# D213 Time Series Modeling Performance Assessment, Task 1

**Alexa R. Fisher**

**Western Governors University**

**Degree: M.S. Data Analytics**

## Table of Contents

## Part I: Research Question

### A1. Proposal of Question

The research question for this thesis was, “Can the telecommunication company's daily revenue be accurately forecasted in comparison to the actual observed daily revenue?”. This analysis was completed with the use of ARIMA time series modeling. The ARIMA time series model forecasted the data to predict observations.

### A2. Defined Goal

The main goal of this analysis was to effectively forecast the WGU-provided telecommunication company’s daily revenue. The data set was first split into two datasets, training and testing. The training data called “train_tc” was 80% of the original dataset. This was represented as the data from 01-01-2020 to 08-07-2021. The remaining 20% of the original was the testing dataset called “test_tc”, which encompassed the dates from 08-08-2021 to 12-31-2021. This analysis aimed to utilize the training set to predict the test data set’s observed values. Upon completion, the forecasted observations via the ARIMA model were compared to the original for evaluation.

## Part II: Method Justification

### B1. Assumptions of Times Series Model

There were a few assumptions about the ARIMA time series model. The assumptions included the following points.

- The time series data must have stationarity.
- There are no outliers or anomalies in the data.

- Autocorrelation is constant.

The first assumption was stationarity. The data of the time series analysis has a mean and variance that do not vary across time (Wu, 2021). If there was a strong trend or seasonality observed, it would deem the data non-stationary.  The WGU provided telecommunication company's original data had a noted upward trend. This observation identified the data as non-stationary. Once the data was differenced, the assumption was satisfied as the trend was no longer apparent. It allowed for the time series model to be performed.

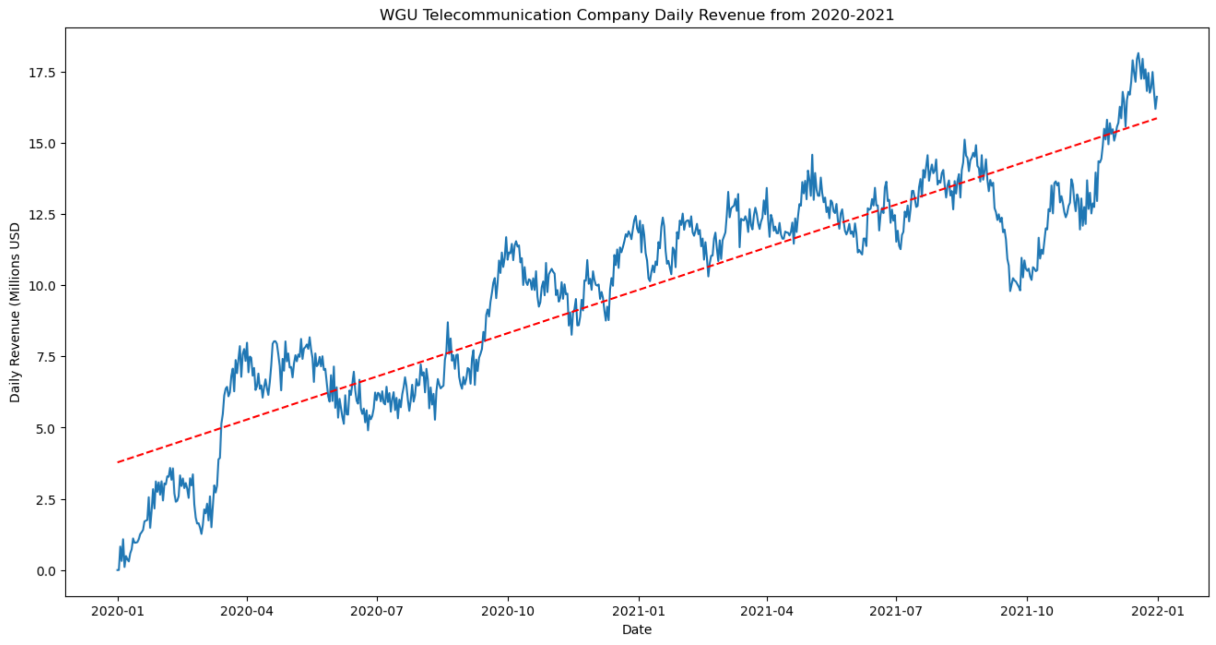
The second assumption was the absence of outliers or anomalies (Reider, n.d.). The presence of these could provide skewed or inaccurate results within the modeling. The provided dataset was free of any anomalies as all the dates within those two years were accounted for.

The last assumption was autocorrelation was constant. Autocorrelated referred to the correlation between the datapoint and their lagged values. Lagged values were previous data points at different time intervals (Reider, n.d.). In this time series model, the time interval was daily. This noted that the previous day's revenue was related to the next day's revenue. The observed values from January 2020 to August 7, 2020, showed to have a positive correlation at step 2.

## Part III: Data Preparation

### C1. Realization of Time Series

The times series model of the telecommunication data set had a total of 731 observations. These observations were over the course of two years. As there are 365 days of the year, which would be a total of 730 days, it was assumed the captured years included a leap year. The data dictionary did not specify a particular year, the years 2020-2021 were assigned as it met the criteria. The additional variable was "Revenue" which captured monetary value in increments of a million US dollars. After the data set was cleaned, the below line graph was plotted. It had an upward trend over the course of the two years.



## C2. Time Step Format

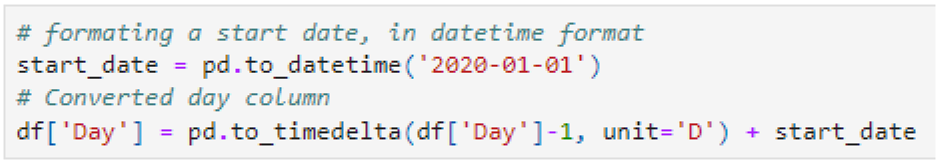
The realization of the time series included the following time step formatting: DateTime index. The DataTime index was based on daily observations from January 1, 2020 to December 31, 2021. There were no gaps noted within the measurement. The length of the sequence was noted as 731 data points or days, which included the leap year (February 29th) for the year 2020. The formatted data as a DataTime object allowed for the data to be reviewed daily, weekly, monthly, and yearly as needed.

## C3. Evaluation of Stationarity

The observed data of the telecommunication was not stationary. Stationarity referenced data without a trend or variance (Wu, 2021). The above line graph showed an obvious upward trend within the data. The telecommunication company's revenue trend line started at around $3 million in the first quarter of the year in 2020. The daily revenue trend continued to rise to around $10 million by the end of the same year. The upward trend did not stop there. The rise continued into 2021 from $10 million to $15 million.

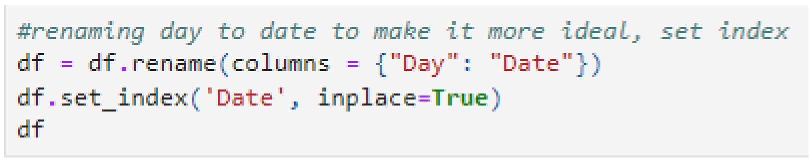
## C4. Steps of Data Preparation

There were several steps to prepare the data for analysis. The original data set notated the attributes of "Day" and "Revenue". The "Day" variable was converted to a DateTime object with the use of Pandas' to_datetime() function.  The start date was set, and each subsequent day was added to populate the remaining dates with the use of Pandas' to_timedelta() function.



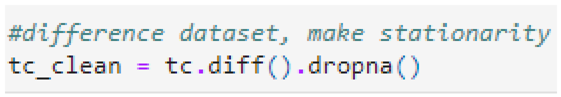
```python
# formating a start date, in datetime format
start_date = pd.to_datetime('2020-01-01')
# Converted day column
df['Day'] = pd.to_timedelta(df['Day']-1, unit='D') + start_date
```

After the variable "Day" was converted, the attribute was renamed to "Date" to make it clear to reference. It was completed with the use of the .rename() function. The data set's index was set to the newly renamed "Date" variable using the .set_index() function.



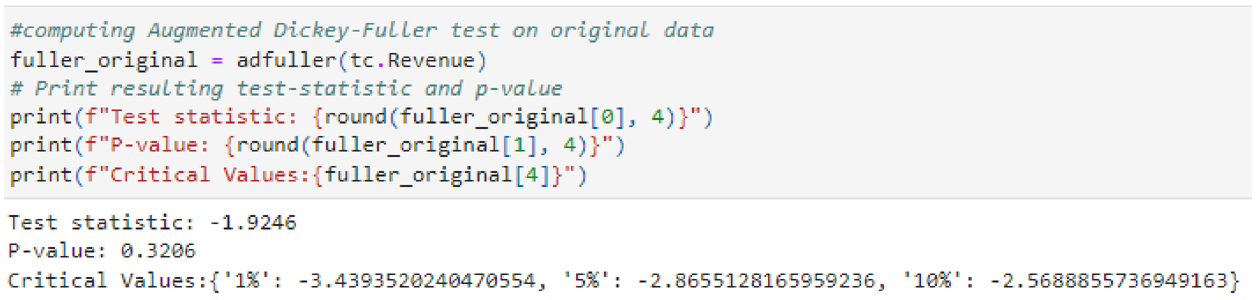
```python
#renaming day to date to make it more ideal, set index
df = df.rename(columns = {"Day": "Date"})
df.set_index('Date', inplace=True)
df
```

To proceed with the time series, the original data needed to have stationarity. The original data was transformed by differencing the data (Reider, n.d.). Differencing was completed using the .diff() function, which took the difference between each consecutive data point. The first data point did not have preceding data to compute the difference, so it was notated as a null value. The use of the .dropna() function removed the null value from the data set. It left the data set with only 730 data points remaining. The new data set was saved as the cleaned data set called “tc_clean”.



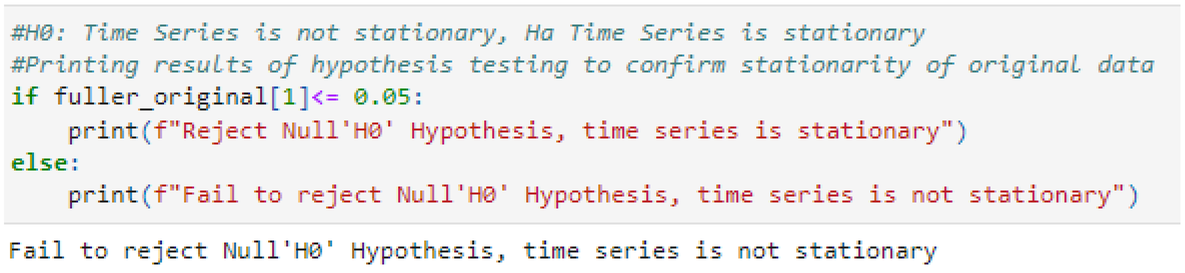
```
#difference dataset, make stationarity
tc_clean = tc.diff().dropna()
```

Stationarity was confirmed with the use of the Augmented Dickey-Fuller (AD Fuller) test. In the analysis, the test was first run on the original data’s “Revenue”. To perform the AD Fuller test, a Null hypothesis was set. This stated the time series was not stationary and the Alternative hypothesis was the time series was stationary. It provided the following results: Test statistic: -1.9246, P-value: 0.3206, Critical Values: {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}. The greater the negative test statistic the higher the likelihood of the data to be stationary (Reider, n.d.). In the original data, the test statistic was -1.9246 which was relatively small. The p-value of the original data set was also greater than the 0.05 threshold, deeming the data non-stationary as well. It was noted as a failure to reject the Null hypothesis.



```
#computing Augmented Dickey-Fuller test on original data
fuller_original = adfuller(tc.Revenue)
# Print resulting test-statistic and p-value
print(f"Test statistic: {round(fuller_original[0], 4)}")
print(f"P-value: {round(fuller_original[1], 4)}")
print(f"Critical Values:{fuller_original[4]}")
```

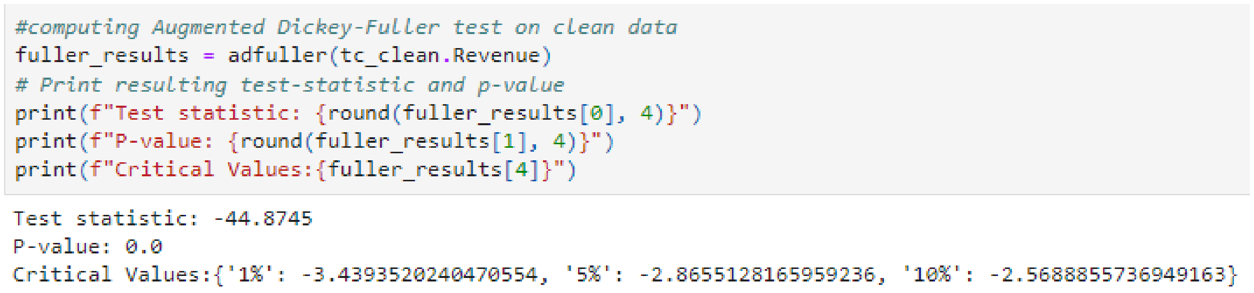
```
Test statistic: -1.9246
P-value: 0.3206
Critical Values:{'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
```



```
#H0: Time Series is not stationary, Ha Time Series is stationary
#Printing results of hypothesis testing to confirm stationarity of original data
if fuller_original[1]<= 0.05:
    print(f"Reject Null'H0' Hypothesis, time series is stationary")
else:
    print(f"Fail to reject Null'H0' Hypothesis, time series is not stationary")
```

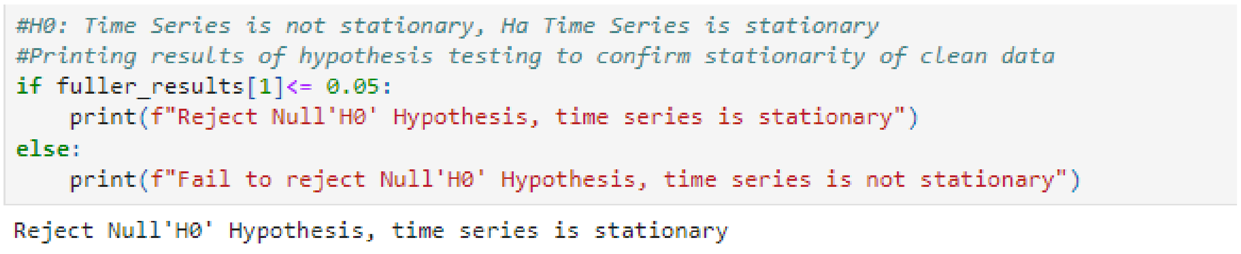
```
Fail to reject Null'H0' Hypothesis, time series is not stationary
```

The test was re-performed on the differenced data's "Revenue" variable to provide the updated test statistic, p-value, and critical values. The Augmented Dickey-Fuller results for the telecommunication's clean data were as follows: Test statistic: -44.8745, P-value: 0.0, Critical Values: {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}. The updated test statistic was a vast increase to -44.8745, which met the higher likelihood of stationarity. The p-value of less than or equal to 0.05 deemed the time series stationary. The updated p-value was 0.0. It met the noted threshold as a result the Null hypothesis could be rejected.



```
#computing Augmented Dickey-Fuller test on clean data
fuller_results = adfuller(tc_clean.Revenue)
# Print resulting test-statistic and p-value
print(f"Test statistic: {round(fuller_results[0], 4)}")
print(f"P-value: {round(fuller_results[1], 4)}")
print(f"Critical Values:{fuller_results[4]}")
```

```
Test statistic: -44.8745
P-value: 0.0
Critical Values:{'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
```



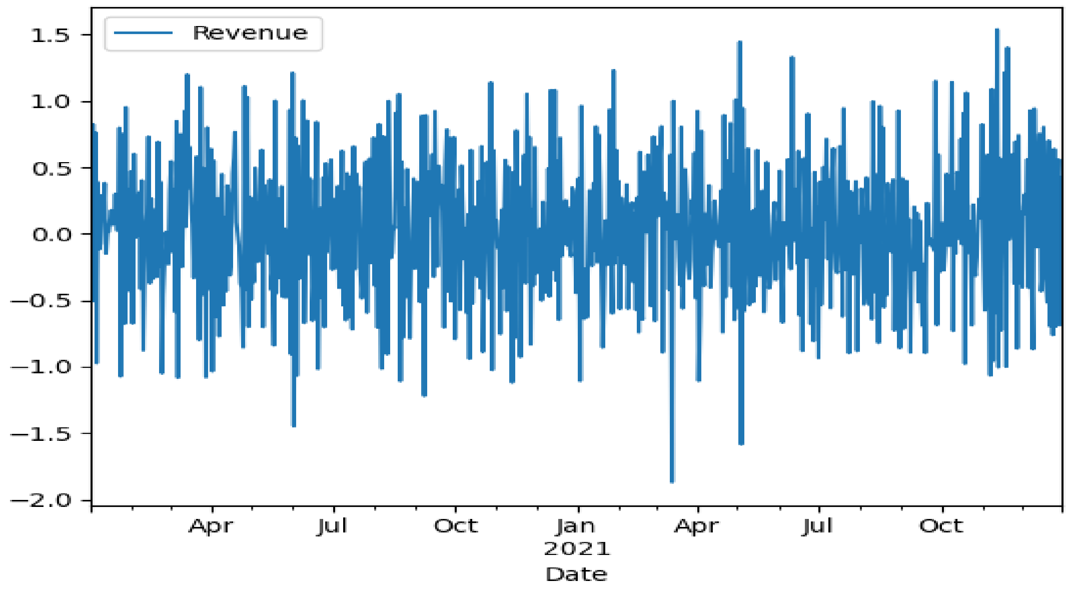
```
#H0: Time Series is not stationary, Ha Time Series is stationary
#Printing results of hypothesis testing to confirm stationarity of clean data
if fuller_results[1]<= 0.05:
    print(f"Reject Null'H0' Hypothesis, time series is stationary")
else:
    print(f"Fail to reject Null'H0' Hypothesis, time series is not stationary")
```

```
Reject Null'H0' Hypothesis, time series is stationary
```

The cleaned data were plotted to show the lack of any trends or seasonality.

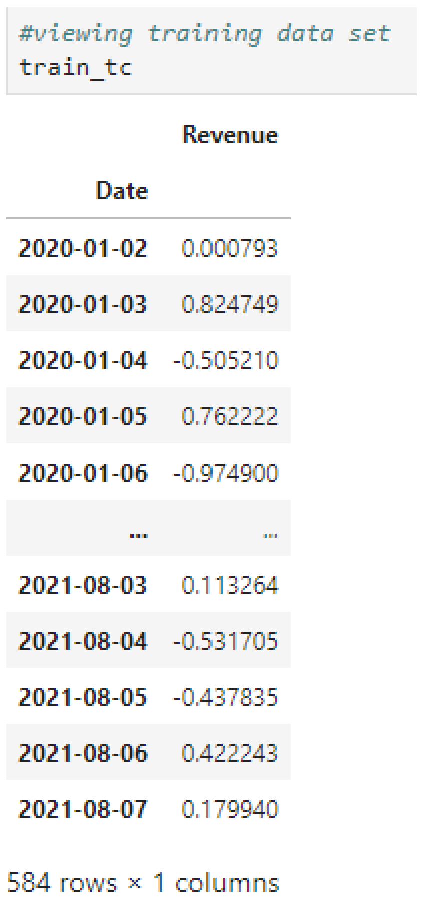
```
# Plot to verify stationarity
tc_clean.plot()
```

```
<AxesSubplot: xlabel='Date'>
```



Lastly, the cleaned dataset was split into a training and testing data set. This was completed with the use of the train_test_split function from SciKit Learn. The testing data set was set to 20% of the data points, this was notated as the dates between 08-08-2021 to 12-31-2021. The training data was 80% of the data set. This was represented as the data from 01-01-2020 to 08-07-2021. The parameter of shuffle was set to False to ensure the data would not be rearranged during the split.

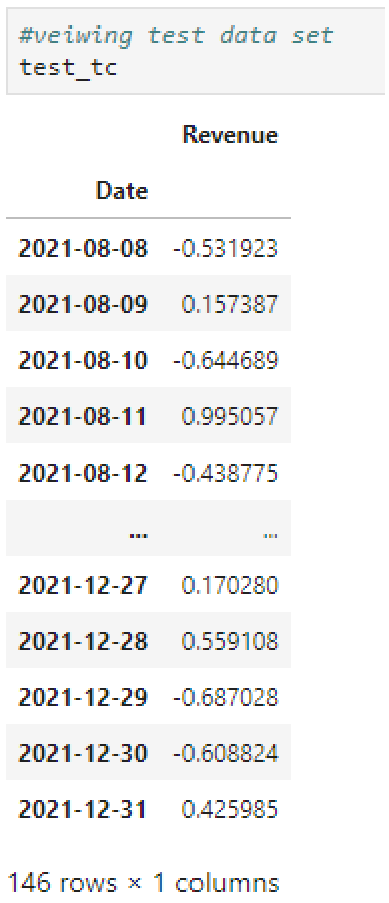
```
# Split time series into a training set and a test set
train_tc, test_tc = train_test_split(tc_clean, test_size=0.2, shuffle=False, random_state=397)
```



```
#viewing training data set
train_tc
```

| Date | Revenue |
|---|---|
| 2020-01-02 | 0.000793 |
| 2020-01-03 | 0.824749 |
| 2020-01-04 | -0.505210 |
| 2020-01-05 | 0.762222 |
| 2020-01-06 | -0.974900 |
| ... | ... |
| 2021-08-03 | 0.113264 |
| 2021-08-04 | -0.531705 |
| 2021-08-05 | -0.437835 |
| 2021-08-06 | 0.422243 |
| 2021-08-07 | 0.179940 |

584 rows × 1 columns



```
#veiwing test data set
test_tc
```

| Date | Revenue |
|---|---|
| 2021-08-08 | -0.531923 |
| 2021-08-09 | 0.157387 |
| 2021-08-10 | -0.644689 |
| 2021-08-11 | 0.995057 |
| 2021-08-12 | -0.438775 |
| ... | ... |
| 2021-12-27 | 0.170280 |
| 2021-12-28 | 0.559108 |
| 2021-12-29 | -0.687028 |
| 2021-12-30 | -0.608824 |
| 2021-12-31 | 0.425985 |

146 rows × 1 columns

### C5. Cleaned Dataset

Please see the attached CSV files to view the results of the cleaned data sets. The clean datasets included the original cleaned data set, the training data set, and the test data set.

The cleaned dataset was called “AFCodeD213Tk1_clean.csv”. The cleaned dataset was split into training and test datasets. The training data set included 80% of the cleaned data set and was called “AFD213Tk1_train_clean.csv”. The test data set included 20% of the cleaned data and was named “AFD213Tk1_test_clean.csv”.

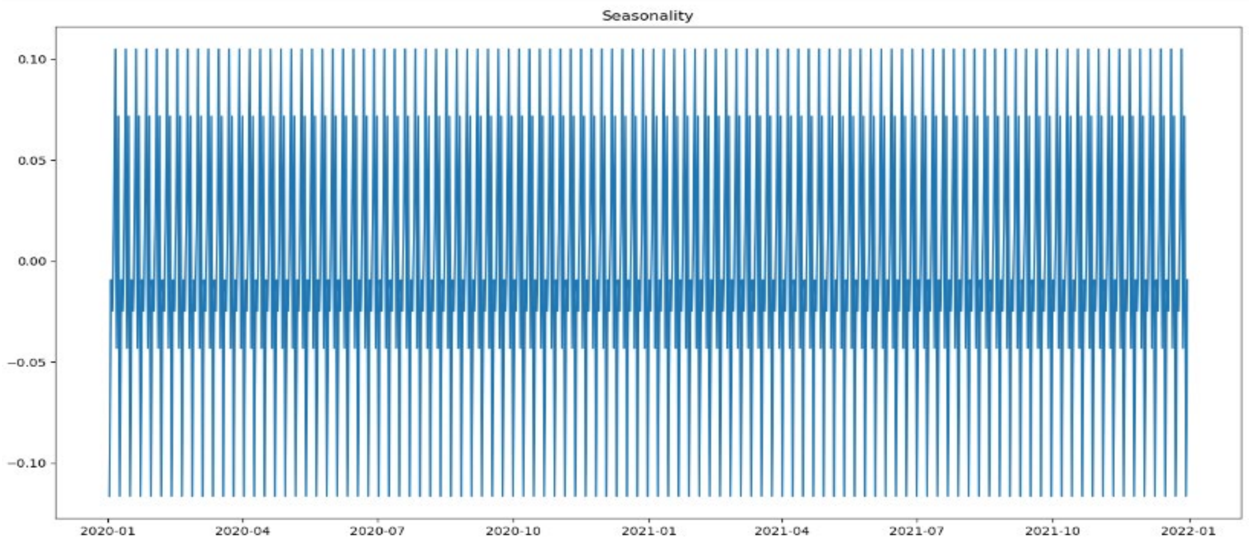
## Part IV: Model Identification and Analysis

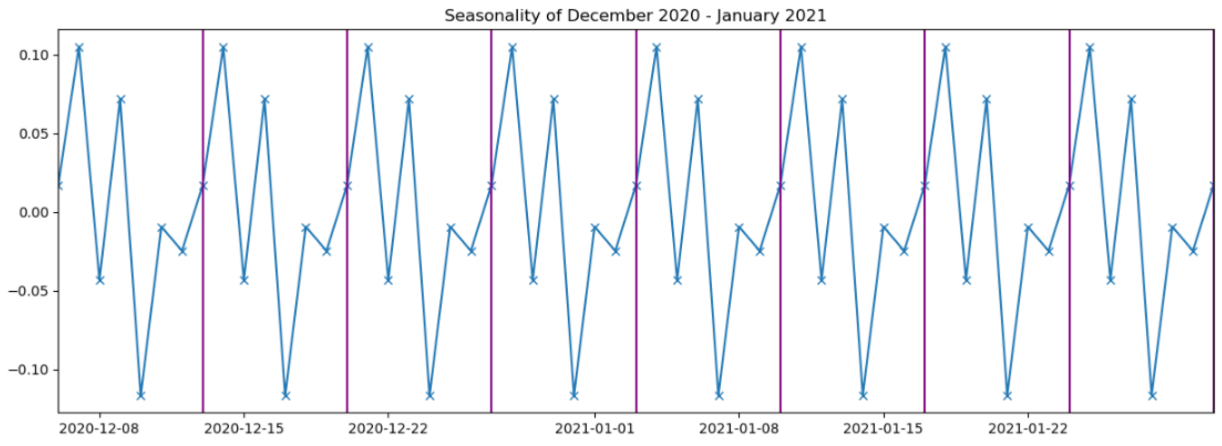
## D1. Annotated Findings

The annotated findings within the time series analysis included the following elements. These were all plotted to show each visualization.

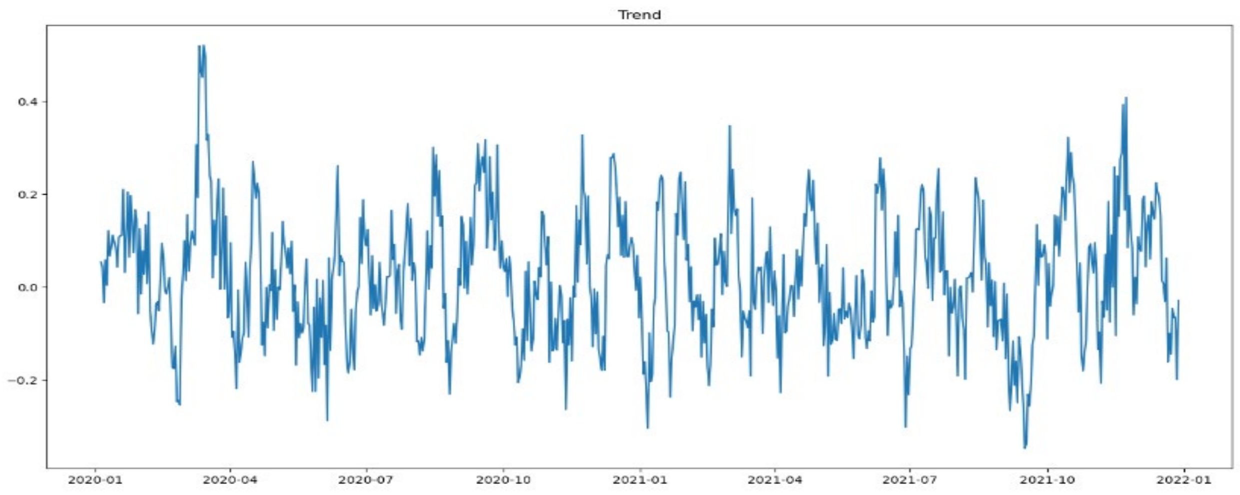
- Seasonality
- Trends
- Autocorrelation Function
- Spectral Density
- Decomposed Time Series
- Residuals of Decomposed Series

Seasonality refers to data that recurs regularly over time (Reider, n.d.). The seasonal component noted within the visualization was a repeating pattern. This confirmed the indication of seasonality. The magnitude of the seasonality was somewhat small at around 0.10 in either positive or negative directions. In the view of the graph, the seasonality seemed to occur quite often, it suggested that it could be related to the days itself. Reviewing the seasonality at a monthly level provided additional insight. It confirmed the seasonality pattern at a zoomed view. The months of December 2020 through January 2021 were used to represent this revised observation. The first day of the week, Sunday was marked with a purple line to show a focused view of the pattern. The repetitive pattern was noted as a weekly occurrence.

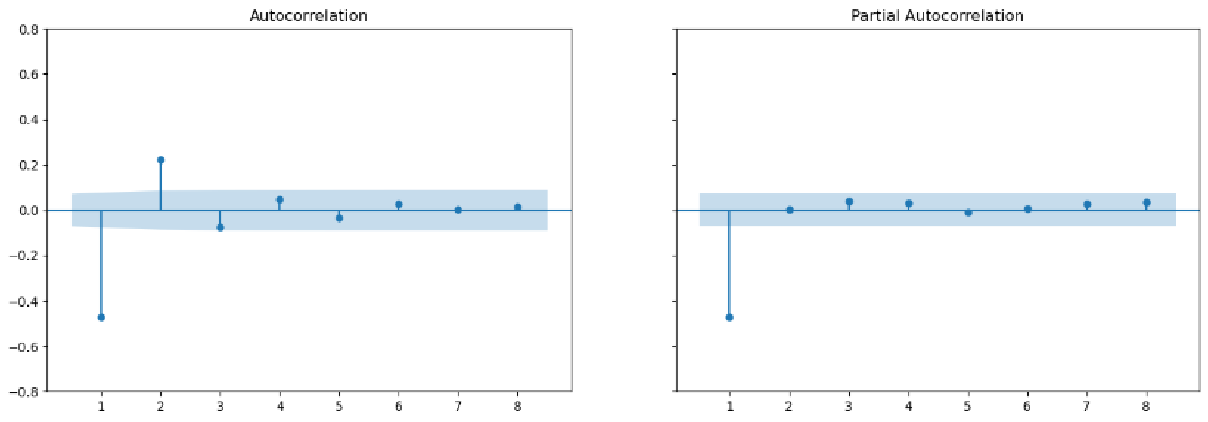




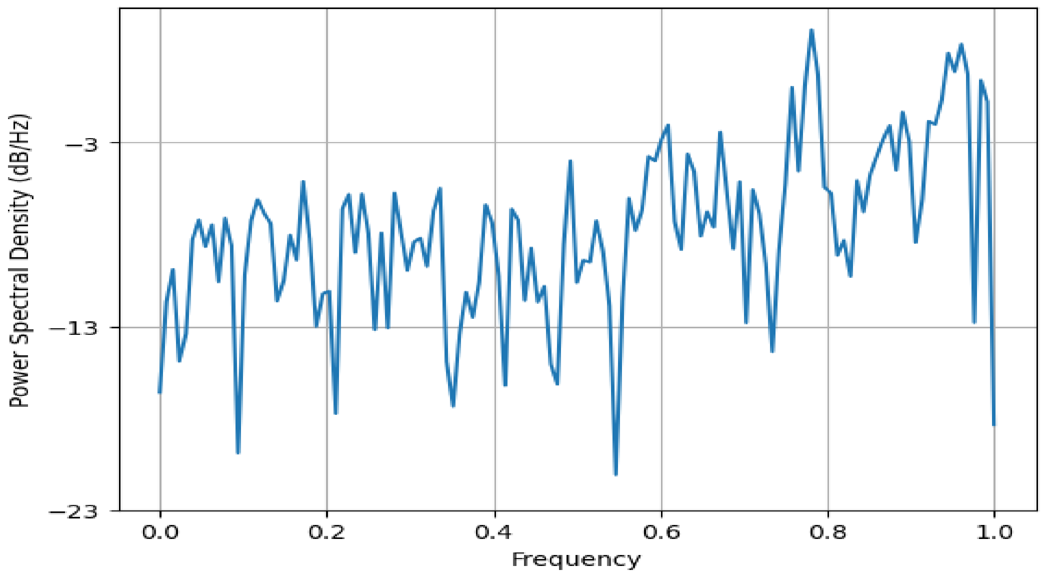
Trends were patterns in the data which showed the movement of the data points over a prolonged period (Tejalkadam18m, 2022). The below visualization showed no apparent trend in the data. There was a peak of over 0.40 in the first quarter of the year 2020, but that could be an outlier.



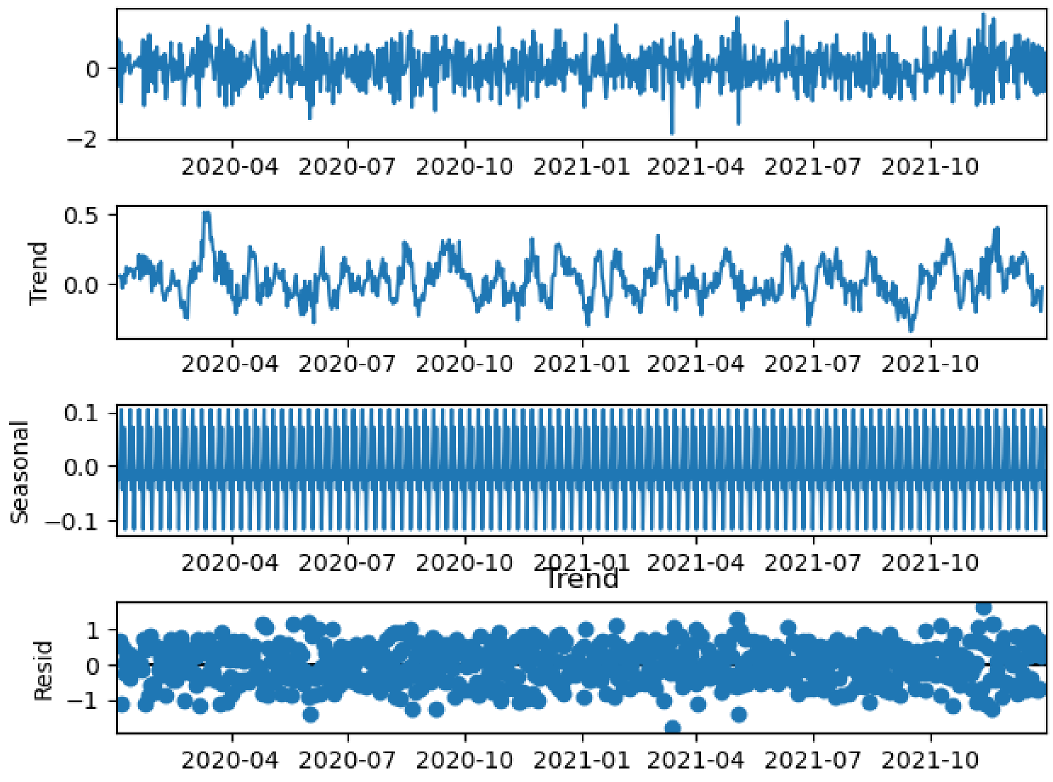
Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF) were used to determine if the data was an autoregression(AR) or moving average(MA) model. The shaded blue area within the visualization referred to statistically insignificant points (Fulton, n.d.). Those data points could be ignored. For example, in the below representation, the ACF tails off at 2 lag steps, and the PACF tails off at 1 lag step. If an Autocorrelation tails off and the Partial Autocorrelation Function cuts off after lag *p*, then the model was noted as an AR *p* model (Fulton, n.d.). The visualization showed an AR (1) model.



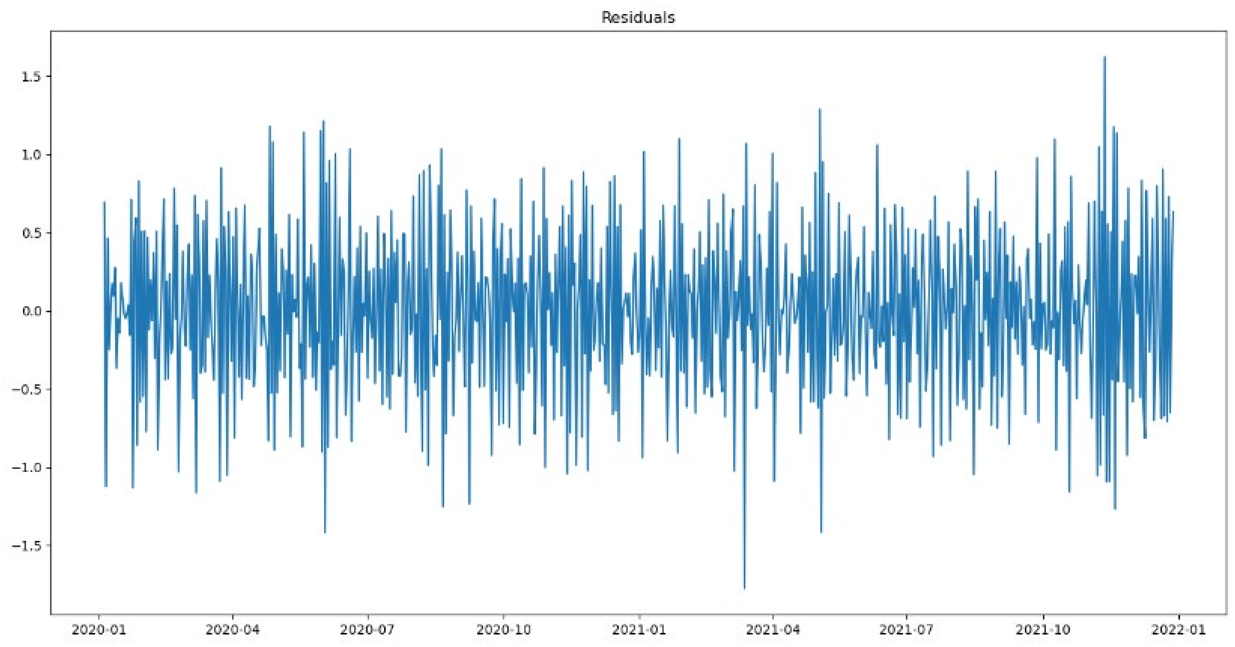
The below graph showed the power spectral density.



The below graph showed the decomposed components of the times series which included the three major components of Seasonality, Trends, and Residuals.



The below visualization confirmed the lack of trends in the residuals of the decomposition. There was no distinguishable trend observed.



## D2. Identification of ARIMA

The ARIMA model identification was noted in two ways. The first way was noted previously based on the ACF and PACF visualization. The second was the use of the auto_arima() function. This function provided the best parameters of *p, d, and q* for the ARIMA model while taking into account the observed trend of the series data. In the case of the telecommunication data set, the following parameters were noted as the best model fit: ARIMA (1, 0, 0)(0, 0, 0)[0].

```
#auto_arima to get best parameters(Pulagam,2020).
auto_arima(
    train_tc["Revenue"],
    start_p = 1, max_p = 6,
    start_q = 1, max_q = 6,
    seasonal = False, trace = True).summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=776.989, Time=0.17 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=909.948, Time=0.07 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=774.990, Time=0.08 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=801.099, Time=0.19 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=776.989, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=778.497, Time=1.03 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=773.893, Time=0.16 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=910.790, Time=0.15 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=775.886, Time=0.37 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=775.888, Time=0.31 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=799.464, Time=0.23 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=777.694, Time=1.77 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 4.648 seconds
```

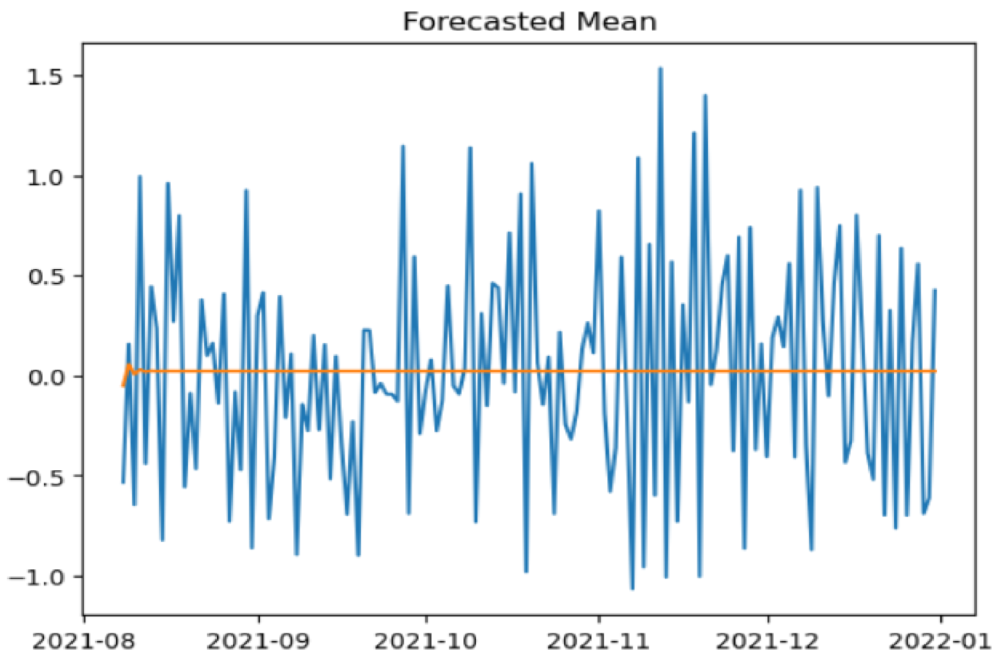
```
#ARIMA model of time series using best parameters from auto arima(Elleh, n.d.).
arima_model = ARIMA(train_tc, order=(1, 0, 0), freq='D')
results = arima_model.fit()
print(results.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  584
Model:                 ARIMA(1, 0, 0)   Log Likelihood                -383.946
Date:                Thu, 03 Aug 2023   AIC                            773.893
Time:                        20:49:56   BIC                            787.002
Sample:                    01-02-2020   HQIC                           779.002
                         - 08-07-2021
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. Forecast Model

The forecasted model utilized the derived ARIMA model, AR(1) as shown below. As the test dataset was only noted to have 146 days from August 08, 2021, to December 31, 2021, the forecast was limited to that time period. The start and end were calculated by subtracting 730 ( the last observation overall) – 146 (days) = 584, which would note the start date.  The forecasted predicted mean was the result of the difference from the previous day to the next. For example, the predicted mean from December 27th through 31st was static with a 0.023356 million daily revenue, which meant the daily revenue increased by the noted amount each subsequent day.

```
#forecasted mean(Pathak, 2020).
forecasted = results.get_prediction(start = 584, end = 729, dynamic = True)
plt.plot(test_tc)
plt.title('Forecasted Predicted Mean')
plt.plot(forecasted.predicted_mean);
```



```
print(forecasted.predicted_mean)
```

```
2021-08-08   -0.048621
2021-08-09    0.056441
2021-08-10    0.008147
2021-08-11    0.030347
2021-08-12    0.020142
                ...
2021-12-27    0.023356
2021-12-28    0.023356
2021-12-29    0.023356
2021-12-30    0.023356
2021-12-31    0.023356
Freq: D, Name: predicted_mean, Length: 146, dtype: float64
```

The training data was utilized to forecast the testing data with the use of the predicted means as well as the confidence intervals from those forecasted values. Confidence Intervals were the range of values the predictions were expected to fall between (Bevans, 2020). The confidence intervals were very large during this range. For example, this was shown with the lower revenue and upper revenue noting a 129.212872 -163.116910 million daily revenue for December 27, 2021.

```
# forecasted differences into a temp dataframe(Pathak, 2020).
fore_temp = pd.DataFrame(forecasted.predicted_mean)
fore_temp.rename(columns={'predicted_mean' : 'Revenue'}, inplace=True)

# Concat copy of train data & forecasted values
tc_forecast = pd.concat([train_tc.copy(), fore_temp.copy()])
tc_forecast = tc_forecast.cumsum()
tc_forecast
```

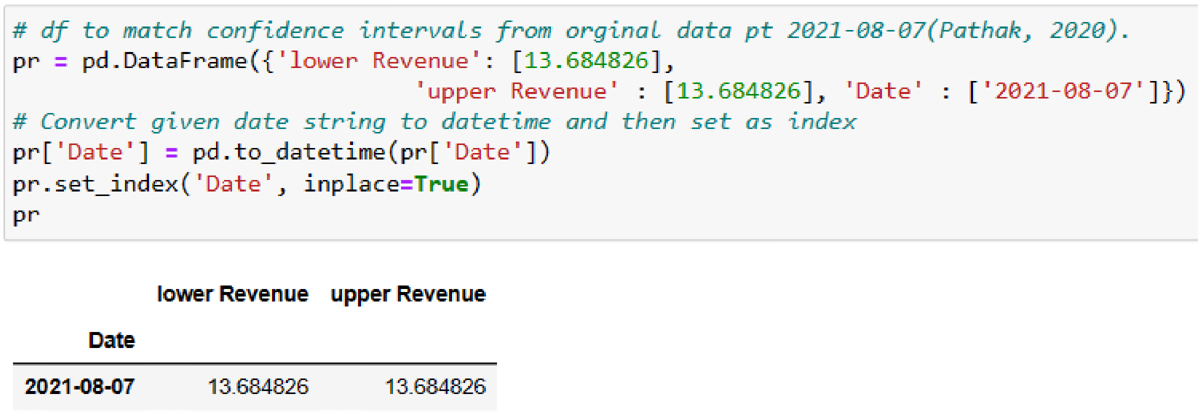
| | Revenue |
|---|---|
| **2020-01-02** | 0.000793 |
| **2020-01-03** | 0.825542 |
| **2020-01-04** | 0.320332 |
| **2020-01-05** | 1.082554 |
| **2020-01-06** | 0.107654 |
| **...** | ... |
| **2021-12-27** | 16.952019 |
| **2021-12-28** | 16.975375 |
| **2021-12-29** | 16.998730 |
| **2021-12-30** | 17.022086 |
| **2021-12-31** | 17.045442 |

730 rows × 1 columns

```
# confidence intervals from forecasted
confid_intv = forecasted.conf_int()
confid_intv
```

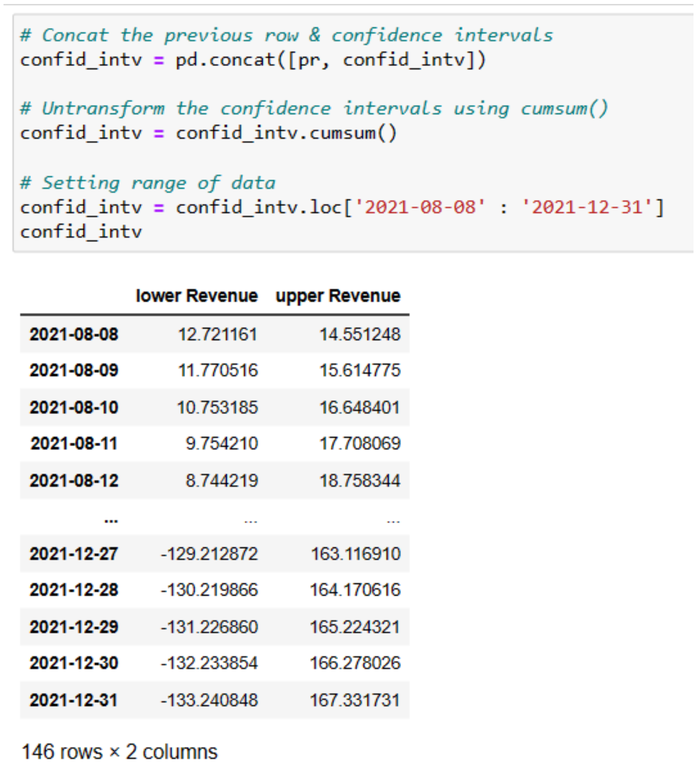
| | lower Revenue | upper Revenue |
|---|---|---|
| **2021-08-08** | -0.963665 | 0.866422 |
| **2021-08-09** | -0.950645 | 1.063528 |
| **2021-08-10** | -1.017331 | 1.033625 |
| **2021-08-11** | -0.998976 | 1.059669 |
| **2021-08-12** | -1.009990 | 1.050275 |
| **...** | ... | ... |
| **2021-12-27** | -1.006994 | 1.053705 |
| **2021-12-28** | -1.006994 | 1.053705 |
| **2021-12-29** | -1.006994 | 1.053705 |
| **2021-12-30** | -1.006994 | 1.053705 |
| **2021-12-31** | -1.006994 | 1.053705 |

146 rows × 2 columns



```
# df to match confidence intervals from orginal data pt 2021-08-07(Pathak, 2020).
pr = pd.DataFrame({'lower Revenue': [13.684826],
                   'upper Revenue' : [13.684826], 'Date' : ['2021-08-07']})
# Convert given date string to datetime and then set as index
pr['Date'] = pd.to_datetime(pr['Date'])
pr.set_index('Date', inplace=True)
pr
```

| | lower Revenue | upper Revenue |
|---|---|---|
| **Date** | | |
| **2021-08-07** | 13.684826 | 13.684826 |



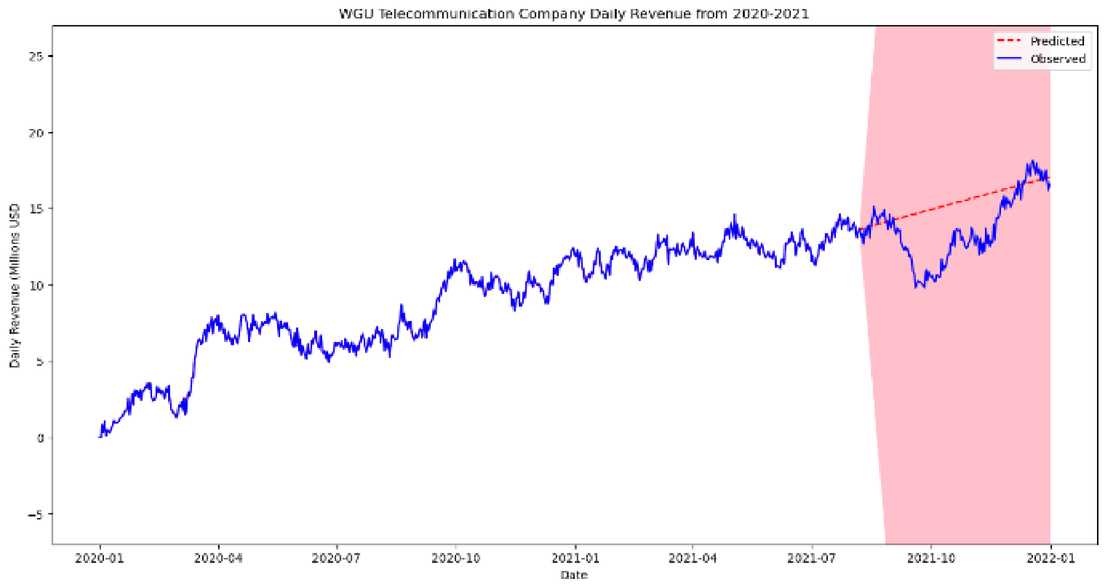
```
# Concat the previous row & confidence intervals
confid_intv = pd.concat([pr, confid_intv])

# Untransform the confidence intervals using cumsum()
confid_intv = confid_intv.cumsum()

# Setting range of data
confid_intv = confid_intv.loc['2021-08-08' : '2021-12-31']
confid_intv
```

| | lower Revenue | upper Revenue |
|---|---|---|
| **2021-08-08** | 12.721161 | 14.551248 |
| **2021-08-09** | 11.770516 | 15.614775 |
| **2021-08-10** | 10.753185 | 16.648401 |
| **2021-08-11** | 9.754210 | 17.708069 |
| **2021-08-12** | 8.744219 | 18.758344 |
| **...** | ... | ... |
| **2021-12-27** | -129.212872 | 163.116910 |
| **2021-12-28** | -130.219866 | 164.170616 |
| **2021-12-29** | -131.226860 | 165.224321 |
| **2021-12-30** | -132.233854 | 166.278026 |
| **2021-12-31** | -133.240848 | 167.331731 |

146 rows × 2 columns

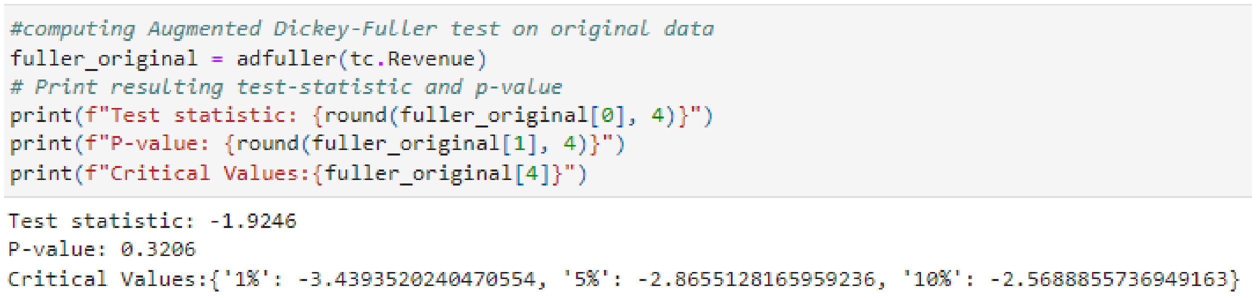
The forecasted visualization showed the telecommunication company's revenue in million per day. The data forecasted the dates of August 08, 2021, through December 31, 2021, which captured the 146 days of the testing data. The red dashed line represented the predicted values and the blue line represented the actual observed data. The predictions showed a steady increase in revenue during the specific time period.

```
#forecasted(Elleh, n.d.).
plt.figure(figsize = [16,8])
plt.title("WGU Telecommunication Company Daily Revenue from 2020-2021")
plt.xlabel("Date")
plt.ylabel("Daily Revenue (Millions USD")

# forecasted data
plt.plot(tc_forecast, color = 'red', linestyle = 'dashed')
# Plot using original dataset
plt.plot(tc, color = 'blue')
# confidence intervals
plt.fill_between(confid_intv.index, confid_intv['lower Revenue'],
                 confid_intv['upper Revenue'], color = 'pink')
plt.ylim(-7, 27)
plt.legend(['Predicted', 'Observed'])
plt.show();
```



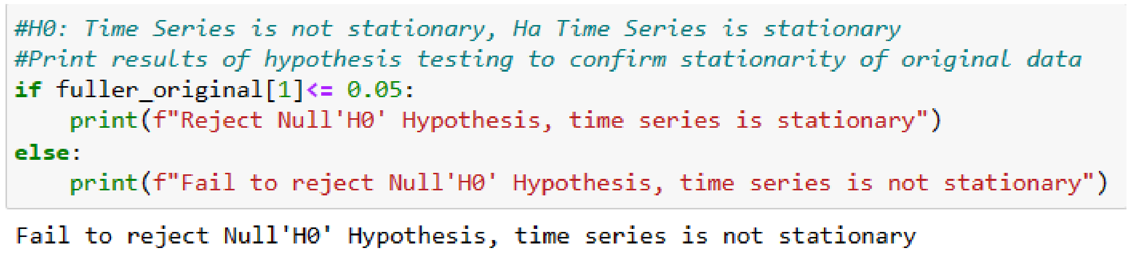
## D4. Outputs of Analysis

Please see below for the outputs and calculations of the analysis performed on the telecommunication data set. Please note that a majority of these are provided above.



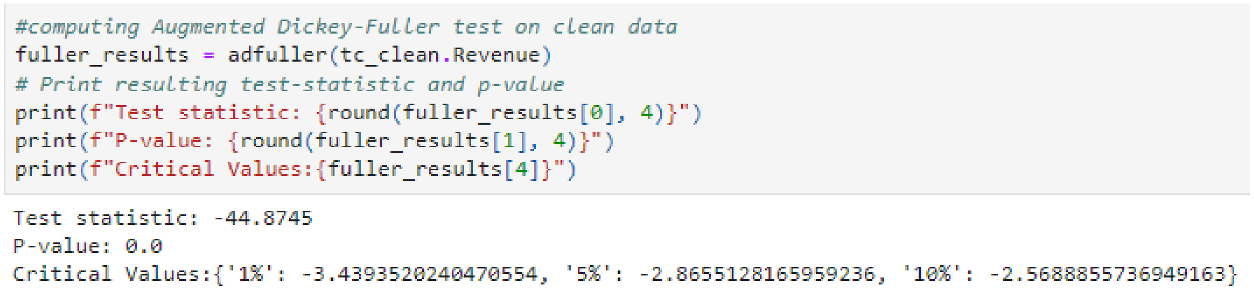
```python
#computing Augmented Dickey-Fuller test on original data
fuller_original = adfuller(tc.Revenue)
# Print resulting test-statistic and p-value
print(f"Test statistic: {round(fuller_original[0], 4)}")
print(f"P-value: {round(fuller_original[1], 4)}")
print(f"Critical Values:{fuller_original[4]}")
```

```
Test statistic: -1.9246
P-value: 0.3206
Critical Values:{'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
```



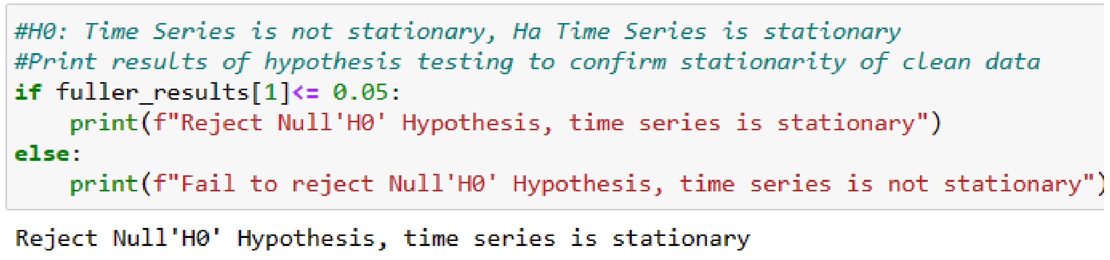
```python
#H0: Time Series is not stationary, Ha Time Series is stationary
#Print results of hypothesis testing to confirm stationarity of original data
if fuller_original[1]<= 0.05:
    print(f"Reject Null'H0' Hypothesis, time series is stationary")
else:
    print(f"Fail to reject Null'H0' Hypothesis, time series is not stationary")
```

```
Fail to reject Null'H0' Hypothesis, time series is not stationary
```



```python
#computing Augmented Dickey-Fuller test on clean data
fuller_results = adfuller(tc_clean.Revenue)
# Print resulting test-statistic and p-value
print(f"Test statistic: {round(fuller_results[0], 4)}")
print(f"P-value: {round(fuller_results[1], 4)}")
print(f"Critical Values:{fuller_results[4]}")
```

```
Test statistic: -44.8745
P-value: 0.0
Critical Values:{'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}
```

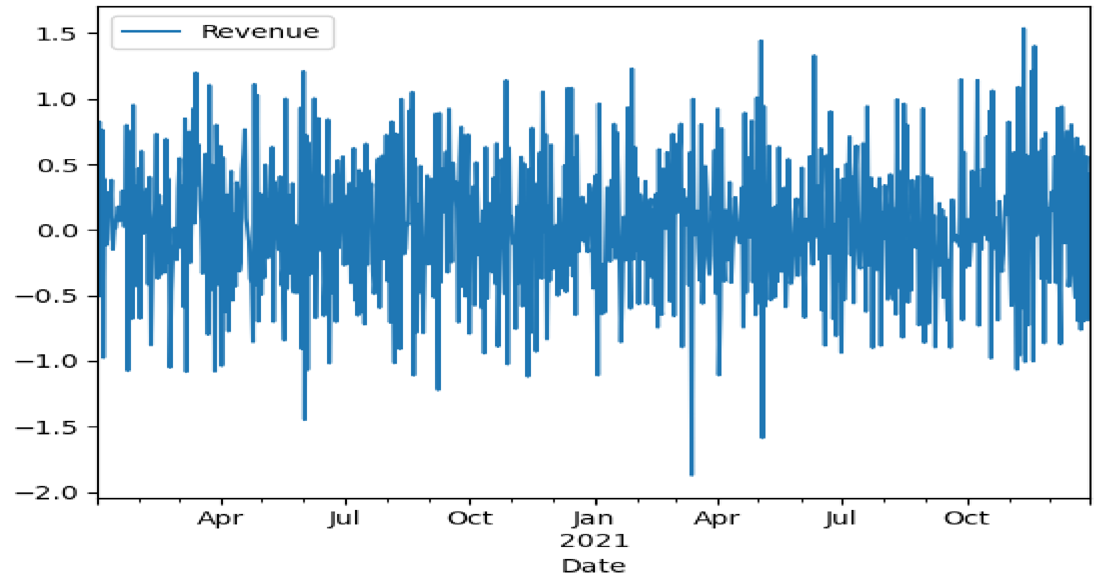


```python
#H0: Time Series is not stationary, Ha Time Series is stationary
#Print results of hypothesis testing to confirm stationarity of clean data
if fuller_results[1]<= 0.05:
    print(f"Reject Null'H0' Hypothesis, time series is stationary")
else:
    print(f"Fail to reject Null'H0' Hypothesis, time series is not stationary")
```

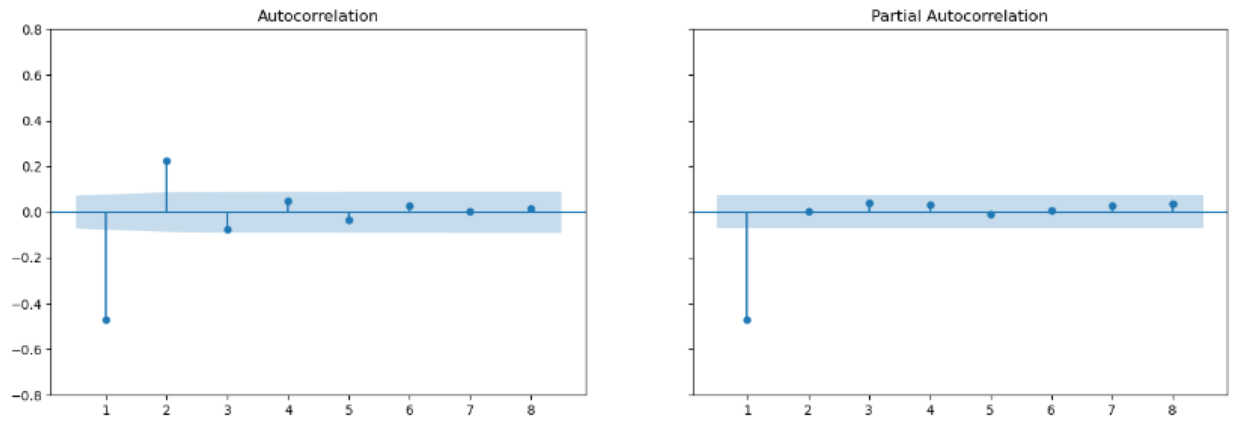
```
Reject Null'H0' Hypothesis, time series is stationary
```

```
# Plot to verify stationarity
tc_clean.plot()
```

```
<AxesSubplot: xlabel='Date'>
```



```
# Plot ACF and PACF(Elleh, n.d.).
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=[16,5], sharey=True)
# Plot ACF  ignore zero (zero always = 1)
plot_acf(tc_clean, lags=8, zero=False, ax=ax1)
# Plot PACF ignore zero (zero always = 1)
plot_pacf(tc_clean, lags=8, zero=False,ax=ax2)
plt.ylim(-0.8, 0.8);
```



```
#auto_arima to get best parameters(Pulagam,2020).
auto_arima(
    train_tc["Revenue"],
    start_p = 1, max_p = 6,
    start_q = 1, max_q = 6,
    seasonal = False, trace = True).summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=776.989, Time=0.17 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=909.948, Time=0.07 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=774.990, Time=0.08 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=801.099, Time=0.19 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=776.989, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=778.497, Time=1.03 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=773.893, Time=0.16 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=910.790, Time=0.15 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=775.886, Time=0.37 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=775.888, Time=0.31 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=799.464, Time=0.23 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=777.694, Time=1.77 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 4.648 seconds
```

SARIMAX Results

| | | | |
|---|---|---|---|
| **Dep. Variable:** | y | **No. Observations:** | 584 |
| **Model:** | SARIMAX(1, 0, 0) | **Log Likelihood** | -383.946 |
| **Date:** | Thu, 03 Aug 2023 | **AIC** | 773.893 |
| **Time:** | 20:49:56 | **BIC** | 787.002 |
| **Sample:** | 01-02-2020 | **HQIC** | 779.002 |
| | - 08-07-2021 | | |
| **Covariance Type:** | opg | | |

| | coef | std err | z | P>\|z\| | [0.025 | 0.975] |
|---|---|---|---|---|---|---|
| **intercept** | 0.0341 | 0.019 | 1.751 | 0.080 | -0.004 | 0.072 |
| **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 |

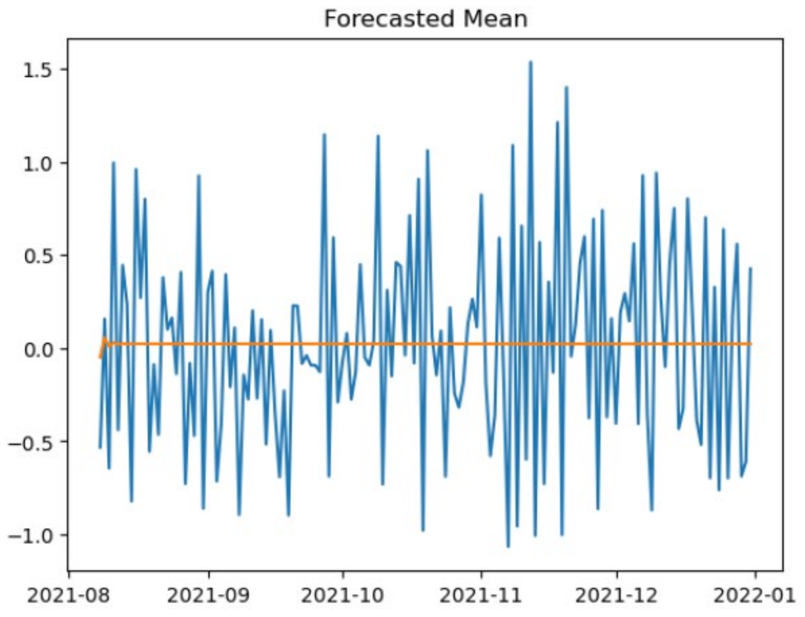
```python
#ARIMA model of time series using best parameters from auto arima(Elleh, n.d.).
arima_model = ARIMA(train_tc, order=(1, 0, 0), freq='D')
results = arima_model.fit()
print(results.summary())
```

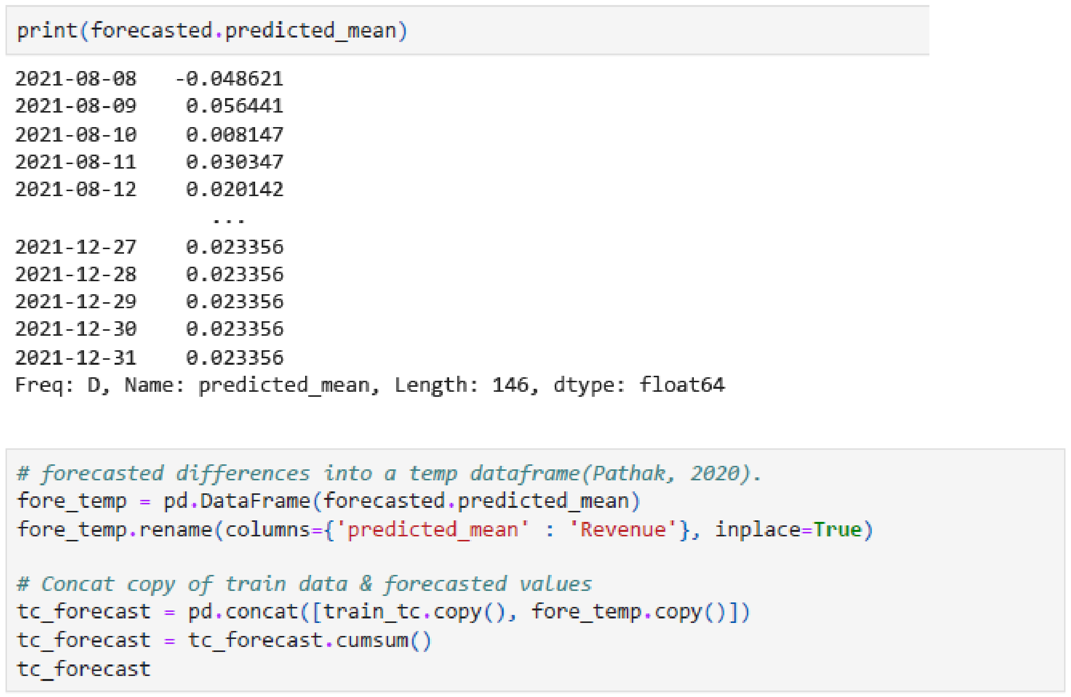
```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  584
Model:                 ARIMA(1, 0, 0)   Log Likelihood                -383.946
Date:                Thu, 03 Aug 2023   AIC                            773.893
Time:                        20:49:56   BIC                            787.002
Sample:                    01-02-2020   HQIC                           779.002
                         - 08-07-2021
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).
```

```python
#AIC
results.aic
```

```
773.8925878321593
```

```python
#forecasted mean(Pathak, 2020).
forecasted = results.get_prediction(start = 584, end = 729, dynamic = True)
plt.plot(test_tc)
plt.title('Forecasted Mean')
plt.plot(forecasted.predicted_mean);
```

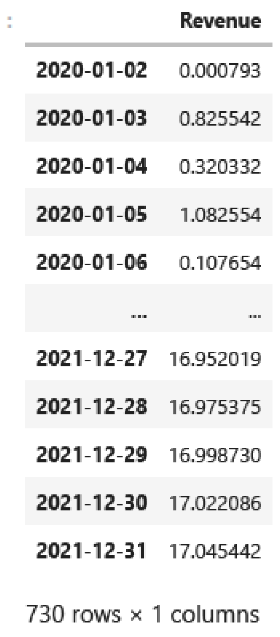




```
print(forecasted.predicted_mean)
```

```
2021-08-08   -0.048621
2021-08-09    0.056441
2021-08-10    0.008147
2021-08-11    0.030347
2021-08-12    0.020142
                ...
2021-12-27    0.023356
2021-12-28    0.023356
2021-12-29    0.023356
2021-12-30    0.023356
2021-12-31    0.023356
Freq: D, Name: predicted_mean, Length: 146, dtype: float64
```

```
# forecasted differences into a temp dataframe(Pathak, 2020).
fore_temp = pd.DataFrame(forecasted.predicted_mean)
fore_temp.rename(columns={'predicted_mean' : 'Revenue'}, inplace=True)

# Concat copy of train data & forecasted values
tc_forecast = pd.concat([train_tc.copy(), fore_temp.copy()])
tc_forecast = tc_forecast.cumsum()
tc_forecast
```



| | Revenue |
|---|---|
| **2020-01-02** | 0.000793 |
| **2020-01-03** | 0.825542 |
| **2020-01-04** | 0.320332 |
| **2020-01-05** | 1.082554 |
| **2020-01-06** | 0.107654 |
| ... | ... |
| **2021-12-27** | 16.952019 |
| **2021-12-28** | 16.975375 |
| **2021-12-29** | 16.998730 |
| **2021-12-30** | 17.022086 |
| **2021-12-31** | 17.045442 |

730 rows × 1 columns

```
# confidence intervals from forecasted
confid_intv = forecasted.conf_int()
confid_intv
```

| | lower Revenue | upper Revenue |
|---|---|---|
| **2021-08-08** | -0.963665 | 0.866422 |
| **2021-08-09** | -0.950645 | 1.063528 |
| **2021-08-10** | -1.017331 | 1.033625 |
| **2021-08-11** | -0.998976 | 1.059669 |
| **2021-08-12** | -1.009990 | 1.050275 |
| ... | ... | ... |
| **2021-12-27** | -1.006994 | 1.053705 |
| **2021-12-28** | -1.006994 | 1.053705 |
| **2021-12-29** | -1.006994 | 1.053705 |
| **2021-12-30** | -1.006994 | 1.053705 |
| **2021-12-31** | -1.006994 | 1.053705 |

146 rows × 2 columns

```
# df to match confidence intervals from orginak data pt 2021-08-07(Pathak, 2020).
pr = pd.DataFrame({'lower Revenue': [13.684826],
                                    'upper Revenue' : [13.684826], 'Date' : ['2021-08-07']})
# Convert given date string to datetime and then set as index
pr['Date'] = pd.to_datetime(pr['Date'])
pr.set_index('Date', inplace=True)
pr
```

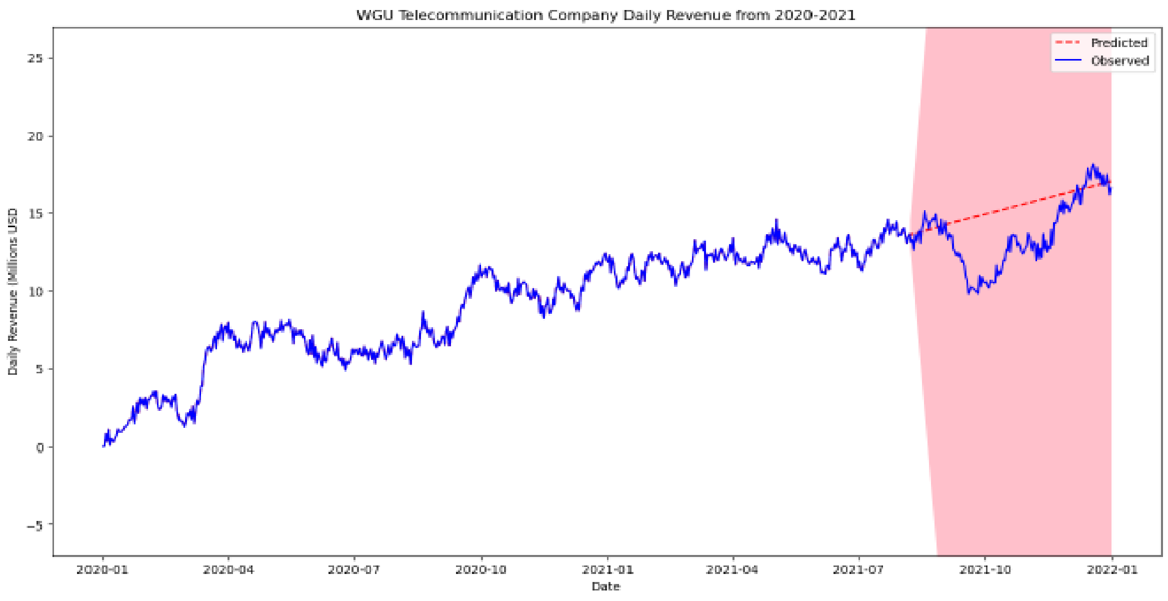
| | lower Revenue | upper Revenue |
|---|---|---|
| **Date** | | |
| **2021-08-07** | 13.684826 | 13.684826 |

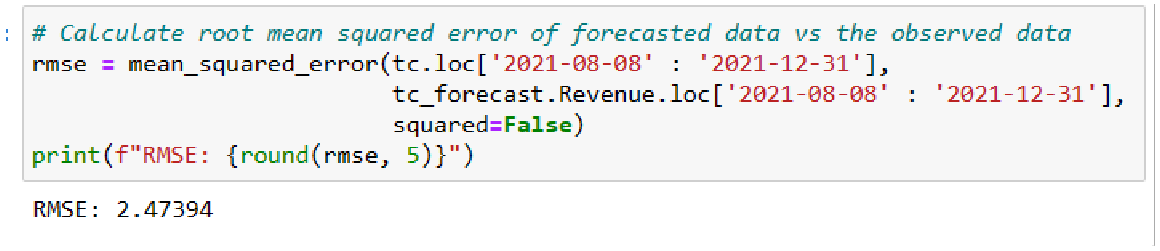
```
# Concat the previous row & confidence intervals
confid_intv = pd.concat([pr, confid_intv])

# Untransform the confidence intervals using cumsum()
confid_intv = confid_intv.cumsum()

# Setting range of data
confid_intv = confid_intv.loc['2021-08-08' : '2021-12-31']
confid_intv
```

| | lower Revenue | upper Revenue |
|---|---|---|
| 2021-08-08 | 12.721161 | 14.551248 |
| 2021-08-09 | 11.770516 | 15.614775 |
| 2021-08-10 | 10.753185 | 16.648401 |
| 2021-08-11 | 9.754210 | 17.708069 |
| 2021-08-12 | 8.744219 | 18.758344 |
| ... | ... | ... |
| 2021-12-27 | -129.212872 | 163.116910 |
| 2021-12-28 | -130.219866 | 164.170616 |
| 2021-12-29 | -131.226860 | 165.224321 |
| 2021-12-30 | -132.233854 | 166.278026 |
| 2021-12-31 | -133.240848 | 167.331731 |

146 rows × 2 columns

```
#forecasted(Elleh, n.d.).
plt.figure(figsize = [16,8])
plt.title("WGU Telecommunication Company Daily Revenue from 2020-2021")
plt.xlabel("Date")
plt.ylabel("Daily Revenue (Millions USD")

# forecasted data
plt.plot(tc_forecast, color = 'red', linestyle = 'dashed')
# Plot using original dataset
plt.plot(tc, color = 'blue')
# confidence intervals
plt.fill_between(confid_intv.index, confid_intv['lower Revenue'],
                 confid_intv['upper Revenue'], color = 'pink')
plt.ylim(-7, 27)
plt.legend(['Predicted', 'Observed'])
plt.show();
```





```
# Calculate root mean squared error of forecasted data vs the observed data
rmse = mean_squared_error(tc.loc['2021-08-08' : '2021-12-31'],
                          tc_forecast.Revenue.loc['2021-08-08' : '2021-12-31'],
                          squared=False)
print(f"RMSE: {round(rmse, 5)}")
```

```
RMSE: 2.47394
```

## D5. Prepared Code

Please see the below code for the implementation of the time series model for the telecommunication data set. This included the differencing of the data set, the computation of the Augmented Dickey-Fuller test, the splitting of the data, the selection of best parameters via .auto_arima() function, and plotting of autocorrelation/ partial autocorrelation. The code also includes the ARIMA model, forecasted mean, and final forecasted model. Please note all code was also included in the attached Jupyter Notebook PDF file as well.

```
##difference dataset, make stationarity
tc_clean = tc.diff().dropna()
# Save cleaned tc to csv
tc_clean.to_csv('AFD213Tk1_clean.csv')
#computing Augmented Dickey-Fuller test on original data
fuller_original = adfuller(tc.Revenue)
# test statistic, p-value, crit. values of original data
print(f"Test statistic: {round(fuller_original[0], 4)}")
print(f"P-value: {round(fuller_original[1], 4)}")
print(f"Critical Values:{fuller_original[4]}")
#H0: Time Series is not stationary, Ha Time Series is stationary
#Print results of hypothesis testing to confirm stationarity of original data
if fuller_original[1]<= 0.05:
   print(f"Reject Null'H0' Hypothesis, time series is stationary")
else:
    print(f"Fail to reject Null'H0' Hypothesis, time series is not stationary")
#computing Augmented Dickey-Fuller test on clean data
fuller_results = adfuller(tc_clean.Revenue)
#test statistic, p-value, crit. values of clean data
print(f"Test statistic: {round(fuller_results[0], 4)}")
print(f"P-value: {round(fuller_results[1], 4)}")
print(f"Critical Values:{fuller_results[4]}")
#H0: Time Series is not stationary, Ha Time Series is stationary
#Print results of hypothesis testing to confirm stationarity of clean data
if fuller_results[1]<= 0.05:
    print(f"Reject Null'H0' Hypothesis, time series is stationary")
else:
    print(f"Fail to reject Null'H0' Hypothesis, time series is not stationary")
# Plot to verify stationarity
tc_clean.plot()
# Split time series into a training set and a testing set
train_tc, test_tc = train_test_split(tc_clean, test_size=0.2, shuffle=False,
random_state=397)


#viewing training data set
train_tc
#viewing test data set
test_tc
# Save train tc to csv
train_tc.to_csv('AFD213Tk1_train_clean.csv')
# Save test tc to csv
test_tc.to_csv('AFD213Tk1_test_clean.csv')
##spectral density
plt.psd(x=tc_clean.Revenue);
# Plot ACF and PACF(Elleh, n.d.).
```

```
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=[16,5], sharey=True)
# Plot ACF  ignore zero
plot_acf(tc_clean, lags=8, zero=False, ax=ax1)
# Plot PACF ignore zero
plot_pacf(tc_clean, lags=8, zero=False,ax=ax2)
plt.ylim(-0.8, 0.8);
# Decompose the clean data set, plot time series
decomp_tc = seasonal_decompose(tc_clean)
decomp_tc.plot()
plt.title('Trend')
plt.show();
# viewing plot of trend.
plt.figure(figsize = [16,8])
plt.title('Trend')
plt.plot(decomp_tc.trend);
#viewing plot of seasonality
plt.figure(figsize = [16,8])
plt.title('Seasonality')
plt.plot(decomp_tc.seasonal);
#seasonality on weekly level viewed December-January, purple line: Sundays
(Unknown,n.d.).
plt.figure(figsize = [15,5])
plt.plot(decomp_tc.seasonal, marker='x')
plt.xlim(pd.to_datetime('2020-12-06'), pd.to_datetime('2021-01-31'))
plt.title('Seasonality of December 2020 - January 2021')
plt.axvline(x=pd.to_datetime('2020-12-06'), color='purple')
plt.axvline(x=pd.to_datetime('2020-12-13'), color='purple')
plt.axvline(x=pd.to_datetime('2020-12-20'), color='purple')
plt.axvline(x=pd.to_datetime('2020-12-27'), color='purple')
plt.axvline(x=pd.to_datetime('2021-01-03'), color='purple')
plt.axvline(x=pd.to_datetime('2021-01-10'), color='purple')
plt.axvline(x=pd.to_datetime('2021-01-17'), color='purple')
plt.axvline(x=pd.to_datetime('2021-01-24'), color='purple')
plt.axvline(x=pd.to_datetime('2021-01-31'), color='purple');
# residual components plot
plt.figure(figsize = [16,8])
plt.title('Residuals')
plt.plot(decomp_tc.resid);
#auto_arima to get best parameters(Pulagam,2020).
auto_arima(
   train_tc["Revenue"],
   start_p = 1, max_p = 6,
   start_q = 1, max_q = 6,
   seasonal = False, trace = True).summary()
#ARIMA model of time series using best parameters from auto arima(Elleh, n.d.).
arima_model = ARIMA(train_tc, order=(1, 0, 0), freq='D')
results = arima_model.fit()
print(results.summary())
#AIC
results.aic
#forecasted mean
forecasted = results.get_prediction(start = 584, end = 729, dynamic = True)
plt.plot(test_tc)
plt.title('Forecasted Mean')
plt.plot(forecasted.predicted_mean);
print(forecasted.predicted_mean)
# forecasted differences into a temp dataframe(Pathak, 2020).
fore_temp = pd.DataFrame(forecasted.predicted_mean)
fore_temp.rename(columns={'predicted_mean' : 'Revenue'}, inplace=True)
```

```
# Concat copy of train data & forecasted values

tc_forecast = pd.concat([train_tc.copy(), fore_temp.copy()])

tc_forecast = tc_forecast.cumsum()

tc_forecast

# confidence intervals from forecasted

confid_intv = forecasted.conf_int()

confid_intv

# df to match confidence intervals from original data pt 2021-08-07(Pathak,
2020).

pr = pd.DataFrame({'lower Revenue': [13.684826],

                    'upper Revenue' : [13.684826], 'Date' : ['2021-08-07']})

# Convert given date string to datetime and then set as index

pr['Date'] = pd.to_datetime(pr['Date'])

pr.set_index('Date', inplace=True)

pr

# Concat the previous row & confidence intervals

confid_intv = pd.concat([pr, confid_intv])

# Untransform the confidence intervals using cumsum()

confid_intv = confid_intv.cumsum()

# setting range of data

confid_intv = confid_intv.loc['2021-08-08' : '2021-12-31']

confid_intv

#forecasted(Elleh, n.d.).

plt.figure(figsize = [16,8])

plt.title("WGU Telecommunication Company Daily Revenue from 2020-2021")

plt.xlabel("Date")

plt.ylabel("Daily Revenue (Millions USD")


# forecasted data(Elleh, n.d.).

plt.plot(tc_forecast, color = 'red', linestyle = 'dashed')
```

```
# Plot using original dataset
plt.plot(tc, color = 'blue')
# confidence intervals
plt.fill_between(confid_intv.index, confid_intv['lower Revenue'],
                 confid_intv['upper Revenue'], color = 'pink')
plt.ylim(-7, 27)
plt.legend(['Predicted', 'Observed'])
plt.show();
# Calculate root mean squared error of forecasted data vs the observed data
rmse = mean_squared_error(tc.loc['2021-08-08' : '2021-12-31'],
                         tc_forecast.Revenue.loc['2021-08-08' : '2021-12-31'],
squared=False)
print(f"RMSE: {round(rmse, 5)}")
```

## Part V: Data Summary and Implications

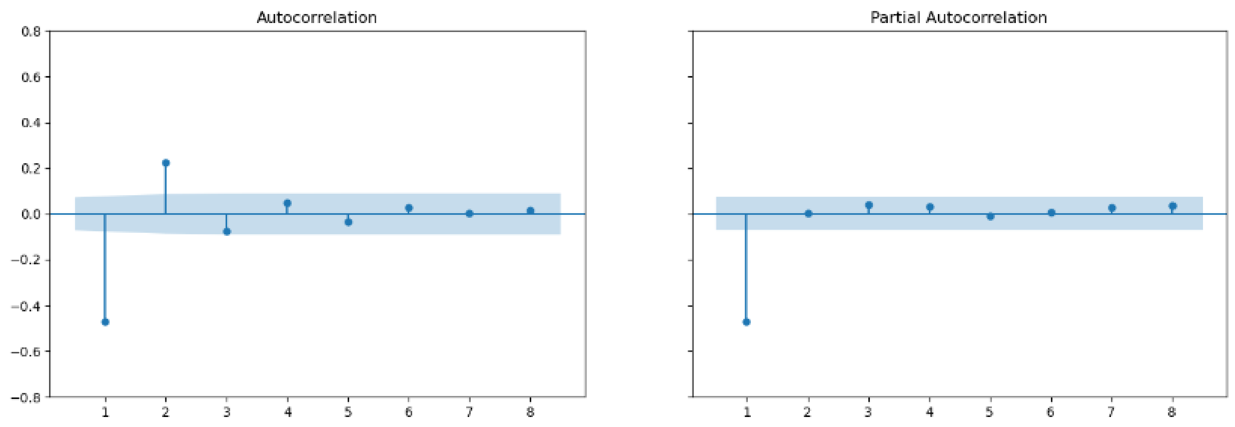
### E1. Results and Implications

The time series analysis of the telecommunication company provided various findings and results. These included the following points,

- Selection of an ARIMA model
- Prediction Interval of Forecast
- Justification of Forecast Length
- Model Evaluation and Error Metric

The use of the autocorrelation and partial autocorrection functions provided the selection results of the data. The telecommunication data was the best fit for the autoregression model. As observed within the chart, the ACF tailed off at 2 lag steps with the PACF cutting off at 1 lag step. This deemed the model the best match for AR(1) model.

```
# Plot ACF and PACF(Elleh, n.d.).
fig, (ax1, ax2) = plt.subplots(1, 2, figsize=[16,5], sharey=True)
# Plot ACF  ignore zero (zero always = 1)
plot_acf(tc_clean, lags=8, zero=False, ax=ax1)
# Plot PACF ignore zero (zero always = 1)
plot_pacf(tc_clean, lags=8, zero=False,ax=ax2)
plt.ylim(-0.8, 0.8);
```

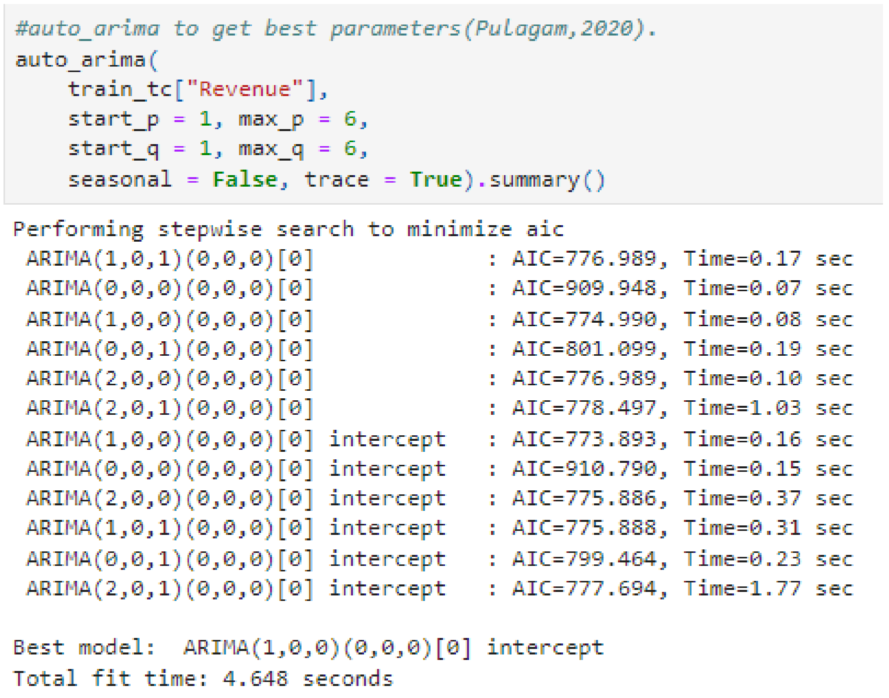


The prediction interval of the forecast was 146 days or steps. These steps represented the 20% of data that made the testing set. The decision to split the original data set into an 80:20 ratio was due to the size of the data and to keep the forecasted values in a small sampling. The training set which represented about 80% or around 19-20 months of the data was utilized in predicting the remaining 20% or 4-5 months of testing data.

The forecast length was justifiable. The length provided an adequate comparison to the observed sampled revenue. This would provide the telecommunication company with an idea of how accurate the forecasting model performed on a small sample size of about 4-5 months. Larger forecast lengths would need additional legacy data to provide adequate forecasting results.

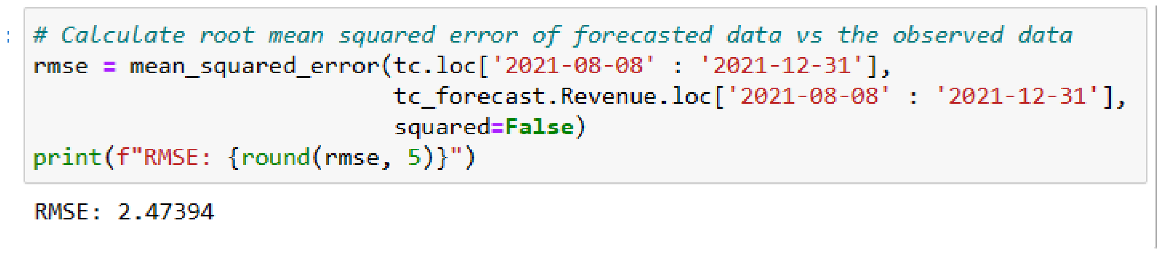
Model evaluation was completed with the use of the auto_arima() function. The function provided a stepwise computation to provide the model parameters with the best AIC value. Akaike Information Criterion (AIC) was a mathematical method for evaluating how well a model fits the data it was created from (Zajic, 2022). The time series model via ARIMA was generated with the use of the parameters providing the lowest AIC results, which was 773.893.

Lastly, the model error metric was computed with the use of root mean squared error(RMSE). The lower the RMSE value the better. It would indicate the greater accuracy of the model. The metric provided the average squared distance between the predicted values and the actual observed values (Bobbitt, 2021). The RMSE value for this analysis was 2.47394, which was pretty low.



```
#auto_arima to get best parameters(Pulagam,2020).
auto_arima(
    train_tc["Revenue"],
    start_p = 1, max_p = 6,
    start_q = 1, max_q = 6,
    seasonal = False, trace = True).summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(0,0,0)[0]             : AIC=776.989, Time=0.17 sec
 ARIMA(0,0,0)(0,0,0)[0]             : AIC=909.948, Time=0.07 sec
 ARIMA(1,0,0)(0,0,0)[0]             : AIC=774.990, Time=0.08 sec
 ARIMA(0,0,1)(0,0,0)[0]             : AIC=801.099, Time=0.19 sec
 ARIMA(2,0,0)(0,0,0)[0]             : AIC=776.989, Time=0.10 sec
 ARIMA(2,0,1)(0,0,0)[0]             : AIC=778.497, Time=1.03 sec
 ARIMA(1,0,0)(0,0,0)[0] intercept   : AIC=773.893, Time=0.16 sec
 ARIMA(0,0,0)(0,0,0)[0] intercept   : AIC=910.790, Time=0.15 sec
 ARIMA(2,0,0)(0,0,0)[0] intercept   : AIC=775.886, Time=0.37 sec
 ARIMA(1,0,1)(0,0,0)[0] intercept   : AIC=775.888, Time=0.31 sec
 ARIMA(0,0,1)(0,0,0)[0] intercept   : AIC=799.464, Time=0.23 sec
 ARIMA(2,0,1)(0,0,0)[0] intercept   : AIC=777.694, Time=1.77 sec

Best model:  ARIMA(1,0,0)(0,0,0)[0] intercept
Total fit time: 4.648 seconds
```

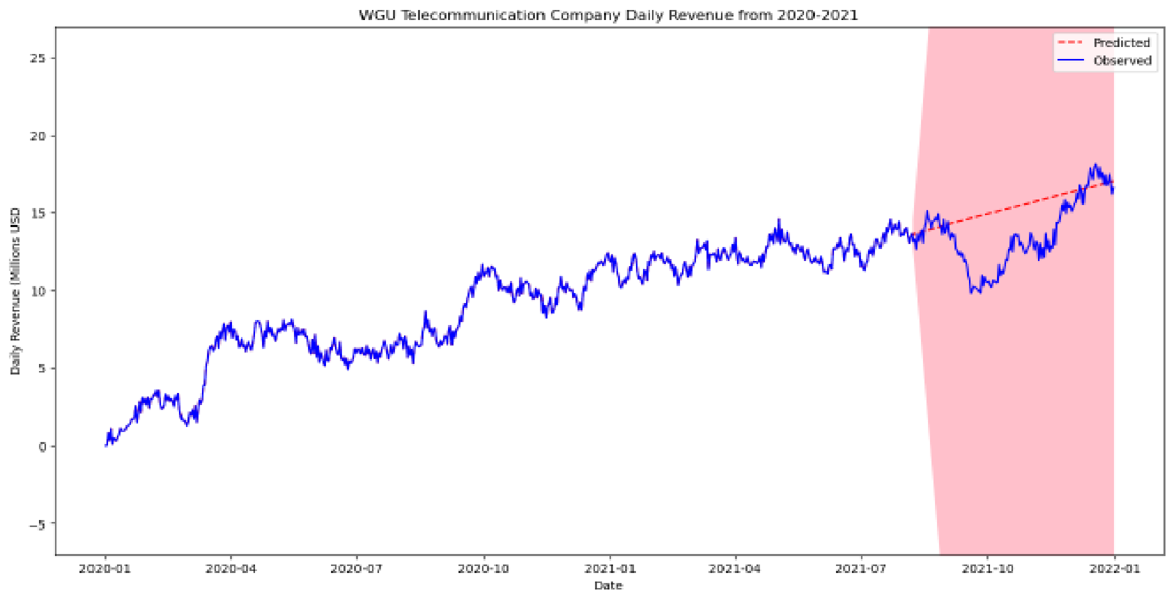


```
# Calculate root mean squared error of forecasted data vs the observed data
rmse = mean_squared_error(tc.loc['2021-08-08' : '2021-12-31'],
                          tc_forecast.Revenue.loc['2021-08-08' : '2021-12-31'],
                          squared=False)
print(f"RMSE: {round(rmse, 5)}")
```

```
RMSE: 2.47394
```

## E2. Visualization of Final Model

The final visualization could be viewed below. It included the plots of both the observations of the telecommunication data set as well as the forecasts generated by the model. The forecasted model plots which were noted in red, showed a steady increase of 10 million to 15 million in "Revenue" from August 2021 to December 2021. The attempt determined forecasting could be done, yet effectively was not noted. This was summarized as such due to the observations during that period having a decrease in "Revenue". The observed Revenue dropped as low as 5 million and did not increase more than 10 million in the same forecasted months.



## E3. Course of Action

The recommended course of action for the telecommunication company would be the following suggestion, further investigation of the daily revenue fluctuations during the end of 2021. Although the time series via ARIMA could forecast the testing data's daily revenue, it seemed as though it could not accurately do so for all months being forecasted. The observed daily revenue had noted a drop in revenue during the months of September through November 2021; whereas, during the same period, the predicted daily revenue showed a steady upward trend. The upward trend aligned with the prior 18- 19 months' observed daily revenue. The telecommunication company's investigation efforts could include additional data from prior annual revenue to determine if the noted observations were outliers or a norm for the company during those months.

## Part VI: Reporting

### F. Industry Relevant Interactive Report

Please reference included copy of executed Jupyter Notebook in PDF format: "AFCodeD213Tk1.PDF"

### G. Third-Party Web Sources

Elleh, Festus. (n.d.). *Advanced Data Analytics - Task 1* [Video]. College of Information Technology. Western Governors University. https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=efceba6c-e8ef-47a2-b859-aec400fe18e7

Pathak, Prabhat. (2020, October 29). *How to Create an ARIMA Model for Time Series Forecasting in Python.*Retrieved from Analytics Vidhya: https://www.analyticsvidhya.com/blog/2020/10/how-to-create-an-arima-model-for-time-series-forecasting-in-python/

Pulagam, Sushmitha. (2020, June 26). *Time Series forecasting using Auto ARIMA in Python.* Retrieved from TowardDataScience: https://towardsdatascience.com/time-series-forecasting-using-auto-arima-in-python-bb83e49210cd

Unknown.(n.d.). *How do you plot a vertical line on a time series plot in Pandas?*. Retrieved from TutorialsPoint: https://www.tutorialspoint.com/how-do-you-plot-a-vertical-line-on-a-time-series-plot-in-pandas

## H. References

Bevans, R. (2020, August 7). *Understanding Confidence Intervals*. Retrieved from Scribbr: https://www.scribbr.com/statistics/confidence-interval/

Bobbitt, Z. (2021, May 10). *How to Interpret Root Mean Square Error (RMSE)*. Retrieved from Statology: https://www.statology.org/how-to-interpret-rmse/

Fulton, J. (n.d.). *Intro to ACF and PACF*. Retrieved from DataCamp: https://campus.datacamp.com/courses/arima-models-in-python/the-best-of-the-best-models?ex=1&learningMode=course

Reider, R. (n.d.). *Time Series Analysis in Python*. Retrieved from Datacamp: https://app.datacamp.com/learn/courses/time-series-analysis-in-python

Tejalkadam18m. (2022, May 30). *What is trend in time series?* Retrieved from Geeks for Geeks: https://www.geeksforgeeks.org/what-is-a-trend-in-time-series/

Wu, S. (2021, July 4). *Stationarity Assumption in Time Series Data*. Retrieved from Toward DataScience: https://towardsdatascience.com/stationarity-assumption-in-time-series-data-67ec93d0f2f

Zajic, A. (2022, November 29). *What is Akaike Information Criterion (AIC)?* Retrieved from BuiltIn: https://builtin.com/data-science/what-is-aic