Eric Yarger

D213 Task 1

Initial Setup

In [1]:
```
# Import Initial Libraries
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
from scipy import stats
from statsmodels.tsa.stattools import adfuller
import statsmodels
import datetime
```

Environment

In [2]:
```
# Windows 10, Anaconda, JupyterLab, JupyterNotebook
# Jupyter environment version
!jupyter --version
```

```
Selected Jupyter core packages...
IPython          : 7.31.1
ipykernel        : 6.15.2
ipywidgets       : not installed
jupyter_client   : 7.3.5
jupyter_core     : 4.10.0
jupyter_server   : 1.18.1
jupyterlab       : 3.4.4
nbclient         : 0.5.13
nbconvert        : 6.4.4
nbformat         : 5.5.0
notebook         : 6.4.12
qtconsole        : not installed
traitlets        : 5.1.1
```

In [3]:
```
# Python Version
import platform
print(platform.python_version())
```

```
3.7.13
```

In [4]:
```
#Load Medical Dataset
df = pd.read_csv('C:/Users/ericy/Desktop/medical_time_date.csv')
```

EDA

In [5]:
```
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 731 entries, 0 to 730
Data columns (total 2 columns):
 #   Column   Non-Null Count  Dtype
---  ------   --------------  -----
 0   Day      731 non-null    object
 1   Revenue  731 non-null    float64
dtypes: float64(1), object(1)
memory usage: 11.5+ KB
```

In [6]:
```
df.shape
```

Out[6]:
```
(731, 2)
```

In [7]:
```
df.describe()
```

Out[7]:

	Revenue
count	731.000000
mean	14.179608
std	6.959905
min	-4.423299
25%	11.121742
50%	15.951830
75%	19.293506
max	24.792249

In [8]:
```
df.head()
```

Out[8]:

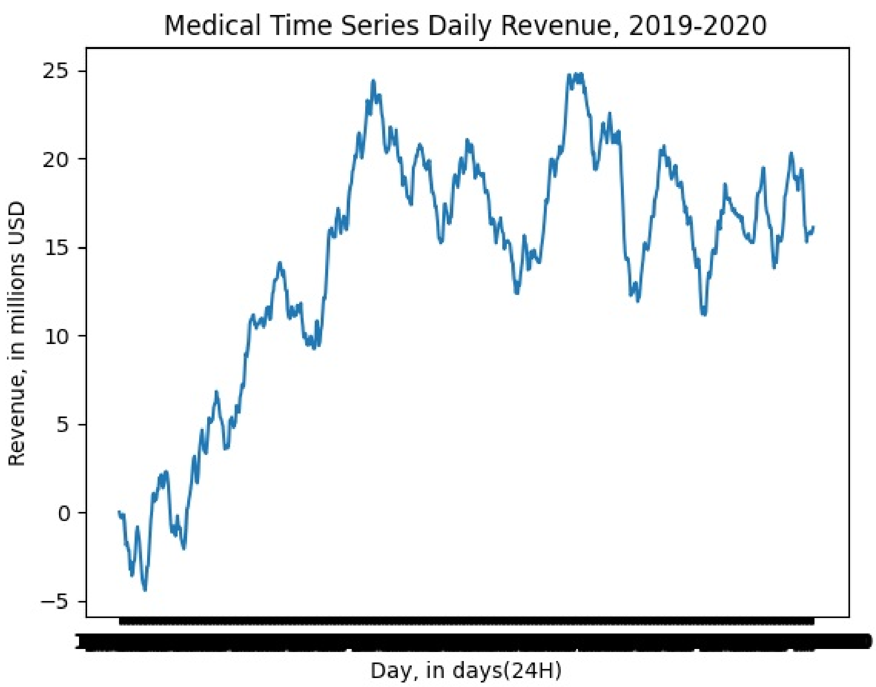
	Day	Revenue
0	1/1/2019	0.000000
1	1/2/2019	-0.292356
2	1/3/2019	-0.327772
3	1/4/2019	-0.339987
4	1/5/2019	-0.124888

In [9]:
```
df.isnull().any()
```

Out[9]:
```
Day          False
Revenue      False
dtype: bool
```

C1: Line Graph Visualization

In [10]:
```
plt.plot(df['Day'],df['Revenue'])
plt.title('Medical Time Series Daily Revenue, 2019-2020')
plt.xlabel('Day, in days(24H)')
plt.ylabel('Revenue, in millions USD')
plt.show()
```



Data Cleaning

In [11]:
```
# Drop any null columns
df = df.dropna()
```

In [12]:
```
# Export cleaned data to excel file
#df.to_excel('C:/Users/ericy/Desktop/medical_time_clean.xlsx')
```

C2: Time Step Formatting

Set df['Date'] to Index

In [13]:
```
# Day to datetime
df['Day'] = pd.to_datetime(df['Day'])
```

In [14]:
```
# Set Day as Index
df.set_index('Day',inplace=True)
```

In [15]:
```
df.isnull().any()
```

Out[15]:
```
Revenue    False
dtype: bool
```

In [16]:
```
# Export cleaned data to excel file
df.to_excel('C:/Users/ericy/Desktop/medical_time_clean.xlsx')
```

C3: Stationarity Analysis

Augmented Dickey Fuller (ADF) Test

Assess stationarity of dataset

In [17]:
```
# Code Reference (Making time series stationary | Python, n.d.)
dicky_fuller_test = adfuller(df)
```

In [18]:
```
dicky_fuller_test
```

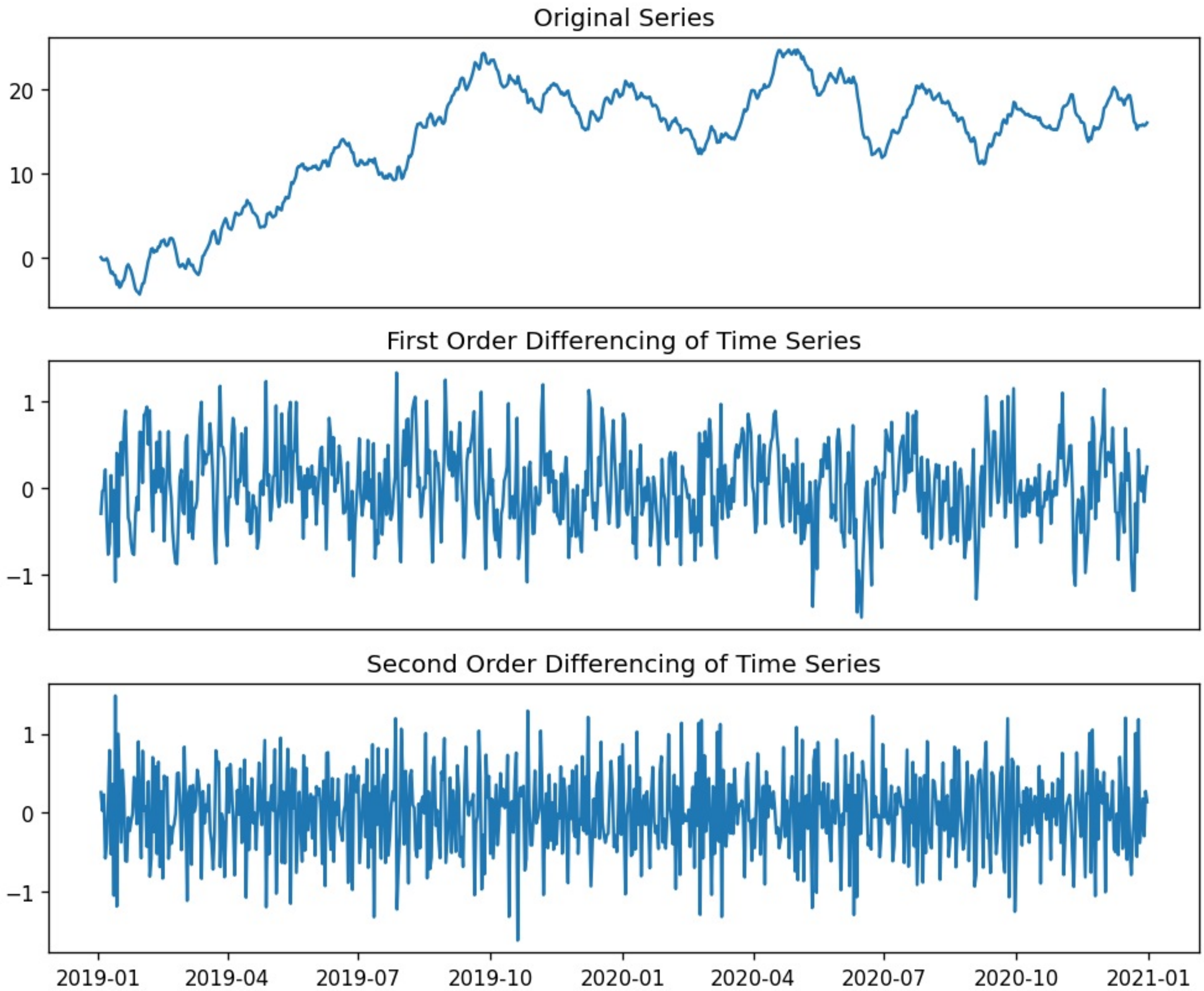
Out[18]:
```
(-2.2183190476089485,
 0.19966400615064228,
 1,
 729,
 {'1%': -3.4393520240470554,
  '5%': -2.8655128165959236,
  '10%': -2.5688855736949163},
 842.4530276176408)
```

In [19]:
```
# Results show p = .19964
# Data does not reject null hypothesis at p < .05
# Therefore, Time series is non-stationary
```

1st and 2nd order Differencing

finding 'd' for ARIMA model

In [20]:
```
# Set plot parameters for multi-ax subplots
plt.rcParams.update({'figure.figsize':(10,8), 'figure.dpi':120})

# Establish that there are three subplots
fig, (ax1, ax2, ax3) = plt.subplots(3)

# Plot the original dataset
ax1.plot(df); ax1.set_title('Original Series'); ax1.axes.xaxis.set_visible(False)

# First Order differencing of Time Series
ax2.plot(df.diff()); ax2.set_title('First Order Differencing of Time Series'); ax2.axes.xaxis.set_visible(False

# Second Order Differencing of Time Series
ax3.plot(df.diff().diff()); ax3.set_title('Second Order Differencing of Time Series')

# Plot all three graphs
plt.show()
```



In [21]:

```
# Using pmdarima's ndiffs to find differencing term
# Code reference (Verma, 2021)
from pmdarima.arima import ndiffs

kpss_diffs = ndiffs(df, alpha=0.05, test='kpss', max_d=6)
adf_diffs = ndiffs(df, alpha=0.05, test='adf', max_d=6)
n_diffs = max(adf_diffs, kpss_diffs)

print(f"Estimated differencing term: {n_diffs}")
```

```
Estimated differencing term: 1
```

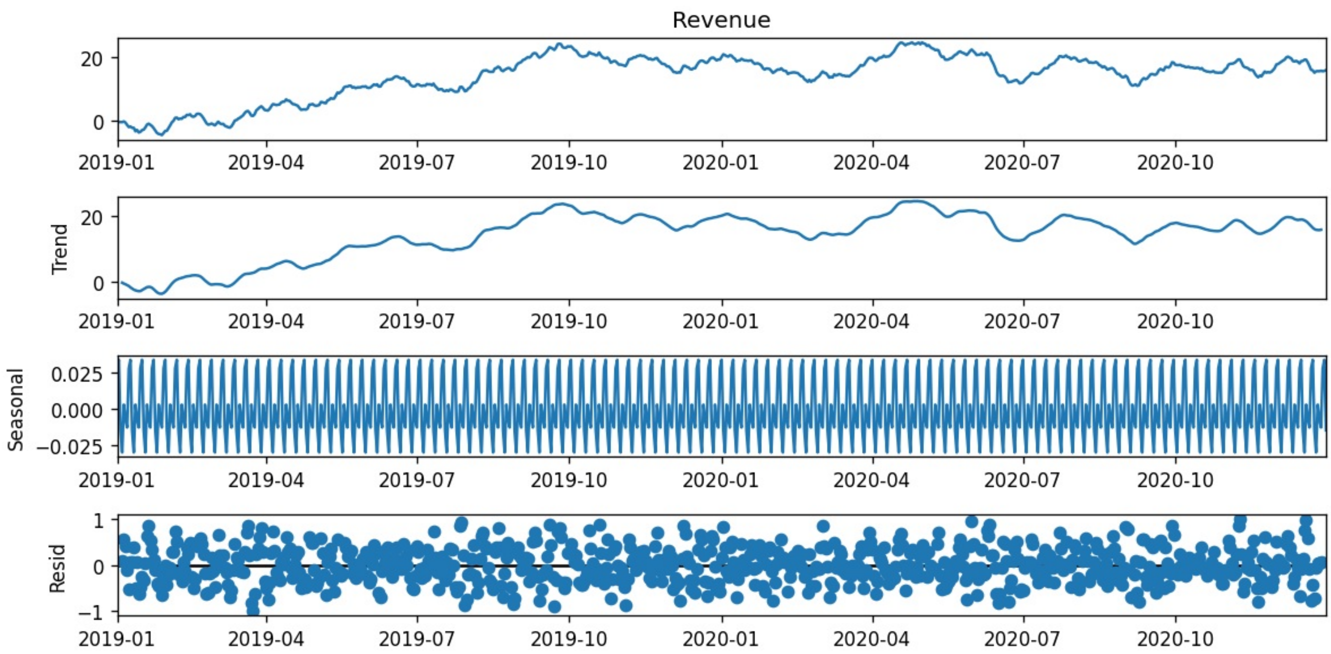
Seasonality Analysis

In [22]:

```
# Code Reference (Boston, 2020)
from statsmodels.tsa.seasonal import seasonal_decompose
result = seasonal_decompose(df['Revenue'])
```

In [23]:

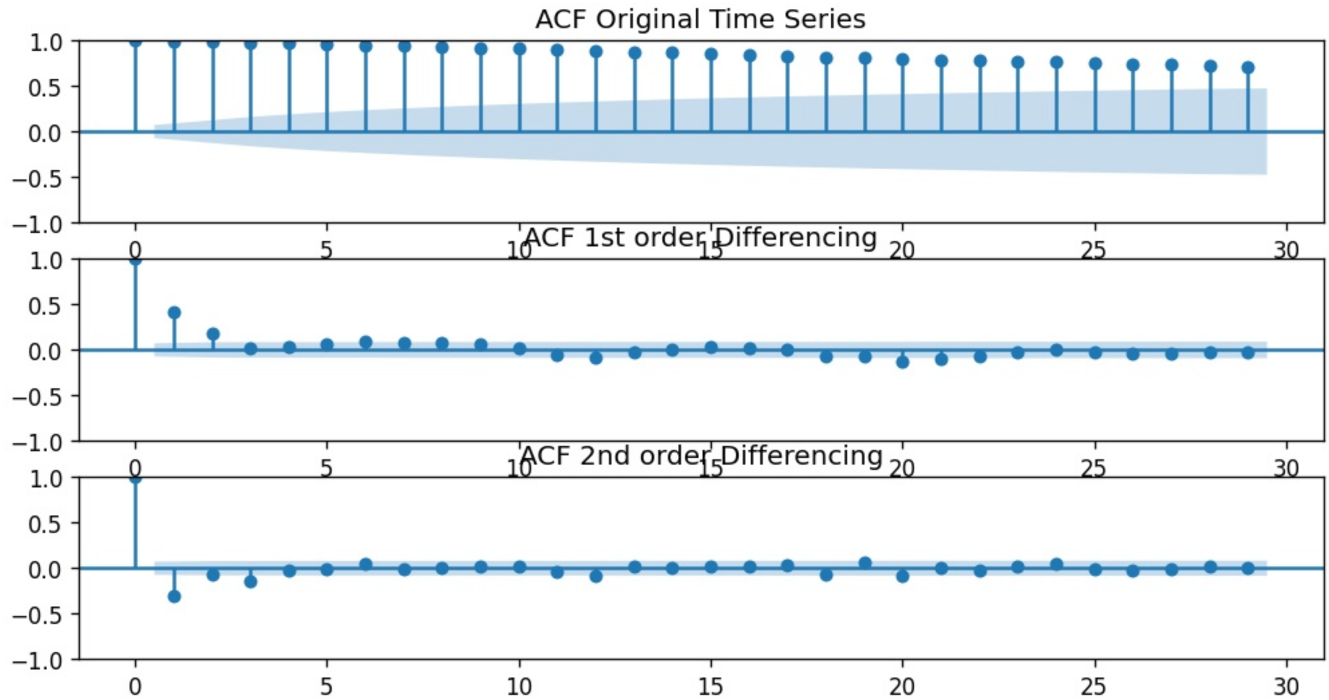
```
# plotting the result of our seasonal decomposition from the step above
from pylab import rcParams
rcParams['figure.figsize'] = 10,5
result.plot();
```



Finding order of MA term 'q'

Using Autocorrelation function (ACF)

In [24]:
```
from statsmodels.graphics.tsaplots import plot_acf

fig, (ax1, ax2, ax3) = plt.subplots(3)
plot_acf(df, ax=ax1, title='ACF Original Time Series');
plot_acf(df.diff().dropna(), ax=ax2, title='ACF 1st order Differencing');
plot_acf(df.diff().diff().dropna(), ax=ax3, title='ACF 2nd order Differencing');
```



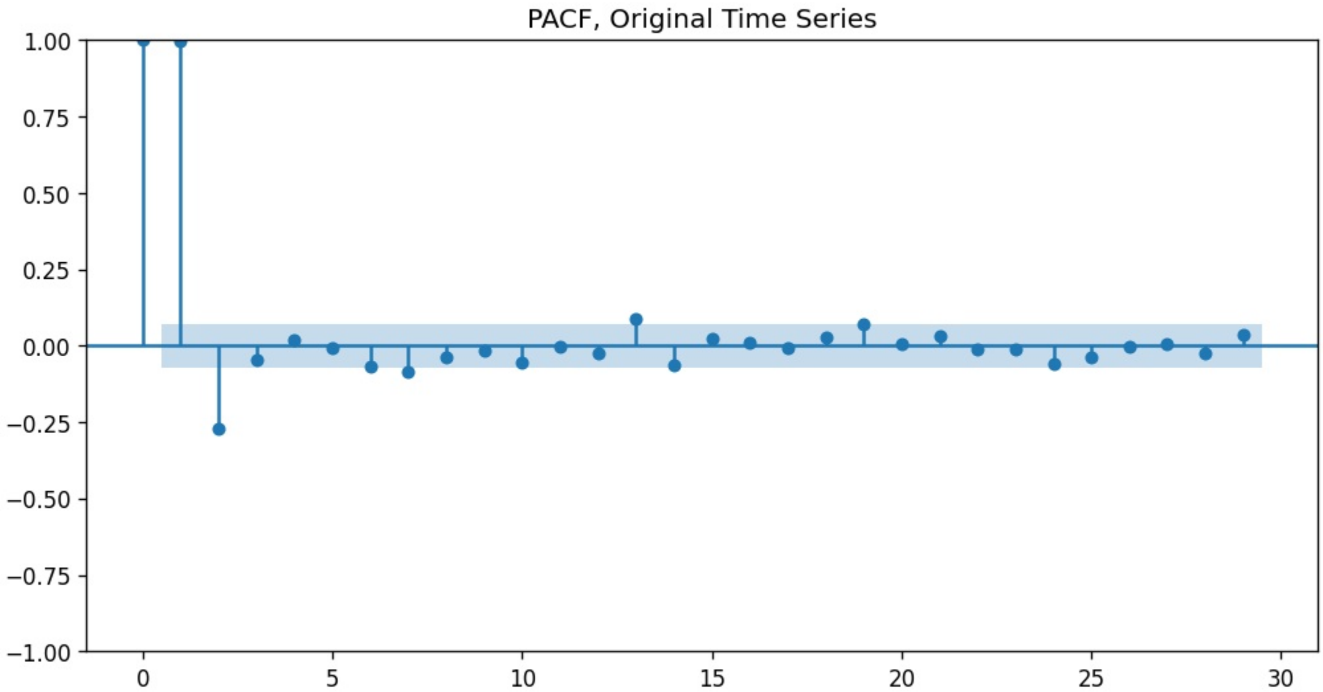
In [ ]:

In [ ]:

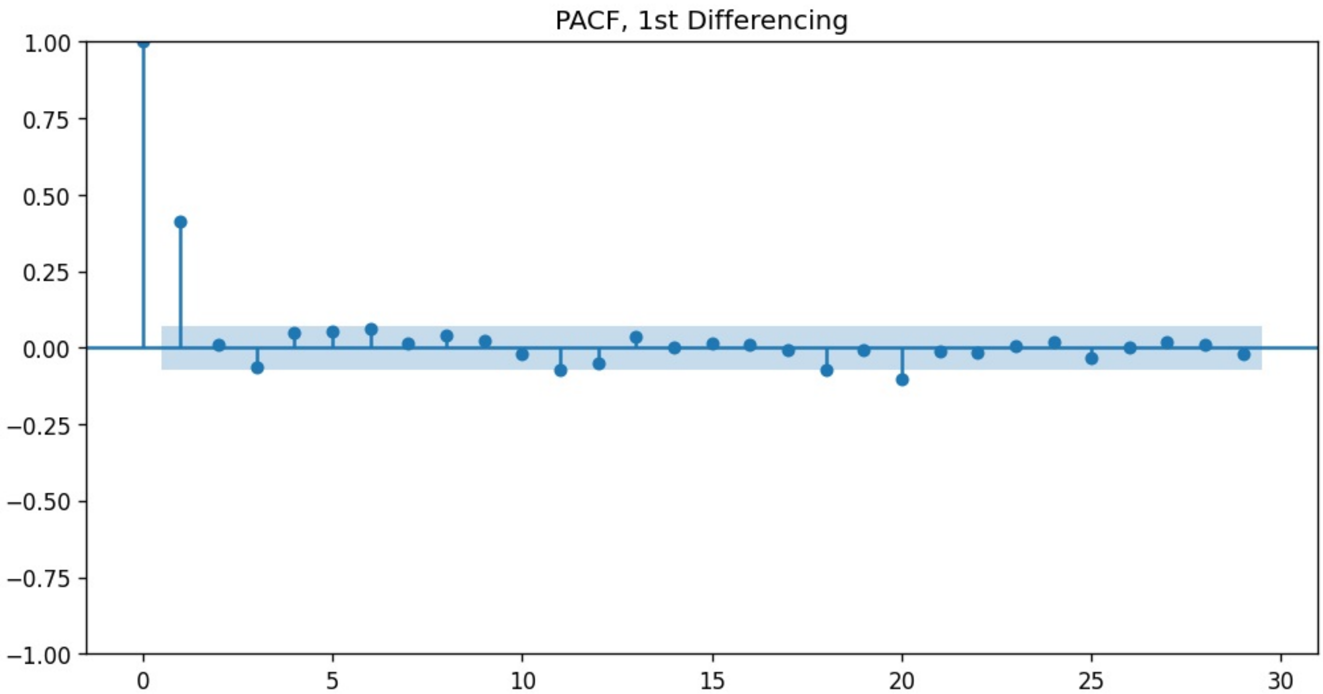
Finding order of AR term 'p'

Using Partial autocorrelation (PACF)

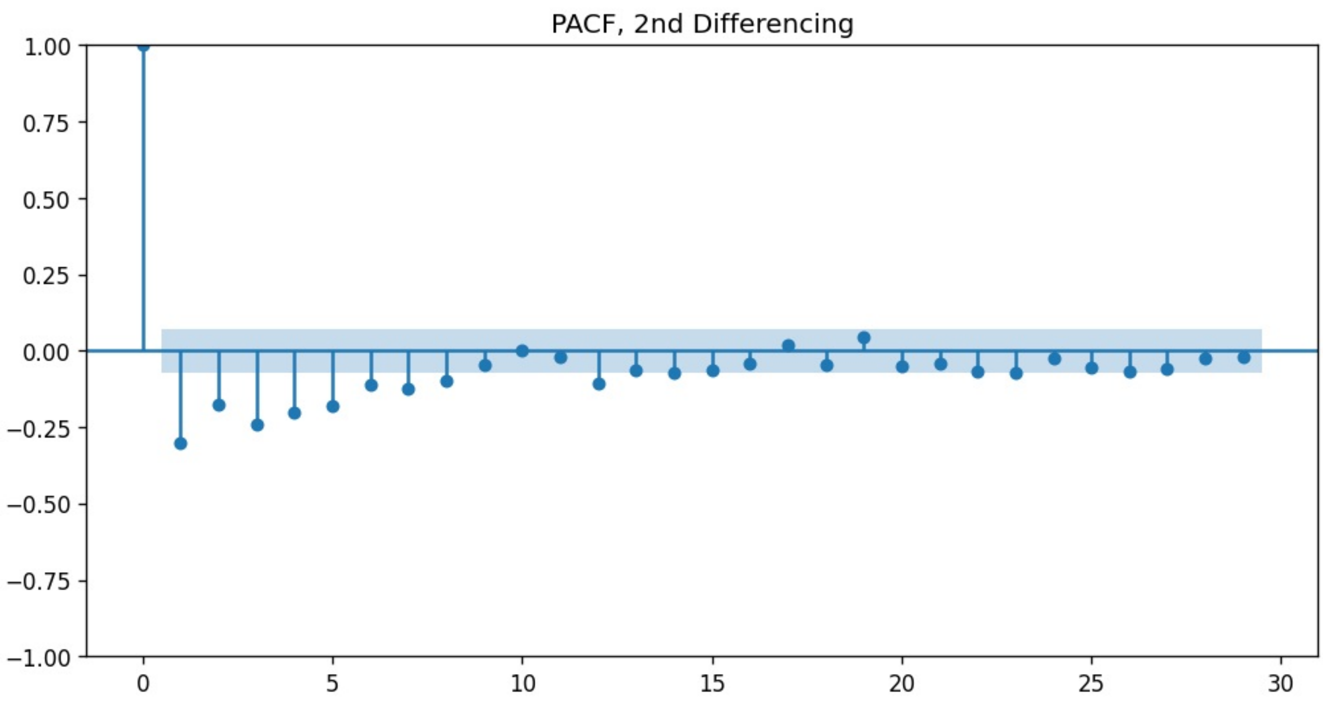
In [25]:
```
from statsmodels.graphics.tsaplots import plot_pacf
#Warnings ignored because this analysis uses deprecated ARIMA model from statsmodels
import warnings
warnings.filterwarnings("ignore")
plot_pacf(df.dropna(), title='PACF, Original Time Series');
```



In [26]:
```
plot_pacf(df.diff().dropna(), title='PACF, 1st Differencing');
```

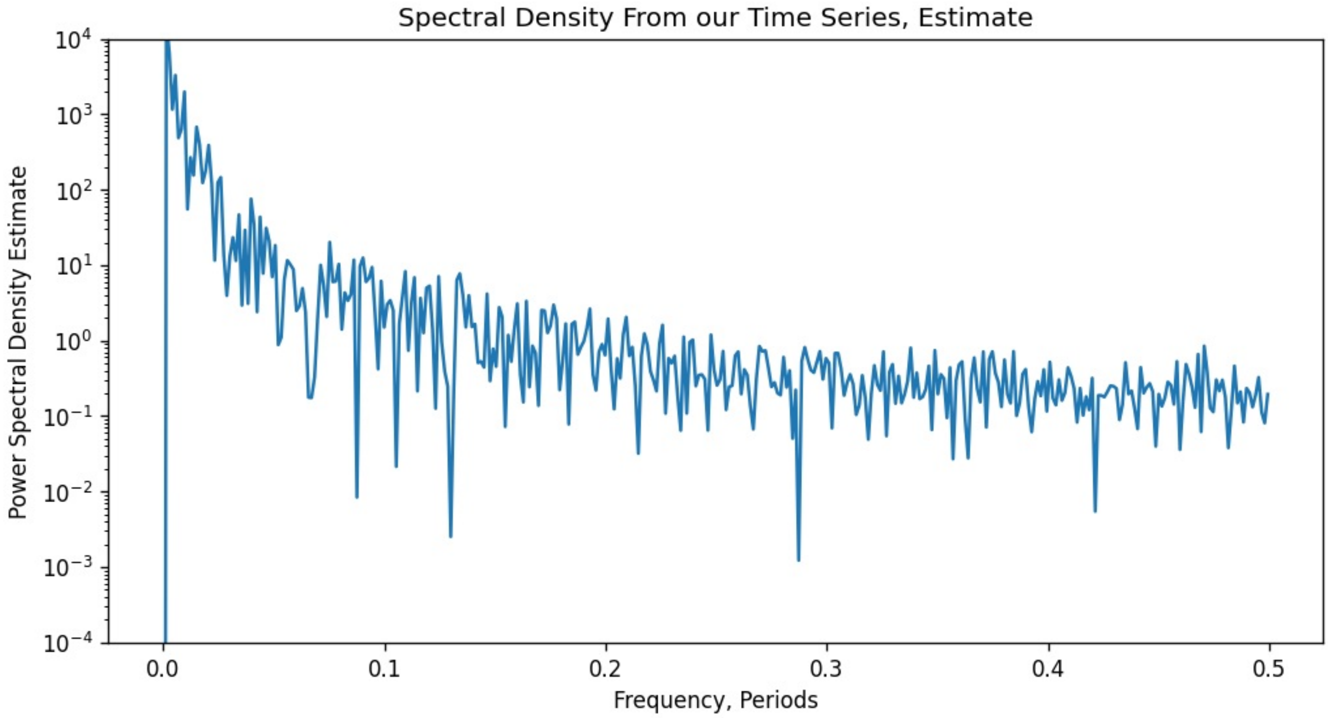


In [27]:
```
plot_pacf(df.diff().diff().dropna(), title='PACF, 2nd Differencing');
```



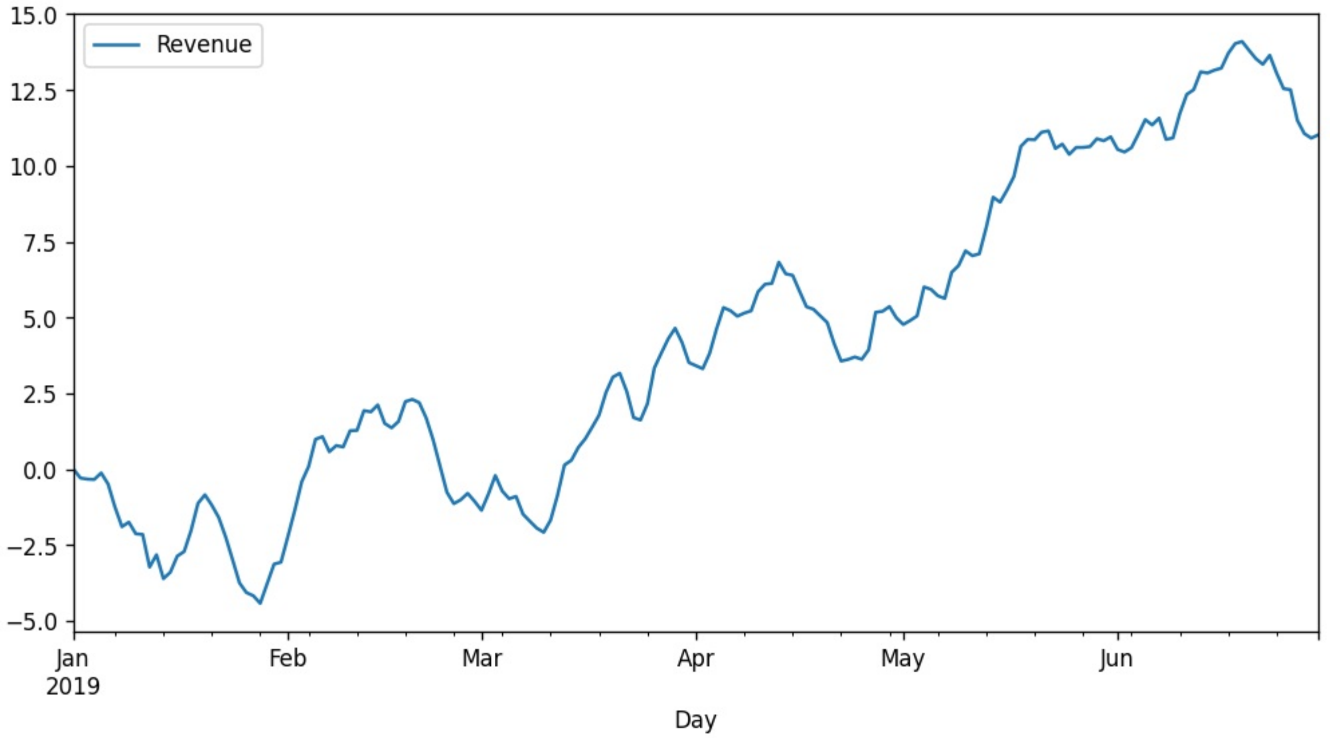
Spectral Density

In [28]:
```
# Spectral Density
# Code Reference (Festus, 2022)
from scipy import signal

# signal periodogram
f, Pxx_den = signal.periodogram(df['Revenue'])

# plotting semilogy - pyplot module used to make a plot with log scaling on the y-axis
plt.semilogy(f, Pxx_den)

# Setting coordinate values and titles for Spectral Density Graph
# setting y-axis min and max value
plt.ylim(1e-4, 1e4)

# Graph Title
plt.title('Spectral Density From our Time Series, Estimate')

# X label for Periods
plt.xlabel('Frequency, Periods')

# Y Label for SD Estimate
plt.ylabel('Power Spectral Density Estimate')
plt.show()
```



In [29]:
```
# Looking at specific location of plot
df.loc[:'2019-6-30'].plot()
```

Out[29]: <AxesSubplot:xlabel='Day'>



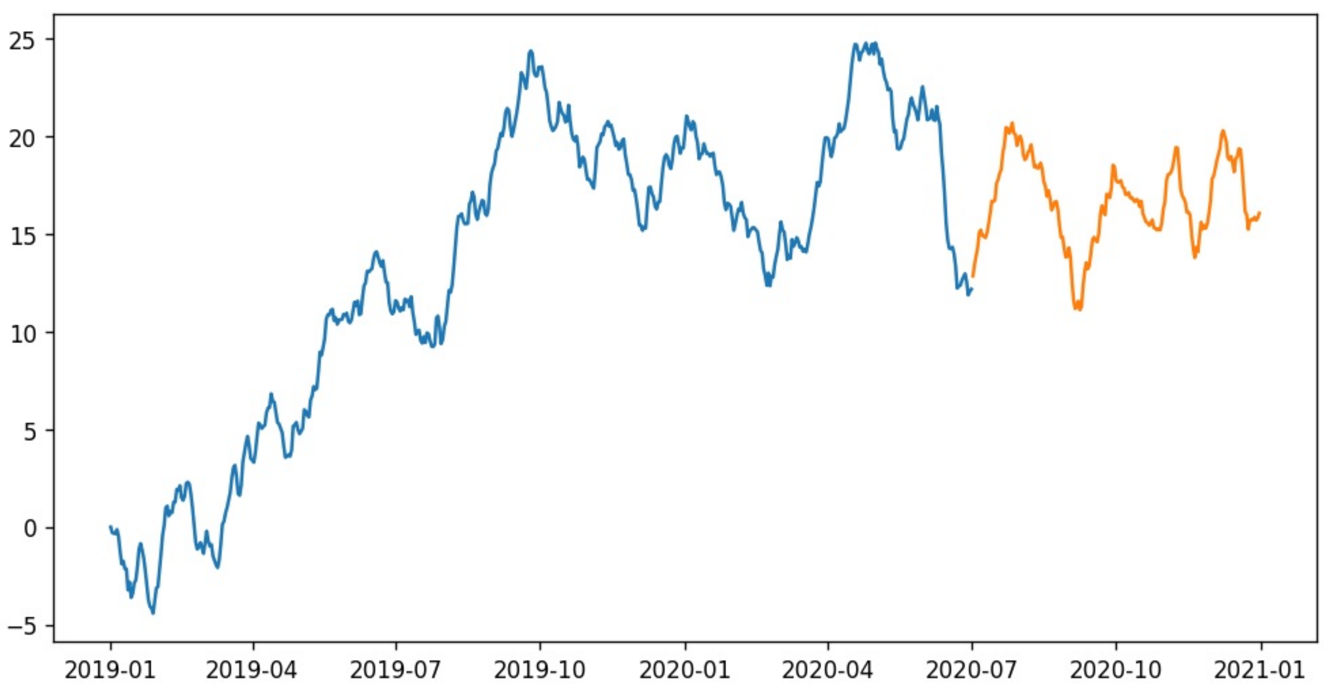
In [ ]:

Create Train/Test Datasets

In [30]:
```
# Splitting data into Test and Train sets using pmdarima's train_test_split
# code reference (Smith, 2019)
from pmdarima.model_selection import train_test_split

y = df
train, test = train_test_split(y, train_size=548)
```

In [31]:
```
# Plot training data
plt.plot(train)

# Plot Test Data
plt.plot(test)
```

Out[31]: [<matplotlib.lines.Line2D at 0x19761d24f48>]



In [32]:
```
print(train.shape)
```

```
print(test.shape)
```

```
(548, 1)
(183, 1)
```

In [33]:
```
# Read out Test and Train sets to Excel file

test.to_excel('C:/Users/ericy/Desktop/medical_time_test_clean.xlsx')
train.to_excel('C:/Users/ericy/Desktop/medical_time_train_clean.xlsx')
```

Auto-arima

Using pmdarima's auto_arima

In [34]:
```
# Fit the model using auto_arima
# Auto-arima code reference (6. Tips to using auto_arima — pmdarima 2.0.1 documentation, n.d.)
# Additional code reference (Pmdarima.arima.AutoARIMA — pmdarima 2.0.1 documentation, n.d.)
# Auto-arima, initial parameter attempt
# Code Reference (Kosaka, 2021)
from pmdarima.arima import StepwiseContext
from pmdarima.arima import auto_arima

# Establish auto_arima to run ARIMA and take into account
# Any Seasonality of the data, and any trends found.
model = auto_arima(train, start_p=1, start_q=1,
                   test='adf',
                   max_p=4,
                   max_q=4,
                   m=1,
                   d=1,
                   seasonal=True,
                   stationarity=False,
                   seasonal_test='ocsb',
                   trace=True,
                   error_action='ignore',
                   suppress_warnings=True,
                   stepwise=True,
                  trend='c')

# Print Summary of Best AIC Minimized SARIMAX Model
print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=672.789, Time=0.14 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=767.938, Time=0.08 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=671.106, Time=0.06 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=691.699, Time=0.07 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=767.938, Time=0.07 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=672.640, Time=0.09 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=671.553, Time=0.28 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=671.106, Time=0.05 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 0.844 seconds
                               SARIMAX Results
==============================================================================
Dep. Variable:                      y   No. Observations:                  548
Model:               SARIMAX(1, 1, 0)   Log Likelihood                -332.553
Date:                Thu, 29 Sep 2022   AIC                            671.106
Time:                        12:06:21   BIC                            684.019
Sample:                    01-01-2019   HQIC                           676.154
                         - 07-01-2020
Covariance Type:                  opg
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
intercept      0.0131      0.019      0.684      0.494      -0.024       0.050
ar.L1          0.4063      0.039     10.399      0.000       0.330       0.483
sigma2         0.1974      0.013     15.436      0.000       0.172       0.223
===================================================================================
Ljung-Box (L1) (Q):                   0.07   Jarque-Bera (JB):                 1.58
Prob(Q):                              0.79   Prob(JB):                         0.45
Heteroskedasticity (H):               1.06   Skew:                            -0.03
Prob(H) (two-sided):                  0.69   Kurtosis:                         2.74
===================================================================================

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

In [35]:
```
model.conf_int()
```

Out[35]:

	0	1
intercept	-0.024364	0.050476
ar.L1	0.329751	0.482925
sigma2	0.172370	0.222511

In [ ]:

In [ ]:

In [ ]:

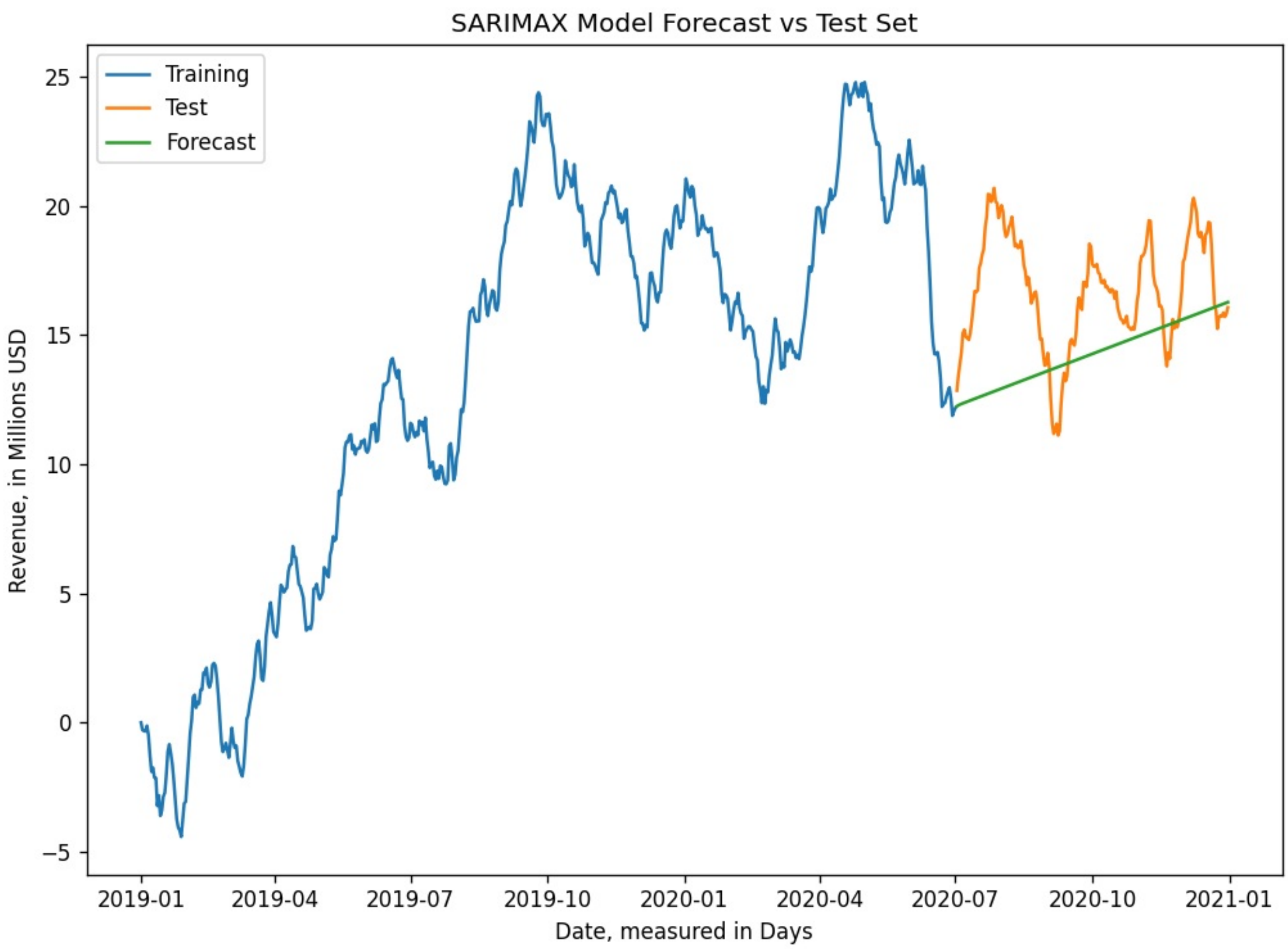
In [ ]:

In [ ]:

In [ ]:

In [36]:

```
# Prediction assignment, predicted revenue column named
# Training, Test, and Predicted data plotted together
# Code Reference (Matplotlib.pyplot.plot — Matplotlib 3.6.0 documentation, n.d.)

# Creating varible with forecast values
forecast = pd.DataFrame(model.predict(n_periods = 183),index=test.index)

# Naming forecast_revenue column in forecast variable
forecast.columns = ['forecast_revenue']

# Establish plot parameters for Forecast

# Plot figure size
plt.figure(figsize=(10,7))

# Training data
plt.plot(train,label="Training")

# Annotate X-axis label
plt.xlabel('Date, measured in Days')

# Annotate Y-axis label
plt.ylabel('Revenue, in Millions USD')

# Annotate Plot Title
plt.title('SARIMAX Model Forecast vs Test Set')

# Plot Test Data
plt.plot(test,label="Test")

# Plot Forecast Data
plt.plot(forecast,label="Forecast")

# Plot legend in upper lefthand corner
plt.legend(loc = 'upper left')

# Show Plot
plt.show()
```



Accuracy Metrics for our forecast

In [37]:
```
# RMSE and MAE to test model accuracy

from sklearn.metrics import mean_squared_error
from math import sqrt
```

In [38]:
```
# Create array of actual Revenue values, stored in Test variable

test_array = test[['Revenue']].to_numpy()
train[-1:]
```

Out[38]:

	Revenue
Day	
2020-07-01	12.18032

In [39]:
```
test_array.shape
```

Out[39]:
```
(183, 1)
```

In [40]:
```
# Predictions to numpy array
predicted_array = forecast[['forecast_revenue']].to_numpy()
```

In [41]:
```
predicted_array
```

Out[41]:
```
array([[12.24289778],
       [12.28138171],
       [12.31007526],
       [12.3347906 ],
       [12.35788945],
       [12.38033145],
       [12.40250655],
       [12.4245732 ],
       [12.44659578],
       [12.46860045],
       [12.49059785],
       [12.51259229],
       [12.53458553],
       [12.55657829],
       [12.57857084],
       [12.60056331],
```

```
[12.62255575],
[12.64454818],
[12.6665406 ],
[12.68853302],
[12.71052544],
[12.73251786],
[12.75451027],
[12.77650269],
[12.79849511],
[12.82048753],
[12.84247994],
[12.86447236],
[12.88646478],
[12.9084572 ],
[12.93044962],
[12.95244203],
[12.97443445],
[12.99642687],
[13.01841929],
[13.0404117 ],
[13.06240412],
[13.08439654],
[13.10638896],
[13.12838137],
[13.15037379],
[13.17236621],
[13.19435863],
[13.21635105],
[13.23834346],
[13.26033588],
[13.2823283 ],
[13.30432072],
[13.32631313],
[13.34830555],
[13.37029797],
[13.39229039],
[13.41428281],
[13.43627522],
[13.45826764],
[13.48026006],
[13.50225248],
[13.52424489],
[13.54623731],
[13.56822973],
[13.59022215],
[13.61221456],
[13.63420698],
[13.6561994 ],
[13.67819182],
[13.70018424],
[13.72217665],
[13.74416907],
[13.76616149],
[13.78815391],
[13.81014632],
[13.83213874],
[13.85413116],
[13.87612358],
[13.89811599],
[13.92010841],
[13.94210083],
[13.96409325],
[13.98608567],
[14.00807808],
[14.0300705 ],
[14.05206292],
[14.07405534],
[14.09604775],
[14.11804017],
[14.14003259],
[14.16202501],
[14.18401742],
[14.20600984],
[14.22800226],
[14.24999468],
[14.2719871 ],
[14.29397951],
[14.31597193],
[14.33796435],
[14.35995677],
[14.38194918],
[14.4039416 ],
[14.42593402],
[14.44792644],
[14.46991886],
[14.49191127],
[14.51390369],
[14.53589611],
[14.55788853],
```

```
       [14.57988094],
       [14.60187336],
       [14.62386578],
       [14.6458582 ],
       [14.66785061],
       [14.68984303],
       [14.71183545],
       [14.73382787],
       [14.75582029],
       [14.7778127 ],
       [14.79980512],
       [14.82179754],
       [14.84378996],
       [14.86578237],
       [14.88777479],
       [14.90976721],
       [14.93175963],
       [14.95375204],
       [14.97574446],
       [14.99773688],
       [15.0197293 ],
       [15.04172172],
       [15.06371413],
       [15.08570655],
       [15.10769897],
       [15.12969139],
       [15.1516838 ],
       [15.17367622],
       [15.19566864],
       [15.21766106],
       [15.23965347],
       [15.26164589],
       [15.28363831],
       [15.30563073],
       [15.32762315],
       [15.34961556],
       [15.37160798],
       [15.3936004 ],
       [15.41559282],
       [15.43758523],
       [15.45957765],
       [15.48157007],
       [15.50356249],
       [15.5255549 ],
       [15.54754732],
       [15.56953974],
       [15.59153216],
       [15.61352458],
       [15.63551699],
       [15.65750941],
       [15.67950183],
       [15.70149425],
       [15.72348666],
       [15.74547908],
       [15.7674715 ],
       [15.78946392],
       [15.81145634],
       [15.83344875],
       [15.85544117],
       [15.87743359],
       [15.89942601],
       [15.92141842],
       [15.94341084],
       [15.96540326],
       [15.98739568],
       [16.00938809],
       [16.03138051],
       [16.05337293],
       [16.07536535],
       [16.09735777],
       [16.11935018],
       [16.1413426 ],
       [16.16333502],
       [16.18532744],
       [16.20731985],
       [16.22931227],
       [16.25130469],
       [16.27329711]])
```

In [42]:

```
#RMSE Calculation

rmse = sqrt(mean_squared_error(test_array, predicted_array))
print ('RMSE = ' + str(rmse))
```

```
RMSE = 3.3752492462072485
```

In [43]:

```
# MAE Calculation

def mae(y_true, predictions):
```

```
    y_true, predictions = np.array(y_true), np.array(predictions)
    return np.mean(np.abs(y_true - predictions))

true = test_array
predicted = predicted_array

print(mae(true, predicted))
```

```
2.716608687477407
```

In [ ]:

E1 Revision

In [ ]:

In [44]:
```
# Model Standard Error calculations, computed numerical Hessian

std_error = model.bse()
print(std_error)
```

```
intercept    0.019092
ar.L1        0.039076
sigma2       0.012791
dtype: float64
```

In [45]:
```
# Generate Model confidence intervals

conf_int = model.conf_int()
```

In [46]:
```
# Generate Forecast Prediction Intervals at 90% Confidence

y_forec, conf_int = model.predict(183, return_conf_int=True, alpha=0.1)
print(conf_int)
```

```
[[ 1.15120186e+01  1.29737769e+01]
 [ 1.10201572e+01  1.35426062e+01]
 [ 1.06042529e+01  1.40158976e+01]
 [ 1.02505605e+01  1.44190207e+01]
 [ 9.94411230e+00  1.47716666e+01]
 [ 9.67322871e+00  1.50874342e+01]
 [ 9.42953411e+00  1.53754790e+01]
 [ 9.20711887e+00  1.56420275e+01]
 [ 9.00177412e+00  1.58914174e+01]
 [ 8.81044196e+00  1.61267590e+01]
 [ 8.63084764e+00  1.63503481e+01]
 [ 8.46125800e+00  1.65639266e+01]
 [ 8.30032245e+00  1.67688486e+01]
 [ 8.14696685e+00  1.69661897e+01]
 [ 8.00032138e+00  1.71568203e+01]
 [ 7.85967039e+00  1.73414562e+01]
 [ 7.72441689e+00  1.75206946e+01]
 [ 7.59405679e+00  1.76950396e+01]
 [ 7.46815985e+00  1.78649214e+01]
 [ 7.34635540e+00  1.80307106e+01]
 [ 7.22832142e+00  1.81927295e+01]
 [ 7.11377599e+00  1.83512597e+01]
 [ 7.00247068e+00  1.85065499e+01]
 [ 6.89418523e+00  1.86588202e+01]
 [ 6.78872323e+00  1.88082670e+01]
 [ 6.68590871e+00  1.89550663e+01]
 [ 6.58558323e+00  1.90993767e+01]
 [ 6.48760355e+00  1.92413412e+01]
 [ 6.39183969e+00  1.93810899e+01]
 [ 6.29817322e+00  1.95187412e+01]
 [ 6.20649590e+00  1.96544033e+01]
 [ 6.11670851e+00  1.97881756e+01]
 [ 6.02871979e+00  1.99201491e+01]
 [ 5.94244562e+00  2.00504081e+01]
 [ 5.85780823e+00  2.01790303e+01]
 [ 5.77473555e+00  2.03060879e+01]
 [ 5.69316066e+00  2.04316476e+01]
 [ 5.61302127e+00  2.05557718e+01]
 [ 5.53425929e+00  2.06785186e+01]
 [ 5.45682045e+00  2.07999423e+01]
 [ 5.38065396e+00  2.09200936e+01]
 [ 5.30571219e+00  2.10390202e+01]
 [ 5.23195040e+00  2.11567669e+01]
 [ 5.15932653e+00  2.12733756e+01]
 [ 5.08780095e+00  2.13888860e+01]
 [ 5.01733627e+00  2.15033355e+01]
 [ 4.94789719e+00  2.16167594e+01]
 [ 4.87945032e+00  2.17291911e+01]
 [ 4.81196405e+00  2.18406622e+01]
 [ 4.74540839e+00  2.19512027e+01]
 [ 4.67975490e+00  2.20608410e+01]
```

```
[ 4.61497655e+00  2.21696042e+01]
[ 4.55104765e+00  2.22775180e+01]
[ 4.48794372e+00  2.23846067e+01]
[ 4.42564145e+00  2.24908938e+01]
[ 4.36411859e+00  2.25964015e+01]
[ 4.30335393e+00  2.27011510e+01]
[ 4.24332717e+00  2.28051626e+01]
[ 4.18401893e+00  2.29084557e+01]
[ 4.12541064e+00  2.30110488e+01]
[ 4.06748454e+00  2.31129598e+01]
[ 4.01022358e+00  2.32142055e+01]
[ 3.95361144e+00  2.33148025e+01]
[ 3.89763243e+00  2.34147664e+01]
[ 3.84227149e+00  2.35141121e+01]
[ 3.78751417e+00  2.36128543e+01]
[ 3.73334653e+00  2.37110068e+01]
[ 3.67975521e+00  2.38085829e+01]
[ 3.62672730e+00  2.39055957e+01]
[ 3.57425039e+00  2.40020574e+01]
[ 3.52231253e+00  2.40979801e+01]
[ 3.47090216e+00  2.41933753e+01]
[ 3.42000816e+00  2.42882542e+01]
[ 3.36961978e+00  2.43826274e+01]
[ 3.31972664e+00  2.44765053e+01]
[ 3.27031872e+00  2.45698981e+01]
[ 3.22138631e+00  2.46628154e+01]
[ 3.17292004e+00  2.47552665e+01]
[ 3.12491085e+00  2.48472605e+01]
[ 3.07734994e+00  2.49388062e+01]
[ 3.03022882e+00  2.50299122e+01]
[ 2.98353925e+00  2.51205866e+01]
[ 2.93727324e+00  2.52108374e+01]
[ 2.89142305e+00  2.53006725e+01]
[ 2.84598118e+00  2.53900992e+01]
[ 2.80094033e+00  2.54791248e+01]
[ 2.75629344e+00  2.55677566e+01]
[ 2.71203365e+00  2.56560012e+01]
[ 2.66815428e+00  2.57438654e+01]
[ 2.62464885e+00  2.58313557e+01]
[ 2.58151108e+00  2.59184783e+01]
[ 2.53873483e+00  2.60052394e+01]
[ 2.49631415e+00  2.60916449e+01]
[ 2.45424325e+00  2.61777006e+01]
[ 2.41251649e+00  2.62634122e+01]
[ 2.37112839e+00  2.63487851e+01]
[ 2.33007360e+00  2.64338248e+01]
[ 2.28934691e+00  2.65185363e+01]
[ 2.24894327e+00  2.66029248e+01]
[ 2.20885772e+00  2.66869952e+01]
[ 2.16908546e+00  2.67707523e+01]
[ 2.12962178e+00  2.68542008e+01]
[ 2.09046211e+00  2.69373453e+01]
[ 2.05160197e+00  2.70201902e+01]
[ 2.01303702e+00  2.71027400e+01]
[ 1.97476299e+00  2.71849989e+01]
[ 1.93677574e+00  2.72669710e+01]
[ 1.89907120e+00  2.73486604e+01]
[ 1.86164542e+00  2.74300710e+01]
[ 1.82449453e+00  2.75112067e+01]
[ 1.78761474e+00  2.75920713e+01]
[ 1.75100237e+00  2.76726685e+01]
[ 1.71465380e+00  2.77530019e+01]
[ 1.67856550e+00  2.78330751e+01]
[ 1.64273402e+00  2.79128914e+01]
[ 1.60715598e+00  2.79924543e+01]
[ 1.57182809e+00  2.80717670e+01]
[ 1.53674709e+00  2.81508328e+01]
[ 1.50190984e+00  2.82296549e+01]
[ 1.46731324e+00  2.83082363e+01]
[ 1.43295425e+00  2.83865802e+01]
[ 1.39882991e+00  2.84646893e+01]
[ 1.36493730e+00  2.85425668e+01]
[ 1.33127359e+00  2.86202153e+01]
[ 1.29783598e+00  2.86976378e+01]
[ 1.26462174e+00  2.87748369e+01]
[ 1.23162819e+00  2.88518152e+01]
[ 1.19885270e+00  2.89285756e+01]
[ 1.16629271e+00  2.90051204e+01]
[ 1.13394568e+00  2.90814523e+01]
[ 1.10180914e+00  2.91575736e+01]
[ 1.06988067e+00  2.92334869e+01]
[ 1.03815788e+00  2.93091946e+01]
[ 1.00663844e+00  2.93846988e+01]
[ 9.75320052e-01  2.94600021e+01]
[ 9.44200472e-01  2.95351065e+01]
[ 9.13277490e-01  2.96100143e+01]
[ 8.82548938e-01  2.96847277e+01]
[ 8.52012691e-01  2.97592488e+01]
[ 8.21666658e-01  2.98335796e+01]
```

```
 [ 7.91508788e-01  2.99077223e+01]
 [ 7.61537065e-01  2.99816789e+01]
 [ 7.31749512e-01  3.00554513e+01]
 [ 7.02144183e-01  3.01290414e+01]
 [ 6.72719168e-01  3.02024513e+01]
 [ 6.43472589e-01  3.02756827e+01]
 [ 6.14402603e-01  3.03487375e+01]
 [ 5.85507394e-01  3.04216176e+01]
 [ 5.56785182e-01  3.04943246e+01]
 [ 5.28234213e-01  3.05668604e+01]
 [ 4.99852764e-01  3.06392267e+01]
 [ 4.71639142e-01  3.07114252e+01]
 [ 4.43591679e-01  3.07834575e+01]
 [ 4.15708737e-01  3.08553252e+01]
 [ 3.87988704e-01  3.09270301e+01]
 [ 3.60429994e-01  3.09985737e+01]
 [ 3.33031046e-01  3.10699574e+01]
 [ 3.05790325e-01  3.11411830e+01]
 [ 2.78706320e-01  3.12122518e+01]
 [ 2.51777543e-01  3.12831655e+01]
 [ 2.25002532e-01  3.13539253e+01]
 [ 1.98379845e-01  3.14245328e+01]
 [ 1.71908064e-01  3.14949894e+01]
 [ 1.45585792e-01  3.15652965e+01]
 [ 1.19411654e-01  3.16354555e+01]
 [ 9.33842956e-02  3.17054677e+01]
 [ 6.75023833e-02  3.17753345e+01]
 [ 4.17646033e-02  3.18450571e+01]
 [ 1.61696616e-02  3.19146369e+01]
 [-9.28371659e-03  3.19840751e+01]
 [-3.45967871e-02  3.20533730e+01]
 [-5.97707874e-02  3.21225318e+01]
 [-8.48069369e-02  3.21915528e+01]
 [-1.09706437e-01  3.22604371e+01]
 [-1.34470473e-01  3.23291860e+01]
 [-1.59100210e-01  3.23978006e+01]
 [-1.83596801e-01  3.24662820e+01]
 [-2.07961379e-01  3.25346314e+01]
 [-2.32195063e-01  3.26028499e+01]
 [-2.56298955e-01  3.26709387e+01]
 [-2.80274142e-01  3.27388987e+01]
 [-3.04121696e-01  3.28067311e+01]
 [-3.27842675e-01  3.28744369e+01]]
```

In [47]:
```
# Assign Predictions to pandas DataFrame

conf_pd = pd.DataFrame(conf_int, columns =['Low_Prediction','High_Prediction'])

#Assign Low predictions to variable
low_prediction = conf_pd['Low_Prediction']

#Assign High predictions to variable
high_prediction = conf_pd['High_Prediction']
```

In [48]:
```
# Read out Test and Train sets to csv file
# Open csv files in Google Sheets, Add Day Column
# Dates align with 'test' variable, which contains actual revenue figures

low_prediction.to_csv('C:/Users/ericy/Desktop/Low_Prediction.csv')
high_prediction.to_csv('C:/Users/ericy/Desktop/High_Prediction.csv')
```

In [49]:
```
#Load predictions, date column added

low_pred = pd.read_csv('C:/Users/ericy/Desktop/Low_Prediction_dt.csv')
high_pred = pd.read_csv('C:/Users/ericy/Desktop/High_Prediction_dt.csv')
```

In [50]:
```
# Variable exploration to ensure compatability with 'test' datetime timeframe
low_pred.head()
```

Out[50]:

	Day	Revenue
0	2020-07-02 00:00:00	11.372002
1	2020-07-03 00:00:00	10.778540
2	2020-07-04 00:00:00	10.277463
3	2020-07-05 00:00:00	9.851277
4	2020-07-06 00:00:00	9.481697

In [51]:
```
# Variable exploration to ensure compatability with 'test' datetime timeframe
high_pred.head()
```

Out[51]:

	Day	Revenue
0	2020-07-02 00:00:00	13.113794
1	2020-07-03 00:00:00	13.784223
2	2020-07-04 00:00:00	14.342688
3	2020-07-05 00:00:00	14.818304
4	2020-07-06 00:00:00	15.234082

Convert Low and High Prediction 'Day' column to datetime and index

In [52]:

```
# Lower Predictions, Day to datetime
low_pred['Day'] = pd.to_datetime(low_pred['Day'])
```

In [53]:

```
# Lower Predictions, Set Day as Index
low_pred.set_index('Day',inplace=True)
```

In [54]:

```
# High Predictions, Day to datetime
high_pred['Day'] = pd.to_datetime(high_pred['Day'])
```

In [55]:

```
# High Predictions, Set Day as Index
high_pred.set_index('Day',inplace=True)
```

In [56]:

```
low_pred.head()
```

Out[56]:

Day	Revenue
2020-07-02	11.372002
2020-07-03	10.778540
2020-07-04	10.277463
2020-07-05	9.851277
2020-07-06	9.481697

In [57]:

```
high_pred.head()
```

Out[57]:

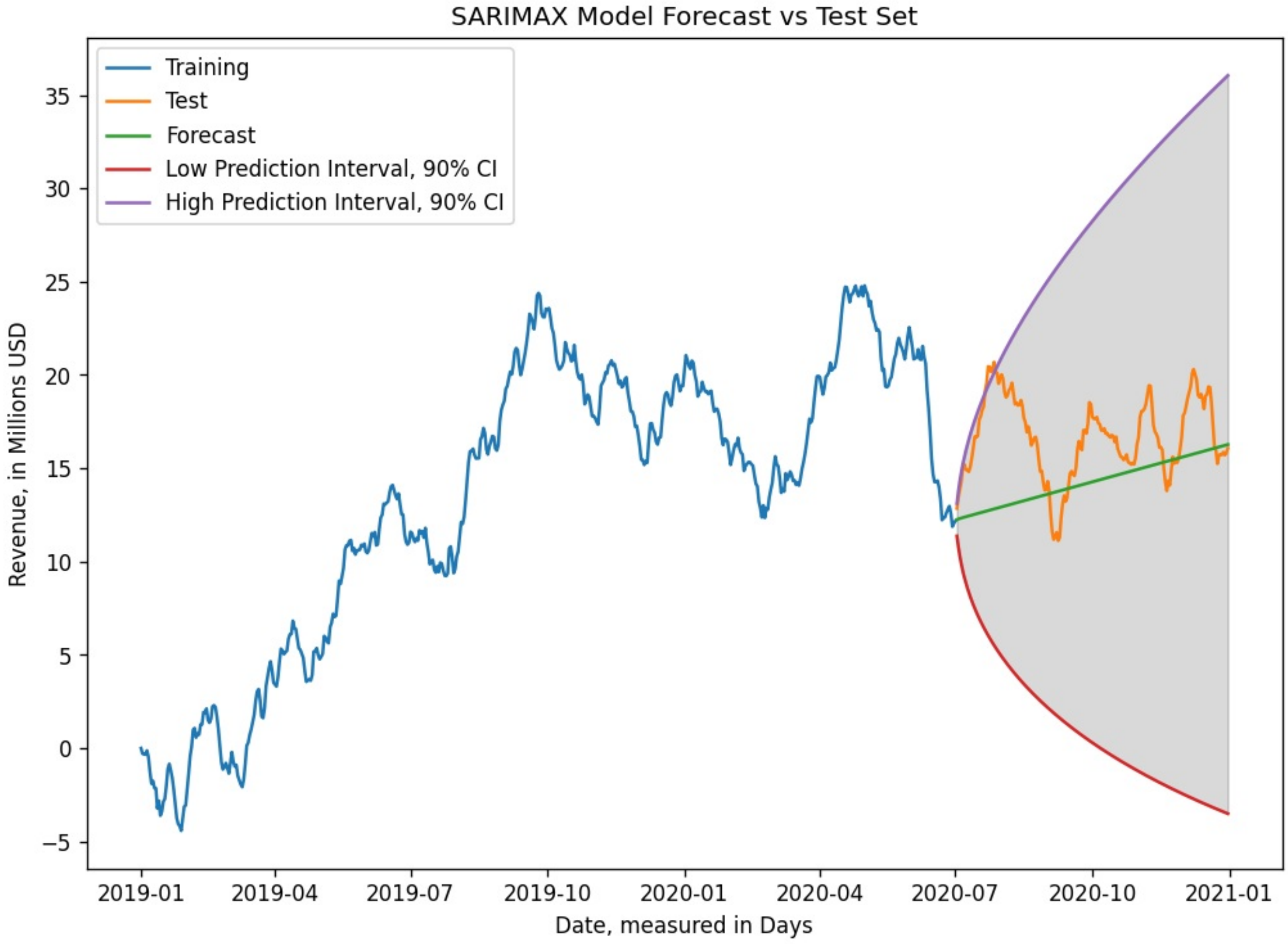
Day	Revenue
2020-07-02	13.113794
2020-07-03	13.784223
2020-07-04	14.342688
2020-07-05	14.818304
2020-07-06	15.234082

SARIMAX Model Forecast, With Prediction Interval (CI 90%), Vs Test Set

In [58]:

```
# Prediction assignment, predicted revenue column named
# Training, Test, and Predicted data plotted together
# Code Reference (Matplotlib.pyplot.plot — Matplotlib 3.6.0 documentation, n.d.)

# Creating varible with forecast values
forecast = pd.DataFrame(model.predict(n_periods = 183),index=test.index)

# Naming forecast_revenue column in forecast variable
forecast.columns = ['forecast_revenue']

# Establish plot parameters for Forecast

# Plot figure size
plt.figure(figsize=(10,7))

# Training data
plt.plot(train,label="Training")

# Annotate X-axis label
plt.xlabel('Date, measured in Days')

# Annotate Y-axis label
plt.ylabel('Revenue, in Millions USD')
```

```python
# Annotate Plot Title
plt.title('SARIMAX Model Forecast vs Test Set')

# Plot Test Data
plt.plot(test,label="Test")

# Plot Forecast Data
plt.plot(forecast,label="Forecast")

# Add Prediction Interval at 95% CI
plt.plot(low_pred,label='Low Prediction Interval, 90% CI')
plt.plot(high_pred,label='High Prediction Interval, 90% CI')
plt.fill_between(low_pred.index, low_pred['Revenue'], high_pred['Revenue'], color='k', alpha=.15)

# Plot legend in upper lefthand corner
plt.legend(loc = 'upper left')


# Show Plot
plt.show()
```



In [ ]:

Loading [MathJax]/jax/output/CommonHTML/fonts/TeX/fontdata.js