WGU D213 PA 1

Andrew Shrestha

Part I: Research Question

A1. Question: Utilizing the Teleco Time Series Data Set that we have, are we able to project the revenue for the following year based off previous revenue trends? This will be accomplished by using the time series analysis technique.

A2. Objectives and Goals: The goal of this research analysis is to add a greater insight into stakeholder decision making with the aim in increasing customer loyalty as well as revenue growth. With our data model created through this analysis, we will be able to identify significant patterns that give us a good sense of what will influence the projected revenue in the future. From these results, we can confidently make recommendations that can directly influence customer churn as well as revenue.

Part II: Method Justification

B1. Summary of the Assumptions for Time Series Model

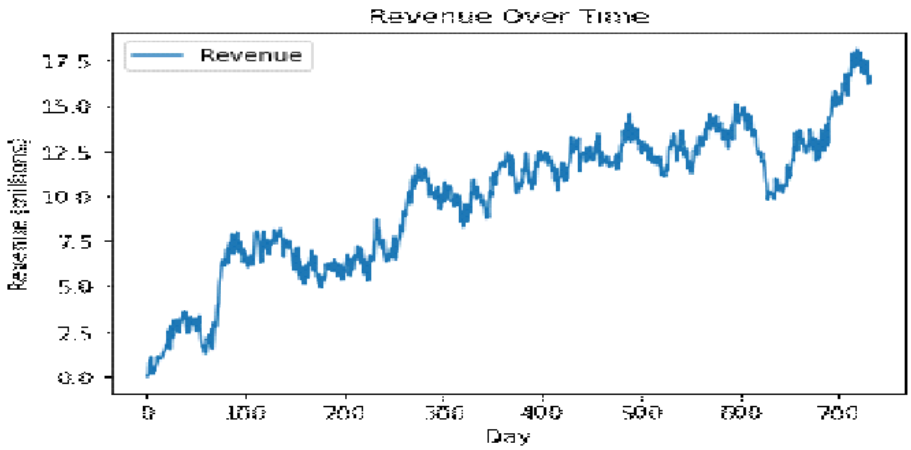
A time series is essentially a series of data points that are ordered via time. In this kind of model, time is usually the independent variable with the goal of making a forecast/prediction for the future. Below are 2 important assumptions for the Time Series Model (**Songhao Wu, 2021).**

I. Stationary: Stationarity is a significant assumption as the data and its statistically properties should not change over time. This will translate into data having a constant mean and variance, as well as covariance being independent of time. In the case that not all the data points display stationarity, we are able to use different transformations to make them have this property.
II. Autocorrelation: is a significant assumption of data points in a time series are linearly related to a lagged version of itself. This assumption will help to identify hidden patterns in our data as well as obtaining correct forecasting **(Anais Dotis , 2019).**

III. Part III: Data Preparation

C1. Line Graph Visualization of Time Series

```
# Import standard libraries needed for cleaning and analysis
import pandas as pd
import numpy as np
%matplotlib inline
import matplotlib.pyplot as plt

#Visualization libraries
import seaborn as sns

#Statistics
from statsmodels.tsa.stattools import adfuller
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf
from statsmodels.tsa.statespace.sarimax import SARIMAX
from scipy import signal
import statsmodels.api as sm
from pylab import rcParams
import warnings
warnings.filterwarnings('ignore') # Ignore warning messages for readability


# Read the Teleco Time Series Data Set
#Need to parse dates and change index col to accurately get line graph
visualizations
df = pd.read_csv(r"C:\Users\andre\OneDrive\Desktop\teleco_time_series.csv",
index_col = 'Day', parse_dates = True)


df.head()


# Visualize Time Series
df.plot()
plt.title("Revenue Over Time")
plt.ylabel("Revenue (millions)")
plt.show();
```



C2. Time Step Formatting

```
Teleco_df =
pd.read_csv(r"C:\Users\andre\OneDrive\Desktop\teleco_time_series.csv")


Teleco_df.info


Teleco_df.dtypes


Day          int64
Revenue    float64
dtype: object


#Checking for missing values
print(Teleco_df.isnull().values.any())


False


#Checking for NA values
print(Teleco_df.isna().values.any())


False


#Checking for Duplicate values
print(Teleco_df.Day.duplicated().sum())


0


# Checking for gaps in day count, None as well since Count = Max for length
of sequence both Day and Revenue
print(Teleco_df.describe())


                Day     Revenue
count   731.000000  731.000000
mean    366.000000    9.822901
std     211.165812    3.852645
min       1.000000    0.000000
25%     183.500000    6.872836
50%     366.000000   10.785571
75%     548.500000   12.566911
max     731.000000   18.154769
```

C3. Stationarity of Time Series Evaluation

```
# Evaluate via Dicky-Fuller Test

#
result = adfuller(df['Revenue'])

# Print test statistic
print("The t-statistic is:", round(result[0],2))

# Print p-value
print("The p-value is:", round(result[1],2))

# Print critical values
crit_vals = result[4]
print("The critical value of the t-statistic for a 95% confidence level is:",
round(crit_vals['5%'],2))
```

```
The t-statistic is: -1.92
The p-value is: 0.32
The critical value of the t-statistic for a 95% confidence level is: -2.87
```

We are using the Augmented Dicky-Fuller test because it is a fundamentally statistical significance test where there is a null and alternative hypothesis. Through computing the p-values and test statistic, we can determine as to whether a given series is stationary or not **(Selva Prabhakaran, 2019).**

The null hypothesis assumes $\alpha=1$, thus the p-value should be less than the significance level of 0.05 for us to reject the null.

The results above indicate that the t-statistic = -1.92 and p-value of 0.32. Since our p-value is higher than the significance level of 0.05, we do not reject the null hypothesis, and conclude that the time series is non-stationary.

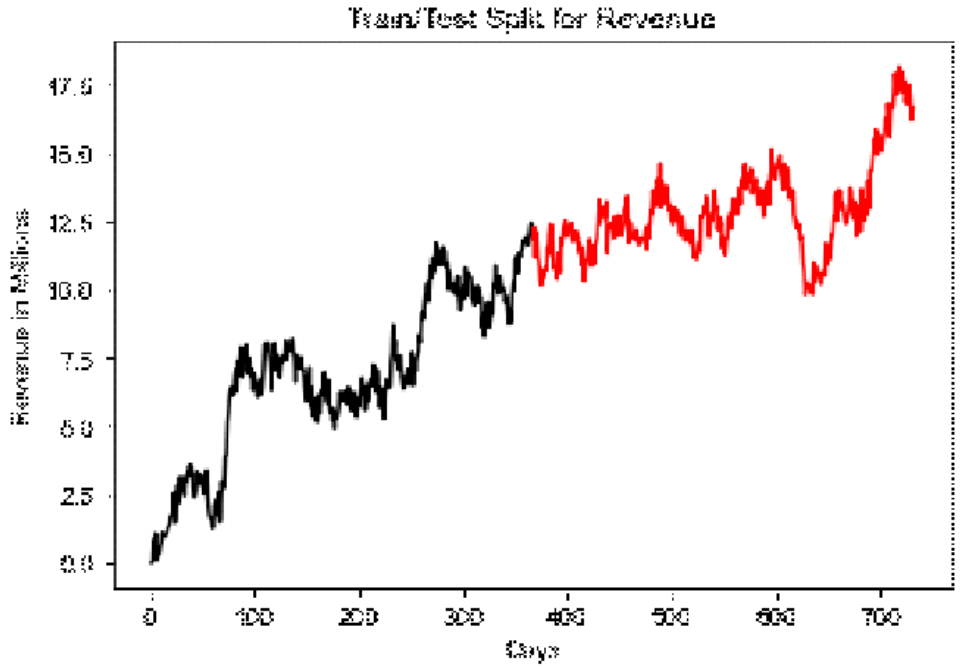
With this result of non-stationary, we will need to transform the data in order to continue with the ARIMA model.

C4. Steps Used to Prepare Data for Analysis

The two main preparation steps that will be utilized will be as follows:

i. Checking for null/missing values, gaps in measurements, or duplicates in the data
ii. Creating dummy date/time variables for our ARIMA analysis
iii. Creating split data sets where 80% will be training and 20% will be testing

```
# We first begin by splitting dataframe in exactly half
df['Day'] = df.index
teleco_train = df.iloc[:len(df) - 365]

teleco_train['teleco_train'] = teleco_train['Revenue']
del teleco_train['Day']
del teleco_train['Revenue']



# Now we want to create a test set
teleco_test = df.iloc[len(df) - 365:]

teleco_test['teleco_test'] = teleco_test['Revenue']
del teleco_test['Day']
del teleco_test['Revenue']



# We can visualize the training and test split that we created by splitting
the data in half
plt.plot(teleco_train, color='black')
plt.plot(teleco_test, color='red')
plt.title('Train/Test Split for Revenue')
plt.xlabel('Days')
plt.ylabel('Revenue in Millions')
sns.set()
plt.show()
```



```
#We will then split the data into both training and testing sets
df1 = df.values.flatten()

# Create dummy dates for the our Arima modules
dates = pd.date_range('1900-1-1', periods=len(df1), freq='D')

# Add the dates/data to our new dataframe
ts = pd.DataFrame({'dates': dates, 'Revenue': df1})

# Set our index to be the dates column
df_ts = ts.set_index('dates')
df_ts.head()


# Determine what the cut off would be for an 80% training/20% testing split
cutoff = round(len(df_ts)* 0.8)
cutoff_date = df_ts.iloc[[585]].index.values
Y, M, D, h, m, s = [cutoff_date.astype(f"M8[{x}]") for x in "YMDhms"]

print("80% of the data includes", cutoff, "records.")
print ("The date for at index", cutoff, "is:", D)


80% of the data includes 1170 records.
The date for at index 1170 is: ['1901-08-09']


# We now split the data into 80% training and 20% testing
df_train = df_ts.iloc[:cutoff + 1]
df_test = df_ts.iloc[cutoff + 1:]
```

Part IV: Model Identification & Analysis

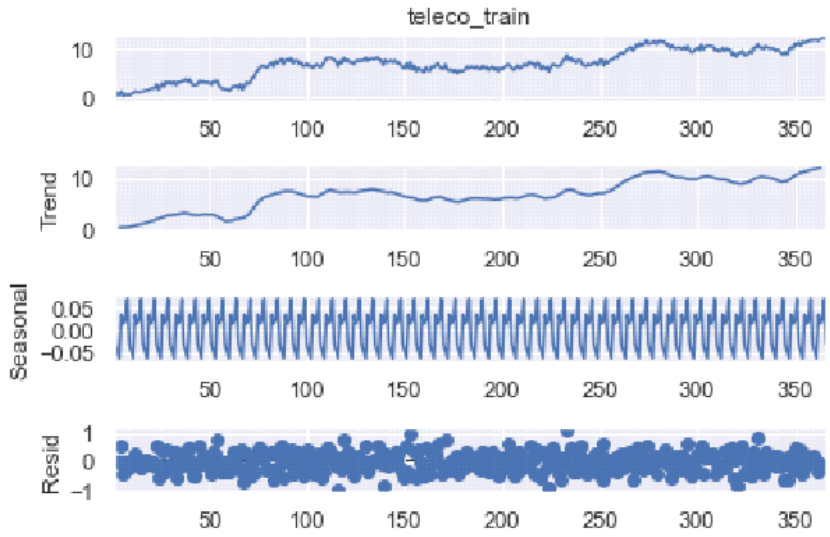
D1. Annotated Findings with Visualizations

I. **Seasonality**

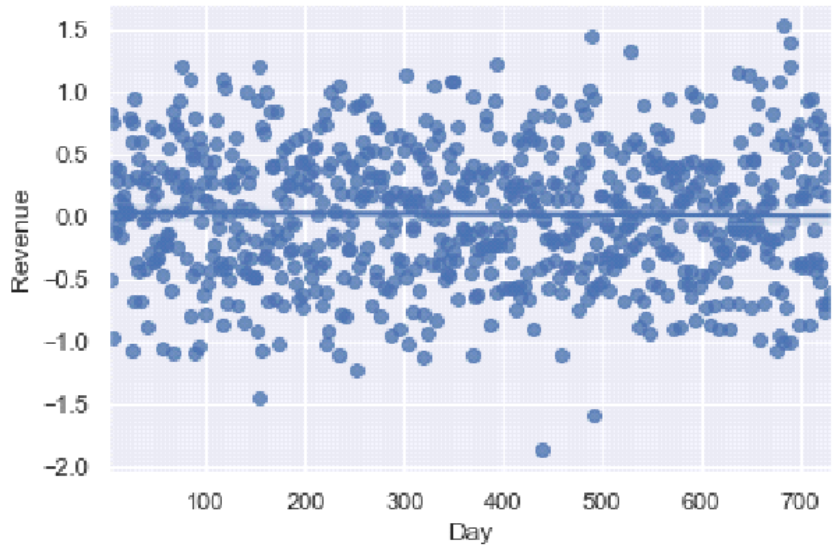
This is a particular characteristic of a Time Series where the data observed has regular and predictable changes that reoccur each calendar year. Any of the predictable fluctuations that occur over at minimum one year period is said to have the presence of seasonality **(Will Kenton, 2020).**

We can see below in our decomposition that there is a clear repeating and predicable fluctuation that occurs in our data.

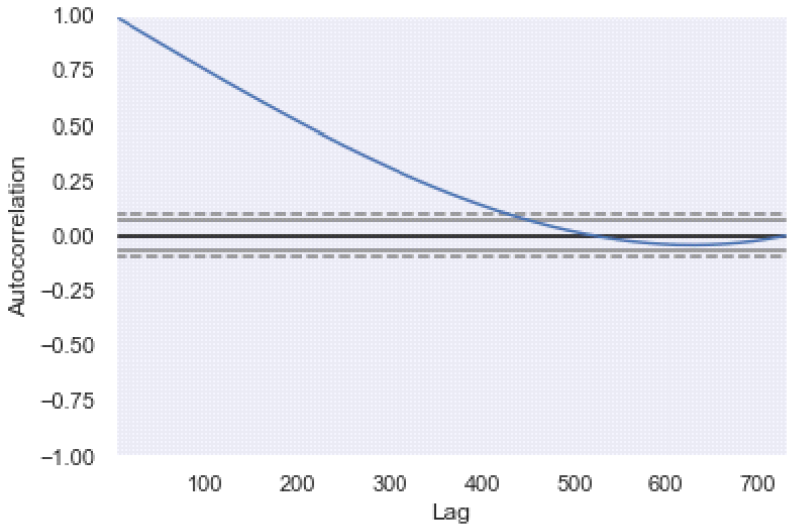
```
# Decompose our Training Set
from statsmodels.tsa.seasonal import seasonal_decompose
decompose = seasonal_decompose(teleco_train['teleco_train'],
model='additive', period=7)
decompose.plot()
plt.show()
```




```
#Check for Data Trends
sns.regplot(x=df_diff.index,y='Revenue',data=df_diff, fit_reg=True);
```



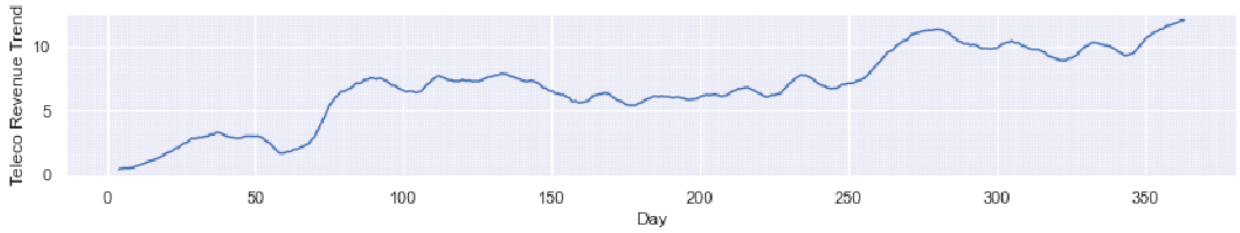
```
# Seasonality with autocorrelation plot
pd.plotting.autocorrelation_plot(df);
```



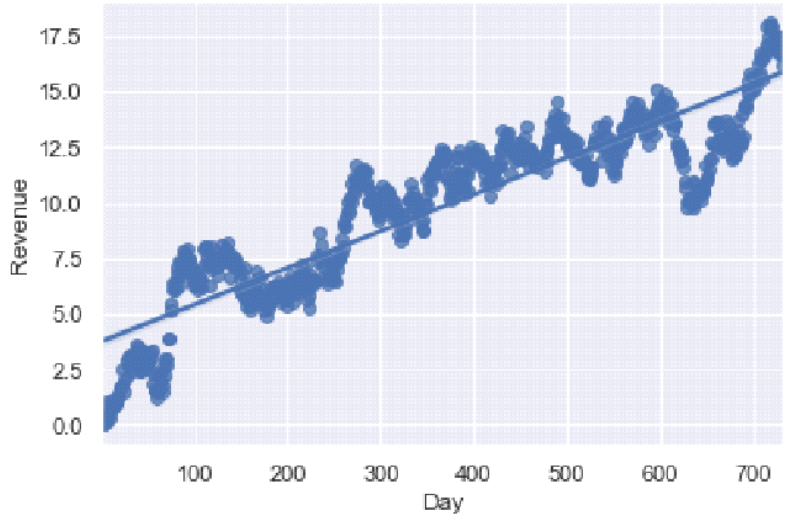
II. Trends

Trends are simply the patterns that are found in a Time Series whereby used to describe an upward or downward movement in the data set for either a part or the whole data set.

```
# Decompose Trend
decompose_trend = decompose.trend

ax = decompose_trend.plot(figsize=(14,2))
ax.set_xlabel('Day')
ax.set_ylabel('Teleco Revenue Trend')
```




```
#Check for Data Trends
sns.regplot(x=df.index,y='Revenue',data=df, fit_reg=True);
```



III. Autocorrelation

The Autocorrelation refers to the degree of similarity between a variable's present value, with that of any past values that we can obtain access to. This can also be described as the similarity between a given time series or a lagged version of itself **(Anais Dotis, 2019).**

As we see below in our results, we tend to have high correlation numbers with regards to one month, two months, and six month intervals.

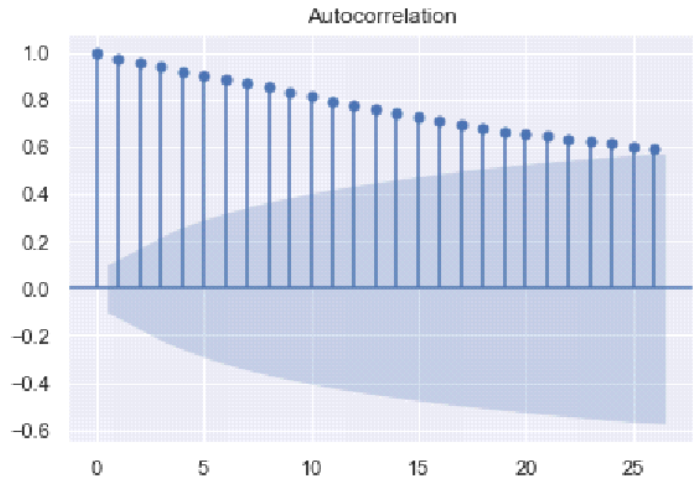
```
# Autocorrelation coefficient with a lag of one month
autocorrelation_lag1 = Teleco_df['Revenue'].autocorr(lag=31)
print("One Month (31 days) Lag: ", autocorrelation_lag1)
```

```
One Month (31 days) Lag:  0.8690526347331858
```

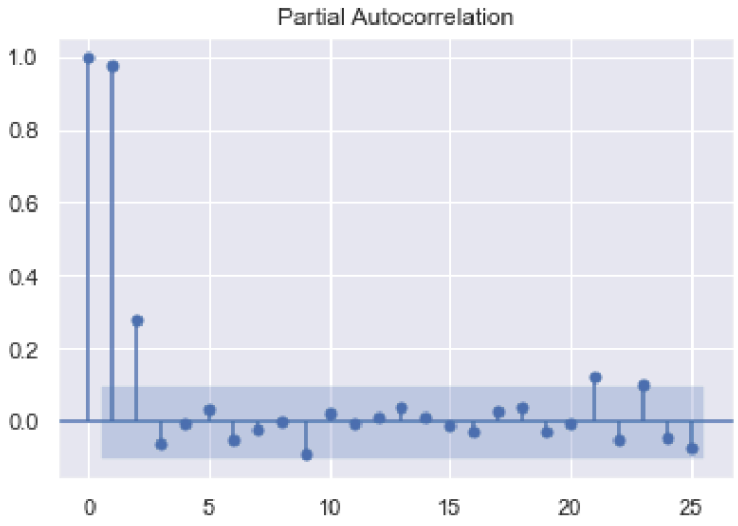
```
# Autocorrelation coefficient with two and six months lag
autocorrelation_lag3 = Teleco_df['Revenue'].autocorr(lag=62)
print("Two Months Lag: ", autocorrelation_lag3)

autocorrelation_lag6 = Teleco_df['Revenue'].autocorr(lag=182)
print("Six Months Lag: ", autocorrelation_lag6)
```

```
Two Months Lag:  0.7758400879703405
Six Months Lag:  0.8139870392893127
```

```
# Plot the function with the training set
from statsmodels.graphics.tsaplots import plot_acf
plot_acf(teleco_train)
plt.show()
```



```
# Plot partial autocorrelation for the lag = 25
plot_pacf(teleco_train, lags=25)
plt.show()
```



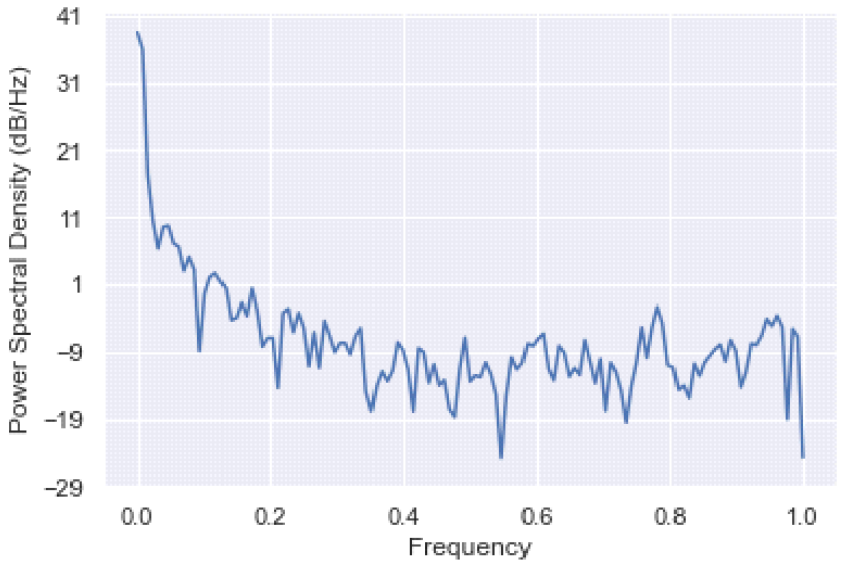
IV. Spectral Density

The spectral density is regarded as a frequency domain representation of a time series that is directly related to the autocovariance time domain representation **(Eberly College of Science, 2022).**

```
# Spectral density function
plt.psd(Teleco_df['Revenue'])
```

```
(array([6.97387711e+03, 3.91439441e+03, 5.47611144e+01, 1.10791953e+01,
        4.25439684e+00, 8.90767103e+00, 9.32436541e+00, 5.07699490e+00,
        4.50820186e+00, 1.97142105e+00, 3.17930644e+00, 2.05465010e+00,
        1.27856111e-01, 9.12357670e-01, 1.58928494e+00, 1.82600308e+00,
        1.37345831e+00, 1.08602805e+00, 3.71664573e-01, 4.01447712e-01,
        6.67019362e-01, 4.28134847e-01, 1.08303874e+00, 5.05461139e-01,
        1.49612021e-01, 2.01156666e-01, 2.01929017e-01, 3.62482513e-02,
        4.74386760e-01, 5.46619399e-01, 2.44460915e-01, 4.75225583e-01,
        2.85007556e-01, 7.59631735e-02, 2.43183009e-01, 7.19503505e-02,
        3.60475608e-01, 2.18971480e-01, 1.23421972e-01, 1.69234109e-01,
        1.69371921e-01, 1.14748035e-01, 2.14144408e-01, 2.82736486e-01,
        3.17059554e-02, 1.62458482e-02, 3.98324693e-02, 6.38895068e-02,
        4.57061837e-02, 6.54718394e-02, 1.72598760e-01, 1.32802101e-01,
        6.74716970e-02, 1.61089989e-02, 1.42266347e-01, 1.23640967e-01,
        4.30737122e-02, 8.17619769e-02, 3.95102804e-02, 4.74833201e-02,
        1.73085344e-02, 1.32485092e-02, 7.14595244e-02, 2.04348554e-01,
        4.51051907e-02, 5.55609932e-02, 5.26341931e-02, 8.68327071e-02,
        5.86340275e-02, 2.88124318e-02, 3.23398046e-03, 2.85277556e-02,
        1.03859619e-01, 6.85761289e-02, 8.53467489e-02, 1.65815053e-01,
        1.53444767e-01, 1.97776090e-01, 2.33413022e-01, 6.95161339e-02,
        4.68937467e-02, 1.53353843e-01, 1.21391009e-01, 5.34892233e-02,
        7.17250734e-02, 5.68272472e-02, 1.87708648e-01, 8.92439810e-02,
        4.25453689e-02, 9.79721726e-02, 1.63975102e-02, 8.61687997e-02,
        6.31239429e-02, 3.34886298e-02, 1.09875888e-02, 3.83717041e-02,
        8.81504336e-02, 2.89832055e-01, 1.01621880e-01, 2.88982189e-01,
        5.81784635e-01, 3.29483923e-01, 8.00152896e-02, 7.31104074e-02,
        3.39216170e-02, 3.93543037e-02, 2.55189466e-02, 8.36160222e-02,
        5.48857747e-02, 8.78315482e-02, 1.10183077e-01, 1.37188728e-01,
        1.60562556e-01, 9.01524835e-02, 1.87554502e-01, 1.29971998e-01,
        3.71787543e-02, 6.31650566e-02, 1.65356465e-01, 1.58164919e-01,
        2.17681939e-01, 3.85248684e-01, 3.04221135e-01, 4.32125608e-01,
        2.94750935e-01, 1.22618052e-02, 2.74845469e-01, 2.10555482e-01,
        3.28604031e-03]),
 array([0.       , 0.0078125, 0.015625 , 0.0234375, 0.03125  , 0.0390625,
        0.046875 , 0.0546875, 0.0625   , 0.0703125, 0.078125 , 0.0859375,
        0.09375  , 0.1015625, 0.109375 , 0.1171875, 0.125    , 0.1328125,
        0.140625 , 0.1484375, 0.15625  , 0.1640625, 0.171875 , 0.1796875,
        0.1875   , 0.1953125, 0.203125 , 0.2109375, 0.21875  , 0.2265625,
        0.234375 , 0.2421875, 0.25     , 0.2578125, 0.265625 , 0.2734375,
        0.28125  , 0.2890625, 0.296875 , 0.3046875, 0.3125   , 0.3203125,
        0.328125 , 0.3359375, 0.34375  , 0.3515625, 0.359375 , 0.3671875,
        0.375    , 0.3828125, 0.390625 , 0.3984375, 0.40625  , 0.4140625,
        0.421875 , 0.4296875, 0.4375   , 0.4453125, 0.453125 , 0.4609375,
        0.46875  , 0.4765625, 0.484375 , 0.4921875, 0.5      , 0.5078125,
        0.515625 , 0.5234375, 0.53125  , 0.5390625, 0.546875 , 0.5546875,
        0.5625   , 0.5703125, 0.578125 , 0.5859375, 0.59375  , 0.6015625,
        0.609375 , 0.6171875, 0.625    , 0.6328125, 0.640625 , 0.6484375,
        0.65625  , 0.6640625, 0.671875 , 0.6796875, 0.6875   , 0.6953125,
        0.703125 , 0.7109375, 0.71875  , 0.7265625, 0.734375 , 0.7421875,
        0.75     , 0.7578125, 0.765625 , 0.7734375, 0.78125  , 0.7890625,
```

```
0.796875 , 0.8046875, 0.8125   , 0.8203125, 0.828125 , 0.8359375,
0.84375  , 0.8515625, 0.859375 , 0.8671875, 0.875    , 0.8828125,
0.890625 , 0.8984375, 0.90625  , 0.9140625, 0.921875 , 0.9296875,
0.9375   , 0.9453125, 0.953125 , 0.9609375, 0.96875  , 0.9765625,
0.984375 , 0.9921875, 1.       ]))
```



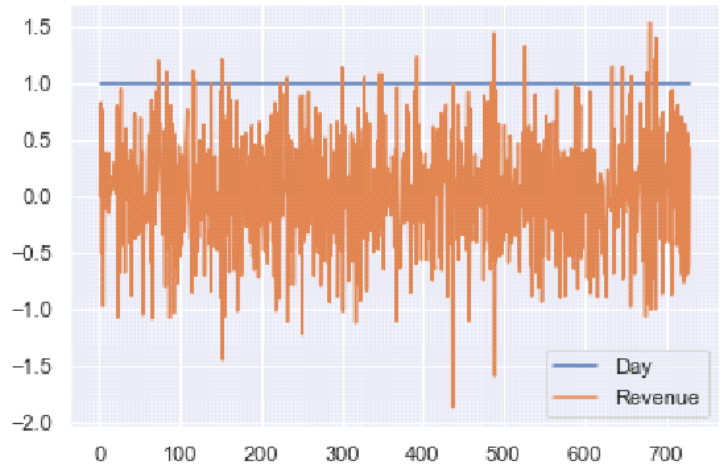
V. Residuals

The Residuals are the differences between the observed and predicted data. From our visual we obtained via the decomposed data above, we see that there is no trend from the residuals.

D2. ARIMA Model

Due to us finding out earlier that the Time Series was non-stationary, we must under go the process of converting the data into stationary before proceeding with the ARIMA Model.

```
# Difference the data to convert it to stationary data
first_diff = Teleco_df.diff().dropna()


# Run ADF test
result = adfuller(first_diff['Revenue'])


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




```
# Print our test statistic and p-value obtained
print('test statistic:', result[0])
print('p-value:', result[1])


test statistic: -44.874527193876
p-value: 0.0


# Import auto_arima class
!pip install pmdarima
```

```
  Downloading pmdarima-1.8.5-cp38-cp38-win_amd64.whl (602 kB)
Requirement already satisfied: setuptools!=50.0.0,>=38.6.0 in c:\users\and
re\anaconda3\lib\site-packages (from pmdarima) (50.3.1.post20201107)
Requirement already satisfied: scipy>=1.3.2 in c:\users\andre\anaconda3\li
b\site-packages (from pmdarima) (1.5.2)
Requirement already satisfied: joblib>=0.11 in c:\users\andre\anaconda3\li
b\site-packages (from pmdarima) (0.17.0)
Requirement already satisfied: scikit-learn>=0.22 in c:\users\andre\anacon
da3\lib\site-packages (from pmdarima) (0.23.2)
Collecting numpy>=1.19.3
  Downloading numpy-1.22.4-cp38-cp38-win_amd64.whl (14.8 MB)
Requirement already satisfied: urllib3 in c:\users\andre\anaconda3\lib\sit
e-packages (from pmdarima) (1.25.11)
Collecting statsmodels!=0.12.0,>=0.11
  Downloading statsmodels-0.13.2-cp38-cp38-win_amd64.whl (9.1 MB)
Requirement already satisfied: pandas>=0.19 in c:\users\andre\anaconda3\li
b\site-packages (from pmdarima) (1.1.3)
Requirement already satisfied: Cython!=0.29.18,>=0.29 in c:\users\andre\an
aconda3\lib\site-packages (from pmdarima) (0.29.21)
Requirement already satisfied: threadpoolctl>=2.0.0 in c:\users\andre\anac
onda3\lib\site-packages (from scikit-learn>=0.22->pmdarima) (2.1.0)
Collecting patsy>=0.5.2
  Downloading patsy-0.5.2-py2.py3-none-any.whl (233 kB)
Collecting packaging>=21.3
  Downloading packaging-21.3-py3-none-any.whl (40 kB)
Requirement already satisfied: pytz>=2017.2 in c:\users\andre\anaconda3\li
b\site-packages (from pandas>=0.19->pmdarima) (2020.1)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\andre\an
aconda3\lib\site-packages (from pandas>=0.19->pmdarima) (2.8.1)
Requirement already satisfied: six in c:\users\andre\anaconda3\lib\site-pa
ckages (from patsy>=0.5.2->statsmodels!=0.12.0,>=0.11->pmdarima) (1.15.0)
Requirement already satisfied: pyparsing!=3.0.5,>=2.0.2 in c:\users\andre\
anaconda3\lib\site-packages (from packaging>=21.3->statsmodels!=0.12.0,>=0
.11->pmdarima) (2.4.7)
Installing collected packages: numpy, patsy, packaging, statsmodels, pmdar
ima
  Attempting uninstall: numpy
    Found existing installation: numpy 1.19.2
    Uninstalling numpy-1.19.2:
      Successfully uninstalled numpy-1.19.2
  Attempting uninstall: patsy
    Found existing installation: patsy 0.5.1
    Uninstalling patsy-0.5.1:
      Successfully uninstalled patsy-0.5.1
  Attempting uninstall: packaging
    Found existing installation: packaging 20.4
    Uninstalling packaging-20.4:
      Successfully uninstalled packaging-20.4
  Attempting uninstall: statsmodels
    Found existing installation: statsmodels 0.12.0
    Uninstalling statsmodels-0.12.0:
      Successfully uninstalled statsmodels-0.12.0
```

```
from pmdarima import auto_arima


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

auto_arima_fit.summary()
```

```
Performing stepwise search to minimize aic
 ARIMA(2,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=6.40 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=2367.159, Time=0.07 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=1419.537, Time=1.43 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=1969.738, Time=1.12 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=2399.547, Time=0.07 sec
 ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=1568.311, Time=0.33 sec
 ARIMA(1,0,0)(2,1,0)[12] intercept   : AIC=1320.755, Time=3.38 sec
 ARIMA(1,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=9.26 sec
 ARIMA(1,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=4.06 sec
 ARIMA(0,0,0)(2,1,0)[12] intercept   : AIC=2339.965, Time=1.34 sec
 ARIMA(2,0,0)(2,1,0)[12] intercept   : AIC=1147.041, Time=5.99 sec
 ARIMA(2,0,0)(1,1,0)[12] intercept   : AIC=1256.245, Time=2.29 sec
 ARIMA(2,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=8.37 sec
 ARIMA(2,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=4.74 sec
 ARIMA(3,0,0)(2,1,0)[12] intercept   : AIC=1148.348, Time=6.64 sec
 ARIMA(2,0,1)(2,1,0)[12] intercept   : AIC=1148.544, Time=6.39 sec
 ARIMA(1,0,1)(2,1,0)[12] intercept   : AIC=1195.703, Time=5.24 sec
 ARIMA(3,0,1)(2,1,0)[12] intercept   : AIC=1132.917, Time=16.20 sec
 ARIMA(3,0,1)(1,1,0)[12] intercept   : AIC=1235.930, Time=8.73 sec
 ARIMA(3,0,1)(2,1,1)[12] intercept   : AIC=inf, Time=17.33 sec
 ARIMA(3,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=10.14 sec
 ARIMA(3,0,2)(2,1,0)[12] intercept   : AIC=1132.712, Time=16.79 sec
 ARIMA(3,0,2)(1,1,0)[12] intercept   : AIC=1236.061, Time=9.34 sec
 ARIMA(3,0,2)(2,1,1)[12] intercept   : AIC=inf, Time=19.43 sec
 ARIMA(3,0,2)(1,1,1)[12] intercept   : AIC=inf, Time=11.58 sec
 ARIMA(2,0,2)(2,1,0)[12] intercept   : AIC=1142.858, Time=8.70 sec
 ARIMA(3,0,3)(2,1,0)[12] intercept   : AIC=1136.935, Time=24.86 sec
 ARIMA(2,0,3)(2,1,0)[12] intercept   : AIC=1140.290, Time=18.11 sec
 ARIMA(3,0,2)(2,1,0)[12]             : AIC=1135.369, Time=5.38 sec

Best model:  ARIMA(3,0,2)(2,1,0)[12] intercept
Total fit time: 233.749 seconds
```

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	731
Model:	SARIMAX(3, 0, 2)x(2, 1, [], 12)	**Log Likelihood**	-557.356
Date:	Mon, 20 Jun 2022	**AIC**	1132.712
Time:	20:45:51	**BIC**	1173.912
Sample:	0	**HQIC**	1148.619
	- 731		
Covariance Type:	opg		

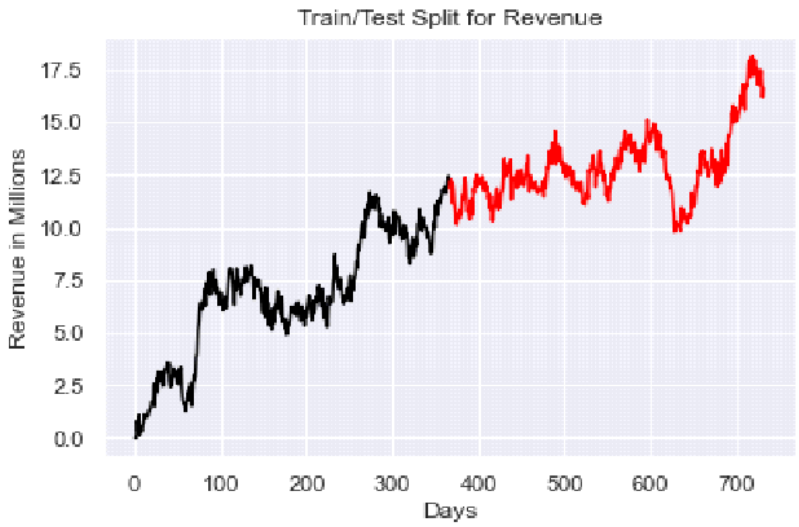
	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0063	0.003	1.825	0.068	-0.000	0.013
ar.L1	1.4351	0.074	19.435	0.000	1.290	1.580
ar.L2	-0.0293	0.126	-0.234	0.815	-0.275	0.217
ar.L3	-0.4174	0.070	-5.943	0.000	-0.555	-0.280
ma.L1	-0.9698	0.082	-11.780	0.000	-1.131	-0.808
ma.L2	0.1058	0.078	1.363	0.173	-0.046	0.258
ar.S.L12	-0.7100	0.038	-18.581	0.000	-0.785	-0.635
ar.S.L24	-0.3822	0.039	-9.753	0.000	-0.459	-0.305
sigma2	0.2711	0.015	17.713	0.000	0.241	0.301

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	1.89
Prob(Q):	0.94	**Prob(JB):**	0.39
Heteroskedasticity (H):	1.06	**Skew:**	0.01

Prob(H) (two-sided):	0.64	**Kurtosis:**	2.75

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

```
# Again begin by splitting dataframe in exactly half
df['Day'] = df.index
teleco_train = df.iloc[:len(df) - 365]

teleco_train['teleco_train'] = teleco_train['Revenue']
del teleco_train['Day']
del teleco_train['Revenue']


# Now we want to create a test set
teleco_test = df.iloc[len(df) - 365:]

teleco_test['teleco_test'] = teleco_test['Revenue']
del teleco_test['Day']
del teleco_test['Revenue']


# We can visualize the training and test split that we created by splitting
the data in half
plt.plot(teleco_train, color='black')
plt.plot(teleco_test, color='red')
plt.title('Train/Test Split for Revenue')
plt.xlabel('Days')
plt.ylabel('Revenue in Millions')
sns.set()
plt.show()
```



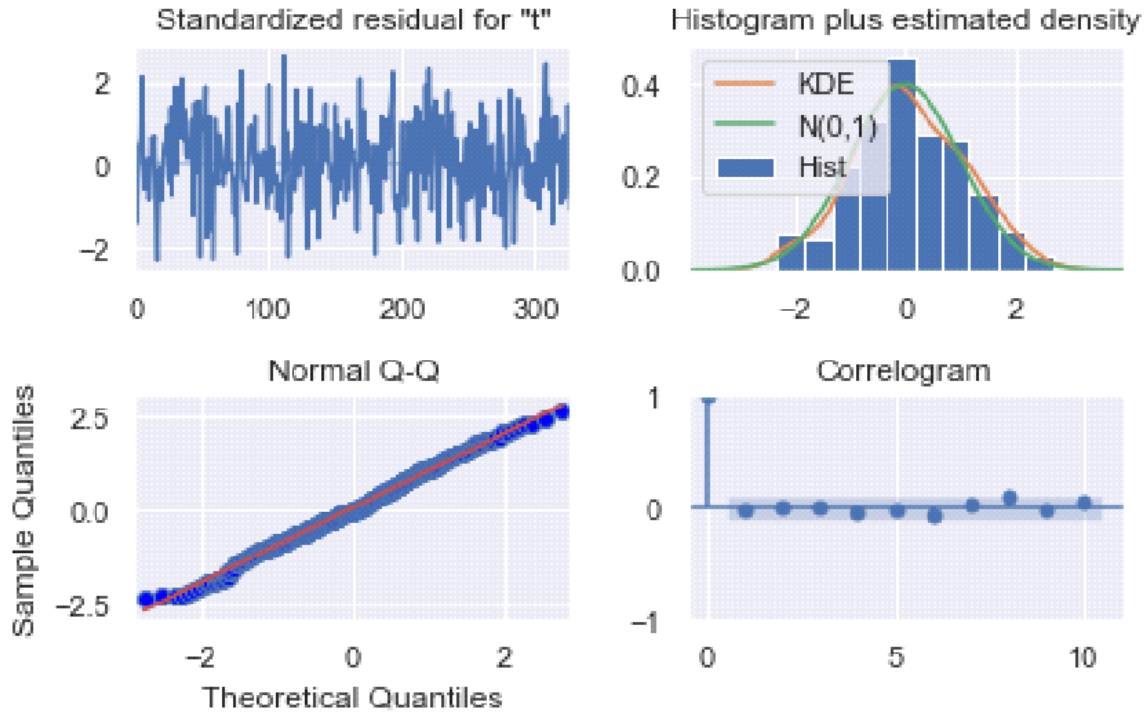
SARIMAX MODEL:

```python
# Import statsmodels API
import statsmodels.api as sm

# Build SARIMAX model
model = sm.tsa.SARIMAX(teleco_train,
                       order=(3, 0, 2),
                       seasonal_order=(2, 1, 0, 12),
                       enforce_stationarity=False,
                       enforce_invertibility=False)

SARIMAX_results = model.fit()

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

```
                                     SARIMAX Results
==========================================================================================
Dep. Variable:                        teleco_train   No. Observations:
366
Model:             SARIMAX(3, 0, 2)x(2, 1, [], 12)   Log Likelihood
-247.972
Date:                             Mon, 20 Jun 2022   AIC
511.944
Time:                                     20:53:29   BIC
542.263
Sample:                                          0   HQIC
524.042
                                               - 366
Covariance Type:                               opg
==========================================================================================
====
                 coef    std err          z      P>|z|      [0.025      0.
975]
------------------------------------------------------------------------------
----
ar.L1          1.3589      0.117     11.582      0.000       1.129       1
.589
ar.L2          0.0669      0.181      0.369      0.712      -0.288       0
.422
ar.L3         -0.4370      0.110     -3.989      0.000      -0.652      -0
.222
ma.L1         -0.8504      0.127     -6.682      0.000      -1.100      -0
.601
ma.L2          0.0595      0.119      0.500      0.617      -0.174       0
.293
ar.S.L12      -0.8340      0.063    -13.318      0.000      -0.957      -0
.711
ar.S.L24      -0.4170      0.060     -6.909      0.000      -0.535      -0
.299
sigma2         0.2662      0.023     11.628      0.000       0.221       0
.311
```

```
================================================================================
=========
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):
1.22
Prob(Q):                              0.89   Prob(JB):
0.54
Heteroskedasticity (H):               1.03   Skew:
0.03
Prob(H) (two-sided):                  0.88   Kurtosis:
2.71
================================================================================
=========

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

```
SARIMAX_results.plot_diagnostics()
plt.tight_layout()
```



D3. ARIMA Model Forecasting

```
# Forecast
result = SARIMAX_results.get_forecast()

# Summary of forecast results and confidence intervals
test_1 = teleco_test['teleco_test'].values.astype('float32')
forecast = result.predicted_mean
print('Expected: %.2f' % forecast)
print('Forecast: %.2f' % test_1[0])
print('Standard Error: %.2f' % result.se_mean)
```

```
Expected: 12.24
Forecast: 11.85
Standard Error: 0.52
```

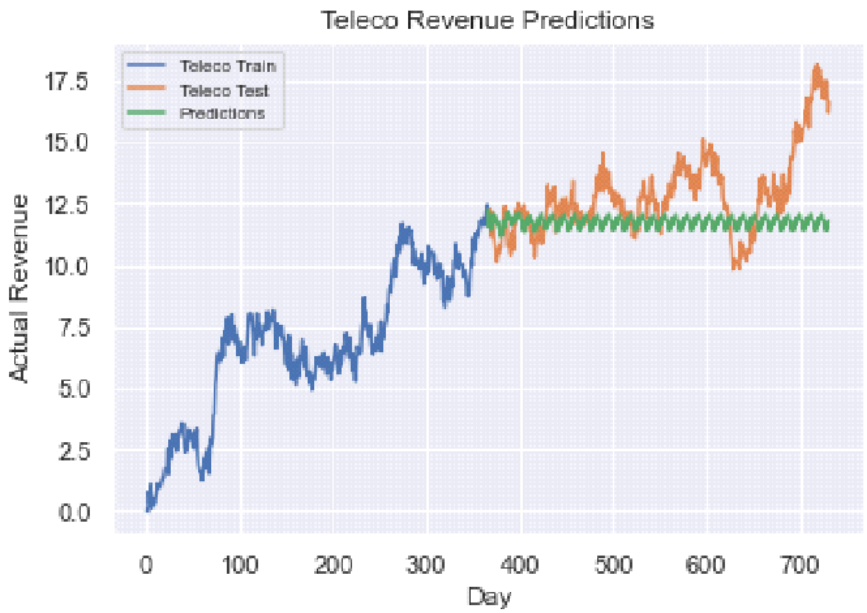
```
# Create intervals
intervals = [0.2, 0.1, 0.05, 0.01]
for a in intervals:
    ci = result.conf_int(alpha=a)
    print('%.1f%% Confidence Interval: %.2f between %.2f and  %.2f' % ((1 -
a) * 100, forecast, ci['lower teleco_train'], ci['upper teleco_train']))
```

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

Out[110]:

	lower teleco_train	upper teleco_train
366	10.909483	13.567237

```
#Prediction using test
start = len(teleco_train)
end = len(teleco_train) + len(teleco_test) - 1

# Predict with respect to test set
predictions = SARIMAX_results.predict(start, end, typ =
'levels').rename('Predictions')
plt.plot(teleco_train, label = 'Teleco Train')
plt.plot(teleco_test, label = 'Teleco Test')
plt.plot(predictions, label = 'Predictions')
plt.title('Teleco Revenue Predictions')
plt.xlabel('Day')
plt.ylabel('Actual Revenue')
plt.legend(loc='upper left', fontsize = 8)
plt.show()
```




```
#Prediction using the complete data set
model = sm.tsa.statespace.SARIMAX(Teleco_df['Revenue'],
                                  order=(3, 0, 2),
                                  seasonal_order=(2, 1, 0, 12),
                                  enforce_stationarity=False,
                                  enforce_invertibility=False)

results = model.fit()


# Result of prediction for the next year
forecast = results.predict(start = len(Teleco_df['Revenue']),
                           end = (len(Teleco_df['Revenue']) - 1) + 365,
                           typ = 'level').rename('Teleco Forecast Result')


Teleco_df['Revenue'].plot(figsize = (12, 5), legend = True)
forecast.plot(legend = True)
```

D4. Output and Calculations + D5. Supporting Codes

```
# Import packages for our evaluation
from sklearn.metrics import mean_squared_error
from statsmodels.tools.eval_measures import rmse

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

```
MSE:  4.0187
```

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

```
RMSE:  2.0047
```

Part V: Summary and Implications

E1. Results of Analysis

i. Selection of the ARIMA Model

In section D2 of this analysis, the 'Best Model' calculated was ARIMA (3,0,2)(2,1,0)[12]. This was then the values used in our SARIMAX model as we fit the data in our AUTO_ARIMA codes, thus giving us the optimal ARIMA model.

ii. Prediction Forecast Interval

In section D3 of this analysis, we had 4 confidence intervals that we choose to test. These being 80%, 90%, 95%, and 99% intervals. We also printed the lower and upper data from the teleco_train sets for a greater understanding of the data range. For example, we see that with our 95% confidence interval, there is only a 5% change that the data will fall outside the calculated range of 11.23 and 13.25.

iii. Forecast Length Justification

We are using the forecast length of a single year in our prediction to grant our users the insights of the more present and immediate short-term results. Looking at the forecasts, it is also shown that there are predicted highs and lows for that single year in terms of revenue which can be significant enough for upper-level management/shareholders.

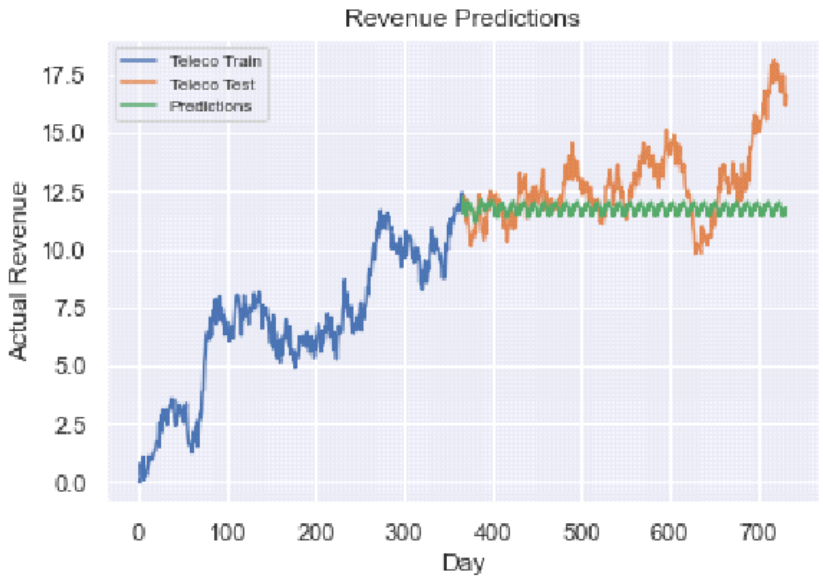
iv. Model Evaluations and Error Metrics

The evaluation that we choose to use for the SARIMAX model above was the MSE and root MSE. The results of both errors are shown section D4 of 4.018 and 2.004 respectively. Due to the fact that both of these evaluations are used as a measure of the quality of an estimator, we can conclude that the SARIMAX model was an effective model used for predictions since both values are relatively small compared to the data.

E2. Annotated Visualization of Forecast of Final vs Test Set

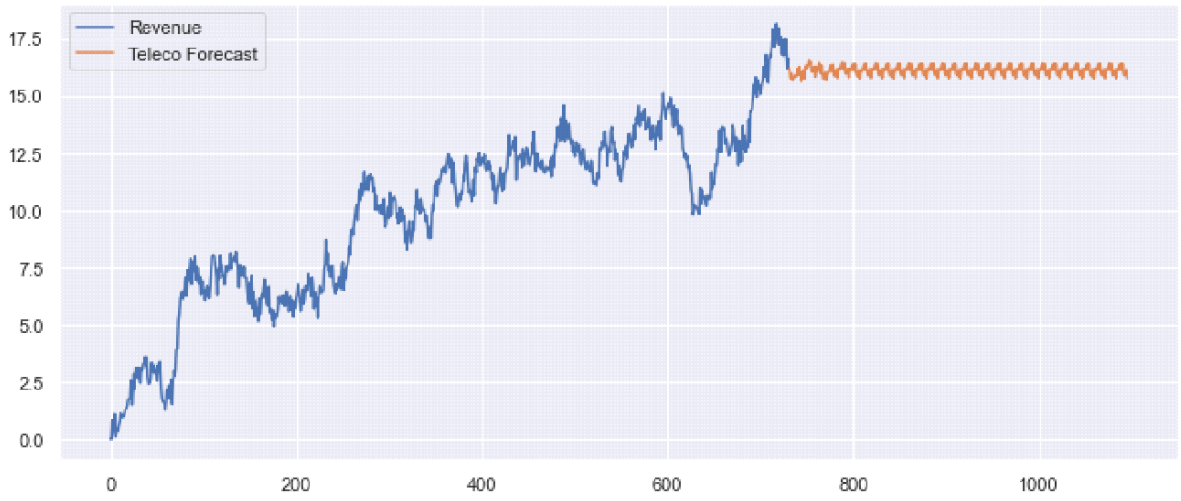
TEST SET Forecast:

```
start = len(teleco_train)
end = len(teleco_train) + len(teleco_test) - 1

predictions = SARIMAX_results.predict(start, end, typ =
'levels').rename('Predictions')
plt.plot(teleco_train, label = 'Teleco Train')
plt.plot(teleco_test, label = 'Teleco Test')
plt.plot(predictions, label = 'Predictions')
plt.title('Revenue Predictions')
plt.xlabel('Day')
plt.ylabel('Actual Revenue')
plt.legend(loc='upper left', fontsize = 8)
plt.show()
```



Final Model Forecast:

```
# Creating the forecast for the next revenue year
forecast = results.predict(start = len(Teleco_df['Revenue']),
                           end = (len(Teleco_df['Revenue']) - 1) + 365,
                           typ = 'level').rename('Teleco Forecast')

Teleco_df['Revenue'].plot(figsize = (12, 5), legend = True)
forecast.plot(legend = True)
```



E3. Recommendation Based on Results

Based on our time series prediction analysis, we can see that the data supports a continuation of an uptrend with regards to Revenue. There are also observe that there are clear seasonality fluctuations within the Revenue, however this is predictable and expected.

It also seems to solidify that the more customers spend on products and services within the telecom company, the less chance of them churning, as also inferred by this upward trend increase.

Attention should be paid attention to by some of the various sudden drops in revenue. It will be very important for the shareholders and company management to investigate what was the reason behind these drops as they signify customers who have churned from the company. Therefore, Recommendation would be to do significant data collection and figure out what is happening during times of sudden revenue decreases and apply a strategy to address them. Perhaps various forms of discounts or product bundling could be used to incentivize customers to stay if reasons is due to competition or pricing of products and services.

Code	000120121
Password	aLTGBCWECn7u

Part VI: Reporting

F. Jupyter Notebook Reporting

A copy of the Jupyter Notebook used is attached to the assignment.

G. Third Party Code

Selva Prabhakaran (2019). Time Series Analysis in Python – A Comprehensive Guide with Examples. https://www.machinelearningplus.com/time-series/time-series-analysis-python/

Project Pro (2022). How to Build ARIMA Model in Python for Time Series Forecasting?. https://www.projectpro.io/article/how-to-build-arima-model-in-python/544

Yugesh Verma (2021). Complete Guide to SARIMAX in Python for Time Series Modeling. https://analyticsindiamag.com/complete-guide-to-sarimax-in-python-for-time-series-modeling/

H. Acknowledged Sources

Songhao Wu (2021). Stationarity Assumption in Time Series Data. https://towardsdatascience.com/stationarity-assumption-in-time-series-data-67ec93d0f2f

Anais Dotis (2019). Autocorrelation in Time Series. https://dganais.medium.com/autocorrelation-in-time-series-c870e87e8a65

Selva Prabhakaran (2019). Augmented Dickey Fuller Test (ADF Test) – Must Read Guide. https://www.machinelearningplus.com/time-series/augmented-dickey-fuller-test/

Will Kenton (2020). Seasonality. https://www.investopedia.com/terms/s/seasonality.asp#:~:text=Seasonality%20is%20a%20characteristic%20of,is%20said%20to%20be%20seasonal.

Eberly College of Science (2022). Estimating the Spectral Density. https://online.stat.psu.edu/stat510/lesson/12/12.1#:~:text=The%20spectral%20density%20is%20a,express%20it%20in%20different%20ways.