D213 Task 1: Time Series Modeling

In [2]:
```
#Install Libraries and Packages

import pandas as pd
import matplotlib.pyplot as plt
import numpy as np
```

In [3]:
```
# Load dataset
telco_df = pd.read_csv('teleco_time_series .csv')
```

In [4]:
```
#Check data
telco_df.head()
```

Out[4]:

	Day	Revenue
0	1	0.000000
1	2	0.000793
2	3	0.825542
3	4	0.320332
4	5	1.082554

Add date column to the dataset and set as index

In [5]:
```
telco_df.set_index('Day', inplace=True)
telco_df.index=pd.to_datetime(telco_df.index, unit = 'D', origin = '20
telco_df.head()
```

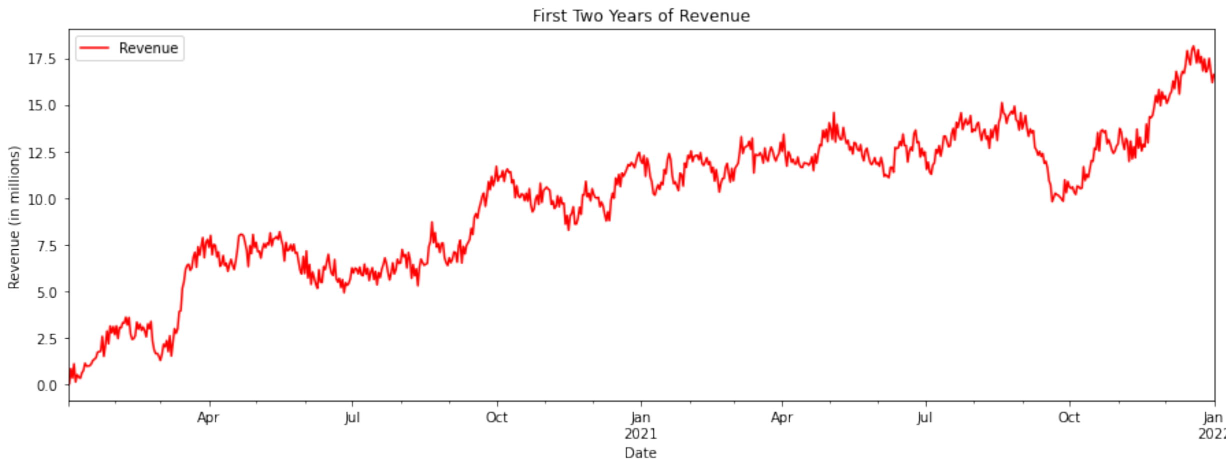
Out[5]:

	Revenue
Day	
2020-01-02	0.000000
2020-01-03	0.000793
2020-01-04	0.825542
2020-01-05	0.320332
2020-01-06	1.082554

Data Preparation

Provide a line graph visualizing the realization of the time series

In [6]:
```
#Visualize the dataset
telco_df.plot(figsize=(15,5), color = 'red')
plt.title('First Two Years of Revenue')
plt.xlabel('Date')
plt.ylabel('Revenue (in millions)')
plt.show()
```



Describe the time step formatting of the realization

In [7]:
```
# Check basic structure of the dataset
telco_df.info
```

Out[7]:
```
<bound method DataFrame.info of               Revenue
Day
2020-01-02   0.000000
2020-01-03   0.000793
2020-01-04   0.825542
2020-01-05   0.320332
2020-01-06   1.082554
...               ...
2021-12-28  16.931559
2021-12-29  17.490666
2021-12-30  16.803638
2021-12-31  16.194814
2022-01-01  16.620798

[731 rows x 1 columns]>
```

In [8]:
```python
# Check datatypes
telco_df.dtypes
```

Out[8]:
```
Revenue    float64
dtype: object
```

In [9]:
```python
# Check for missing data
telco_df.isnull().sum()
```

Out[9]:
```
Revenue    0
dtype: int64
```

In [10]:
```python
# Check for NAs
telco_df.isna().sum()
```

Out[10]:
```
Revenue    0
dtype: int64
```

In [11]:
```python
# Check for duplicates
telco_df.duplicated().sum()
```

Out[11]:
```
0
```

In [12]:
```python
# Check descriptive statistics for dataset
telco_df.describe()
```

Out[12]:

	Revenue
count	731.000000
mean	9.822901
std	3.852645
min	0.000000
25%	6.872836
50%	10.785571
75%	12.566911
max	18.154769

Evaluate the stationarity of the time series

In [13]:
```
# Apply augmented Dickey-Fuller test
from statsmodels.tsa.stattools import adfuller
test = adfuller(telco_df['Revenue'], autolag = 'AIC')
print(test)
```

```
(-1.92461215731018, 0.32057281507939817, 1, 729, {'1%': -3.4393520240
470554, '5%': -2.8655128165959236, '10%': -2.5688855736949163}, 965.0
609576707513)
```

Make the time series stationary

In [14]:
```
# Make stationary by taking the differance and drop na
stat_df = telco_df.diff().dropna()
stat_df.head()
```

Out[14]:

	Revenue
Day	
2020-01-03	0.000793
2020-01-04	0.824749
2020-01-05	-0.505210
2020-01-06	0.762222
2020-01-07	-0.974900

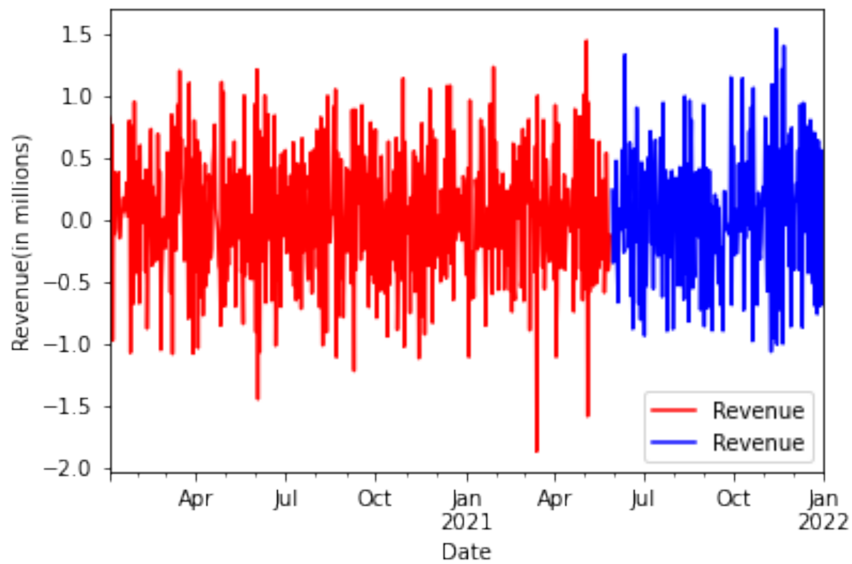
In [15]:
```
#Check if stationary
from statsmodels.tsa.stattools import adfuller
test = adfuller(stat_df['Revenue'], autolag = 'AIC')
print(test)
```

```
(-44.874527193875984, 0.0, 0, 729, {'1%': -3.4393520240470554, '5%':
-2.8655128165959236, '10%': -2.5688855736949163}, 965.5032159185916)
```

Create train - test split using 70/30 model

In [16]:
```
telco_train = stat_df.iloc[:512]
telco_test = stat_df.iloc[513:]
```

In [17]:
```
# plot train and test sets
fig, ax = plt.subplots()
telco_train.plot(ax = ax, color = 'red')
telco_test.plot(ax = ax, color = 'blue')
plt.xlabel('Date')
plt.ylabel('Revenue(in millions)')
plt.show()
```



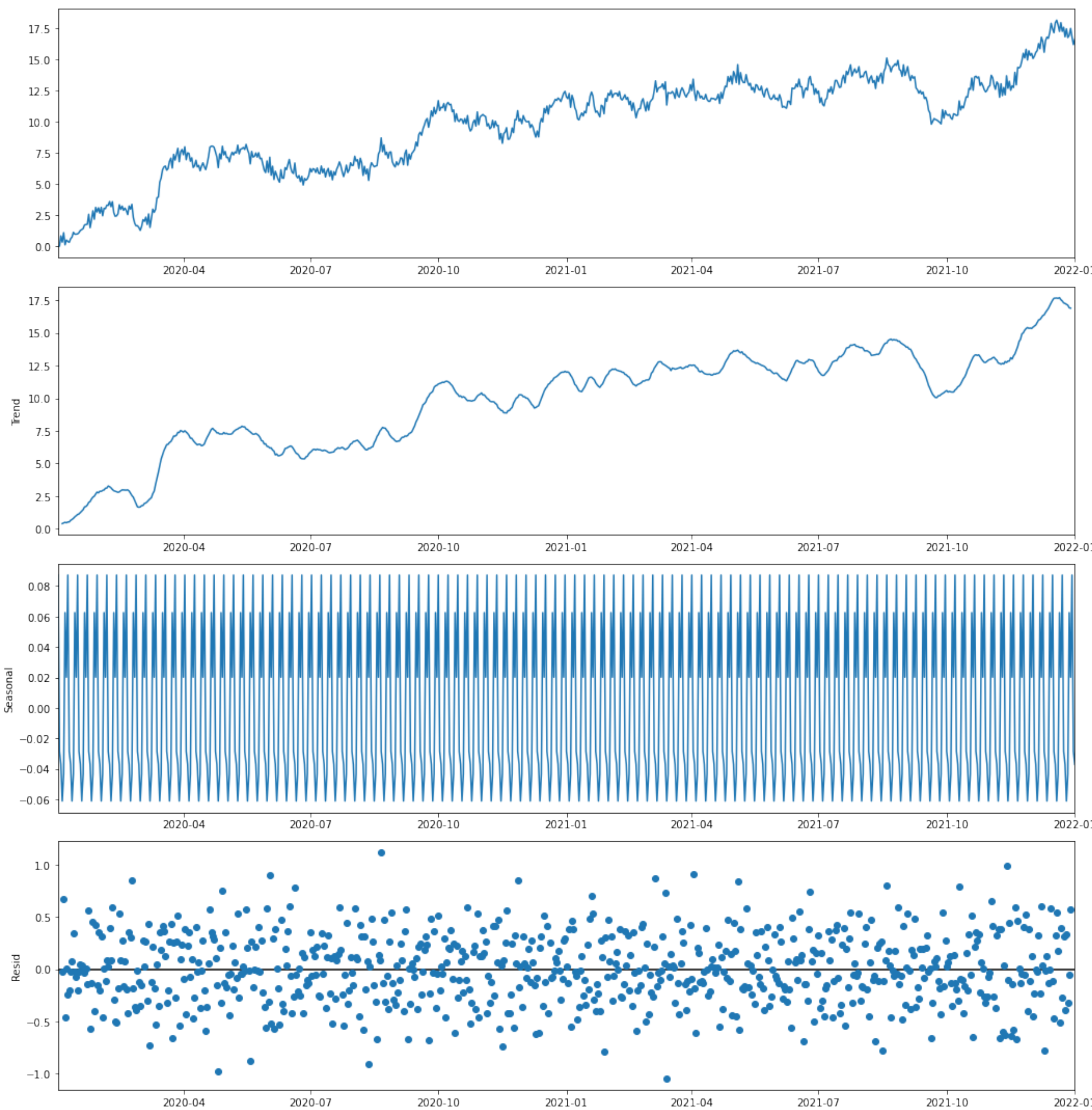
Export copy of prepared data

In [18]:
```
stat_df.to_csv('time_series_prepared_data.csv')
```

Model Identification and Analysis

Check for seasonality

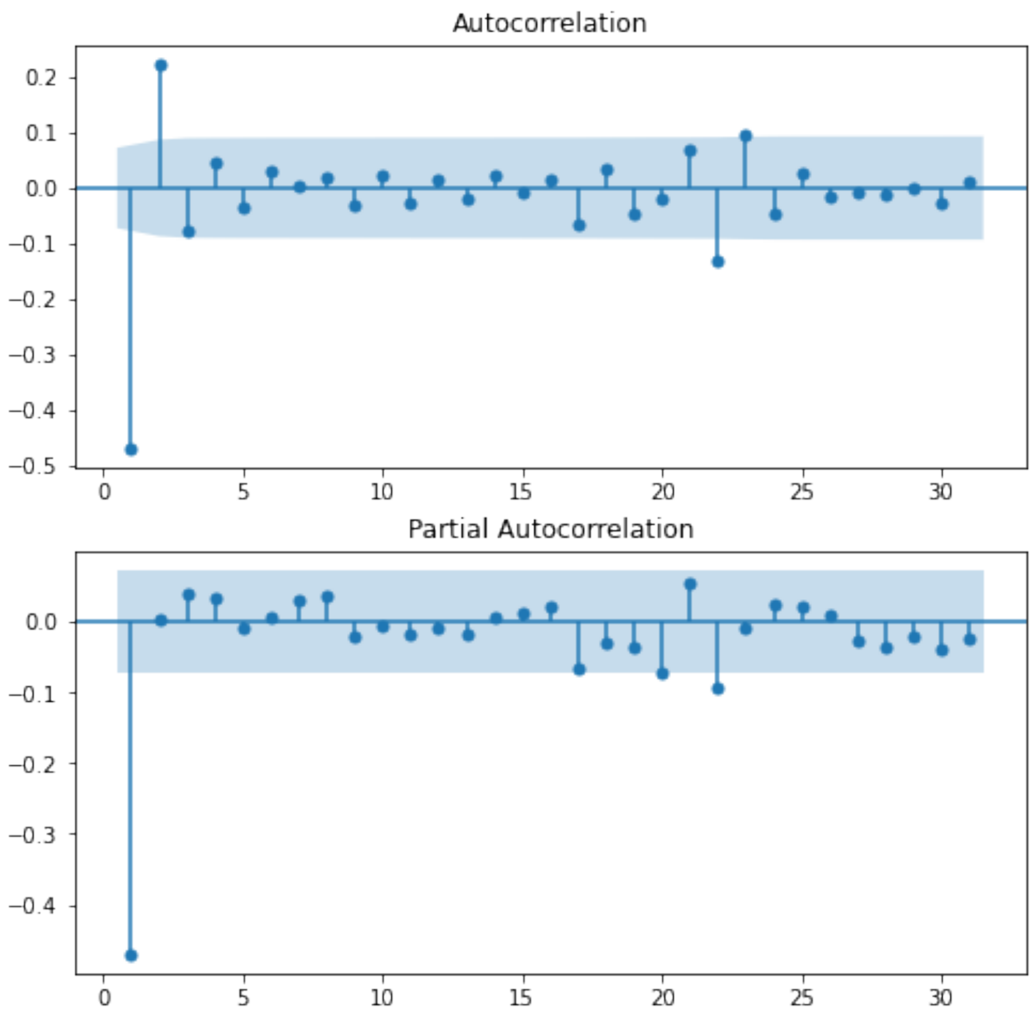
In [19]:
```
from statsmodels.tsa.seasonal import seasonal_decompose
plt.rcParams['figure.figsize'] = 15, 15
decomp_results = seasonal_decompose(telco_df, model = 'additive')
decomp_results.plot()
plt.show()
```



Plot Autocorrelation and Partical Autocorrelation

In [20]:
```
from statsmodels.graphics.tsaplots import plot_acf, plot_pacf

fig, (ax1, ax2) = plt.subplots(2, 1, figsize = (8,8))
plot_acf(stat_df, lags = 31, zero = False, ax = ax1)
plot_pacf(stat_df, lags = 31, zero = False, ax = ax2)
plt.show()
```



Spactral Density

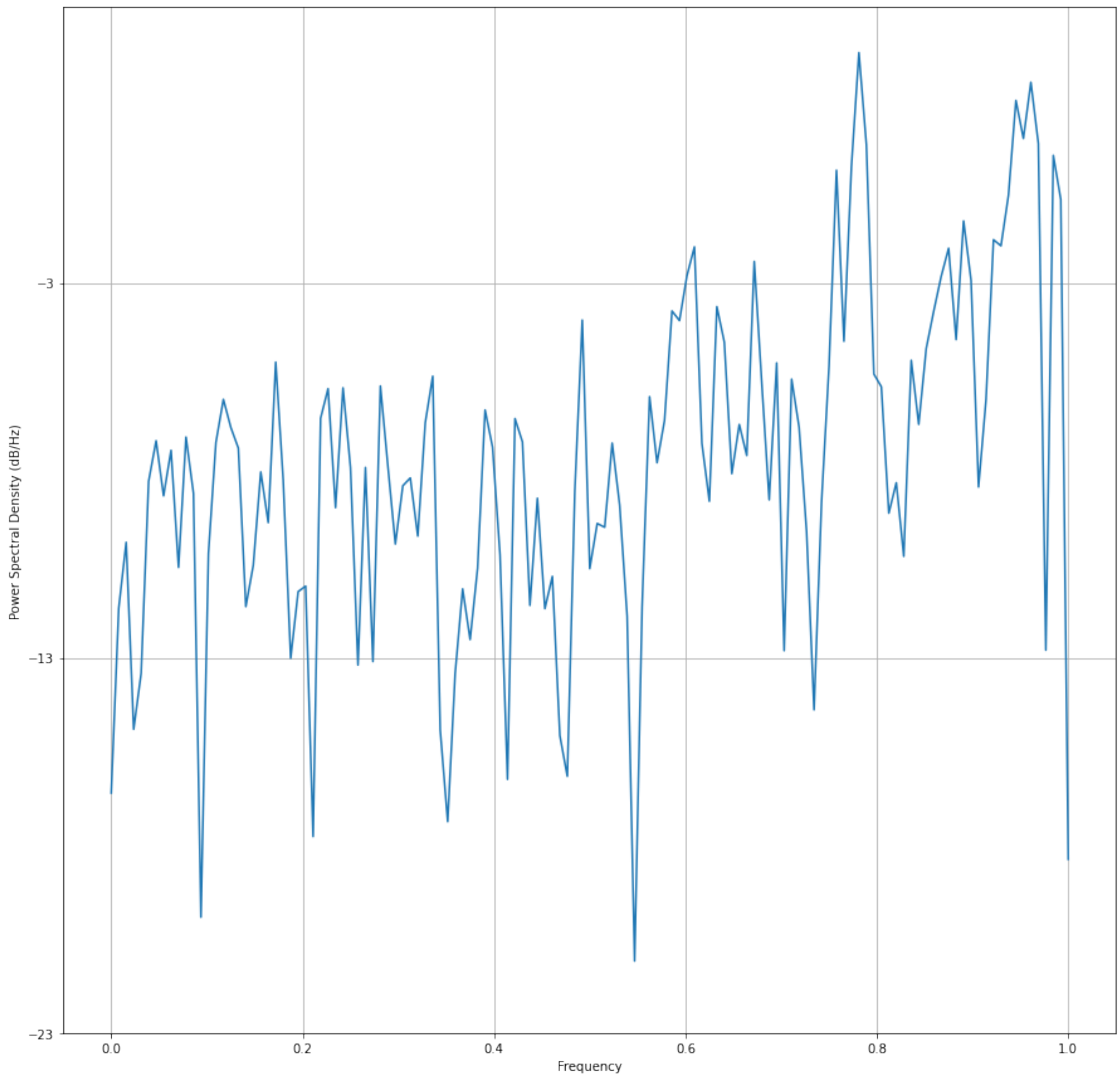
In [21]:
```
plt.psd(stat_df['Revenue'])
```

Out[21]:
```
(array([0.02187272, 0.06794442, 0.10193972, 0.03234469, 0.04559398,
        0.14815969, 0.19015645, 0.13556855, 0.17930696, 0.08735005,
        0.19447871, 0.13755823, 0.01020268, 0.09512023, 0.18805592,
        0.24517453, 0.20614533, 0.18171205, 0.06867317, 0.08851738,
        0.15697924, 0.11499682, 0.30811184, 0.14944065, 0.05001068,
        0.07531453, 0.07789058, 0.01673615, 0.21814643, 0.26177285,
        0.12608986, 0.26308319, 0.16121103, 0.04795751, 0.1613957 ,
        0.04902511, 0.26613974, 0.16274877, 0.10082987, 0.14421741,
        0.15131872, 0.10587624, 0.21235552, 0.28259121, 0.03221009,
        0.01834018, 0.04559876, 0.07659406, 0.05610271, 0.08696616,
        0.22968462, 0.1827402 , 0.09436169, 0.0237871 , 0.21783212,
        0.18875707, 0.06917957, 0.13361489, 0.06783029, 0.08269334,
```

```
        0.03112555, 0.02422832, 0.13926167, 0.39890482, 0.08670748,

        0.11441993, 0.11177872, 0.18740897, 0.12808631, 0.06475304,
        0.0077952 , 0.06857082, 0.24922119, 0.16611852, 0.21520259,
        0.42182914, 0.39754655, 0.52374424, 0.62512939, 0.18726519,
        0.13111332, 0.43322009, 0.34818613, 0.1553137 , 0.21013366,
        0.17361039, 0.57161035, 0.27249034, 0.13228186, 0.30658681,
        0.05239826, 0.27755592, 0.20651373, 0.10902572, 0.03644409,
        0.13032478, 0.30017193, 1.00082731, 0.35010851, 1.02384551,
        2.06051044, 1.16975022, 0.28636722, 0.26486441, 0.12188377,
        0.14693411, 0.09351289, 0.31160769, 0.21024373, 0.3346019 ,
        0.42069878, 0.52108706, 0.61996018, 0.35376199, 0.73309531,
        0.50884317, 0.1431019 , 0.24463468, 0.65323034, 0.62892871,
        0.85769067, 1.5358352 , 1.2161517 , 1.716833  , 1.1779039 ,
        0.05256369, 1.09660037, 0.83633103, 0.01455748]),
 array([0.       , 0.0078125, 0.015625 , 0.0234375, 0.03125  , 0.0390
625,
        0.046875 , 0.0546875, 0.0625   , 0.0703125, 0.078125 , 0.0859
375,
        0.09375  , 0.1015625, 0.109375 , 0.1171875, 0.125    , 0.1328
125,
        0.140625 , 0.1484375, 0.15625  , 0.1640625, 0.171875 , 0.1796
875,
        0.1875   , 0.1953125, 0.203125 , 0.2109375, 0.21875  , 0.2265
625,
        0.234375 , 0.2421875, 0.25     , 0.2578125, 0.265625 , 0.2734
375,
        0.28125  , 0.2890625, 0.296875 , 0.3046875, 0.3125   , 0.3203
125,
        0.328125 , 0.3359375, 0.34375  , 0.3515625, 0.359375 , 0.3671
875,
        0.375    , 0.3828125, 0.390625 , 0.3984375, 0.40625  , 0.4140
625,
        0.421875 , 0.4296875, 0.4375   , 0.4453125, 0.453125 , 0.4609
375,
        0.46875  , 0.4765625, 0.484375 , 0.4921875, 0.5      , 0.5078
125,
        0.515625 , 0.5234375, 0.53125  , 0.5390625, 0.546875 , 0.5546
875,
        0.5625   , 0.5703125, 0.578125 , 0.5859375, 0.59375  , 0.6015
625,
        0.609375 , 0.6171875, 0.625    , 0.6328125, 0.640625 , 0.6484
375,
        0.65625  , 0.6640625, 0.671875 , 0.6796875, 0.6875   , 0.6953
125,
        0.703125 , 0.7109375, 0.71875  , 0.7265625, 0.734375 , 0.7421
875,
        0.75     , 0.7578125, 0.765625 , 0.7734375, 0.78125  , 0.7890
625,
        0.796875 , 0.8046875, 0.8125   , 0.8203125, 0.828125 , 0.8359
375,
        0.84375  , 0.8515625, 0.859375 , 0.8671875, 0.875    , 0.8828
125,
        0.890625 , 0.8984375, 0.90625  , 0.9140625, 0.921875 , 0.9296
```

```
875,
         0.9375    , 0.9453125, 0.953125 , 0.9609375, 0.96875   , 0.9765
625,
         0.984375 , 0.9921875, 1.        ]))
```



Identify the model paramaters

In [22]:
```python
#! pip install pmdarima
from pmdarima import auto_arima
```

In [23]:
```python
 # Fit auto_arima to dataset
auto_arima_fit = auto_arima(telco_df['Revenue'], start_p=1,
                           start_q=1,max_p=3, max_q=3, m=12, seasonal=
                            d=0,D=1,trace=True, error_action='ignore',
                            suppress_warnings=True, stepwise=True)
auto_arima_fit.summary()
```

```
Performing stepwise search to minimize aic
```

```
Performing stepwise search to minimize aic
 ARIMA(1,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=2.72 sec
 ARIMA(0,0,0)(0,1,0)[12] intercept   : AIC=2367.159, Time=0.03 sec
 ARIMA(1,0,0)(1,1,0)[12] intercept   : AIC=1419.537, Time=0.41 sec
 ARIMA(0,0,1)(0,1,1)[12] intercept   : AIC=1969.738, Time=0.30 sec
 ARIMA(0,0,0)(0,1,0)[12]             : AIC=2399.547, Time=0.03 sec
 ARIMA(1,0,0)(0,1,0)[12] intercept   : AIC=1568.311, Time=0.12 sec

 ARIMA(1,0,0)(2,1,0)[12] intercept   : AIC=1320.755, Time=0.93 sec
 ARIMA(1,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=4.34 sec
 ARIMA(1,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=1.87 sec
 ARIMA(0,0,0)(2,1,0)[12] intercept   : AIC=2339.965, Time=0.56 sec
 ARIMA(2,0,0)(2,1,0)[12] intercept   : AIC=1147.041, Time=1.79 sec
 ARIMA(2,0,0)(1,1,0)[12] intercept   : AIC=1256.245, Time=0.78 sec
 ARIMA(2,0,0)(2,1,1)[12] intercept   : AIC=inf, Time=4.23 sec
 ARIMA(2,0,0)(1,1,1)[12] intercept   : AIC=inf, Time=1.88 sec
 ARIMA(3,0,0)(2,1,0)[12] intercept   : AIC=1148.348, Time=2.21 sec
 ARIMA(2,0,1)(2,1,0)[12] intercept   : AIC=1148.544, Time=2.20 sec
 ARIMA(1,0,1)(2,1,0)[12] intercept   : AIC=1195.703, Time=1.76 sec
 ARIMA(3,0,1)(2,1,0)[12] intercept   : AIC=1132.930, Time=5.46 sec
 ARIMA(3,0,1)(1,1,0)[12] intercept   : AIC=1235.930, Time=3.31 sec
 ARIMA(3,0,1)(2,1,1)[12] intercept   : AIC=inf, Time=7.56 sec
 ARIMA(3,0,1)(1,1,1)[12] intercept   : AIC=inf, Time=2.84 sec
 ARIMA(3,0,2)(2,1,0)[12] intercept   : AIC=1132.774, Time=5.37 sec
 ARIMA(3,0,2)(1,1,0)[12] intercept   : AIC=1236.060, Time=2.81 sec
 ARIMA(3,0,2)(2,1,1)[12] intercept   : AIC=inf, Time=6.14 sec
 ARIMA(3,0,2)(1,1,1)[12] intercept   : AIC=inf, Time=3.13 sec
 ARIMA(2,0,2)(2,1,0)[12] intercept   : AIC=1142.858, Time=2.54 sec
 ARIMA(3,0,3)(2,1,0)[12] intercept   : AIC=1138.471, Time=6.76 sec
 ARIMA(2,0,3)(2,1,0)[12] intercept   : AIC=1138.255, Time=4.92 sec
 ARIMA(3,0,2)(2,1,0)[12]             : AIC=1135.369, Time=1.92 sec

Best model:  ARIMA(3,0,2)(2,1,0)[12] intercept
Total fit time: 78.959 seconds
```

Out[23]:

SARIMAX Results

Dep. Variable:	y	**No. Observations:**	731
Model:	SARIMAX(3, 0, 2)x(2, 1, [], 12)	**Log Likelihood**	-557.387
Date:	Tue, 08 Feb 2022	**AIC**	1132.774
Time:	11:09:25	**BIC**	1173.974
Sample:	0 - 731	**HQIC**	1148.680
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
intercept	0.0061	0.003	1.760	0.078	-0.001	0.013
ar.L1	1.4416	0.076	18.875	0.000	1.292	1.591
ar.L2	-0.0437	0.128	-0.341	0.733	-0.295	0.208

ar.L3	-0.4090	0.072	-5.681	0.000	-0.550	-0.268
ma.L1	-0.9710	0.085	-11.419	0.000	-1.138	-0.804
ma.L2	0.1044	0.079	1.316	0.188	-0.051	0.260
ar.S.L12	-0.7059	0.038	-18.357	0.000	-0.781	-0.631
ar.S.L24	-0.3798	0.039	-9.629	0.000	-0.457	-0.303
sigma2	0.2728	0.015	17.605	0.000	0.242	0.303

Ljung-Box (L1) (Q):	0.00	**Jarque-Bera (JB):**	1.88
Prob(Q):	0.95	**Prob(JB):**	0.39
Heteroskedasticity (H):	1.06	**Skew:**	0.01
Prob(H) (two-sided):	0.65	**Kurtosis:**	2.75

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

Fit the model

In [24]:
```
from statsmodels.tsa.statespace.sarimax import SARIMAX

#Create train and test set
train = telco_df.iloc[:512]
test = telco_df.iloc[512:]

#create and fit model
model =SARIMAX(train, order=(3, 0, 2),seasonal_order=(2, 1, 0, 12),
               enforce_stationarity=False, enforce_invertibility=False
results = model.fit()
```

```
/opt/anaconda3/envs/D213/lib/python3.6/site-packages/statsmodels/tsa/
base/tsa_model.py:527: ValueWarning: No frequency information was pro
vided, so inferred frequency D will be used.
  % freq, ValueWarning)
/opt/anaconda3/envs/D213/lib/python3.6/site-packages/statsmodels/tsa/
base/tsa_model.py:527: ValueWarning: No frequency information was pro
vided, so inferred frequency D will be used.
  % freq, ValueWarning)
/opt/anaconda3/envs/D213/lib/python3.6/site-packages/statsmodels/base
/model.py:568: ConvergenceWarning: Maximum Likelihood optimization fa
iled to converge. Check mle_retvals
  ConvergenceWarning)
```

In [25]:
```
#Print summary of the results
results.summary()
```

Out[25]:

SARIMAX Results

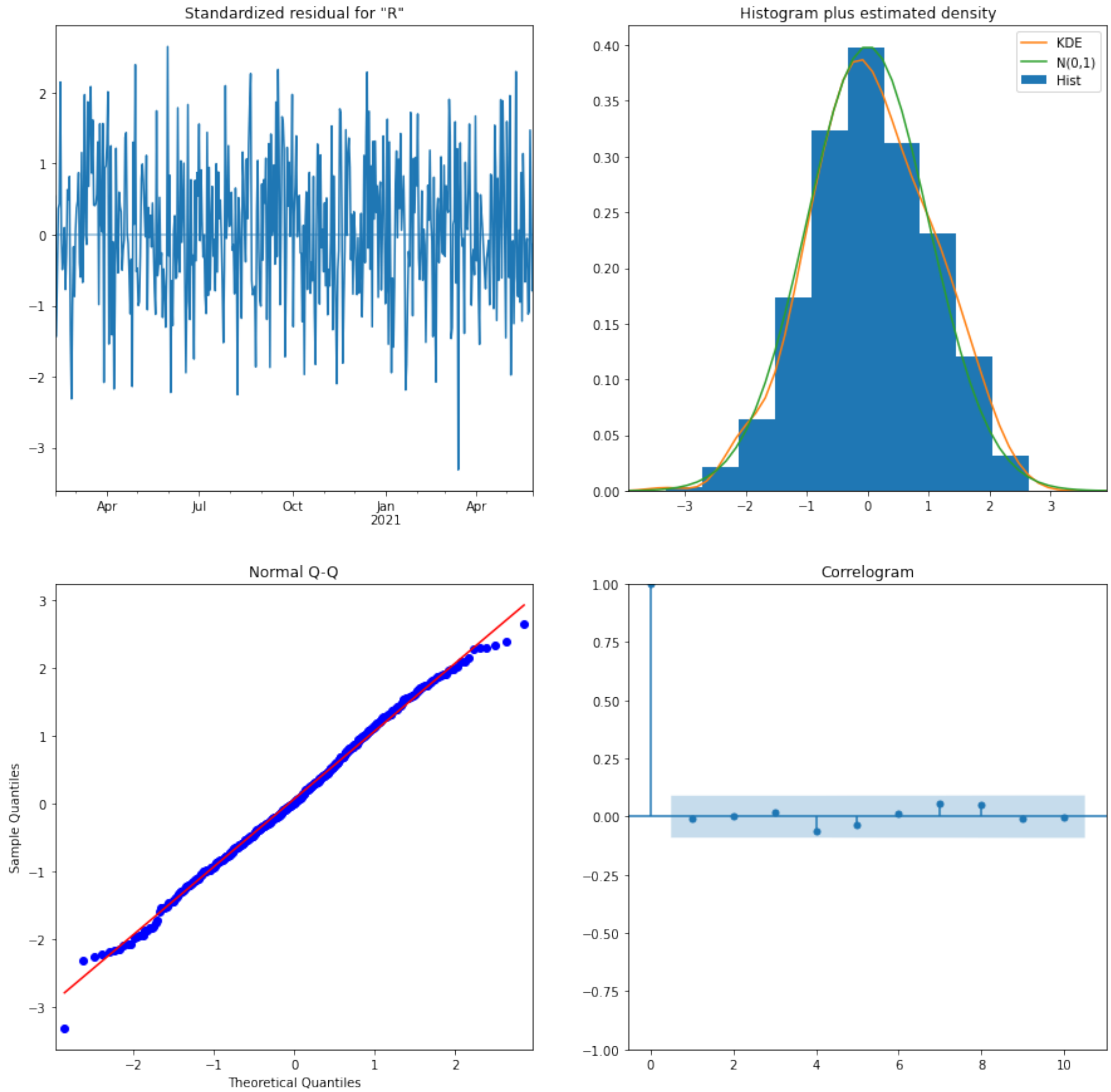
Dep. Variable:	Revenue	**No. Observations:**	512
Model:	SARIMAX(3, 0, 2)x(2, 1, [], 12)	**Log Likelihood**	-364.295
Date:	Tue, 08 Feb 2022	**AIC**	744.589
Time:	11:09:27	**BIC**	777.862
Sample:	01-02-2020	**HQIC**	757.676
	- 05-27-2021		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	1.3543	0.136	9.922	0.000	1.087	1.622
ar.L2	-0.0399	0.157	-0.255	0.799	-0.347	0.267
ar.L3	-0.3323	0.111	-2.998	0.003	-0.550	-0.115
ma.L1	-0.8597	0.140	-6.124	0.000	-1.135	-0.585
ma.L2	0.1297	0.105	1.235	0.217	-0.076	0.336
ar.S.L12	-0.7545	0.048	-15.698	0.000	-0.849	-0.660
ar.S.L24	-0.4066	0.048	-8.469	0.000	-0.501	-0.313
sigma2	0.2728	0.019	14.191	0.000	0.235	0.311

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	1.18
Prob(Q):	0.88	**Prob(JB):**	0.55
Heteroskedasticity (H):	0.99	**Skew:**	-0.03
Prob(H) (two-sided):	0.95	**Kurtosis:**	2.76

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

In [26]:
```
#Create diagnostics plot
results.plot_diagnostics()
plt.show()
```

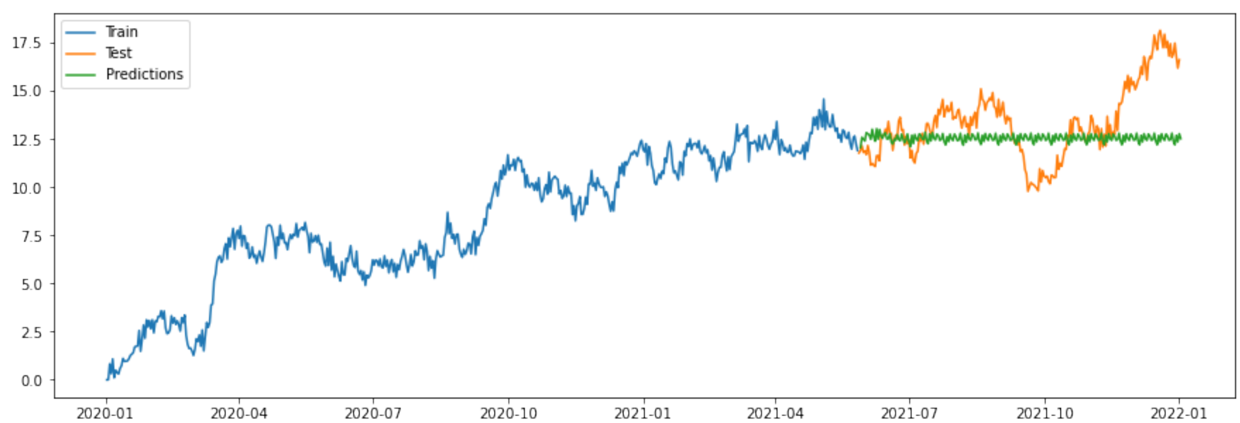


In [27]:
```
#Find the confidence interval for 0.05
#https://machinelearningmastery.com/time-series-forecast-uncertainty-u
result = results.get_forecast()
forecast = result.predicted_mean
test_1 = test['Revenue'].values.astype('float32')
print('Expected: %.3f' % forecast)
print('Forecast: %.3f' % test['Revenue'][0])
print('Standard Error: %.3f' % result.se_mean)
ci = result.conf_int(0.05)
print(ci)
```

```
Expected: 12.633
Forecast: 11.784
Standard Error: 0.522
            lower Revenue  upper Revenue
2021-05-28      11.609233      13.656771
```

Forcasting on the ARIMA model

In [28]:
```
#Plot the train, test, and prediction sets on the same graph
pred = results.predict(start = 513, end = 731, dynamic = False)
plt.figure(figsize = (15,5))
plt.plot(train['Revenue'], label = 'Train')
plt.plot(test['Revenue'], label = "Test")
plt.plot(pred, label = 'Predictions')
plt.legend(loc = 'best')
plt.show()
```



Evaluation and error metrics

In [29]:
```
#Import calculation packages
from sklearn.metrics import mean_squared_error
import math

#Find Mean Squares Error
MSE = mean_squared_error(test, pred)
print('MSE:', round(MSE, 2))
```

```
MSE: 4.28
```

In [30]:
```
#Find Root Mean Squared Error

RMSE = math.sqrt(MSE)
print('RMSE:', round(RMSE, 2))
```

```
RMSE: 2.07
```

In [31]:
```
#Find the Mean Forecast Error
#https://machinelearningmastery.com/time-series-forecasting-performanc

forecast_errors = [test['Revenue'][i]-pred[i] for i in range(len(test)
mean_forecast_error = sum(forecast_errors)*1.0/len(test)
print('Mean Forecast Error:' , round(mean_forecast_error,2))
```

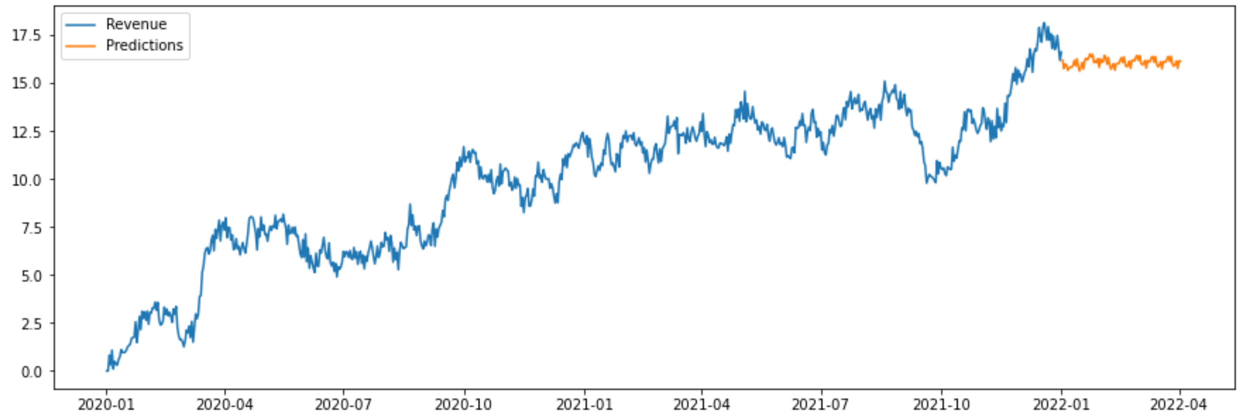
```
Mean Forecast Error: 0.79
```

Train model on full dataset

In [32]:
```
#Train model with full dataset
full_model =SARIMAX(telco_df, order=(3, 0, 2),seasonal_order=(2, 1, 0,
               enforce_stationarity=False, enforce_invertibility=False
full_results = full_model.fit()
```

```
/opt/anaconda3/envs/D213/lib/python3.6/site-packages/statsmodels/tsa/
base/tsa_model.py:527: ValueWarning: No frequency information was pro
vided, so inferred frequency D will be used.
  % freq, ValueWarning)
/opt/anaconda3/envs/D213/lib/python3.6/site-packages/statsmodels/tsa/
base/tsa_model.py:527: ValueWarning: No frequency information was pro
vided, so inferred frequency D will be used.
  % freq, ValueWarning)
```

In [33]:
```
# Forecast revenue for the next quarter (3 months)
full_pred = full_results.predict(start = 731, end = 821, dynamic = Fal
plt.figure(figsize = (15,5))
plt.plot(telco_df['Revenue'], label = 'Revenue')
plt.plot(full_pred, label = 'Predictions')
plt.legend(loc = 'best')
plt.show()
```



In [ ]: