Task 1: Time Series Modeling - ARIMA – D213

Abhishek Aern

Western Governor University

Contents

Part I - Research Question

A1: Question for analysis

What is the optimal ARIMA model that accurately forecasts the future daily revenue for the next 180 days for a telecom company, based on the historical data of daily revenue over the first two years of operation?

A2: Goal of analysis

The goal of this research question is to identify the optimal ARIMA model that accurately forecasts the future daily revenue for the next 180 days for a telecom company. By utilizing historical data of daily revenue over the first two years of operation, the objective is to develop an ARIMA model that can effectively capture the underlying patterns and trends in the revenue data. The aim is to provide accurate and reliable forecasts of daily revenue, which can assist the telecom company in financial planning, budgeting, and decision-making related to resource allocation, pricing strategies, and overall business operations.

The research seeks to optimize the ARIMA model's parameters and configuration to ensure the most accurate predictions of daily revenue for the company's short-term and long-term planning and performance evaluation.

Part II - Method Justification

B: Summary of Assumptions

Here are some of the assumptions for Time series models (Pandian, 2021)–

- Time series models assume that the underlying data follows a stationary process. Stationarity means that the statistical properties of the data, such as mean, variance, and covariance, remain constant over time.

- Time series data often exhibit autocorrelation, which means that the observations at different time points are correlated with each other. Autocorrelation implies that the value of a variable at a specific time is influenced by its past values.

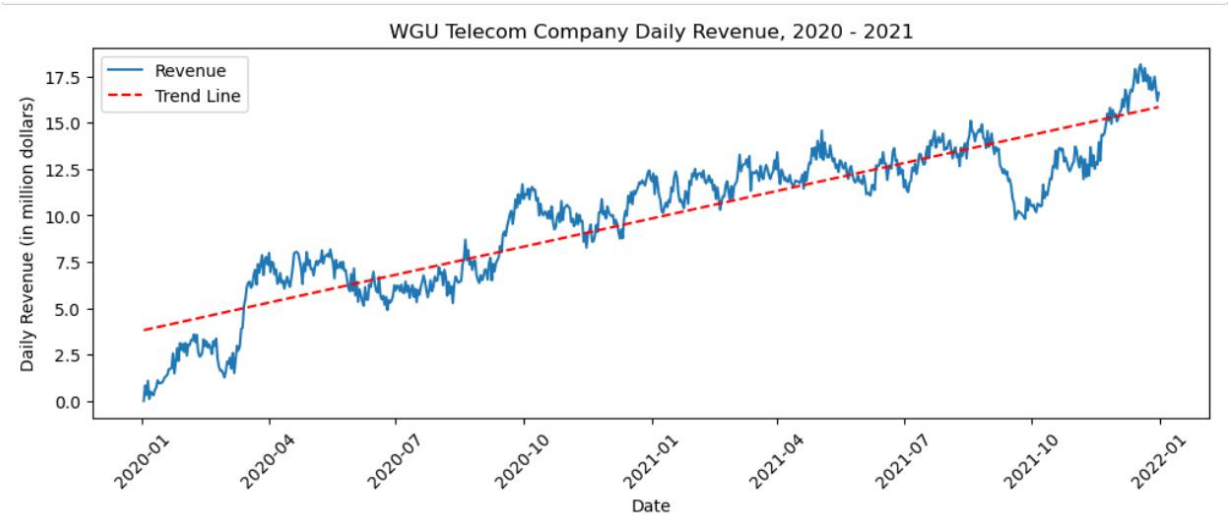
- Time series models assume that the residuals (or errors) are independent of each other. The residuals should not contain any further information or patterns that can be used to predict future values beyond what is captured by the model.

- Time series models often assume that the variance of the residuals remains constant over time. This assumption is called homoscedasticity.

Part III - Data Preparation

C1: Line Graph Visualization

Below is the line graph visualizing for "WGU Telecom Company Daily Revenue, 2020 - 2021".



In this graph, Daily Revenue is shown on the y-axis over the date on the x-axis. The upward trend line(red color) suggests that the revenue for the given period is growing or getting higher as time progresses but a comprehensive analysis is required to consider additional factors and statistical measures to draw meaningful conclusions.

C2: Time Step Formatting

Based on the given data, the time step formatting of the realization appears to be sequential with a uniform time interval of one day between each measurement for Revenue. The length of the sequence is 731, indicating that there are 731 data points

in the time series divided into 2 years which suggests that 1 year is a leap year. There are no explicit gaps in the measurements.

To make the interpretation more meaningful I am initializing the date with 2020-01-01 as 2020 was the leap year so this data is representing daily revenue for the years 2020 and 2021.

C3: Stationarity

The upward trend line suggests that this data is not stationary. There are various methods to evaluate the stationarity of a time series. I am using Augmented Dickey-Fuller (ADF) Test.

The ADF test is a statistical test commonly used to assess stationarity. It tests the null hypothesis that a unit root is present in a time series (indicating non-stationarity). If the test results in a p-value below a certain threshold (e.g., 0.05 in this case), the null hypothesis can be rejected, indicating stationarity.

I created the function adfuller_test (series) that performs the Augmented Dickey-Fuller (ADF) test on a given time series to evaluate its stationarity. The function adfuller() from the statsmodels library is called to perform the ADF test on the input series. The ADF test returns several statistical values used to assess stationarity. A loop is used to iterate over the result values and corresponding labels. The code checks the p-value (result[1]) obtained from the ADF test. If the p-value is less than or equal to 0.05, it indicates strong evidence against the null hypothesis (Ho) and suggests that the data has no unit root, making it stationary. If the p-value is greater than 0.05, it indicates weak evidence against the null hypothesis and suggests that the time series has a unit root, making it non-stationary.

In my case, the code prints a message indicating weak evidence against the null hyp othesis, the time series has a unit root, indicating it is non-stationary.

```
#Ho: It is non stationary
#H1: It is stationary

def adfuller_test(series):
    result=adfuller(series)
    labels = ['ADF Test Statistic','p-value','#Lags Used','Number of Observations Used']
    for value,label in zip(result,labels):
        print(label+' : '+str(value) )
    if result[1] <= 0.05:
        print("strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationa
    else:
        print("weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary ")
```

```
adfuller_test(df_telco['Revenue'])
```

```
ADF Test Statistic : -1.924612157310184
p-value : 0.3205728150793963
#Lags Used : 1
Number of Observations Used : 729
weak evidence against null hypothesis, time series has a unit root, indicating it is non-stationary
```

C4: Steps to Prepare the Data

After the data is imported into DataFrame df_telco, the following steps are performed (Pathak, 2020)–

- Check for null using the .isnull() function and no null value is found.
- Check for missing values using the .isna() function and no missing value is found.
- Converted the 'Day' columns to the ‘Date’ for time series analysis to help interpret the results in a more meaningful way. The code uses the pd.to_timedelta() function to convert the numeric values in the "Day" column to timedelta objects representing the number of days. By subtracting 1 from each value (df_telco ['Day'] - 1) to align the first day with the starting date(2020-01-01). The resulting timedelta objects are then added to the start_date, and the dates are stored in the new "Date" column.
- Removed the original 'Days' column using the .drop method and set the ‘Date’ column as an index.

As the data is not stationary, I need to make it stationary so transform it using the differencing technique to achieve stationarity before applying ARIMA. The differencing technique helps in stabilizing the mean and removing any systematic patterns or trends in the data. It involves subtracting the previous observation from the current observation to remove trends or seasonality. The resulting series is called the differenced series.

I am performing first-order differencing by subtracting the previous observation from the current observation. The periods parameter specifies the number of periods to shift for the differencing. In this case, periods=1 indicates a first-order difference, where each value is subtracted from its previous value. The .dropna() removes any rows with missing values (NaN) from the resulting DataFrame.
The result of this code is a new DataFrame df_telco_stationary, which contains the differenced series of the 'Revenue' column range from 2020-01-02 to 2021-12-31.

```
# Make the data stationary

df_telco_stationary = df_telco.diff(periods=1).dropna()
df_telco_stationary
```

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
...	...
2021-12-27	0.170280
2021-12-28	0.559108
2021-12-29	-0.687028
2021-12-30	-0.608824
2021-12-31	0.425985

730 rows × 1 columns

The differenced series will have one less data point than the original series. After applying the first differencing, checking for stationarity again using the ADF function created earlier, and found that data is now stationary.

Here are the results –

```
#test for stationary again
adfuller_test(df_telco_stationary['Revenue'])
```

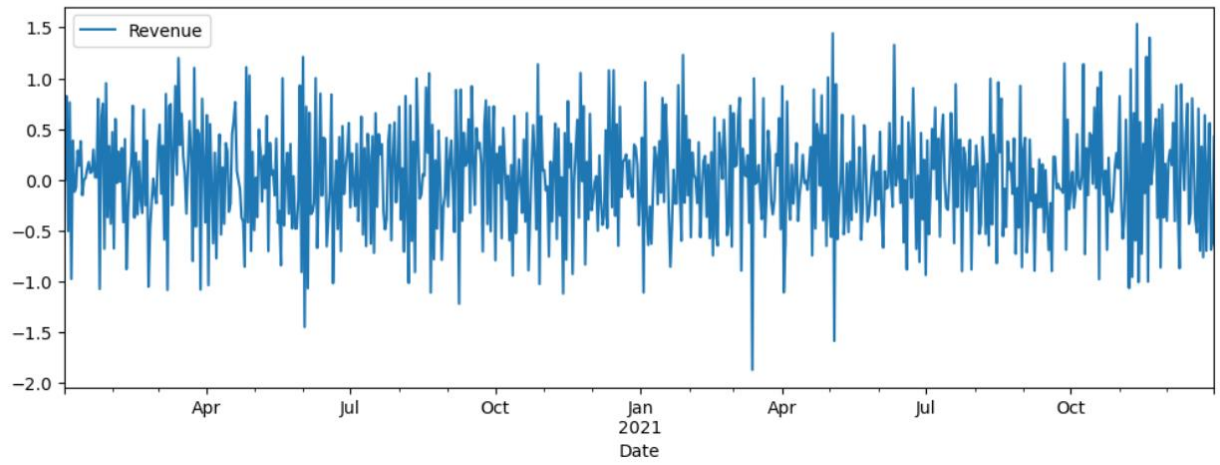
```
ADF Test Statistic : -44.87452719387599
p-value : 0.0
#Lags Used : 0
Number of Observations Used : 729
strong evidence against the null hypothesis(Ho), reject the null hypothesis. Data has no unit root and is stationary
```

Results : This data is now stationary

```
# Plot to verify the stationary

df_telco_stationary.plot(figsize=(12, 4))
```

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



The above data plot looks like stationary and I also do not see any seasonality in this graph.

Train/Test split –

To perform a test and train split on time series data I need to ensure that the splitting is done in a way that preserves the temporal order of the observations. I am using the train_test_split() function from the sklearn.model_selection module is used to split the data into training and testing sets. The test_size parameter is set to 0.2, indicating that 20% of the data will be allocated for testing, while 80% will be used for training. The shuffle parameter is set to False to ensure that the splitting is done based on the temporal order of the observations. This is crucial for time series data to maintain the integrity of the sequence (Parikh) .

A total of 584 records from 2020-01-01 to 2021-08-06 are used for Training data an d the remaining 147 records from 2021-08-07 to 2021-12-31 will be used for testin g data.

```
In [14]: # Perform test and train split
         train_data, test_data = train_test_split(df_telco, test_size=0.2, shuffle=False)

         # Print the split datasets
         print("Training Data:")
         print(train_data)
         print("\nTesting Data:")
         print(test_data)
```

```
Training Data:
              Revenue
Date
2020-01-01   0.000000
2020-01-02   0.000793
2020-01-03   0.825542
2020-01-04   0.320332
2020-01-05   1.082554
...               ...
2021-08-02  13.938920
2021-08-03  14.052184
2021-08-04  13.520478
2021-08-05  13.082643
2021-08-06  13.504886

[584 rows x 1 columns]

Testing Data:
              Revenue
Date
2021-08-07  13.684826
2021-08-08  13.152903
2021-08-09  13.310290
2021-08-10  12.665601
2021-08-11  13.660658
...               ...
2021-12-27  16.931559
2021-12-28  17.490666
2021-12-29  16.803638
2021-12-30  16.194813
2021-12-31  16.620798

[147 rows x 1 columns]
```

C5: Prepared Dataset

I am providing prepared original clean data in ‘telco_clean.csv’, prepared clean stationary data into ‘telco_clean_stationary.csv’ along with training data in train_data_clean.csv and test data in test_data_clean.csv files.

Part IV - Model Identification and Analysis

D1: Report Findings and Visualizations

Decomposed time series -

The seasonal_decompose function will decompose a time series data into its components (trend, seasonality, and residuals) to visualize their contributions to the overall time series (Hirantha, 2021).

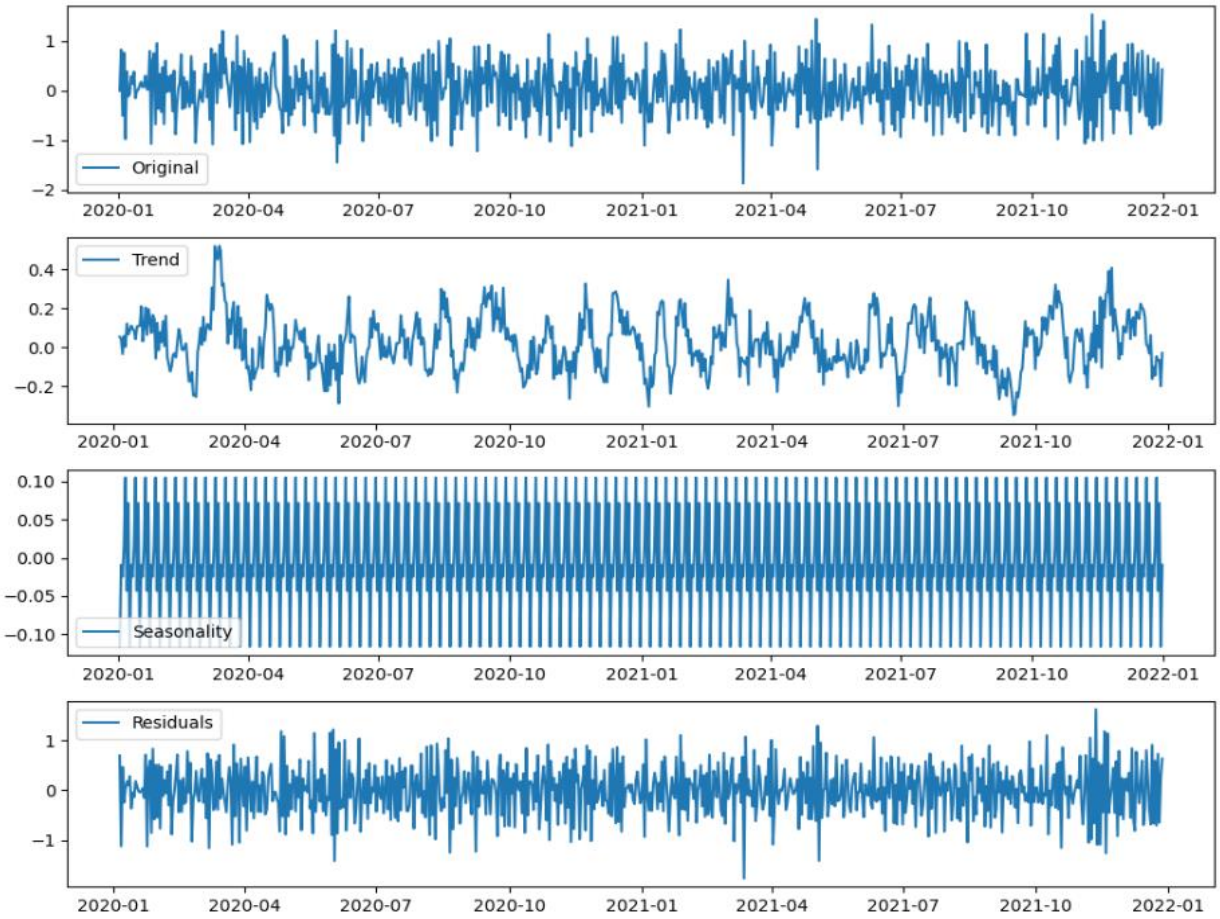
The seasonal_decompose() function is applied to the "Revenue" column of the DataFrame. The model parameter is set to 'additive', indicating that the components of the time series will be additive. The code then plots the original time series, trend component, seasonality component, and residuals component using Matplotlib. The plt.subplot() function is used to create subplots for each component, and the plt.plot() function is used to plot the respective component.

```
# Perform seasonal decomposition
result = seasonal_decompose(df_telco_stationary['Revenue'], model='additive')

result
```

```
<statsmodels.tsa.seasonal.DecomposeResult at 0x1f0b20b2d30>
```

```
# Plot the decomposition components
plt.figure(figsize=(10, 8))
plt.subplot(411)
plt.plot(df_telco_stationary['Revenue'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(result.trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(result.seasonal, label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(result.resid, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```



Observations :

Observed: The observed component represents the original time series data. It shows the actual values of the time series without any decomposition. This graph shows that the data is stationary.

Trend: The trend component captures the long-term variation or the overall direction of the time series. It represents the underlying trend or pattern that persists over an extended period, regardless of seasonal or irregular fluctuations.
By analyzing this trend, there is no trend in df_telco_stationary as all the values are centered around zero with a random trend.

Seasonal: The seasonal component represents the periodic pattern or cycles within the time series. It captures the regular fluctuations that occur at fixed intervals, such as daily, weekly, monthly, or yearly patterns. I do not see any regular fluctuations so Looks like no seasonality is present in my dataset.

Residual: The residual component, also known as the remainder or error component, represents the random or irregular fluctuations that remain after removing the trend and seasonal components from the observed data. It contains unexplained or unpredictable variations in the time series. I do not see any notable patterns or correlations in the residual component that suggests that the trend and seasonal components have been effectively removed from the observed data. The residuals show no significant autocorrelation.

The presence or lack of a seasonal component

I did not see any seasonality in the data as per the previous explanation. Looking more closely into the DataFrame by selecting the specific period for the first 180 days for the years 2020 and 2021 to find any related seasonal patterns.

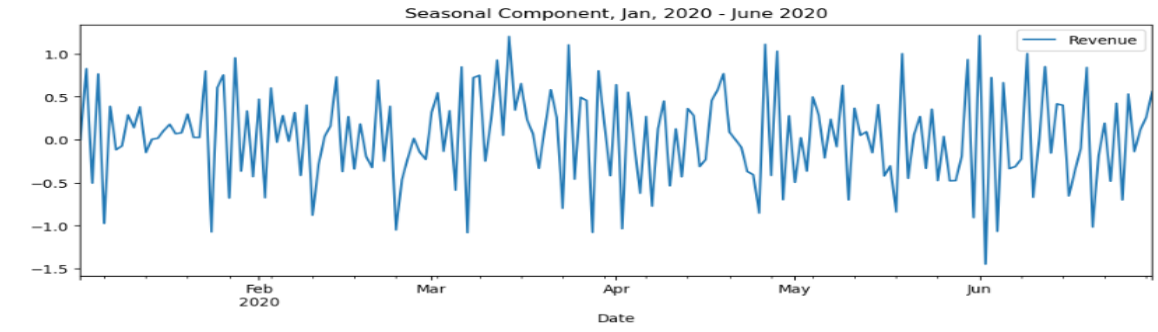
I can only see the irregular peaks and troughs in both seasonal components of a time series. Based on this I am concluding that no seasonality is present in my dataset.

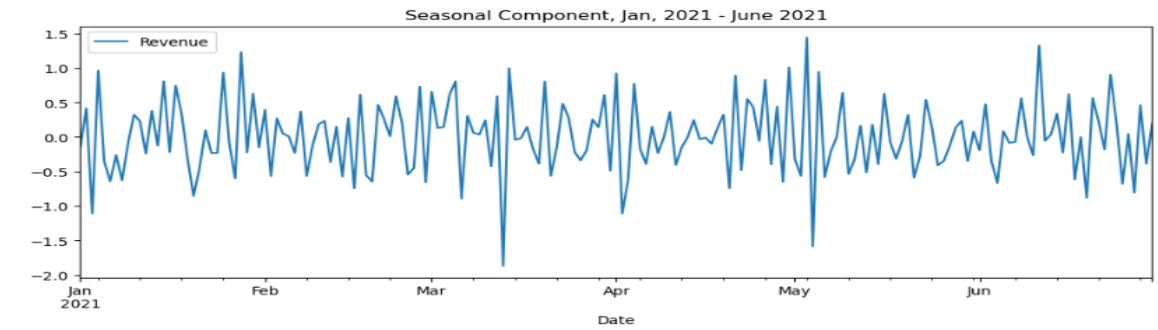
```
df_telco_stationary.loc['2020-01-02' :'2020-06-30'].plot(figsize=(12,4))

# Plot the seasonal component,
plt.title('Seasonal Component, Jan, 2020 - June 2020')

df_telco_stationary.loc['2021-01-01' :'2021-06-30'].plot(figsize=(12,4))
# Plot the seasonal component,
plt.title('Seasonal Component, Jan, 2021 - June 2021')
```

```
Text(0.5, 1.0, 'Seasonal Component, Jan, 2021 - June 2021')
```



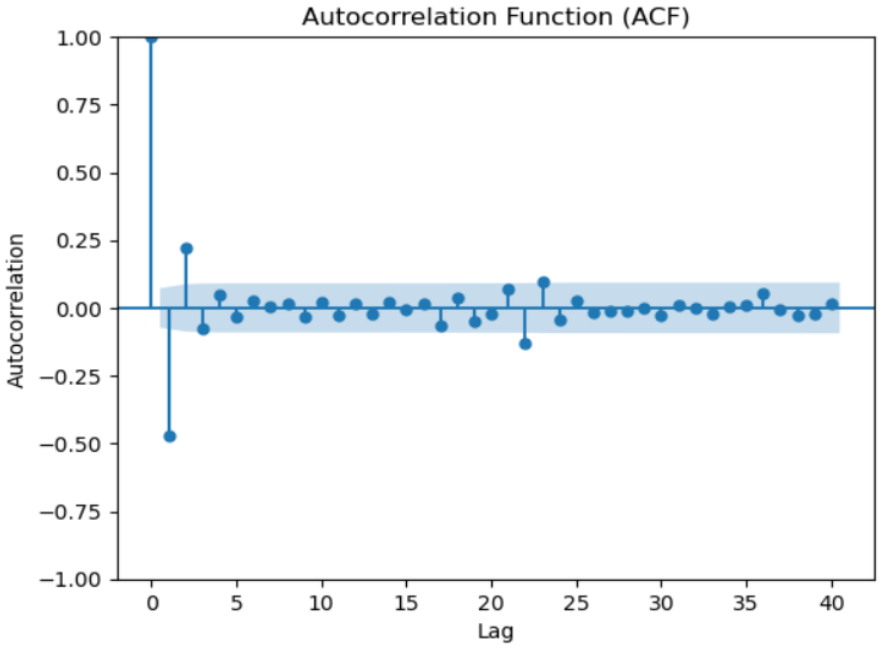


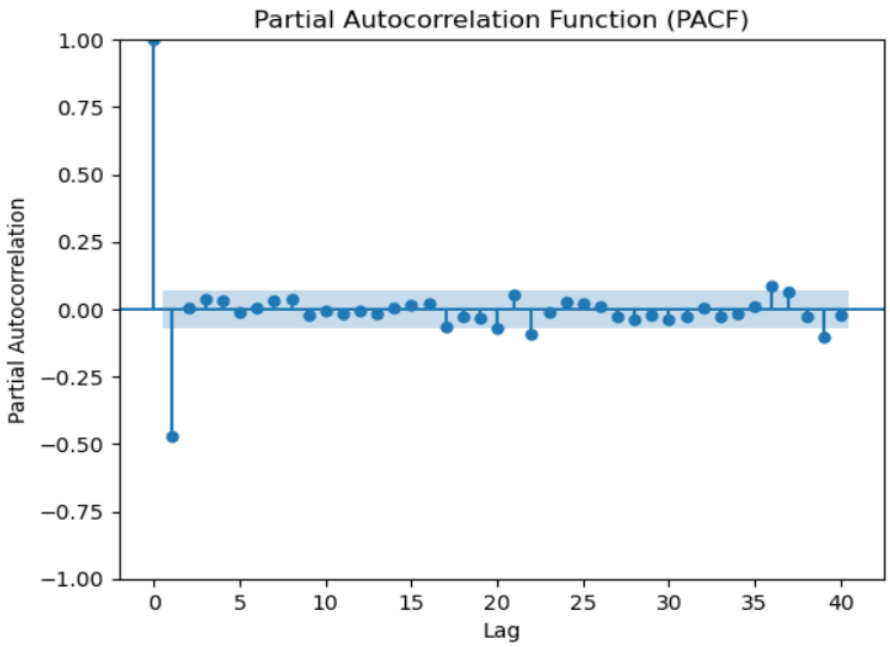
Confirmation of the lack of trends in the residuals of the decomposed series -

The residual component is centered around zero with no notable patterns or correlations indicating a good fit of the decomposition model. It suggests that the model has successfully removed the trend and seasonal components, leaving behind random fluctuations that cannot be explained by the identified patterns. This is desirable as it implies that the residual errors are random and do not contain any systematic information that could be used to improve the model's performance.

Auto Correlation Function (ACF and PACF):

I am using plot_acf and plot_pacf functions from the statsmodels.graphics.tsaplots module to plot the Autocorrelation Function (ACF) and Partial Autocorrelation Function (PACF).

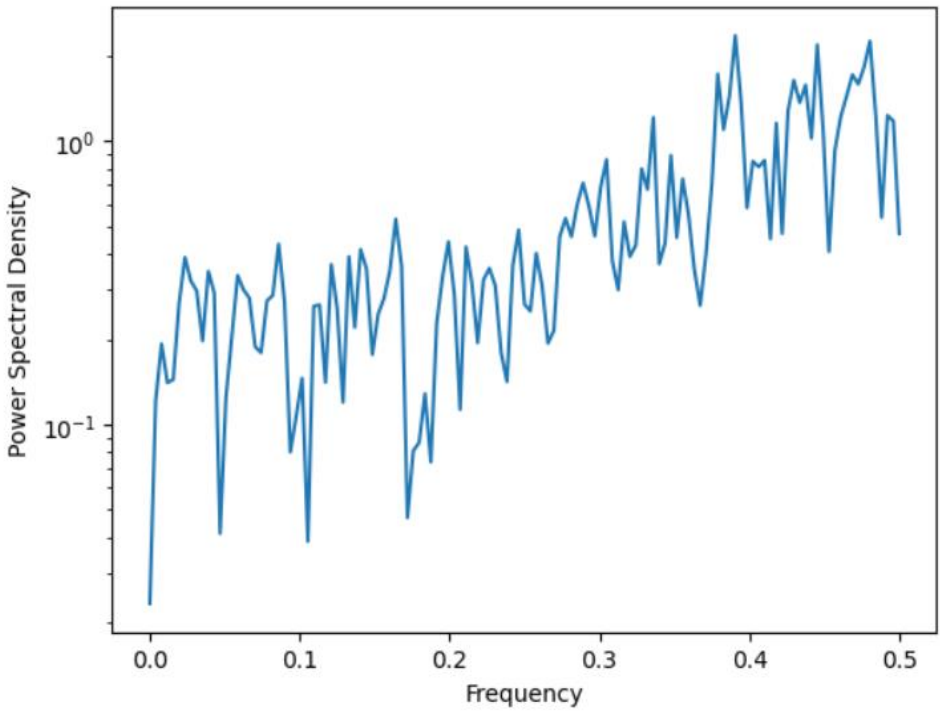




By analyzing the ACF and PACF plots, we gain insights into the presence of autocorrelation and determine the suitable lag order for autoregressive terms in the ARIMA model. The ACF plot helps identify significant lag values and provides insights into potential AR or MA components for an ARIMA model.

Spectral Density

The Power Spectral Density (PSD) plot visualizes the spectral characteristics of a time series. It helps identify dominant frequencies, periodic patterns, and overall patterns of variability.



As per the plot, the power spectrum density increases as frequency increases.

D2: ARIMA Model

The autocorrelation function (ACF) and partial autocorrelation function (PACF) of the differenced series helped to find the order of the autoregressive (AR) and moving average (MA) components of the ARIMA model.

I am also doing auto_arima which uses a stepwise algorithm to search for the best ARIMA model by considering different combinations of parameters. It automatically determines the optimal orders (p, d, q) based on evaluation metrics like AIC (Akaike Information Criterion) or BIC (Bayesian Information Criterion).

Here is the output from auto_arima which gives Best model: ARIMA(1,1,0)(0,0,0) [0]

```
# Auto ARIMA on training data
stepwise_fit = auto_arima(train_data, trace=True, suppress_warning=True)
stepwise_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,1,2)(0,0,0)[0] intercept   : AIC=871.585, Time=0.53 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1015.830, Time=0.07 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=867.657, Time=0.10 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=897.650, Time=0.13 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1014.845, Time=0.03 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=869.544, Time=0.11 sec
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=869.566, Time=0.17 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=870.440, Time=0.54 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=868.235, Time=0.05 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0] intercept
Total fit time: 1.741 seconds
```

This output indicates that the model selected has an autoregressive (AR) order of 1, a differencing (I) order of 1, and a moving average (MA) order of 0. The numbers within the parentheses specify the values for the corresponding orders. In this case, ARIMA(1,1,0) suggests that the current observation depends on the previous observation (AR(1)) and that differencing is applied once to the data (d=1) to make it stationary.

(0,0,0)[0] indicates that there is no seasonal component in the model. The numbers within the square brackets represent the seasonal orders, and in this case, they are set to 0, indicating no seasonal pattern. The term "intercept" indicates that the model includes an intercept term, which allows for a non-zero mean in the time series.

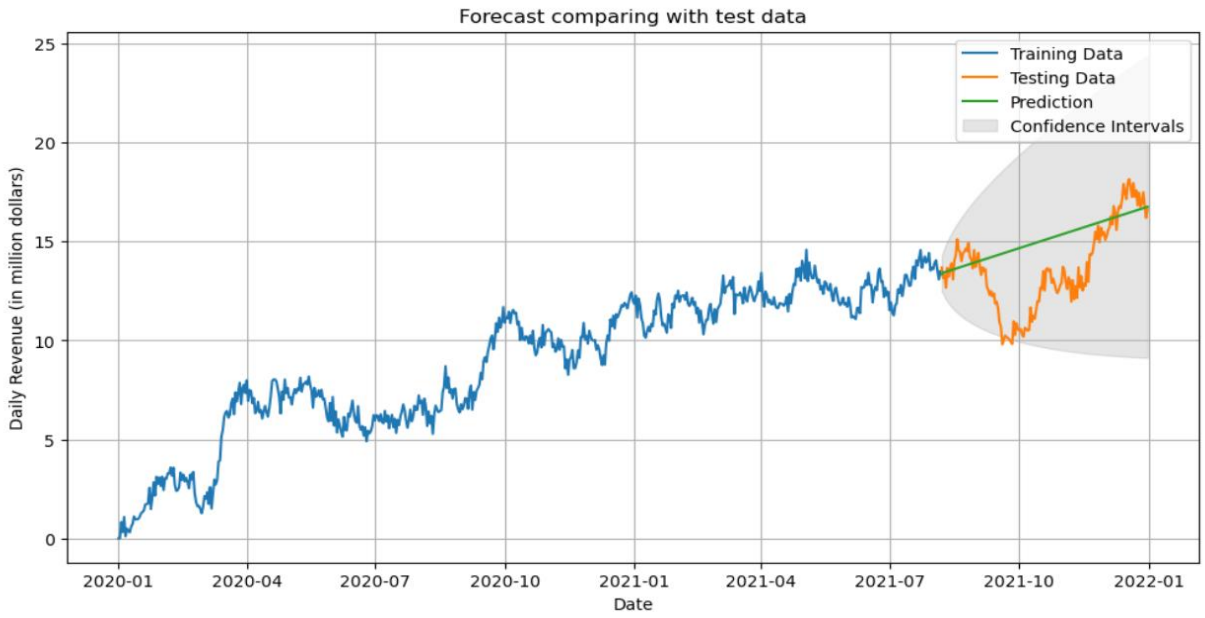
It's a good practice to consider other evaluation metrics as well so I am trying different values of p,d, and q for the ARIMA model manually and comparing the results to make sure which model provides the overall goodness-of-fit. I will be using the new version of arima – SARIMAX.

I tried Model 1 using p,d,q = (1,1,1), Model 2 using p,d,q = (1,1,0), Model 3 using p,d,q = (1,0,0), and Model 4 using p,d,q = (1,0,1) and compared the output and found that the best ARIMA model is Model 2 using p,d,q = (1,1,0). This is the same value I received from the auto arima model as well.

D3: Forecasting using the ARIMA model

Validate the model using test data –

Once the model is fitted I am using the get_forecast() function to generate forecasts for the test data. This provides predictions for the periods covered by the test data. Created below plot shows the forecasted values and the actual values from the test data.

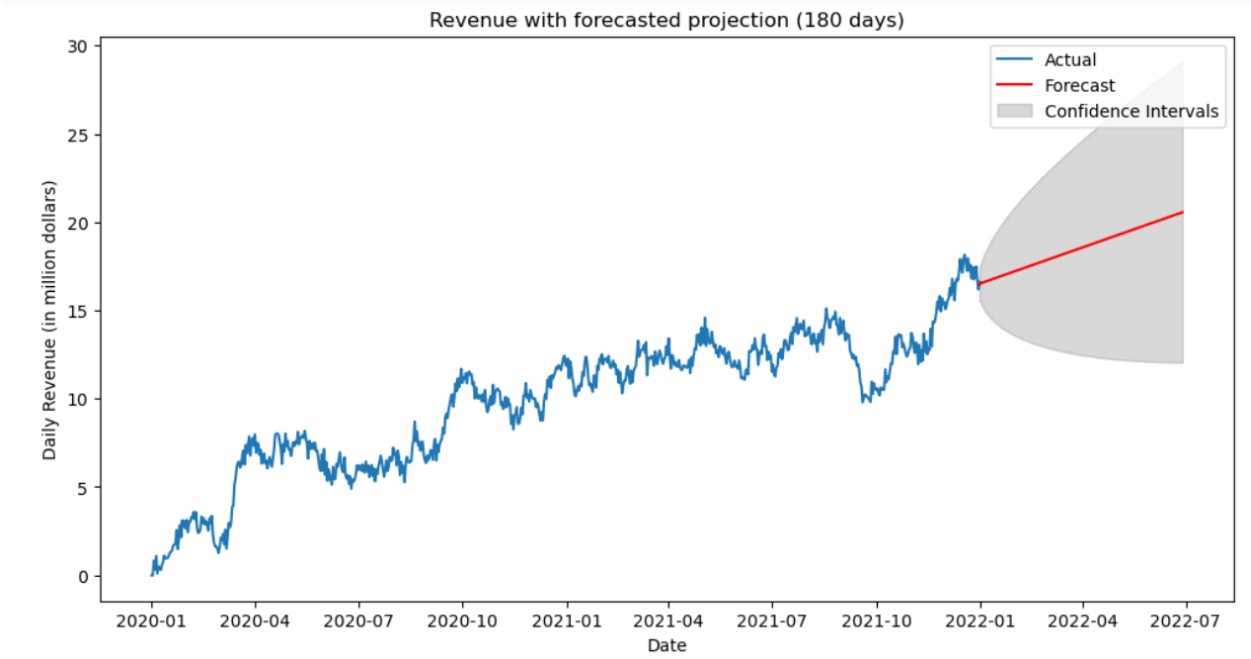


This model continues to show the upward trend for the forecasted value as well with low mean absolute error.

Forecasting for the next 180 days –

To create the forecasting for the future 180 days I am building the model again with the full dataset using the same p,d, and q values and then predicting the revenue for the next 180 days using the get_forecast() function.

Here is the below plot that shows the forecasted values for the next 180 days -



D4: Output and Calculations

Here is the output summary of my best ARIMA model –

```
# Create ARIMA model
#arima_model2 = SARIMAX(train_data, order=(1, 1, 0))  # Specify the order as (p, d, q)
model2 = SARIMAX(train_data, order=(1,1,0), trend = 'c')
# Fit the model
result2 = model2.fit()

# Print the model summary
print(result2.summary())
```

```
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  584
Model:               SARIMAX(1, 1, 0)   Log Likelihood                -383.523
Date:                Tue, 30 May 2023   AIC                            773.046
Time:                        16:12:56   BIC                            786.151
Sample:                  01-01-2020   HQIC                           778.154
                       - 08-06-2021
Covariance Type:                 opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0335      0.019      1.720      0.085      -0.005       0.072
ar.L1         -0.4605      0.036    -12.663      0.000      -0.532      -0.389
sigma2         0.2181      0.014     16.020      0.000       0.191       0.245
===================================================================================
Ljung-Box (L1) (Q):                   0.00   Jarque-Bera (JB):                 1.79
Prob(Q):                              0.96   Prob(JB):                         0.41
Heteroskedasticity (H):               0.97   Skew:                            -0.07
Prob(H) (two-sided):                  0.85   Kurtosis:                         2.77
===================================================================================

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

AIC (Akaike Information Criterion) and BIC (Bayesian Information Criterion) are used to assess the goodness of fit of the model. Lower values of AIC and BIC indicate better model fit. In this case, the AIC is 773.046 and BIC is 786.151, suggesting that the model has a relatively good fit.

The intercept coefficient represents the constant term in the ARIMA model. The p-value helps assess the significance of the intercept term. In this case, the p-value is greater than 0.05, indicating that the intercept term may not be statistically significant.

The AR coefficient represents the autoregressive term in the ARIMA model. In this case, the AR coefficient (ar.L1) is -0.4605 with a standard error of 0.036 and a p-value of 0.000. The negative coefficient suggests a negative relationship between the current observation and the previous observation.

The sigma2 coefficient represents the variance of the error term in the ARIMA model. In this case, the sigma2 coefficient is 0.2181 with a standard error of 0.014 and a p-value of 0.000. It provides an estimate of the variability or dispersion of the residuals.

Overall, the model seems to have a reasonable fit based on the AIC and BIC values. The Ljung-Box (L1) (Q) value is low, indicating no autocorrelation at lag 1. The JB value is also low, indicating a good fit to the normal distribution. The significant ar.L1 coefficient suggests that the previous observation at lag 1 has an impact on the current observation.

Model Evaluation –

I am using the mean absolute error (MAE) metric to measure the average absolute difference between the predicted values and the actual values. It provides a measure of how well the model's predictions match the actual data, regardless of the direction (positive or negative) of the errors.

In my case, the mean absolute error is reported as 0.38, which means that, on average, the model's predictions deviate from the actual values by approximately 0.38 units. The MAE is a scale-dependent metric, meaning that its value depends on the scale of the data.

```
# Caclulate the residuals
result2.resid
```

```
Date
2020-01-01   -0.022957
2020-01-02   -0.022164
2020-01-03    0.791585
2020-01-04   -0.158935
2020-01-05    0.496040
                 ...
2021-08-02    0.273374
2021-08-03    0.235404
2021-08-04   -0.513075
2021-08-05   -0.716219
2021-08-06    0.187087
Length: 584, dtype: float64
```

```
# Calculate the mean absolute error from residuals
mae = np.mean(np.abs(result2.resid))

# Print mean absolute error
print("The mean absolute error is :", round (mae,2))
```

```
The mean absolute error is : 0.38
```

D5: Code Execution

All the codes are provided in the .ipynb file provided with my submission.

Part V- Data Summary and Implications

E1: Results

Selection of ARIMA model: As discussed in the previous section D2, The selection of an ARIMA model involves determining the appropriate values for the p, d, and q parameters. I am using the auto ARIMA algorithm, which automatically determines the optimal values for the p, d, and q parameters based on the provided time series data.
The best ARIMA model selected the one with the lowest AIC, BIC, or error metric value. In my case, the best model selected by auto_arima is an ARIMA(1,1,0) model without a seasonal component. I am using this value to SARIMAX to get the prediction.

Prediction interval of the forecast: The prediction interval indicates the range within which the future values of the time series are expected to fall.

Here are my prediction intervals –

```
confidence_intervals
```

	lower Revenue	upper Revenue
2022-01-01	15.526249	17.378117
2022-01-02	15.516774	17.615111
2022-01-03	15.284093	17.806106
2022-01-04	15.191402	17.985436
2022-01-05	15.061115	18.141191
...	...	...
2022-06-25	12.051031	28.889365
2022-06-26	12.049876	28.935735
2022-06-27	12.048788	28.982038
2022-06-28	12.047765	29.028275
2022-06-29	12.046809	29.074446

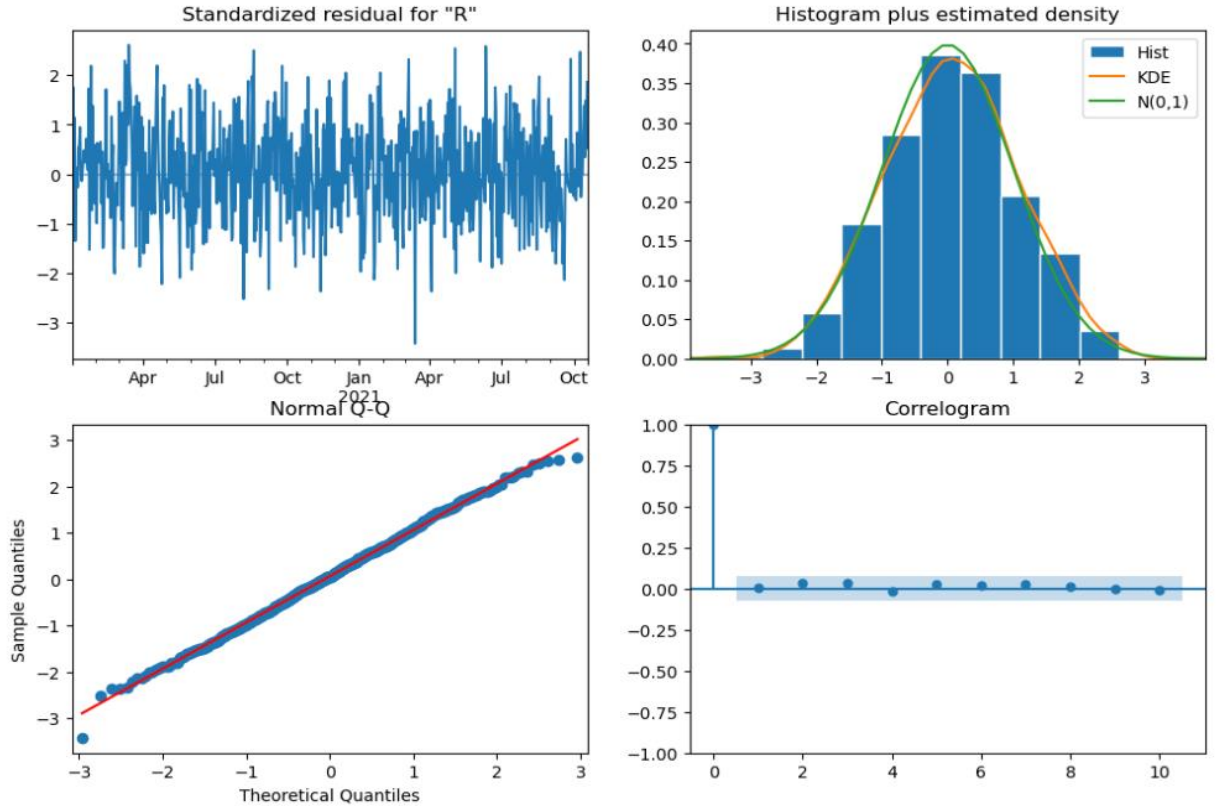
180 rows × 2 columns

Justification of the forecast length: The forecast length refers to the number of periods into the future for which the ARIMA model generates predictions. To answer the research, question my forecast length is 180 days. The forecast length of 180 days provides a reasonable balance between short-term and long-term forecasting. It allows the company to capture the immediate future revenue trends while also considering the potential seasonality and long-term patterns that may impact the telecom company's revenue.

Model evaluation procedure and error metric: Model evaluation involves assessing the performance of the ARIMA model in generating accurate forecasts.

I am using the plot_diagnostics() function in the statsmodels library which generates a set of diagnostic plots that can help assess the quality of the model and identify any potential issues.

```
## Generate the diagnostic plot
model2_fit.plot_diagnostics(figsize=(12,8))
plt.show()
```



The diagnostic plot includes four subplots:

- The standardized residuals show randomness and have a mean of zero with constant variance.
- The histogram and estimated density closely align, indicating the normality of the residuals.
- The points on the normal Q-Q plot fall approximately along a straight line until it reaches the 3rd quantile which indicates the normality of the residuals.
- The autocorrelation function (ACF) plot does not show significant autocorrelation beyond the confidence interval.

I am calculating mean absolute error and RMSE from residuals to evaluate the model performance. The MAE will provide the overall accuracy of a model's predictions. A lower MAE indicates that the model's predictions are, on average, closer to the actual values.

```
# Calculate the mean absolute error from residuals
mae = np.mean(np.abs(result2.resid))

# Print mean absolute error
print("The mean absolute error is :", round (mae,2))
```

```
The mean absolute error is : 0.38
```

The mean absolute error for my model is 0.38, which means that, on average, the model's predictions deviate from the actual values by approximately 0.38 units.

The RMSE value represents the average magnitude of the residuals, indicating the typical difference between the observed values and the predicted values of the model. A lower RMSE indicates better model performance, as it suggests smaller prediction errors.

```
# Calculate the squared residuals
squared_residuals = residuals**2

# Calculate the mean of squared residuals
mean_squared_residuals = np.mean(squared_residuals)

# Calculate the RMSE
rmse = np.sqrt(mean_squared_residuals)
```

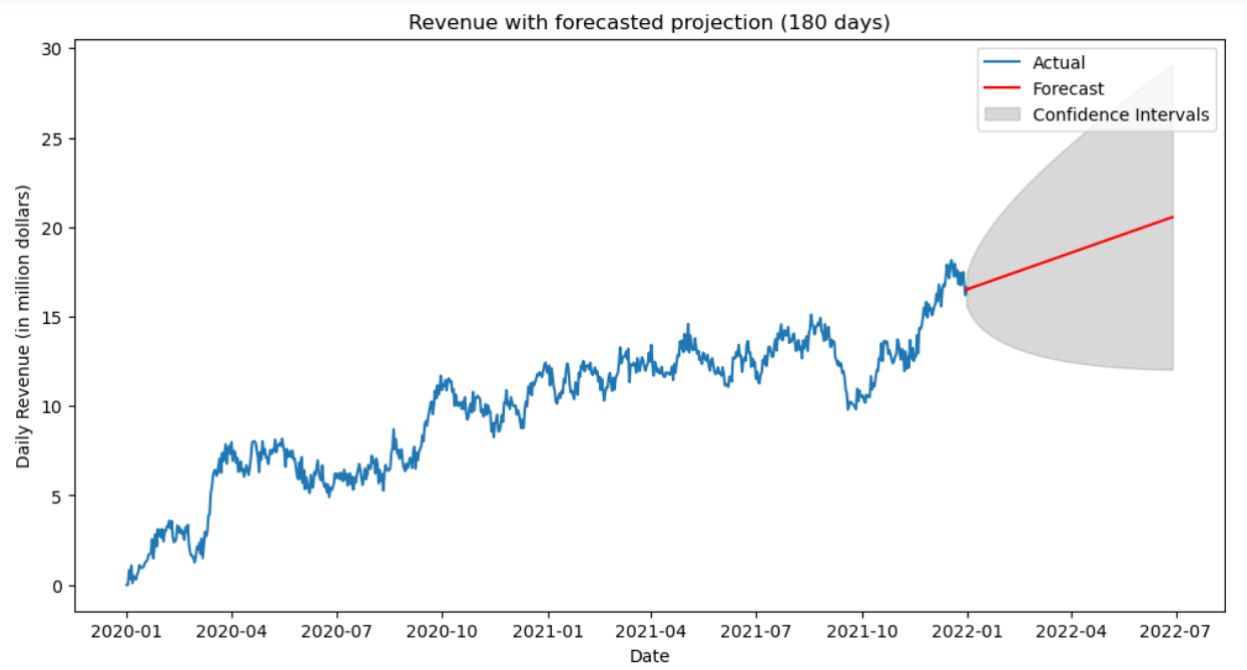
```
rmse
```

```
0.46666055157908537
```

My ARIMA model RMSE score of 0.466 suggests that, on average, the predictions for Revenue of the model deviate from the actual Revenue values by 0.466 units.

E2: Annotated Visualization

Here is the final annotated visualization for the next 180 days showing an upward trend based on all the previous 2 years of data –



E3: Recommendation

All the results from the diagnostic plot suggest that the ARIMA model is performing well in capturing the patterns and variability in the data. The randomness, normality, lack of significant autocorrelation, and constant variance of the residuals indicate that the model is providing a good fit to the data.

The telecom company can utilize the forecasted revenue to inform business decisions and strategies. The accurate forecast can provide insights into revenue projections, budgeting, resource allocation, and planning for the telecom company. Consider incorporating the forecasted revenue into decision-making processes to optimize operational efficiency and financial performance.

I would also recommend continuously monitoring the actual revenue data as it becomes available for the forecasted period. Compare the actual values with the forecasted values to assess the accuracy of the model's predictions. If necessary, re-evaluate the model and consider adjusting the parameters or exploring alternative modeling approaches to improve forecasting accuracy.

Part VI – Reporting

F. Report

The final report in an HTML document is provided with the submission.

G. Third-Party Code References

WGU, R. (n.d.). Advanced Data Analytics - Task 1. Retrieved from https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=efceba6c-e8ef-47a2-b859-aec400fe18e7

WGU, R. (n.d.). Time Series Analysis. Retrieved from https://wgu.hosted.panopto.com/Panopto/Pages/Viewer.aspx?id=551c3203-d37e-4adb-8639-aed60102ce05

Naik, K. (2020, March 19). Forecasting Future Sales Using ARIMA and SARIMAX. https://www.youtube.com/watch?v=2XGSIlgUBDI

H. References

Hirantha, D. (2021, June 5). Time Series Analysis Using ARIMA Model With Python. https://medium.com/geekculture/time-series-analysis-using-arima-model-with-python-afe4b41bbec8

Pandian, S. (2021, October 23). Time Series Analysis and Forecasting | Data-Driven Insights. https://www.analyticsvidhya.com/blog/2021/10/a-comprehensive-guide-to-time-series-analysis/

Parikh, A. B. (n.d.). Machine Learning & Deep Learning in Python & R. https://wgu.udemy.com/course/data_science_a_to_z/learn/lecture/20811912#overview

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