's recommendation of forecast length. The model forecast predicts using one-step-ahead predictions. This allows our forecast to be accurately compared to the actual values from our test data for the time series.

Model evaluation procedure and error metric

Auto-Arima was performed to find a suitable seasonal order. Auto-Arima also was set to select the ARIMA model with the lowest AIC score.

Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) were used to evaluate the error metric of the model. MAE considers the absolute value of error terms from our model and then takes the mean of them. This allows us to know on average how much the predictions deviated from the actual values. RMSE considers the square of the error terms and then takes their mean (Immidi, 2020).

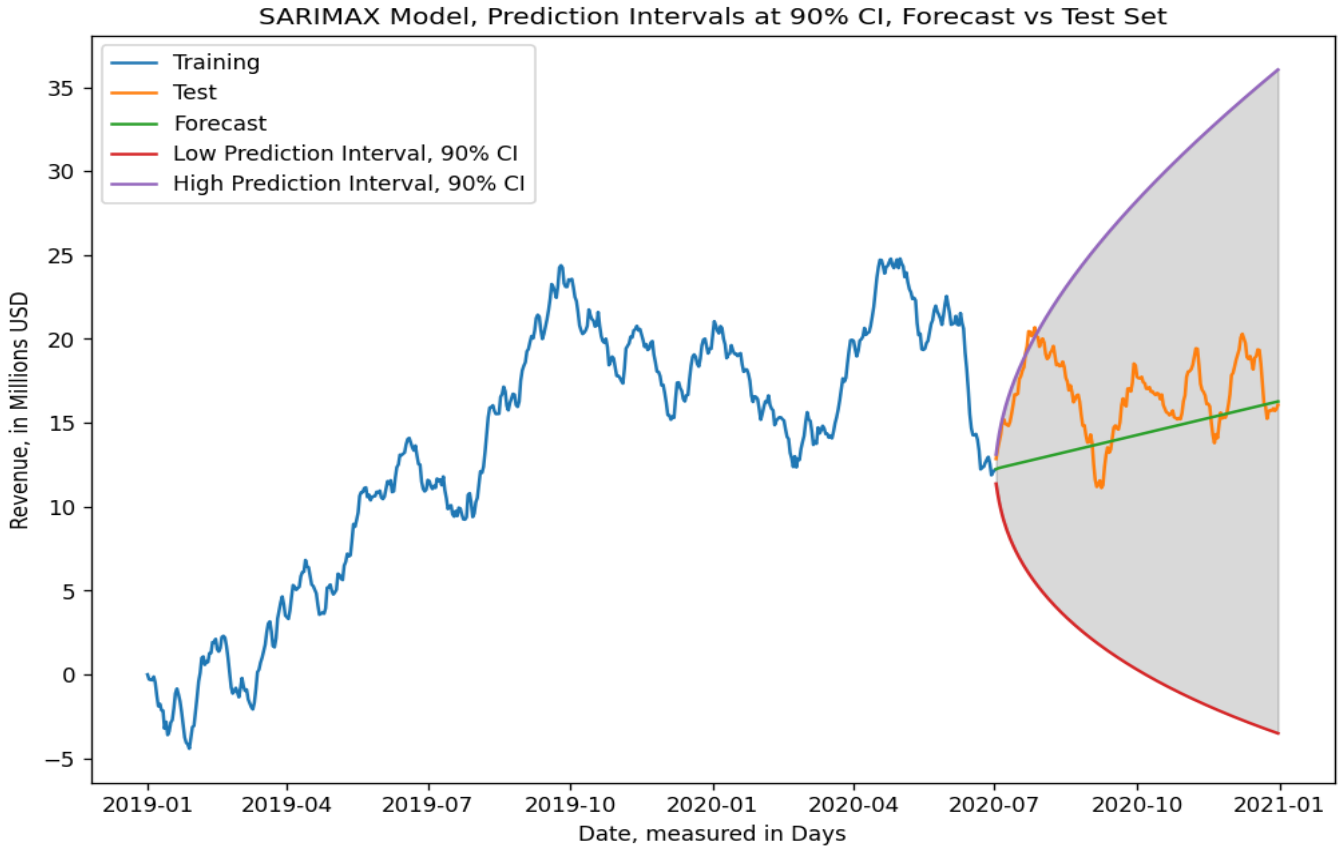
The RMSE of the selected SARIMAX model forecasts is 3.37525. The MAE of the selected ARIMA model is 2.71661. The closer that both evaluation metrics are to 0, is how we evaluate the validity of the model. When viewing the graph of the model forecast we can see that the model fit is not optimal, but is accurate in the direction and overall pricing. The model can be said to exhibit low bias, but with variance: and the low RMSE and MAE values evaluate this correctly.

Annotated Visualization

Provided below is a complete annotated visualization of the final forecast model. This model shows the line of the training data (blue line, 548 data points), and compares the test (orange line, 183 data points) data with the forecasted (green line, 183 data points) data. The Prediction Interval is represented by the shaded area of the graph. The Red line represents the Low Prediction Interval at 90% CI. The Purple line represents the High Prediction Interval at 90% CI.

SARIMAX Model Forecast, With Prediction Interval (CI 90%), Vs Test Set

59]:
```
# Prediction assignment, predicted revenue column named
# Training, Test, and Predicted data plotted together
# Code Reference (Matplotlib.pyplot — Matplotlib 3.6.0 documentation, n.d.)

# Creating varible with forecast values
forecast = pd.DataFrame(model.predict(n_periods = 183),index=test.index)

# Naming forecast_revenue column in forecast variable
forecast.columns = ['forecast_revenue']

# Establish plot parameters for Forecast

# Plot figure size
plt.figure(figsize=(10,7))

# Training data
plt.plot(train,label="Training")

# Annotate X-axis label
plt.xlabel('Date, measured in Days')

# Annotate Y-axis label
plt.ylabel('Revenue, in Millions USD')

# Annotate Plot Title
plt.title('SARIMAX Model, Prediction Intervals at 90% CI, Forecast vs Test Set')

# Plot Test Data
plt.plot(test,label="Test")

# Plot Forecast Data
plt.plot(forecast,label="Forecast")

# Add Prediction Interval at 95% CI
plt.plot(low_pred,label='Low Prediction Interval, 90% CI')
plt.plot(high_pred,label='High Prediction Interval, 90% CI')
plt.fill_between(low_pred.index, low_pred['Revenue'], high_pred['Revenue'], color='k', alpha=.15)

# Plot legend in upper lefthand corner
plt.legend(loc = 'upper left')


# Show Plot
plt.show()
```



Recommendations

To restate the research question for this analysis:

"Using Time Series Analysis, can we create a model that can be used to predict future revenues, which can be used by our Executive Team to better address readmission?"

In answer to this question, my recommendation is that this time series model can be used to predict future revenue, but with special consideration taken for the length of time out the forecast is. As we can see from the visualization in E2, the model forecasts accurately when looking at longer horizons, such as six months. However, it's not suitable for daily revenue forecasting because it doesn't do a good job at predicting the noise and variations in how our actual revenue stream looks.

My first specific recommendation is for our Executive Team to use this model for mid and long term business decisions. The model has the potential to provide valuable insight on future revenue streams, which can in turn give our Executives a valuable tool for helping decide large decisions. Examples of such decisions could be things such as when to expand operations based on revenue generation or if there's a need to raise additional capital to support operating costs. These decisions in turn can have a direct effect and influence on lowering readmission rates.

My second specific recommendation is to use this model with reservation. The initial reason for our executives wanting to see a time series model on revenue was to help address readmission, as part of the readmission project. If this model is adequate for future revenue forecasts then by all means use it. If not, further testing and development will need to be taken to create a more sophisticated model. This comes at the expense of time and labor from our analyst team. The pros and cons of this will need to be weighed in accordance to the urgency and value that this project brings to our Executive Team, the organization, and our shareholders in comparison to our other projects. If other projects in the pipeline have the potential to offer a greater return and chance at lowering readmission rates, we should pursue those before allocating more time to this analysis.

Reporting

Provided with this submission is a PDF of the Jupyter Notebook IDE used to generate this analysis. This was accessed through JupyterLab, which is the latest web-based Interactive Development Environment

for notebooks, code, and data analysis. Jupyter Notebook is an open source web application maintained by Project Jupyter.

Sources For 3rd Party Code

Making time series stationary | Python. (n.d.). Retrieved from https://campus.datacamp.com/courses/arima-models-in-python/chapter-1-arma-models?ex=5

Boston, S. (2020, October 31). ValueError: You must specify a period or X must be a pandas object with a DatetimeIndex with a freq not set to none. Retrieved from https://stackoverflow.com/questions/64617482/valueerror-you-must-specify-a-period-or-x-must-be-a-pandas-object-with-a-dateti

6. Tips to using auto_arima — pmdarima 2.0.1 documentation. (n.d.). Retrieved from https://alkaline-ml.com/pmdarima/tips_and_tricks.html#period

Pmdarima.arima.AutoARIMA — pmdarima 2.0.1 documentation. (n.d.). Retrieved from https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.AutoARIMA.html#pmdarima.arima.AutoARIMA

Matplotlib.pyplot.plot — Matplotlib 3.6.0 documentation. (n.d.). Retrieved from https://matplotlib.org/stable/api/_as_gen/matplotlib.pyplot.plot.html

Verma, Y. (2021). Complete guide to SARIMAX in Python for time series modeling. Retrieved from https://analyticsindiamag.com/complete-guide-to-sarimax-in-python-for-time-series-modeling/

Smith, T. G. (2019, December 18). Forecasting the stock market with pmdarima. Retrieved from https://alkaline-ml.com/2019-12-18-pmdarima-1-5-2/

Kosaka, M. (2021, March 3). Efficient time-series using Python's Pmdarima library. Retrieved from https://towardsdatascience.com/efficient-time-series-using-pythons-pmdarima-library-f6825407b7f0

H Sources

Elleh, Festus. (2022, June 19). D213 T1 Jun 19 2022. Retrieved from https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=efceba6c-e8ef-47a2-b859-aec400fe18e7

Augmented Dickey–Fuller test. (2022, August 25). Retrieved from https://en.wikipedia.org/wiki/Augmented_Dickey%E2%80%93Fuller_test

Dail. (2011, August 28). How to test the autocorrelation of the residuals? Retrieved from https://stats.stackexchange.com/questions/14914/how-to-test-the-autocorrelation-of-the-residuals

Immidi, S. V. (2020, September 4). Error metrics used in time series forecasting modeling. Retrieved from https://medium.com/analytics-vidhya/error-metrics-used-in-time-series-forecasting-modeling-9f068bdd31ca

Pmdarima.arima.auto_arima — pmdarima 2.0.1 documentation. (n.d.). Retrieved from https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.arima.auto_arima.html

Lewis, M. (2022, March 24). What is a Prediction Interval? Retrieved from https://study.com/learn/lesson/prediction-interval-overview-formula-examples.html