Advanced Data Analytics - ARIMA – Performance Assessment

Steven Schindler

Advanced Data Analytics – D213

Sewell, Williams; PhD

College of I.T., Western Governors University

June 21, 2023

Part I: Research Question

A1.

Using the telecommunications data file, we can use forecasting techniques to ask: What is the revenue forecast for the upcoming quarter?

A2.

The goal of this analysis is to use time series to forecast revenue for the upcoming first quarter(2022) for the company by analyzing the company's revenue over a two-year period(2020-2021).

Part II: Method Justification

B.

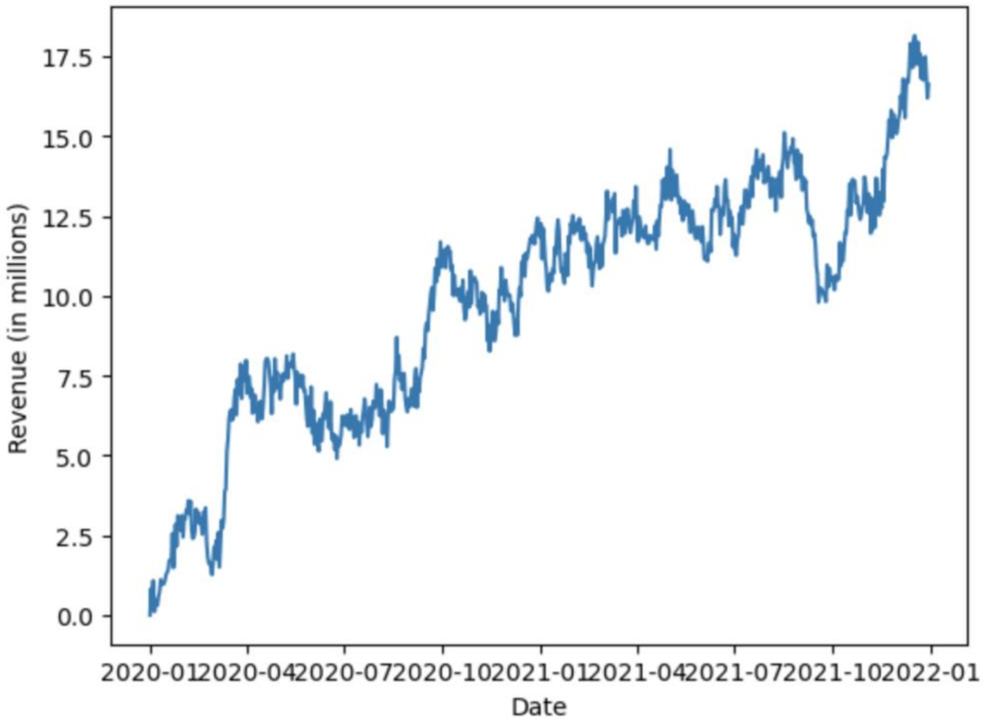
An assumption of the time series analysis is that the data should be stationary. This means that the mean and variance are constant over a period of time and the series has zero trend as it is not growing or shrinking. Stationarity also means that the autocorrelation is constant; how each value in the time series is related to its neighbor stays the same.(Hyndman, R.J., & Athanasopoulos, G. (2018).) Autocorrelation is a mathematical representation between a given time series and a lagged version of itself.(InfluxData. (n.d.).)

Part III: Data Preparation

C1.

Below is a line graph representing the revenue in millions from 2020 to 2022 two years later.

[<matplotlib.lines.Line2D at 0x7fee0a134940>]



C2.

dafr = pd.read_csv('teleco_time_series.csv')

dafr.dropna()
dafr = dafr[dafr['Revenue']> 0]

dafr['Date'] = pd.date_range(start = datetime(2020,1,1),periods = dafr.shape[0],freq='24H')
dafr = dafr.set_index(['Date'])

dafr = dafr.drop(['Day'],axis = 1)
dafr.shape

The above code checks for gaps in the time series by removing null values after the series is read into a data frame using the dropna() function. It also removes anywhere the revenue is 0 or less than 0. Then the data 'Day' is converted to a datetime format and then set as the index. After the removal of zero revenue the length of the sequence is 730 days.

C3.

Running the Dickey-Fuller test on the original data set we see that the p-value is greater than .05 at 0.39. This says that the original data is not stationary.

```
Results of Dickey-Fuller test:
Test Statistic                 -1.774638
p-value                         0.393124
#Lags Used                      1.000000
No. of Observations           728.000000
Critical Value (1%)            -3.439364
Critical Value (5%)            -2.865518
Critical Value (10%)           -2.568888
dtype: float64
```

Using differrencing on the data set and then running the ADfuller test again we now have a p-value of 0.000 this being less than .05 the data is now stationary.

```
Results of Dickey-Fuller test:
Test Statistic                -44.927782
p-value                         0.000000
#Lags Used                      0.000000
No. of Observations           728.000000
Critical Value (1%)            -3.439364
Critical Value (5%)            -2.865518
Critical Value (10%)           -2.568888
dtype: float64
```

C4.

The first step was to read the data into a pandas data frame called df. Then the Day column was converted into a Date column. Null values were than dropped and stationarity was tested using the Adfuller test. Stationarity was coerced using differencing and finally the Data was split into training and test sets. The training set uses 80% and the testing set uses 20%. The code to split the sets is listed below:

Xtrain = df_stationarity.iloc[:-145]

Xtest = df_stationarity.iloc[-145:]

print("X_train shape: ",Xtrain.shape)

print("X_test shape: ",Xtest.shape)

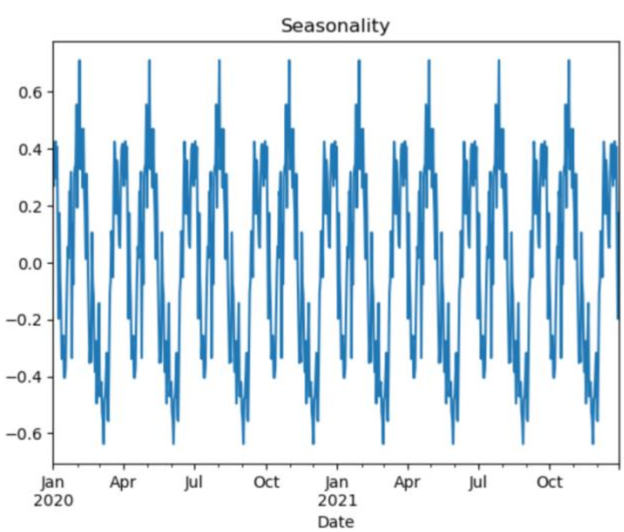
C5.

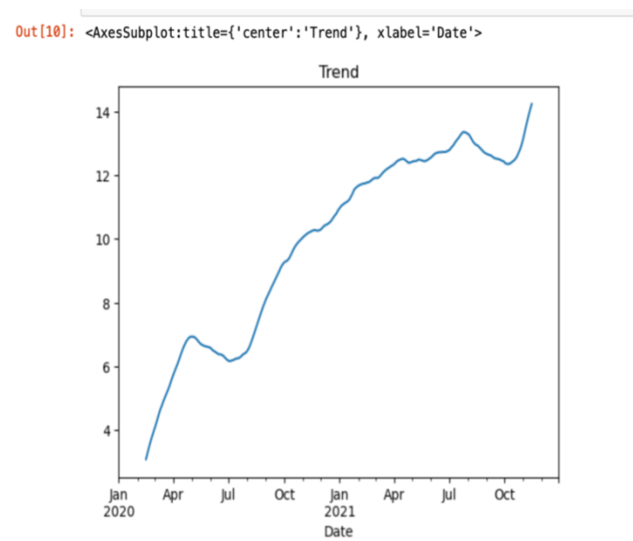
The cleaned dataset is called cleaned_D213.csv and the training and testing sets are called X_train.csv and X_test.csv respectively.

Part IV: Model Identification and Analysis

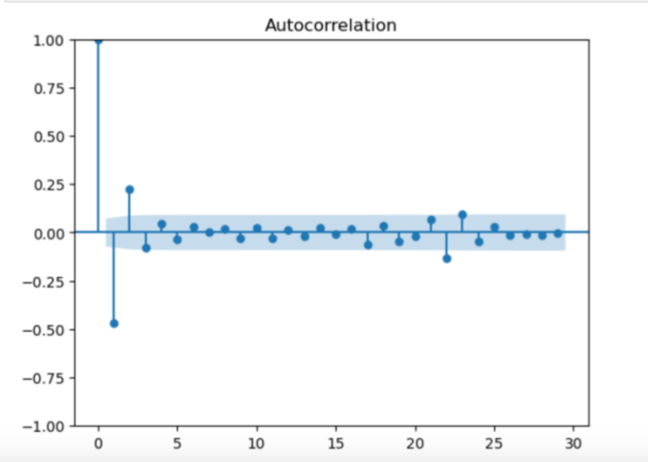
D1.

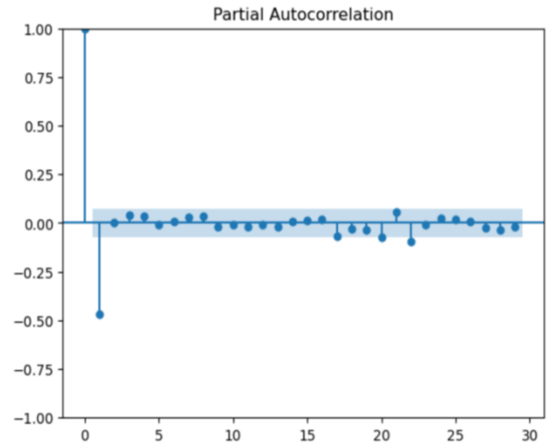
Below are the visuals for seasonality, trends, acf and pacf.



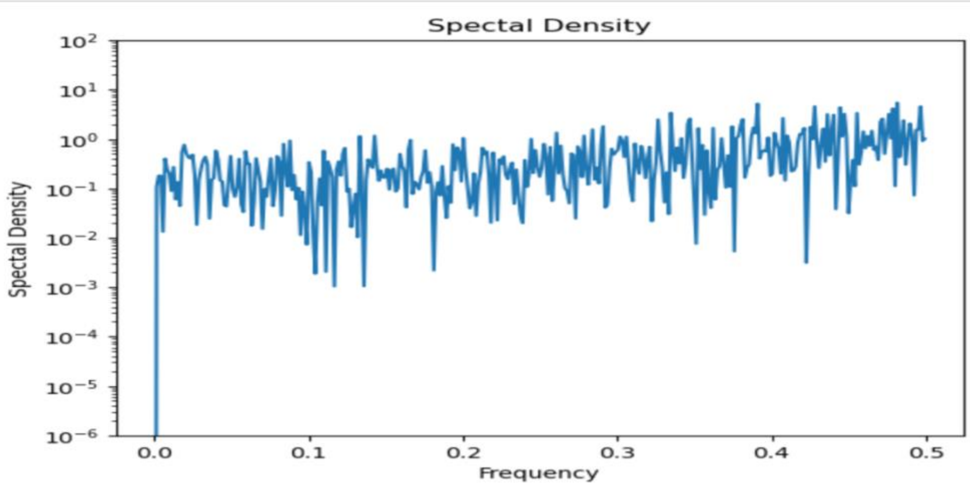


[11]:

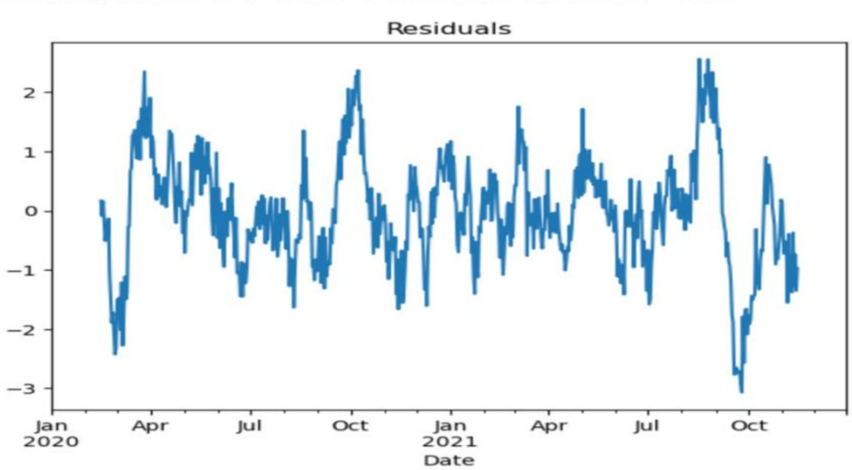


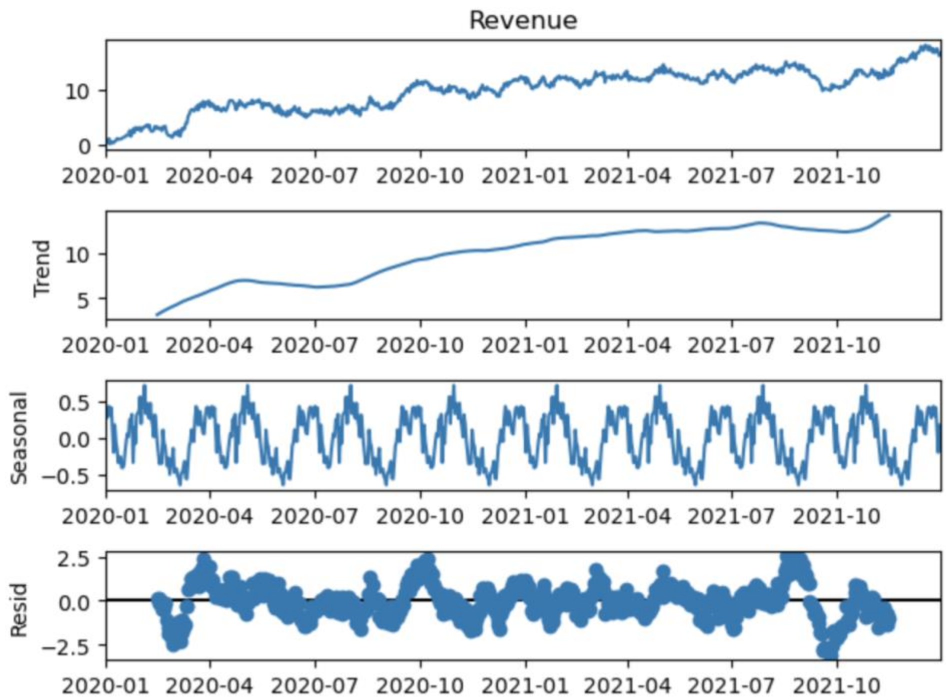


Next are the visualizations for spectral density, the residuals and the decomposed time series.



Out[15]: <AxesSubplot:title={'center':'Residuals'}, xlabel='Date'>





D2.

Using auto_arima we get the best model with the lowest aic is (5,1,0)(0,0,0)[0].

```
Performing stepwise search to minimize aic
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1945.480, Time=0.08 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=1379.929, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.31 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1943.480, Time=0.02 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=1221.799, Time=0.07 sec
 ARIMA(3,1,0)(0,0,0)[0] intercept   : AIC=1153.515, Time=0.10 sec
 ARIMA(4,1,0)(0,0,0)[0] intercept   : AIC=1126.331, Time=0.13 sec
 ARIMA(5,1,0)(0,0,0)[0] intercept   : AIC=1104.274, Time=0.16 sec
 ARIMA(5,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.86 sec
 ARIMA(4,1,1)(0,0,0)[0] intercept   : AIC=inf, Time=0.67 sec
 ARIMA(5,1,0)(0,0,0)[0]             : AIC=1102.281, Time=0.08 sec
 ARIMA(4,1,0)(0,0,0)[0]             : AIC=1124.337, Time=0.07 sec
 ARIMA(5,1,1)(0,0,0)[0]             : AIC=inf, Time=0.51 sec
 ARIMA(4,1,1)(0,0,0)[0]             : AIC=inf, Time=0.55 sec

Best model:  ARIMA(5,1,0)(0,0,0)[0]
Total fit time: 3.686 seconds
```

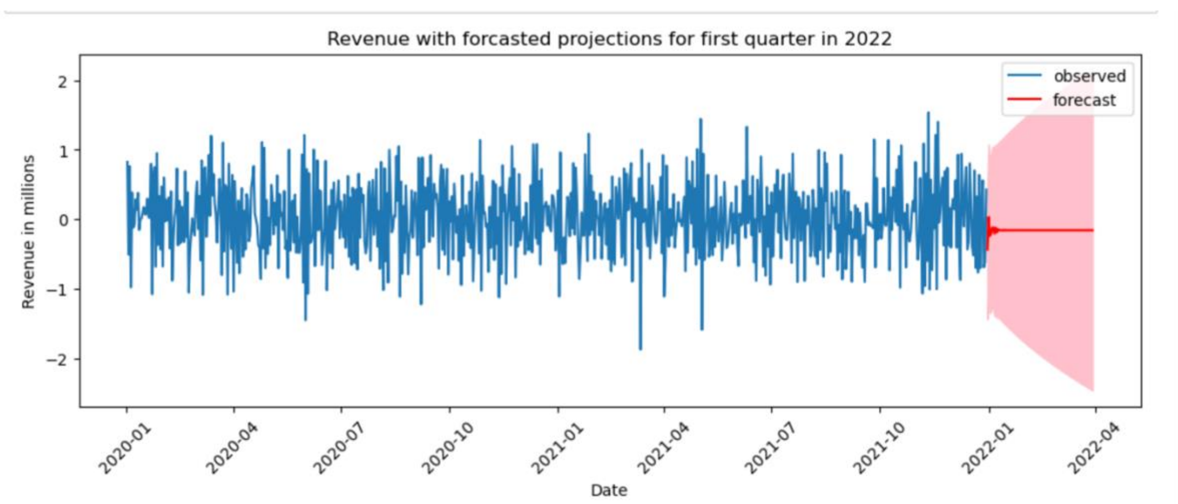
Out[38]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	729
Model:	SARIMAX(5, 1, 0)	**Log Likelihood**	-545.140
Date:	Sun, 18 Jun 2023	**AIC**	1102.281

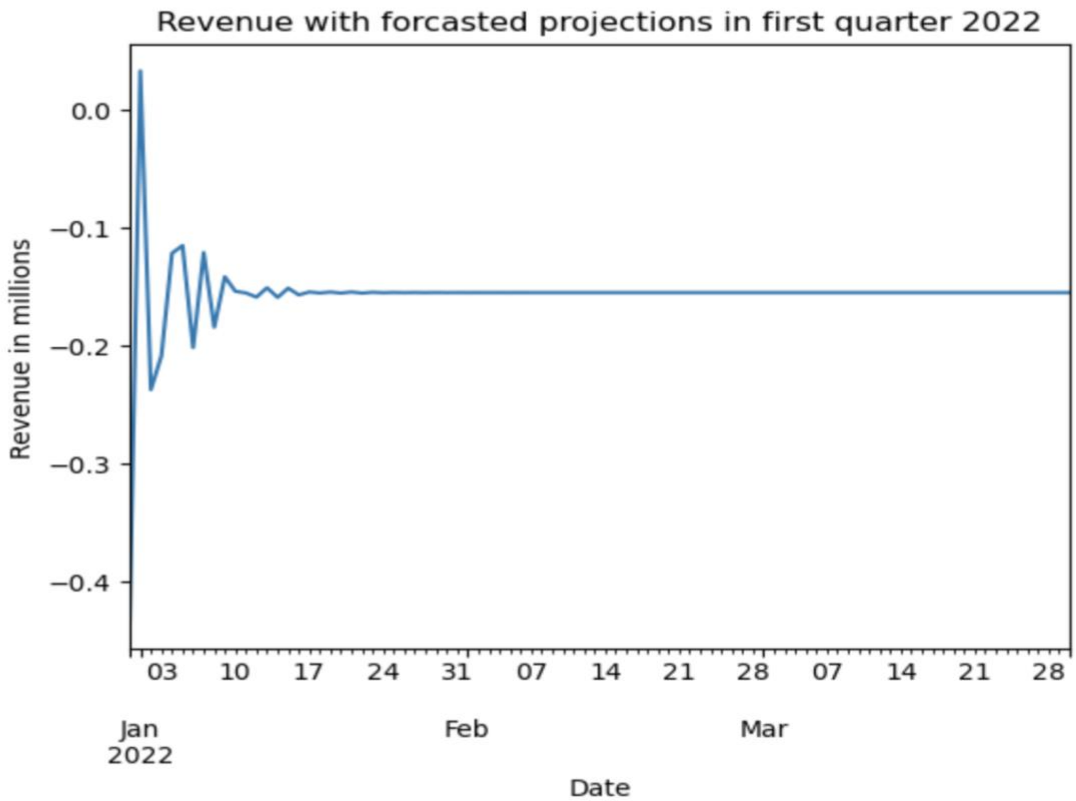
D3.

Below is a graph of the revenue with the forecasted model using the derived ARIMA model:



Next is a picture of just the forecasted first quarter for 2022:

```
: <AxesSubplot:title={'center':'Revenue with forcasted projections in fi
venue in millions'>
```



D4.

The following is my code to make and validate stationarity, autocorrelation plots and the sarimax model:

Stationarity:

from statsmodels.tsa.stattools import adfuller
print ('Dickey-Fuller test: ')

dfte = adfuller(dafr['Revenue'], autolag='AIC')
dfout = pd.Series(dfte[0:4], index=['Test Statistic','pvalue','#Lags','No.Observations'])

for key,value in dfte[4].items():
dfout['Critical Value (%s) '%key] = value # Critical Values should always be more than the test statistic

print(dfout)
df_stationarity = dafr.diff().dropna()
from statsmodels.tsa.stattools import adfuller

```
print ('Dickey-Fuller test: ')

dfte = adfuller(dafr['Revenue'], autolag='AIC')
dfout = pd.Series(dfte[0:4], index=['Test
Statistic','pvalue','#Lags','No.Observations'])

for key,value in dfte[4].items():
        dfout['Critical Value (%s) '%key] = value # Critical Values should always
be more than the test statistic

print(dfout)
```

Autocorrelation plots:

```
from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
plot_acf(df_stationarity)

plot_pacf(df_stationarity)
```

Sarimax:

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
warnings.filterwarnings("ignore")

models = SARIMAX(df_stationarity,order = (5,1,0),seasonal_order =
(0,0,0,0),error_action="ignore",supress_warnings = 'true')
results = models.fit()
print(results.summary())
```

The model summary for the SARIMAX:

```
CONVERGENCE: REL_REDUCTION_OF_F_<=_FACTR*EPSMCH
                               SARIMAX Results
==============================================================================
Dep. Variable:                Revenue   No. Observations:                  729
Model:               SARIMAX(5, 1, 0)   Log Likelihood                -545.140
Date:                Sun, 18 Jun 2023   AIC                           1102.281
Time:                        19:05:53   BIC                           1129.823
Sample:                    01-02-2020   HQIC                          1112.908
                         - 12-30-2021
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
ar.L1         -1.2959      0.036    -35.739      0.000      -1.367      -1.225
ar.L2         -1.0189      0.057    -17.814      0.000      -1.131      -0.907
ar.L3         -0.7097      0.065    -10.954      0.000      -0.837      -0.583
ar.L4         -0.4272      0.058     -7.306      0.000      -0.542      -0.313
ar.L5         -0.1813      0.037     -4.875      0.000      -0.254      -0.108
sigma2         0.2611      0.015     17.940      0.000       0.233       0.290
===================================================================================
Ljung-Box (L1) (Q):                   0.66   Jarque-Bera (JB):                 3.02
Prob(Q):                              0.42   Prob(JB):                         0.22
Heteroskedasticity (H):               0.99   Skew:                            -0.12
Prob(H) (two-sided):                  0.92   Kurtosis:                         2.79
===================================================================================
```

The mean forecast predictions for the first quarter of 2022:

```
2021-12-31   -0.433841
2022-01-01    0.032414
2022-01-02   -0.237616
2022-01-03   -0.208838
2022-01-04   -0.122104
                ...
2022-03-26   -0.155405
2022-03-27   -0.155405
2022-03-28   -0.155405
2022-03-29   -0.155405
2022-03-30   -0.155405
```

D5.

Code is listed below and in the provided Juypter notebook file.

import numpy as np
import pandas as pd
import matplotlib.pylab as plt
import warnings

```
warnings.filterwarnings("ignore")
from datetime import datetime
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.statespace.sarimax import SARIMAX
dafr = pd.read_csv('teleco_time_series.csv')

dafr.dropna()
dafr = dafr[dafr['Revenue']> 0]

dafr['Date'] = pd.date_range(start = datetime(2020,1,1),periods =
dafr.shape[0],freq='24H')
dafr = dafr.set_index(['Date'])

dafr = dafr.drop(['Day'],axis = 1)
dafr.shape
plt.xlabel('Date')
plt.ylabel('Revenue "(in millions)"')
plt.plot(dafr)
from statsmodels.tsa.stattools import adfuller
print ('Dickey-Fuller test: ')

dfte = adfuller(dafr['Revenue'], autolag='AIC')
dfout = pd.Series(dfte[0:4], index=['Test
Statistic','pvalue','#Lags','No.Observations'])

for key,value in dfte[4].items():
        dfout['Critical Value (%s) '%key] = value # Critical Values should always
be more than the test statistic

print(dfout)

df_stationarity = dafr.diff().dropna()

from statsmodels.tsa.stattools import adfuller
print ('Dickey-Fuller test: ')

dfte = adfuller(dafr['Revenue'], autolag='AIC')
dfout = pd.Series(dfte[0:4], index=['Test
Statistic','pvalue','#Lags','No.Observations'])
```

```
for key,value in dfte[4].items():
        dfout['Critical Value (%s) '%key] = value # Critical Values should always
be more than the test statistic

print(dfout)

Xtrain = df_stationarity.iloc[:-145]
Xtest = df_stationarity.iloc[-145:]

print("X_train shape: ",Xtrain.shape)
print("X_test shape: ",Xtest.shape)

df_stationarity.to_csv("cleaned_D213.csv")
Xtrain.to_csv('X_train.csv')
Xtest.to_csv('X_test.csv')

decom = seasonal_decompose(dafr['Revenue'], period=90)

plt.title('Seasonality')
decom.seasonal.plot()
plt.title('Trend')
decom.trend.plot()

from statsmodels.tsa.stattools import acf, pacf
from statsmodels.graphics.tsaplots import plot_acf,plot_pacf
plot_acf(df_stationarity)

plot_pacf(df_stationarity)

from scipy import signal
f, Pxx = signal.periodogram(df_stationarity['Revenue'])
plt.semilogy(f, Pxx)
plt.ylim([1e-6, 1e2])
plt.title('Spectral Density')
plt.xlabel('Frequency')
plt.ylabel('Spectral Density')
plt.show()

decom.plot() # Plot decomposition
plt.show() # Check  for seasonality in the data
```

```
plt.title('Residuals')
decom.resid.plot()

from pmdarima.arima import auto_arima
stepwise_fit =
auto_arima(df_stationarity,start_p=0,start_q=0,d=1,seasonal=True,trace=True,s
uppress_warnings=True)
stepwise_fit.summary()

from statsmodels.tsa.arima.model import ARIMA
warnings.filterwarnings("ignore")
model = ARIMA(df_stationarity['Revenue'], order=(5,1,0))
results_ARIMA = model.fit()

model_fit = model.fit()
print(model_fit.summary())

from statsmodels.tsa.statespace.sarimax import SARIMAX
warnings.filterwarnings("ignore")

models = SARIMAX(df_stationarity,order = (5,1,0),seasonal_order =
(0,0,0,0),error_action="ignore",supress_warnings = 'true')
results = models.fit()
print(results.summary())

pred = results.get_prediction(start = -90)
mean_pred = pred.predicted_mean
confidence = pred.conf_int()
lower = confidence.loc[:,'lower Revenue']
upper = confidence.loc[:,'upper Revenue']
print(mean_pred)

plt.figure(figsize=(12,4))
plt.plot(Xtest.index, Xtest,label='test set')

plt.plot(mean_pred.index, mean_pred,color='r',label='forecast')

plt.fill_between(lower.index,lower,upper,color='pink')
```

```
plt.title('Forcast compared with test data')
plt.xlabel('Date')
plt.ylabel('Revenue in millions')
plt.legend()
plt.show()


forecast = results.get_forecast(steps=145)
meanforecast = forecast.predicted_mean
confidence = forecast.conf_int()
lower = confidence.loc[:,'lower Revenue']
upper = confidence.loc[:,'upper Revenue']
print(meanforecast)

plt.figure(figsize=(12,4))
plt.plot(df_stationarity.index, df_stationarity,label='observed')

plt.plot(meanforecast.index, meanforecast,color='r',label='forecast')

plt.fill_between(lower.index,lower,upper,color='pink')


plt.title('Revenue with forcasted projections for first quarter in 2022')
plt.xlabel('Date')
plt.ylabel('Revenue in millions')
plt.xticks(rotation=45)
plt.legend()
plt.show()

plt.title('Revenue with forcasted projections in first quarter 2022')
plt.xlabel('Date')
plt.ylabel('Revenue in millions')
meanforecast.plot()

from sklearn.metrics import mean_absolute_error
mean_absolute_error(Xtest.values, meanforecast.values)
```

Part V: Data Summary and Implications

E1.

The selection of the Arima model was chosen using auto Arima to find the lowest AIC. This outputted a best Arima model of ARIMA(5,1,0)(0,0,0)[0]. The code to get this best model is shown below:

from pmdarima.arima import auto_arima
stepwise_fit =
auto_arima(df_stationarity,start_p=0,start_q=0,d=1,seasonal=True,trace=True,suppress_warnings=True)
stepwise_fit.summary()

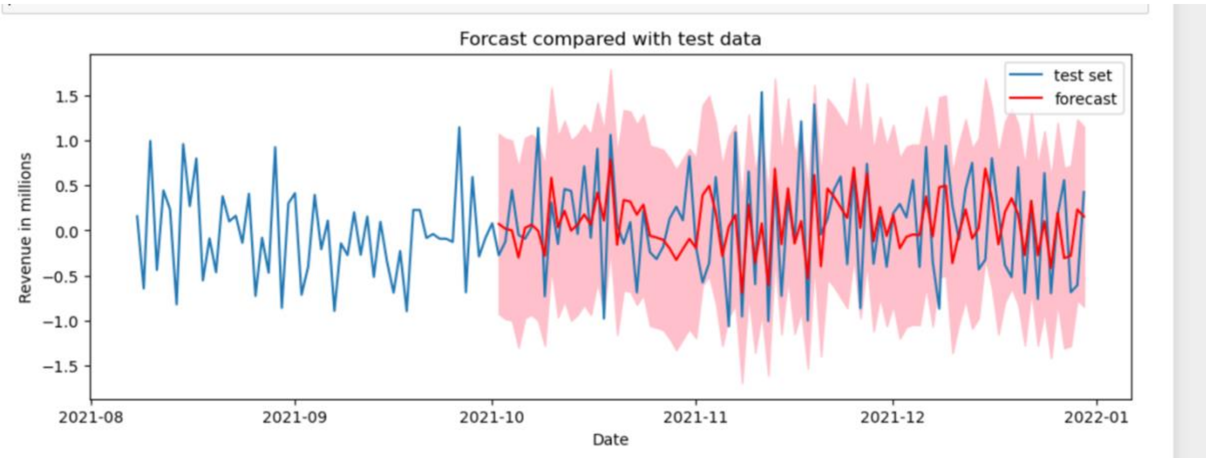
The prediction interval of the forecast is 1 as our data is two years at a daily interval. Because of this the ARIMA model identifies seasonality and correlations of daily revenue.

The model can only predict up to one year as there is only two years of data however, since we are only predicting for a quarter the forecast length is 3 months.

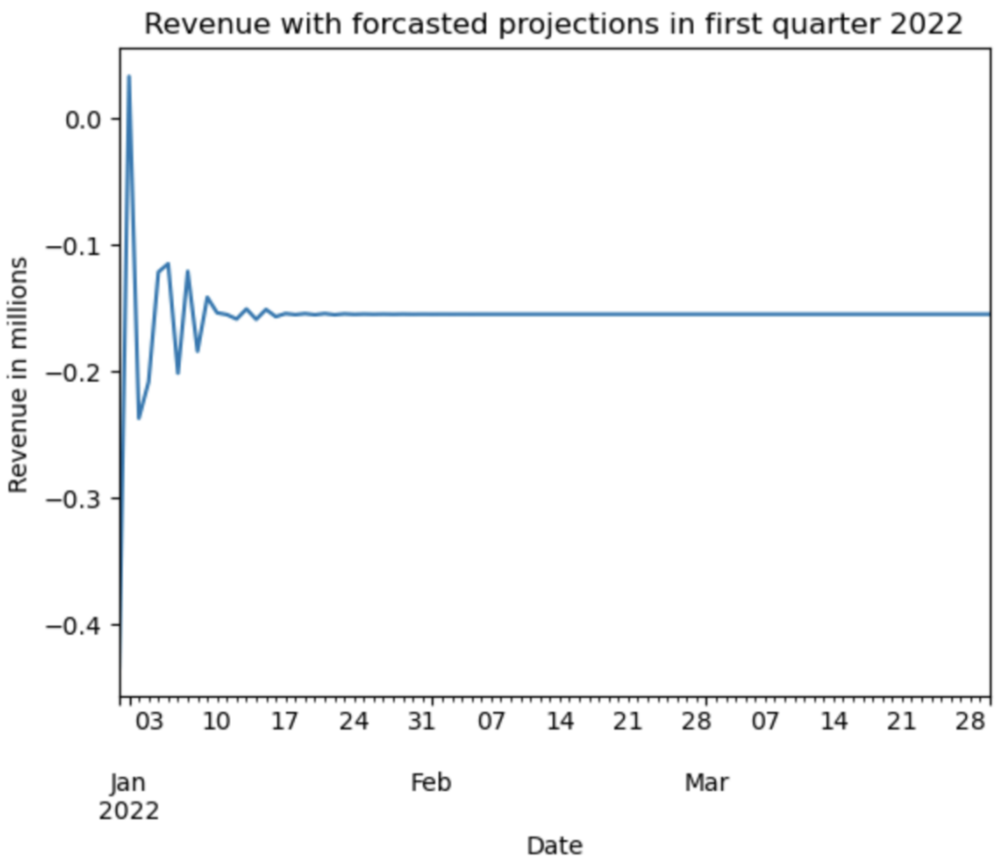
Auto ARIMA was used as the model evaluation procedure to find the lowest AIC. The mean square error and mean absolute error were both calculated at .36 and .48 respectively.

E2.

Below is an annotated visualization of the final model compared to the test set:



E3.



Based on the above visualizations it looks like the revenue is going to be steadily negative around Jan 17th therefore, I recommend for the company to investigate ways of reducing cost and increase efforts for retention.

References:

InfluxData. (n.d.). Autocorrelation in Time Series Data. Retrieved from https://www.influxdata.com/blog/autocorrelation-in-time-series-data/

Hyndman, R.J., & Athanasopoulos, G. (2018). Stationarity. In Forecasting: Principles and Practice (2nd Edition). OTexts. Retrieved from https://otexts.com/fpp2/stationarity.html