D213: Advanced Data Analytics

Time Series Analysis of WGU Telecom and Revenue Forecasts

Andrew Mecchi

Masters of Data Analytics, Western Governors University

The following report covers the Performance Assessment for D213: Advanced Data Analytics. This document is categorized by the questions defined in the rubric.

A: Research Question

A1. Research question — Page 2

A2. Goals of time series analysis — Page 2

B: Method Justification

B1. Assumptions of time series modeling — Page 2

C. Data Preparation

C1. Visualization of time series — Page 3

C2. Description of timestamp formatting — Page 3

C3. Evaluation of stationarity — Page 4

C4. Steps of preparation for analysis — Page 5

C5. Copy of cleaned train and test sets — SEE ATTACHED — Page 10

D. Model Identification and Analysis

D1. Decomposition, seasonality, and trend analysis — Page 11

D2. Identification of ARIMA model — Page 14

D3. ARIMA forecast — Page 15

D4. Model output and calculations — Page 17

D5. Code used for analysis — SEE ATTACHED — Page 23

E. Data Summary and Implications

E1. Discussion of time series model results and analysis — Page 23

E2. Final model visualization — Page 26

E3. Recommended course of action based on results — Page 26

F. Panopto Video

F1. Panopto Presentation — SEE ATTACHED — Page ?

G: Sources and Acknowledgements

G1. Code Sources – Page ?

G1. Literary Sources – Page ?

Part I: Research Question

A1. 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.

Through the use of a time series analysis of the previous two-years of data, can daily revenues for the next two quarters (Q1 & Q2) be accurately forecasted?

A2. Define the objectives or goals of the data analysis. Ensure your objectives or goals are reasonable within the scope of the scenario and are represented in the available data.

The goal of this analysis is to build a time series model using WGU Telecom's data containing daily revenues from January 2019 through December 2020. Through the analysis of observable trends, seasonality, correlations, and data decomposition; an ARIMA model will be employed on stationary data to create a predictive model for projecting revenues for the upcoming year.

Part II: Method Justification

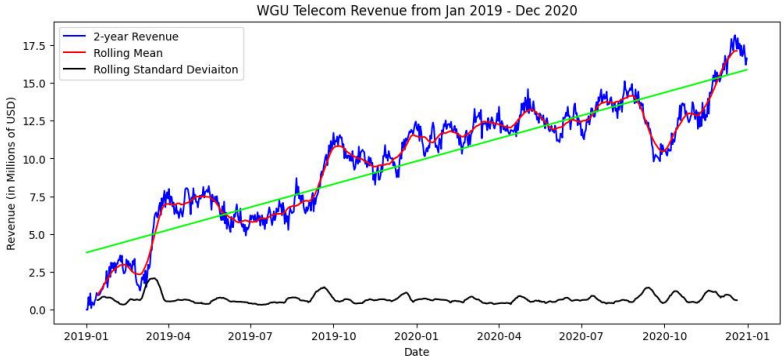
B1. Summarize the assumptions of a time series model including stationarity and autocorrelated data.

Prior to employing an ARIMA model to conduct a time series analysis on the WGU Telecom data, the assumptions of the analysis must be understood. The primary assumption of time series analysis is that the data must be stationary with zero trends; specifically, the mean, variance, and autocorrelation of the data must be constant over time (Statistics Solutions, 2023). The stationarity of the data is characterized by the statistical properties of a process generating a time series that do not change over time, the data itself can change over time, however, the way it changes cannot (Affek, 2019). Also, the autocorrelation of the data refers to the mathematical representation of the degree of similarity between a given time series and a lagged version of itself over successive time intervals; therefore, the autocorrelation must be constant over time to determine stationarity (Smith, 2023). Additionally, the error term is randomly distributed, time series error and residuals are uncorrelated, and the dataset must be free of outliers as they can lead to inaccurate results (Elleh, n.d.). When these assumptions have been met, the data can proceed to modeling and forecasting.

Part III: Data Preparation

C1. Provide a line graph visualizing the realization of the time series.

Figure 1: WGU Telecom Revenue Trend Visualization



C2. Describe the time step formatting of the realization, including *any* gaps in measurement and the length of the sequence.

The provided WGU Telecom dataset summarized the revenue stream of the first two years of business operations. The data dictionary described two columns, Day and Revenue, and contained 731 records which is indicative of two years (730 days) and a day. Therefore, before the datetime conversion, the two years covered must include a leap year in one of those years; thus, confirming no gaps in measurement. Once the dataset was imported, records were explored to identify any missing values or duplicate entries. Following confirmation of no missing values or duplicate entries, the “Day” column was converted into a time series object (Fig. 2). Given the data spanned two years and contained a leap year, it was determined that the dataset covered the years of 2019 through 2020 (the later being the leap year). Figure 3 represents the affirmation of no duplicate or missing data and provides a plot of the time series object.

Figure 2: Conversion of data to Time Series Object

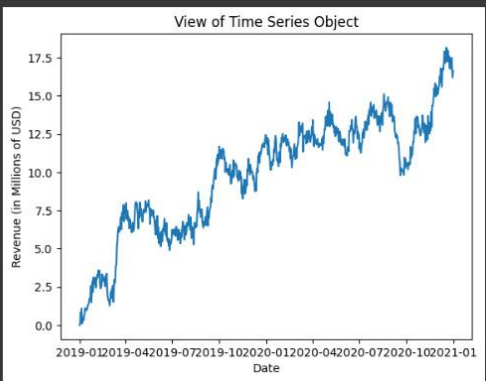
```
[46] # Convert day in to datetime format
     # SOURCE --- (Elleh, n.d)
     df['Day'] = (pd.date_range(start = pd.to_datetime("2019/1/1"),
                  periods = df.shape[0], freq='24H'))

     # Set the Date as an index
     df.set_index('Day',inplace=True)

     # View df after datetime conversion --- should be 731 x 1
     df
```

Day	Revenue
2019-01-01	0.000000
2019-01-02	0.000793
2019-01-03	0.825542
2019-01-04	0.320332
2019-01-05	1.082554
...	...
2020-12-27	16.931559
2020-12-28	17.490666
2020-12-29	16.803638
2020-12-30	16.194813
2020-12-31	16.620798

731 rows × 1 columns

Figure 3: Time Series Object Conversion Plot

```
[38] # Check for null values or missing data
     df.isnull().sum()

     Day        0
     Revenue    0
     dtype: int64

[39] # Check for duplicate values
     df.duplicated().sum()

     0

     # Plot of df
     plt.plot(df)
     plt.title("View of Time Series Object")
     plt.ylabel("Revenue (in Millions of USD)")
     plt.xlabel("Date")
     plt.show()
```



C3. Evaluate the stationarity of the time series.

When observing the visualizations of Figures 1 and 3, it is apparent the data shows an upward trend in revenue, thus indicating the data is not stationary. To analyze the stationarity of the time series mathematically, the augmented Dickey-Fuller test was calculated on the data series (Fig. 4). As stated earlier, in order to conduct an ARIMA time series analysis, the data must be stationary. The augmented Dickey Fuller test confirmed the dataset is indeed non-stationary, resulting with a test statistic of -1.92 and a p-value of 0.32 (alpha = 0.05). Therefore, prior to defining the time series model, differencing must be used to enforce stationarity.

Figure 4: Assess Stationarity with Augmented Dickey-Fuller Test

```
[56] # Compute agumented dickey-fuller to mathmatically test for stationarity of dataset
     # Default autolag argument measurment is AIC
     dickey_fuller_results = adfuller(df['Revenue'])

[57] # Dickey Fuller Results
     # Source --- (Statsmodels, n.d.)
     df_test_output = pd.Series(dickey_fuller_results[0:4], index = ['Test Stastic', 'p-value', 'Number of Lags', 'Number of Observations'])

     for key, value in dickey_fuller_results[4].items():
             df_test_output["Critical Value (%s)" % key] = value

     # View Dickey-Fuller resutls
     print(df_test_output)

     Test Stastic             -1.924612
     p-value                   0.320573
     Number of Lags            1.000000
     Number of Observations  729.000000
     Critical Value (1%)      -3.439352
     Critical Value (5%)      -2.865513
     Critical Value (10%)     -2.568886
     dtype: float64
```

C4. Explain the steps you used to prepare the data for analysis, including the training and test set split.

1) Import WGU Telecom Revenue Dataset

```
# Import dataset
df =
pd.read_csv(r'C:\Users\andrew\Desktop\WGU_MSDA\D213_Advanced_Data_Analytics\data sets\churn\teleco_time_series.csv')
```

2) Search for missing values

```
# Check for null values or missing data
df.isnull().sum()
```

3) Identify any potential duplicate entries

```
# Check for duplicate values
df.duplicated().sum()
```

4) Prepare data for time series modeling. Convert the “Day” column into a datetime time series object and set “Day” as the index.

```
# Convert day in to datetime format
# SOURCE --- (Elleh, n.d)
df['Day'] = (pd.date_range(start = pd.to_datetime("2019/1/1"),
             periods = df.shape[0], freq='24H'))

# Set the Date as an index
df.set_index('Day',inplace=True)
```

5) View summary statistics of dataframe, look for potential outliers as time series analysis can yield inaccurate results with inclusion of outlying data.

```
# View summary statistics of df
df.describe()
```

	Revenue
count	731.000000
mean	9.822901
std	3.852645
min	0.000000
25%	6.872836
50%	10.785571
75%	12.566911
max	18.154769

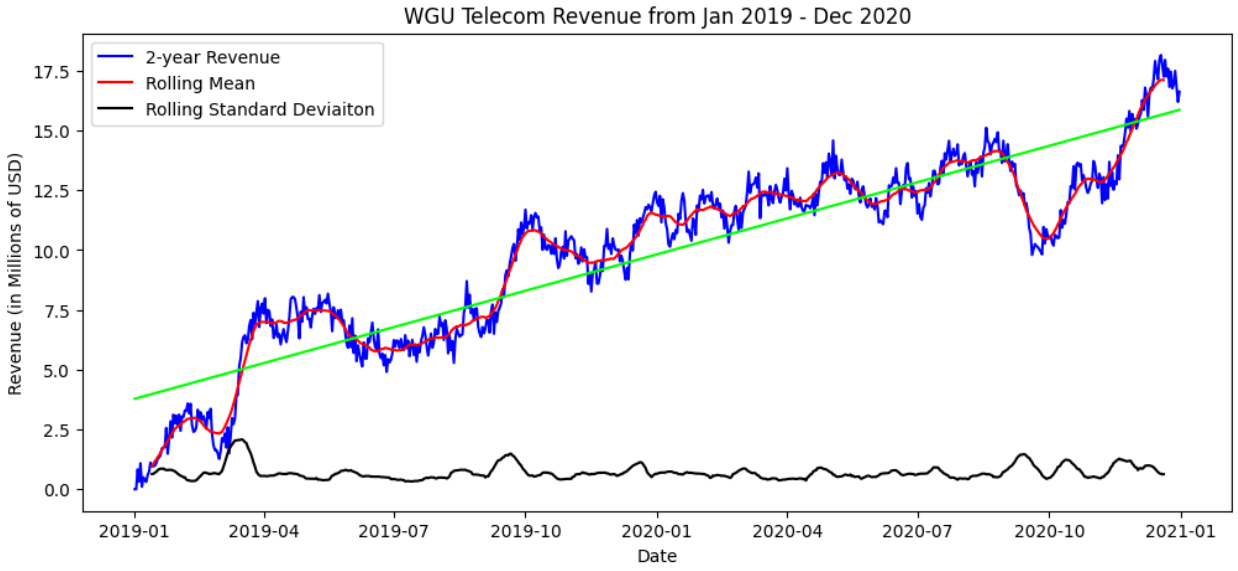
6) Calculate the rolling mean and standard deviation in preparation of data visualization.

```
# Compute rolling Mean and Rolling Standard Deviation
# Data covers 731 rows or two years of data with a leap year, window set
for 24 months
rolling_mean = df['Revenue'].rolling(window = 24, center = True).mean()
rolling_stdv = df['Revenue'].rolling(window = 24, center = True).std()
```

7) Plot revenue data with rolling mean and standard deviation. Add a trend line to observe any potential trends confounding stationarity. The resulting data confirmed an upward trend in revenue, thus, visually determined data is not stationary.

```
# Explore the data by Visualizing the data.
plt.figure(figsize=(12,5))
org = plt.plot(df['Revenue'], color = 'blue', label = '2-year Revenue')
r_mean = plt.plot(rolling_mean, color = 'red', label = 'Rolling Mean')
r_std = plt.plot(rolling_stdv, color = 'black', label = 'Rolling Standard
Deviation')
plt.legend(loc = 'best')
plt.title("WGU Telecom Revenue from Jan 2019 - Dec 2020")
plt.xlabel("Date")
plt.ylabel("Revenue (in Millions of USD)")
```

```
# Create a trend line
# Source --- (Zach, 2022)
x = mdates.date2num(df.index)
y = df['Revenue']
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
# Plot trendline
plt.plot(x, p(x), color = 'lime')
plt.show()

# Data shows an upward trend
```



8) Implement augmented Dickey-Fuller test to compute stationarity mathematically. View results of the test and observe test statistic and p-value. Test statistic -1.92, p-value 0.32 - data confirmed non-stationary.

```
# Compute augmented dickey-fuller to mathematically test for stationarity
of dataset
# Default autolag argument measurement is AIC
dickey_fuller_results = adfuller(df['Revenue'])

# Dickey Fuller Results
# Source --- (Statsmodels, n.d.)
df_test_output = pd.Series(dickey_fuller_results[0:4], index = ['Test
Statistic', 'p-value', 'Number of Lags', 'Number of Observations'])

for key, value in dickey_fuller_results[4].items():
        df_test_output["Critical Value (%s)" % key] = value
```

```
# View Dickey-Fuller results
print(df_test_output)
```

```
Test Statistic            -1.924612
p-value                    0.320573
Number of Lags             1.000000
Number of Observations   729.000000
Critical Value (1%)       -3.439352
Critical Value (5%)       -2.865513
Critical Value (10%)      -2.568886
dtype: float64
```

9) In order to perform an ARIMA model, differencing must be applied to the non-stationary dataset. Detrending the data also results with a null value for the first row as there is no previous data to difference, thus nulls must also be removed from the differenced dataset.

```
# P-value above alpha of 0.05, must reject null hypothesis and proceed to
make data stationary
# Detrend data using diff() function to apply differencing
df_difference = df.diff()

# Differencing will result with null value for first value as no previous
value to difference, proceed to drop nulls
df_difference = df_difference.dropna()
```

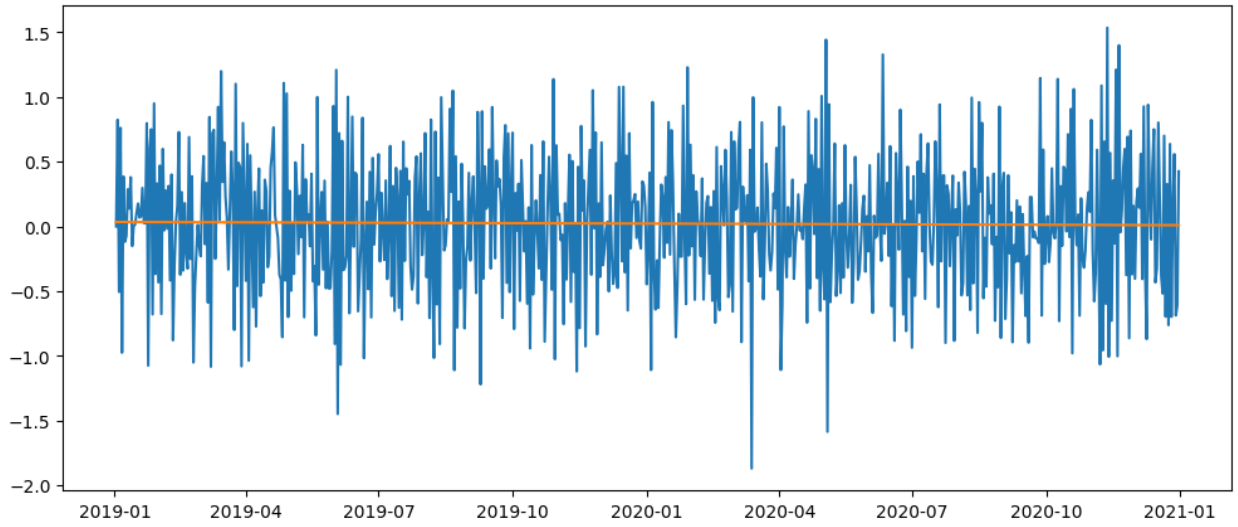
10) Apply augmented Dickey-Fuller test on the differenced dataset to observe for any stationarity. Results confirmed that detrending the data is now stationary; test statistic of -44.87 and a p-value of 0.000.

```
# Compute augmented dickey-fuller test for stationarity of dataset
diff_results = adfuller(df_difference['Revenue'])

# Get results of differencing
diff_test_output = pd.Series(diff_results[0:4], index = ['Test Statistic',
'p-value', 'Number of Lags Used', 'Number of Observations'])

for key, value in diff_results[4].items():
        diff_test_output["Critical Value (%s)" % key] = value

# View Dickey-Fuller results
print(diff_test_output)

# Very good test statistic, and critical values well below test stat are
signs of stationary data
```

```
Test Statistic                -44.874527
p-value                         0.000000
Number of Lags Used             0.000000
Number of Observations        729.000000
Critical Value (1%)            -3.439352
Critical Value (5%)            -2.865513
Critical Value (10%)           -2.568886
dtype: float64
```

11) Visualize plot of stationary data with trendline.

```
# Plot of Stationary data
plt.figure(figsize=(12,5))
plt.plot(df_difference)
x = mdates.date2num(df_difference.index)
y = df_difference['Revenue']
z = np.polyfit(x, y, 1)
p = np.poly1d(z)
# Plot trendline
plt.plot(x, p(x))
plt.show()
```



12) Preparation of train/test split. Calculate the values to split data. Train = 585 (note, this value is rounded up) Train = 146. Therefore, the value of the train set was used to split the data into train and test sets.

```
# Prepare data for test and train sets using an 80/20 split
# Data will be split in sequential order for model

train_size = round(len(df)*0.8)
```

```
test_size = round(len(df)*0.2)

print("Train set size: " + str(train_size))
print("Test set size: " + str(test_size))
```

```
Train set size: 585
Test set size: 146
```

13) Split data based on results from 80/20 split calculations. The train results were rounded up to 585, therefore the size of the test set was used to define each set (146). Results of shape: Train Set Shape: (584, 1) and Test Set Shape: (146, 1). For accurate modeling, the split must not be random, instead, the first 80% of the data will train the model and the remaining 20% represents the test set.

```
# Split stationary detrended data into train and test sets
# First 80% of data for train, and last 20% for test sets
train = df_difference.iloc[:-146]
test = df_difference.iloc[-146:]

# Confirm accurate split for train/test sets
print("Train Set Shape: " + str(train.shape))
print("Test Set Shape: " + str(test.shape))
```

```
Train Set Shape: (584, 1)
Test Set Shape: (146, 1)
```

14) Export cleaned and prepared train and test sets before analysis.

```
# Export clean and prepared train and test sets
train.to_csv(r'C:\\Users\\andrew\\Desktop\\WGU_MSDA\\D213_Advanced_Data_An
alytics\\PA\\Task_1\\train.csv')
test.to_csv(r'C:\\Users\\andrew\\Desktop\\WGU_MSDA\\D213_Advanced_Data_Ana
lytics\\PA\\Task_1\\test.csv')
```

C5. Provide a copy of the cleaned data set.

SEE ATTACHED: train.csv & test.csv

Part IV: Model Identification and Analysis

D1. Report the annotated findings with visualizations of your data analysis, including the following elements:

Seasonality: the presence or lack of a seasonal component

Seasonality is a cyclical change that occurs regularly over a period of time and repeats in a predictable fashion. To plot seasonality, Pythons' seasonal_decompose method is applied to the differenced time series data. The seasonal plot of the decomposed data appears to be seasonal when visually observing the plot, however, it is worth noting that the values of seasonality range from approximately 0.13 to -0.15 (Fig. 5). To rule out seasonality, the auto_arima method was applied to the same data and the results determined there was no seasonality detected. Auto_arima would yield a value in the results as highlighted in Figure 6 if seasonality was detected and the value would represent the "cycle" of seasonality. The results yielded a value of zero, thus, no seasonal component was identified.

Figure 5: Seasonality Plot of WGU Telecom Revenue Jan 2019 - Dec 2020

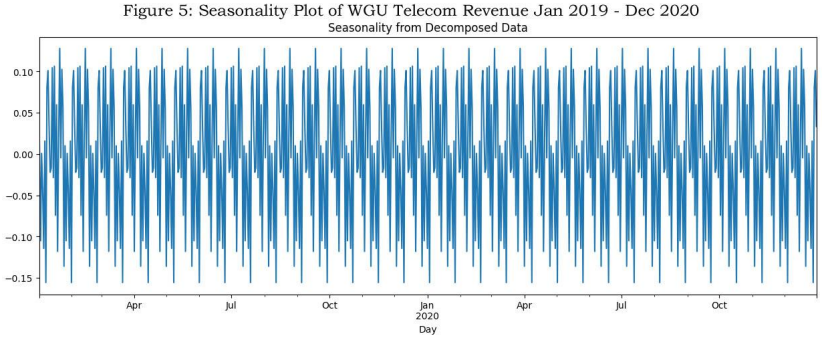


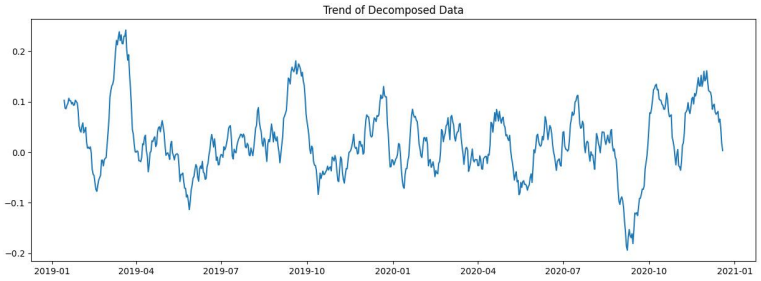
Figure 6: Auto-arima Results Confirming no Seasonality

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=987.305, Time=1.71 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=1162.819, Time=0.12 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=983.122, Time=0.11 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=1019.369, Time=0.17 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=1162.139, Time=0.10 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=985.104, Time=0.21 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=985.106, Time=0.13 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=986.045, Time=1.35 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=984.710, Time=0.09 sec
                                    Seasonality value would indicate "cycle" of seasonal effects.
Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 4.033 seconds
                           SARIMAX Results
Dep. Variable:                      y   No. Observations:                  730
Model:               SARIMAX(1, 0, 0)   Log Likelihood                -488.561
Date:                Sat, 16 Sep 2023   AIC                           983.122
Time:                        23:56:35   BIC                           996.901
Sample:                  01-02-2019   HQIC                          988.438
                       - 12-31-2020
Covariance Type:                  opg
                 coef    std err          z      P>|z|      [0.025      0.975]
intercept      0.0332      0.018      1.895      0.058      -0.001       0.068
ar.L1         -0.4692      0.033    -14.296      0.000      -0.534      -0.405
sigma2         0.2232      0.013     17.801      0.000       0.199       0.248
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 2.05
Prob(Q):                              0.96   Prob(JB):                         0.36
Heteroskedasticity (H):               1.02   Skew:                            -0.02
Prob(H) (two-sided):                  0.85   Kurtosis:                         2.74


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

Trend

Based on the plot of the decomposed data trend, there doesn't appear to be a noticeable, continued trend as the overall direction of the plot irregularly oscillates with upwards and downwards movements (Fig. 7).

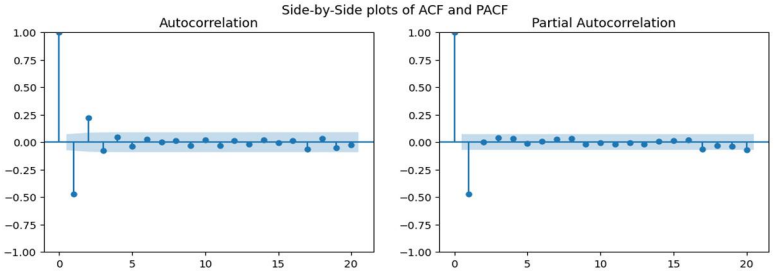
Figure 7: Trend Plot of WGU Telecom Revenue Jan 2019 - Dec 2020



Autocorrelation and Partial Autocorrelation Functions

The autocorrelation function (ACF) represents the correlation of two values at different points within a time series and indicates if a past value influences the current value (Frost, 2022). The autocorrelation plot represents correlation calculations of the time series' observations with observations of previous time steps, or lags. Regarding the WGU Telecom time series, the lags are representative of the influence of the previous day's revenues. The partial autocorrelation function (PACF) is similar to the ACF except that it displays only the correlation between two observations that the shorter lags between those observations do not explain, in other words, the PACF for each lag is the unique correlation between those two observations after partialling out the intervening correlations.(Frost, 2022). The ACF and PACF plots are helpful in determining the stationarity of the time series as well as can aid in defining the autoregressive and moving average values for the ARIMA model. When observing the plots, if an observation resides in the highlighted blue area, the point is statistically insignificant, while observations outside the blue confidence interval are considered statistically significant. If the PACF of the differenced series displays a sharp cutoff, then consider adding an AR term to the model determined by the lag at which the PACF cuts off is the indicated number of AR terms (Nau, n.d.). Based on this rule, the ACF and PACF plots of the differenced WGU Telecom time series suggest an ARIMA model of order AR(1) (Fig. 8).

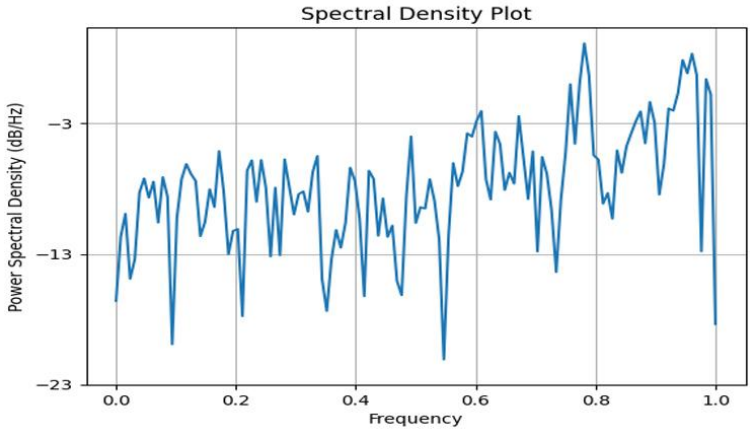
Figure 8: Autocorrelation and Partial Autocorrelation Plots of WGU Telecom Time Series Data



Spectral Density

Often, time series data can exhibit periodic behavior and plotting the spectral density is a technique that allows for the discovery of potential underlying periodicities by measuring time and the variance between the previous and current time periods, or covariance (Jones, 2018). The spectral density is a frequency domain representation of a time series that is directly related to the autocovariance time domain representation whereby the “frequency” is the number of observations before the seasonal pattern repeats (Penn State, n.d.). Figure 9 represents the spectral density of the WGU Telecom time series data.

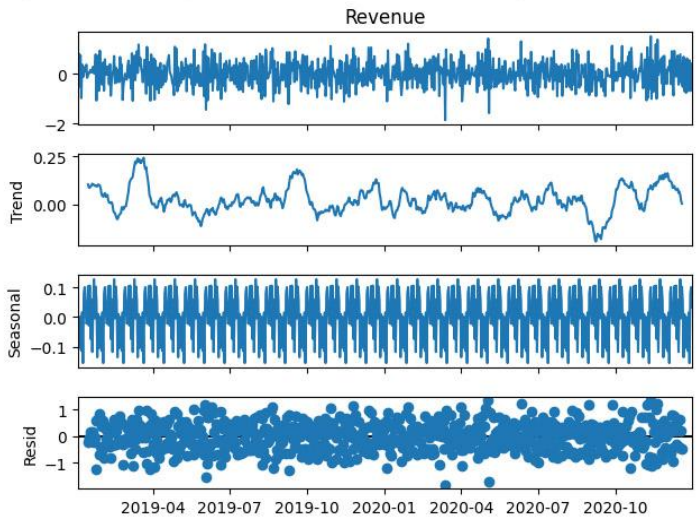
Figure 9: Spectral Density Plot of WGU Telecom Time Series



Decomposed Time Series

The decomposed time series plot yields four plots, the first is the plot of the overall time series followed by decomposed plots of the trend, seasonality, and residuals (from top to bottom) (Fig. 10).

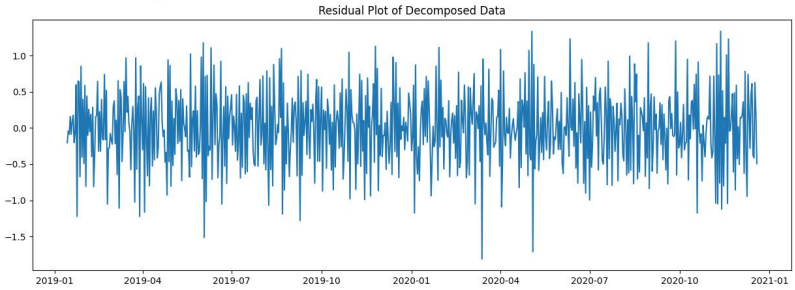
Figure 10: Decomposition Plots: Trend, Seasonality, and Residuals



Residuals - confirm lack of trends

The residual plot represents statistical noise, or, the irregular component consisting of the fluctuations in the time series after removing the previous components (Lewinson, 2022). The residual plot is represented in Figure 11 and shows no apparent trend with a mean centered around zero.

Figure 11: Residual Plot of Decomposed WGU Telecom Time Series



D2. Identify an autoregressive integrated moving average (ARIMA) model that accounts for the observed trend and seasonality of the time series data.

As previously stated, the ACF and PACF plots suggest an order-1 autoregressive model, however, for the most accurate model, the auto_arima function was applied to confirm the best model. The implementation of auto_arima seeks to identify the optimal parameters for the ARIMA model. The algorithm identifies the best fit ARIMA model for the WGU Telecom time series data through a series of calculations and uses the Akaike Information Criteria (AIC) as the performance metric; whereby, the lower the AIC score informs the best performing model (pmdarima, n.d.). The results of the auto_arima confirmed the ARIMA model to be an order-1 autoregressive model with no differencing, and zero-order moving average (Fig. 12). With the optimal model defined, the training dataset is fit to the ARIMA model and result summary completed (Fig. 13).

Figure 12: auto_arima Results - Confirming Best Model - AR(1)

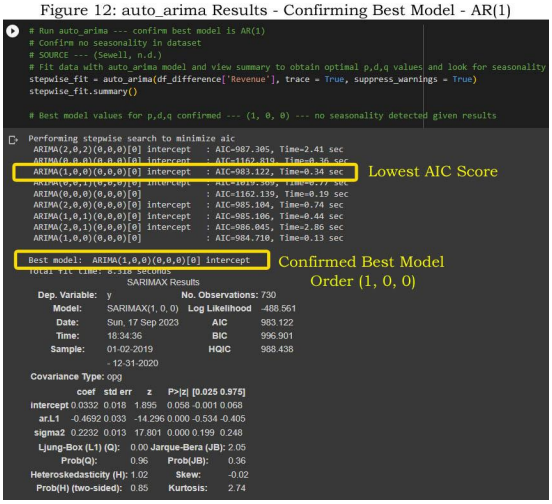


Figure 13: ARIMA Model of Train Dataset with Results

```
# Run training model and fit dataset
model = ARIMA(train['Revenue'], order = (1, 0, 0), freq = 'D')
results = model.fit()

# View result summary
print(results.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  584
Model:                 ARIMA(1, 0, 0)   Log Likelihood                -383.946
Date:                Sun, 17 Sep 2023   AIC                            773.893
Time:                        19:09:23   BIC                            787.002
Sample:                  01-02-2019   HQIC                           779.002
                       - 08-07-2020
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0234      0.013      1.758      0.079      -0.003       0.049
ar.L1         -0.4597      0.036    -12.654      0.000      -0.531      -0.388
sigma2         0.2180      0.014     16.034      0.000       0.191       0.245
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 1.84
Prob(Q):                              0.96   Prob(JB):                         0.40
Heteroskedasticity (H):               0.97   Skew:                            -0.08
Prob(H) (two-sided):                  0.83   Kurtosis:                         2.77
===================================================================================

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

D3. Perform a forecast using the derived ARIMA model identified in part D2.

To perform the forecast using the defined ARIMA model, the results of the fitted train model are used and the get_prediction() method is applied on the remaining 20% of data (146 observations). The predicted mean from the forecasted model were then viewed and plotted (Fig. 14). It is worth noting the values are mean values and level out to a flat line on the plot, therefore, these averages are the expected daily average of revenues, therefore, additional steps are taken to provide a plot of the train and test data sets. These daily revenues are not additive, in that, the cumulative summation of the daily means are not accumulated in a manner that shows the projected increase in revenue. Therefore, a new forecast dataframe is created with these predicted means, applied the cumsum() function to calculate aggregate daily revenues, and then plotted. The get_prediction() function applied to the ARIMA model results also produces confidence intervals and these values were also aggregated in a new dataframe to add to the visualization in Figure 15. Following the plot of the train predictions and test set, the results of the predictive model were compared to the actual values of the test set to assess model accuracy by analyzing the root mean squared error (RMSE) (FIG 16). The RMSE results returned with a fairly accurate model and can now be applied to the two-year data and future forecasts of Q1 and Q2 can be made for the upcoming year.

Figure 14: ARIMA Model Forecast and Predicted Mean Plot

```
# Get forecasted results from model - test data is last 146 values, set start to 584 and end to 729
forecasted = results.get_prediction(start = 584, end = 729, dynamic = True)

# Plot Test data compared to forecasted predicted mean
plt.plot(test)
plt.plot(forecasted.predicted_mean);

# View predicted mean values
print(forecasted.predicted_mean)
```

```
2020-08-08   -0.051234
2020-08-09    0.057239
2020-08-10    0.006341
2020-08-11    0.030223
2020-08-12    0.019017
                ...
2020-12-27    0.022596
2020-12-28    0.022596
2020-12-29    0.022596
2020-12-30    0.022596
2020-12-31    0.022596
Freq: D, Name: predicted_mean, Length: 146, dtype: float64
```

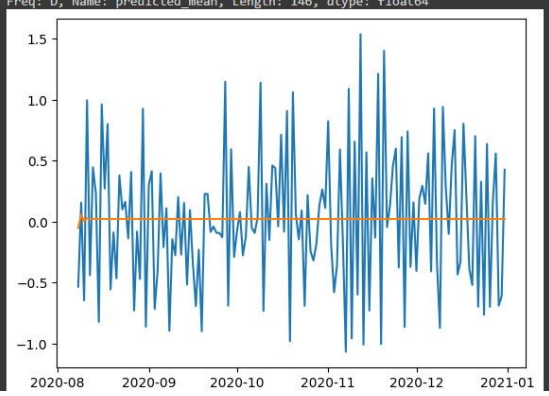


Figure 15: Plot Daily Revenue with Predicted Forecast Relative to Test Data

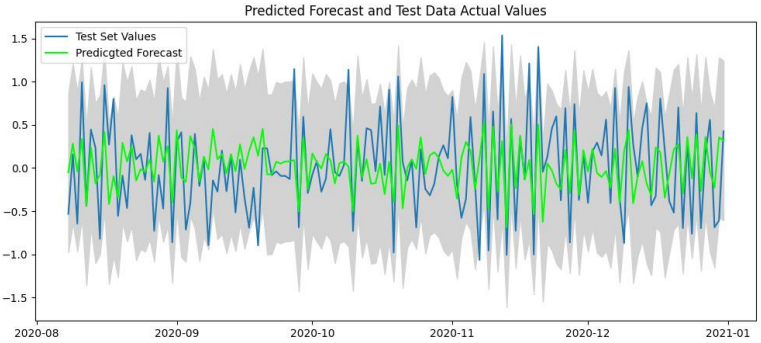


Figure 16: RMSE Calculations for Detrended Data and non-Detrended Data

```
[ ] # Get predictions for RMSE calculation
    predict = results.forecast(len(test))
```

```
[ ] #Calculate RMSE of differenced model

    rmse = sqrt(mean_squared_error(predict, test['Revenue']))
    print(f"The RMSE of the differenced model: {round(rmse,3)}")

    # Confirmed an accurate model
```

```
    The RMSE of the differenced model: 0.57
```

```
[ ] # Calculate root mean squared error of forecasted data against the observed data with no differencing
    rmse_revenue = sqrt(mean_squared_error(df['Revenue'].loc['2020-08-08' : '2020-12-31'], df_forecast['Revenue'].loc['2020-08-08' : '2020-12-31']))
    print(f"The RMSE of the forecasted model: {round(rmse_revenue, 3)}")
```

```
    The RMSE of the forecasted model: 2.474
```

D4. Provide the output and calculations of the analysis you performed.

1) Initial augmented Dickey-Fuller test for stationarity

```
# Compute augmented dickey-fuller to mathematically test for stationarity
of dataset
# Default autolag argument measurement is AIC
dickey_fuller_results = adfuller(df['Revenue'])
# Dickey Fuller Results
# Source --- (Statsmodels, n.d.)
df_test_output = pd.Series(dickey_fuller_results[0:4], index = ['Test
Statistic', 'p-value', 'Number of Lags', 'Number of Observations'])

for key, value in dickey_fuller_results[4].items():
        df_test_output["Critical Value (%s)" % key] = value

# View Dickey-Fuller results
print(df_test_output)
```

```
Test Statistic            -1.924612
p-value                    0.320573
Number of Lags             1.000000
Number of Observations   729.000000
Critical Value (1%)       -3.439352
Critical Value (5%)       -2.865513
Critical Value (10%)      -2.568886
dtype: float64
```

2) Code to detrend data through differencing

```
# P-value above alpha of 0.05, must reject null hypothesis and proceed to
make data stationary
# Detrend data using diff() function to apply differencing
df_difference = df.diff()

# Differencing will result with null value for first value as no previous
value to difference, proceed to drop nulls
df_difference = df_difference.dropna()
```

3) Re-run augmented Dickey-Fuller Test on detrended, differenced data

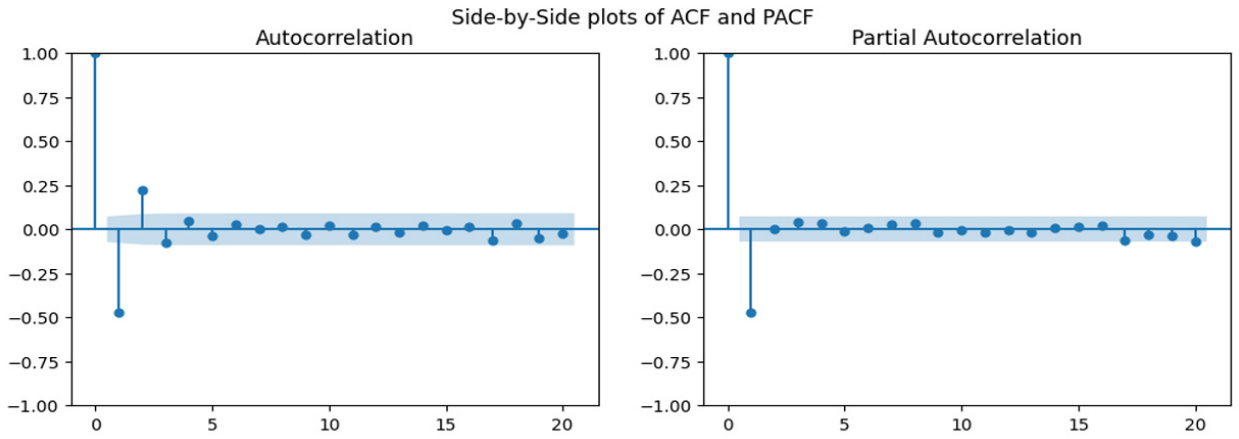
```
# Compute augmented dickey-fuller test for stationarity of dataset
```

```
diff_results = adfuller(df_difference['Revenue'])

# Get results of differencing
diff_test_output = pd.Series(diff_results[0:4], index = ['Test Statistic',
'p-value', 'Number of Lags Used', 'Number of Observations'])

for key, value in diff_results[4].items():
        diff_test_output["Critical Value (%s)" % key] = value

# View Dickey-Fuller results
print(diff_test_output)

# Very good test statistic, and critical values well below test stat are
signs of stationary data
```

```
Test Statistic           -44.874527
p-value                    0.000000
Number of Lags Used        0.000000
Number of Observations   729.000000
Critical Value (1%)       -3.439352
Critical Value (5%)       -2.865513
Critical Value (10%)      -2.568886
dtype: float64
```

4) Plots for autocorrelation function (ACF) and partial autocorrelation function (PACF)

```
# Determine values for ARIMA model order (p, d, q)
# Prepare plot for side by side subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 4))
fig.suptitle('Side-by-Side plots of ACF and PACF')
# Run ACF for value of "q"
plot_acf(df_difference['Revenue'], lags = 20, alpha = 0.05, ax = ax1)
# Run PACF for value of "p"
plot_pacf(df_difference['Revenue'], lags = 20, alpha = 0.05, ax = ax2)
plt.show()
```



5) Calculate to confirm optimal ARIMA model using auto_arima

```
# Run auto_arima --- confirm best model is AR(1)
# Confirm no seasonality in dataset
# SOURCE --- (Sewell, n.d.)
# Fit data with auto_arima model and view summary to obtain optimal p,d,q
values and look for seasonality
stepwise_fit = auto_arima(df_difference['Revenue'], trace = True,
suppress_warnings = True)
stepwise_fit.summary()

# Best model values for p,d,q confirmed --- (1, 0, 0) --- no seasonality
detected given results
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=987.305, Time=3.30 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=1162.819, Time=0.26 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=983.122, Time=0.21 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=1019.369, Time=0.42 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=1162.139, Time=0.15 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=985.104, Time=0.48 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=985.106, Time=0.28 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=986.045, Time=2.20 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=984.710, Time=0.16 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 7.547 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	730
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	-488.561
Date:	Thu, 21 Sep 2023	**AIC**	983.122
Time:	14:50:20	**BIC**	996.901
Sample:	01-02-2019 - 12-31-2020	**HQIC**	988.438
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0332	0.018	1.895	0.058	-0.001	0.068
ar.L1	-0.4692	0.033	-14.296	0.000	-0.534	-0.405
sigma2	0.2232	0.013	17.801	0.000	0.199	0.248

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	2.05
Prob(Q):	0.96	**Prob(JB):**	0.36
Heteroskedasticity (H):	1.02	**Skew:**	-0.02
Prob(H) (two-sided):	0.85	**Kurtosis:**	2.74

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

6) ARIMA model of train data set with print of summary

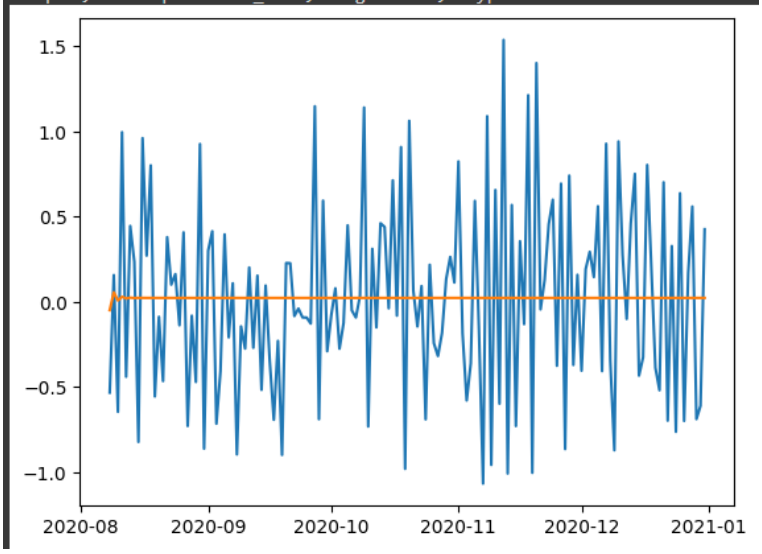
```
# Run training model and fit dataset
model = ARIMA(train['Revenue'], order = (1, 0, 0), freq = 'D')
results = model.fit()

# View result summary
print(results.summary())
```

```
                               SARIMAX Results
==========================================================================================
Dep. Variable:                    Revenue   No. Observations:                  584
Model:                 ARIMA(1, 0, 0)   Log Likelihood                -383.946
Date:                Thu, 21 Sep 2023   AIC                            773.893
Time:                        14:50:20   BIC                            787.002
Sample:                    01-02-2019   HQIC                           779.002
                         - 08-07-2020
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const          0.0234      0.013      1.758      0.079      -0.003       0.049
ar.L1         -0.4597      0.036    -12.654      0.000      -0.531      -0.388
sigma2         0.2180      0.014     16.034      0.000       0.191       0.245
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 1.84
Prob(Q):                              0.96   Prob(JB):                         0.40
Heteroskedasticity (H):               0.97   Skew:                            -0.08
Prob(H) (two-sided):                  0.83   Kurtosis:                         2.77
===================================================================================

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

7) Plot and print of model predicted mean

```
# Get forecasted results from model - test data is last 146 values, set
start to 584 and end to 729
forecasted = results.get_prediction(start = 584, end = 729, dynamic =
True)

# Plot Test data compared to forecasted predicted mean
plt.plot(test)
plt.plot(forecasted.predicted_mean);

# View predicted mean values
print(forecasted.predicted_mean)
```

```
2020-08-08   -0.048621
2020-08-09    0.056441
2020-08-10    0.008147
2020-08-11    0.030347
2020-08-12    0.020142
                ...
2020-12-27    0.023356
2020-12-28    0.023356
2020-12-29    0.023356
2020-12-30    0.023356
2020-12-31    0.023356
Freq: D, Name: predicted_mean, Length: 146, dtype: float64
```



8) Calculate RMSE of train model

```
# Get predictions for RMSE calculation
predict = results.forecast(len(test))
                    #Calculate RMSE of differenced model

rmse = sqrt(mean_squared_error(predict, test['Revenue']))
print(f"The RMSE of the differenced model: {round(rmse,3)}")
```

```
The RMSE of the differenced model: 0.57
```

9) Calculate RMSE of the train model of non-differended data compared to cumulative summation of prediction results.

```
# Calculate root mean squared error of forecasted data against the
observed data with no differencing
rmse_revenue = sqrt(mean_squared_error(df['Revenue'].loc['2020-08-08' :
'2020-12-31'], df_forecast['Revenue'].loc['2020-08-08' : '2020-12-31']))
print(f"The RMSE of the forecasted model: {round(rmse_revenue, 3)}")
```

```
The RMSE of the forecasted model: 2.474
```

D5. Provide the code used to support the implementation of the time series model.

SEE ATTACHED → Python_Mecchi_D213_Task_1_Time_Series.ipynb

Part V: Data Summary and Implications

E1. Discuss the results of your data analysis, including the following points:

The selection of an ARIMA model

WGU Telecom has provided data from the first two years of operation and recorded daily revenue streams with the hopes of having a time series model assist with future revenue projections. The research question asked if the WGU Telecom data could be used to build an ARIMA model to project daily revenues for Q1 and Q2 (next 183 days) of the following year. When initially assessing the data, the augmented Dickey-Fuller test identified the raw data was not stationary, therefore, not suitable to run in an ARIMA model because data must be stationary for accurate projections. To make stationary, the data was detrended by differencing and confirmed stationary by running a second augmented Dickey-Fuller test (Fig. 17).

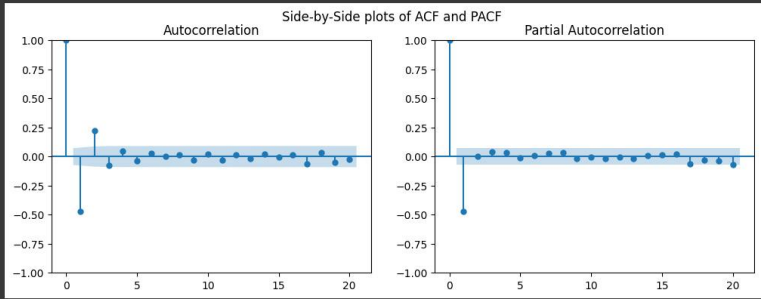
Figure 17: Results of Augmented Dickey-Fuller on Detrended Data

```
# Compute augmented dickey-fuller test for stationarity of dataset
diff_results = adfuller(df_difference['Revenue'])

# Get results of differencing
diff_test_output = pd.Series(diff_results[0:4], index = ['Test Statistic', 'p-value', 'Number of Lags Used', 'Number of Observations'])

for key, value in diff_results[4].items():
       diff_test_output["Critical Value (%s)" % key] = value

# View Dickey-Fuller results
print(diff_test_output)

# Very good test statistic, and critical values well below test stat are signs of stationary data
```

```
Test Statistic            -44.874527
p-value                     0.000000
Number of Lags Used         0.000000
Number of Observations    729.000000
Critical Value (1%)        -3.439352
Critical Value (5%)        -2.865513
Critical Value (10%)       -2.568886
dtype: float64
```

Figure 18: Autocorrelation and Partial Autocorrelation Function Plots

```
# Determine values for ARIMA model order (p, d, q)
# Prepare plot for side by side subplots
fig, (ax1, ax2) = plt.subplots(1, 2, figsize = (12, 4))
fig.suptitle('Side-by-Side plots of ACF and PACF')
# Run ACF for value of "q"
plot_acf(df_difference['Revenue'], lags = 20, alpha = 0.05, ax = ax1)
# Run PACF for value of "p"
plot_pacf(df_difference['Revenue'], lags = 20, alpha = 0.05, ax = ax2)
plt.show()
```



Once the data is stationary, both autocorrelation and partial autocorrelation functions were run to identify the parameters of the ARIMA model (Fig. 18). When observing the ACF and PACF plots, the ACF plot tails off quickly while the PACF starts to tail off at the first lag, suggesting an order-1 AR model. To ensure an accurate model and rule out seasonality, the auto_arima function was applied to the dataset and determined the data contains no seasonality and the optimal model is indeed an order-1 autoregressive ARIMA model (Fig. 19). The auto_arima function identifies the optimal model through iterations of different parameters and uses the lowest value of the Akaike Information Criteria (goodness of fit) as the determinant metric.

Figure 19: Results of auto_arima Function

```
# Run auto_arima --- confirm best model is AR(1)
# Confirm no seasonality in dataset
# SOURCE --- (Sewell, n.d.)
# Fit data with auto_arima model and view summary to obtain optimal p,d,q values and look for seasonality
stepwise_fit = auto_arima(df_difference['Revenue'], trace = True, suppress_warnings = True)
stepwise_fit.summary()

# Best model values for p,d,q confirmed --- (1, 0, 0) --- no seasonality detected given results
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,2)(0,0,0)[0] intercept   : AIC=987.305, Time=2.47 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=1162.819, Time=0.20 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=983.122, Time=0.21 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=1019.369, Time=0.40 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=1162.139, Time=0.08 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=985.104, Time=0.38 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=985.106, Time=0.27 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=986.045, Time=3.30 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=984.710, Time=0.17 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 7.519 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	730
Model:	SARIMAX(1, 0, 0)	**Log Likelihood**	-488.561
Date:	Fri, 22 Sep 2023	**AIC**	983.122
Time:	02:30:27	**BIC**	996.901
Sample:	01-02-2019 - 12-31-2020	**HQIC**	988.438

Covariance Type: opg

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0332	0.018	1.895	0.058	-0.001	0.068
ar.L1	-0.4692	0.033	-14.296	0.000	-0.534	-0.405
sigma2	0.2232	0.013	17.801	0.000	0.199	0.248

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	2.05
Prob(Q):	0.96	**Prob(JB):**	0.36
Heteroskedasticity (H):	1.02	**Skew:**	-0.02
Prob(H) (two-sided):	0.85	**Kurtosis:**	2.74

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

The prediction interval of the forecast

The prediction frequency of the forecast is defined as “D,” or day, and is determined by the original time series data. The WGU Telecom data covers recorded daily revenues from January 2019 through December 2020, therefore, to create an accurate forecast, the prediction interval is input with daily frequency.

The justification of the forecast length

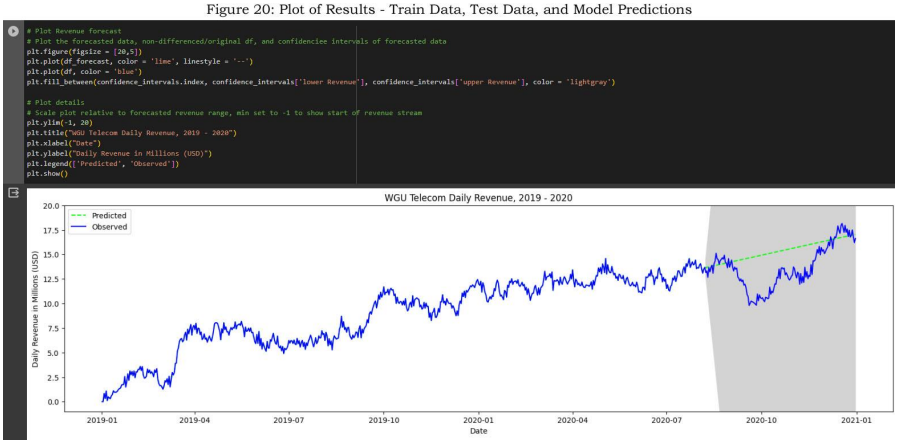
To ensure a successful model, enough historical data must be provided for the training model to learn and yield accurate forecasts. The more data provided for the training of the model the more accurate the model predictions. However, with time series projections, accuracy decreases as the timeframe of the predicted forecast increcreases. For the WGU Telecom ARIMA model, the training set was composed of the first 80% of the data while the remaining 20% was used to test model accuracy. This allows the model to learn time series trend analysis on the first 584 samples while estimating the remaining 146. Regarding the research question, if the training model proves accurate, a new model would be created and trained using the entirety of the WGU Telecom time series data. The projection for the research question looks to identify revenue streams for the first two quarters of the upcoming year. The future forecast is limited to the first two quarters to improve accuracy as compared to ascertaining the projections of revenues for the whole year.

The model evaluation procedure and error metric

The creation of the time series model was curated through a series of functions and refined with efficiency and error metrics. Before the initial model was created, the data was tested for stationarity using the augmented Dickey-Fuller test. The first Dickey-Fuller result identified the base data was not stationary and therefore not suitable for time series modeling. The dataframe was then detrended through differencing and when the augmented Dickey-Fuller test was applied to the differenced data, stationarity was confirmed. With the data prepared for the time series model, the autocorrelation and partial autocorrelations were calculated and plotted. These plots suggested an AR(1) model, however, for confirmation of optimal model and to rule out seasonality, the auto_arima function was utilized. To identify the best ARIMA model, auto_arima identifies model accuracy through evaluation of the Akaike Information Criteria (AIC) for each model iteration. The parameters that yield the lowest AIC identify the best performing model, thus, the results of the auto_arima confirmed an AR(1) model as the optimal parameters (also found no seasonality within the dataset). Using this qualifying criteria, the ARIMA model was fit to the training data and the results of the fit were used to predict the next 146 data points to compare results with the test dataset. The model accuracy was assessed using the root mean squared error (RMSE) metric which measures the average difference between a statistical model’s predicted values and the actual values recorded (Frost, 2023). A model with a low RMSE value indicates the model fits the data with accurate predictions, whereas higher values constitute a model with more error and less accurate predictions. The model returned a RMSE of 0.57, but is scored relative to the differenced dataset, while the RMSE of the actual values and the cumulative summation of values for the predictions yielded a value of 2.474. These values are indicative of a fairly accurate model and can be used for future projections.

E2. Provide an annotated visualization of the forecast of the final model compared to the test set.

Figure 20: Plot of Results - Train Data, Test Data, and Model Predictions



E3. Recommend a course of action based on your results.

The results of the time series forecast yielded moderately accurate results with RMSE values of 0.57 and 2.474 for differenced and non-differenced data respectively. While values closer to zero are more desirable, these values are based on the first two-years of WGU Telecom operations. Given WGU Telecom is in the nascent stages of operation, I would recommend the company use this time series model for future forecasting. More accurate results are unquestionably more desirable, however, the forecast is based on the only data available form the new company. The time series model produced a predicted mean that leveled out with a value of 0.0234 which also represents the average daily increase in revenue. Based on this information, WGU Telecom can generate soft expectations of increased revenue for the upcoming two quarters. With the train/test time series analysis confirmed as a suitable model, I would suggest utilizing the same parameters to train a new model on the entirety of the initial differenced dataset. I would also suggest that WGU Telecom limit the length of their projection to no more than 183 days, or two quarters, because time series predictions become increasingly inaccurate with longer forecasts. WGU Telecom should continue with comparative analysis after the first and second quarter profits are released to assess the accuracy of the future forecast. The influx of the new data will also help provide newer models with more robust data which will yield better results as the company continues to grow.

Part VI: Panopto Video

F1. Panopto Submission

SEE ATTACHED → Panopto_Mecchi_D213_Task_1_Advanced_Data_Analytics

CODE SOURCES

Elleh, F. (n.d.). D213: Advanced Data Analytics - Task 1. [PowerPoint Slides]. Faculty of Masters of Data Analytics, Western Governors University. Retrieved September 15, 2023. https://docs.google.com/presentation/d/1r4DezzerY-rITl3qXNPglmctTRjf0iRi/edit#slide=id.p12.

Statsmodels (n.d.) Stationarity and detrending (ADF/KPSS). Statsmodels 0.15.0 (+59) documentation. Retrieved September 15, 2023. https://www.statsmodels.org/dev/examples/notebooks/generated/stationarity_detrending_adf_kpss.html.

Sewell, W. (n.d.). D213: Advanced Data Analytics - Time Series Analysis - ARIMA Modeling I. [PowerPoint Slides]. Faculty of Masters of Data Analytics, Western Governors University. Retrieved September 15, 2023. https://westerngovernorsuniversity-my.sharepoint.com/:p:/g/personal/william_sewell_wgu_edu/EftnPPOZu8RNvZIbxqAUX0UBJkIAWI8QhKLQ63azgA8a-g?rtime=bvMN_6u320g.

Zach. (2022. March 31). How to Add a Trendline in Matplotlib (With Example). Statology. https://www.statology.org/matplotlib-trendline/.

REFERENCES

Affek, S.P.. (2019, April 8). Stationarity in Time Series Analysis. Towards Data Science. https://towardsdatascience.com/stationarity-in-time-series-analysis-90c94f27322.

Elleh, F. (n.d.). D213: Advanced Data Analytics - Task 1. [PowerPoint Slides]. Faculty of Masters of Data Analytics, Western Governors University. Retrieved September 15, 2023. https://docs.google.com/presentation/d/1r4DezzerY-rITl3qXNPglmctTRjf0iRi/edit#slide=id.p15.

Frost, J. (2022, October 17). Autocorrelation and Partial Autocorrelation in Time Series Data. Statistics by Jim. https://statisticsbyjim.com/time-series/autocorrelation-partial-autocorrelation/#:~:text=Autocorrelation%20is%20the%20correlation%20between,values%20influence%20the%20current%20value.

Frost, J. (2023, May 28). Root Mean Square Error (RMSE). Statistics by Jim. https://statisticsbyjim.com/regression/root-mean-square-error-rmse/.

Jones, J. (2018, February 19). Time Series and Spectral Analysis. Stanford. Retrieved September 15, 2023. https://web.stanford.edu/class/earthsys214/notes/series.html.

Lewinson, E. (2022, March 31).Time Series DIY: Seasonal Decomposition. Towards Data Science. https://towardsdatascience.com/time-series-diy-seasonal-decomposition-f0b469afed44.

Nau, R. (n.d.) Identifying the numbers of AR or MA terms in an ARIMA model. Duke University. Retrieved September 15, 2023. https://people.duke.edu/~rnau/411arim3.htm.

Penn State (n.d.) 12.1 Estimating the Spectral Density. Penn State Eberly College of Science - STAT 510: Applied Time Series Analysis. Retrieved September 15, 2023. https://online.stat.psu.edu/stat510/lesson/12/12.1

pmdarima (n.d.) pmdarima.auto_arima. pmdarima documentation 2.0.3. Retrieved September 15, 2023. https://alkaline-ml.com/pmdarima/modules/generated/pmdarima.ara.auto_arima.html.

Smith, T. (2023, March 19). Autocorrelation: What It Is, How It Works, Tests. Investopedia. https://www.investopedia.com/terms/a/autocorrelation.asp.

Statistics Solutions. (2023, March 9). The Stationary Data Assumption in Time Series Analysis. Statistics Solutions. https://www.statisticssolutions.com/stationary-data-assumption-in-time-series-analysis/.