In [34]:
```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from statsmodels.tsa.stattools import adfuller
from statsmodels.tsa.seasonal import seasonal_decompose
from statsmodels.tsa.arima_model import ARIMA
from pandas.plotting import register_matplotlib_converters
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf
from statsmodels.tsa.stattools import acf, pacf
import statsmodels.api as sm
```

In [9]:
```
import warnings
warnings.filterwarnings("ignore")
```

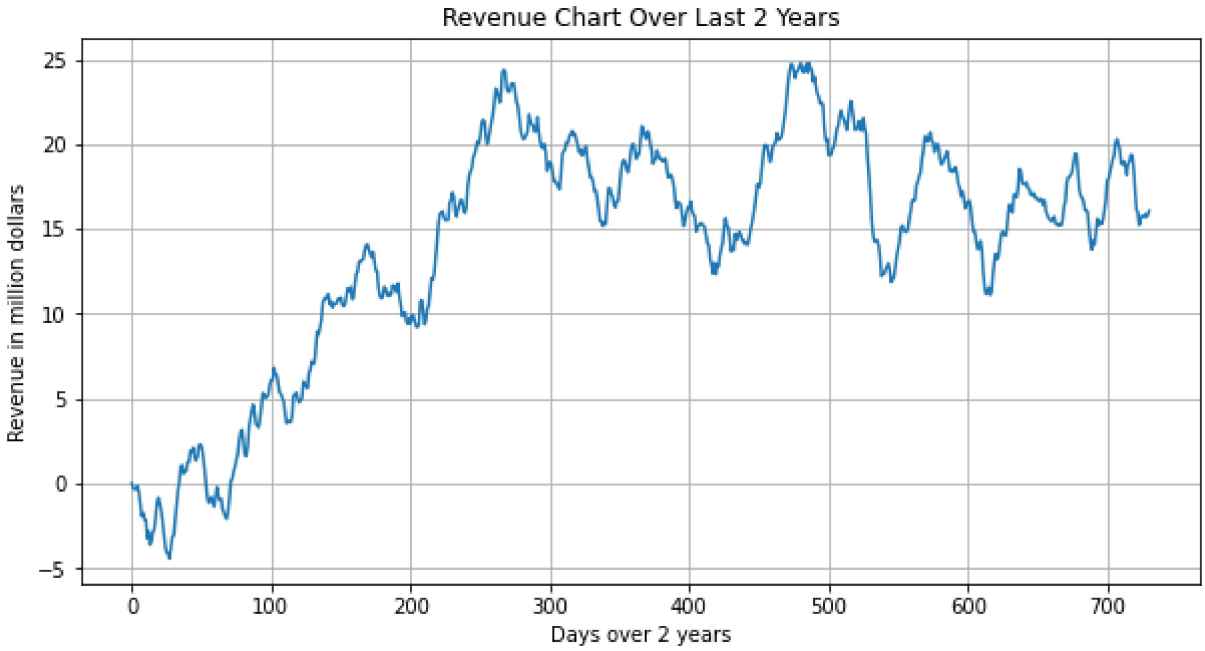
In [10]:
```
#import initial data
df = pd.read_csv('C:/Users/blien/Documents/WGU/D213/Task 1/medical_time_series.cs
```

In [11]:
```
df.head(10)
```

Out[11]:

	Day	Revenue
0	1	0.000000
1	2	-0.292356
2	3	-0.327772
3	4	-0.339987
4	5	-0.124888
5	6	-0.491590
6	7	-1.255250
7	8	-1.896279
8	9	-1.747259
9	10	-2.129558

In [12]:
```
#plot to visualize the time series
plt.figure(figsize = (10,5)) #set figure size
plt.plot(df.Revenue) #plot revenue data
plt.title('Revenue Chart Over Last 2 Years') #Chart Title
plt.xlabel('Days over 2 years') #label x axis
plt.ylabel('Revenue in million dollars') #label y axis
plt.grid(True) #add gridlines
plt.show() #plot
```



In [13]:
```
#checking for nulls
df.isnull().any()
```

Out[13]:
```
Day        False
Revenue    False
dtype: bool
```

In [14]:
```
df['Date'] = (pd.date_range(start = '1/1/2019', periods = df.shape[0], freq = '24
#set date as an index
df.set_index('Date', inplace = True)
df
```

Out[14]:

Date	Day	Revenue
2019-01-01	1	0.000000
2019-01-02	2	-0.292356
2019-01-03	3	-0.327772
2019-01-04	4	-0.339987
2019-01-05	5	-0.124888
...	...	...
2020-12-27	727	15.722056
2020-12-28	728	15.865822
2020-12-29	729	15.708988
2020-12-30	730	15.822867
2020-12-31	731	16.069429

731 rows × 2 columns

In [15]:
```
#drop nulls
clean_df = df.dropna()
```

In [16]:
```
#export to csv
pd.DataFrame(clean_df).to_csv('C:/Users/blien/Documents/WGU/D213_New/cleaned_data
```

In [17]:
```
#ADfuller test to check and make time series stationary
result = adfuller(df['Revenue'])
print("Test statistics: ", result[0])
print("p-value: ", result[1])
print("Critical values: ", result[4])
print("#Lags Used: ", result[2])
print("Number of Observations: ", result[3])
```

```
Test statistics:  -2.21831904760894
p-value:  0.1996640061506454
Critical values:  {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%':
-2.5688855736949163}
#Lags Used:  1
Number of Observations:  729
```

In [18]:
```
#testing whether the time series is non-stationary
if result[1] <= 0.05:
    print("Reject null hypothesis, the time series is stationary")
else:
    print("Fail to reject null hypothesis, the time series in non-stationary")
```

```
Fail to reject null hypothesis, the time series in non-stationary
```

In [19]:
```
df_stationary = df.diff().dropna()
```

In [20]:
```
df_stationary.head(5)
```

Out[20]:

Date	Day	Revenue
2019-01-02	1.0	-0.292356
2019-01-03	1.0	-0.035416
2019-01-04	1.0	-0.012215
2019-01-05	1.0	0.215100
2019-01-06	1.0	-0.366702

In [21]:
```
#ADfuller test to check and make time series stationary
result = adfuller(df_stationary['Revenue'])
print("Test statistics: ", result[0])
print("p-value: ", result[1])
print("Critical values: ", result[4])
print("#Lags Used: ", result[2])
print("Number of Observations: ", result[3])
```

```
Test statistics:  -17.37477230355706
p-value:  5.1132069788403175e-30
Critical values:  {'1%': -3.4393520240470554, '5%': -2.8655128165959236, '10%':
-2.5688855736949163}
#Lags Used:  0
Number of Observations:  729
```

In [22]:
```
#testing whether the time series is non-stationary
if result[1] <= 0.05:
    print("Reject null hypothesis, the time series is stationary")
else:
    print("Fail to reject null hypothesis, the time series in non-stationary")
```

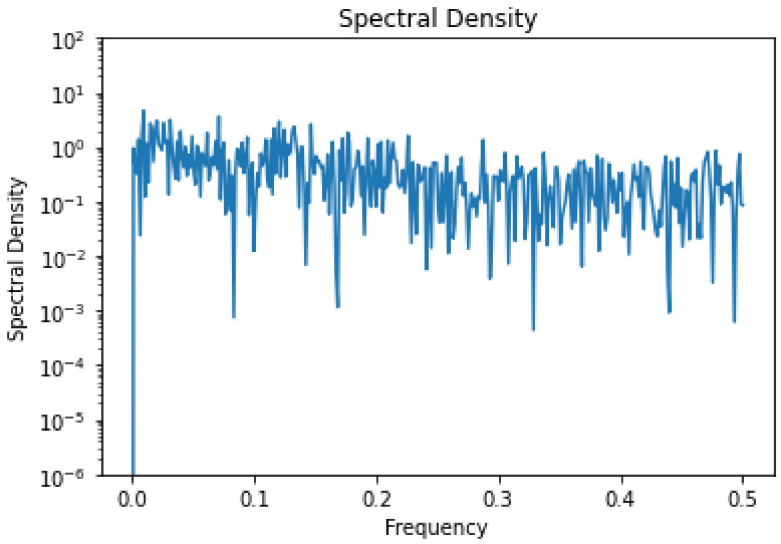
```
Reject null hypothesis, the time series is stationary
```

In [23]:
```
#test and train

x_train = df_stationary.loc[:'2020-09-30']
x_test = df_stationary.loc['2020-10-01':]

print('Train shape ', x_train.shape)
print('Test shape ', x_test.shape)
```

```
Train shape  (638, 2)
Test shape  (92, 2)
```

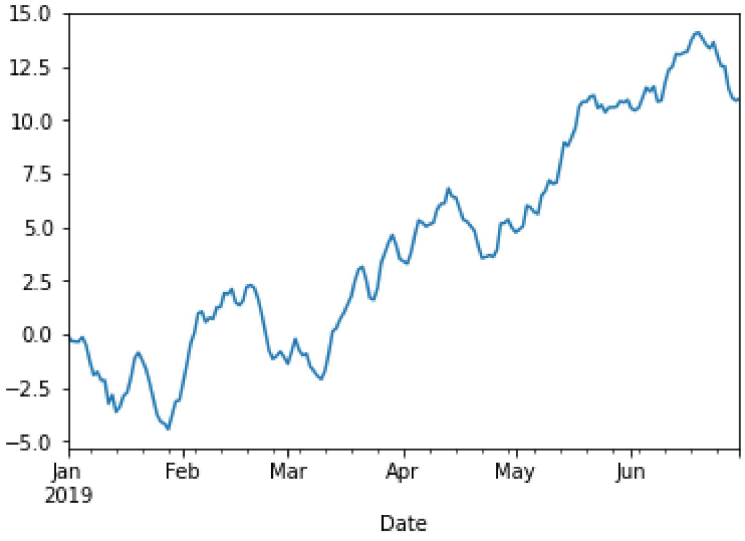
In [24]:
```
#import scipy signal
import scipy.signal
```

In [25]:
```
#spectral density

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



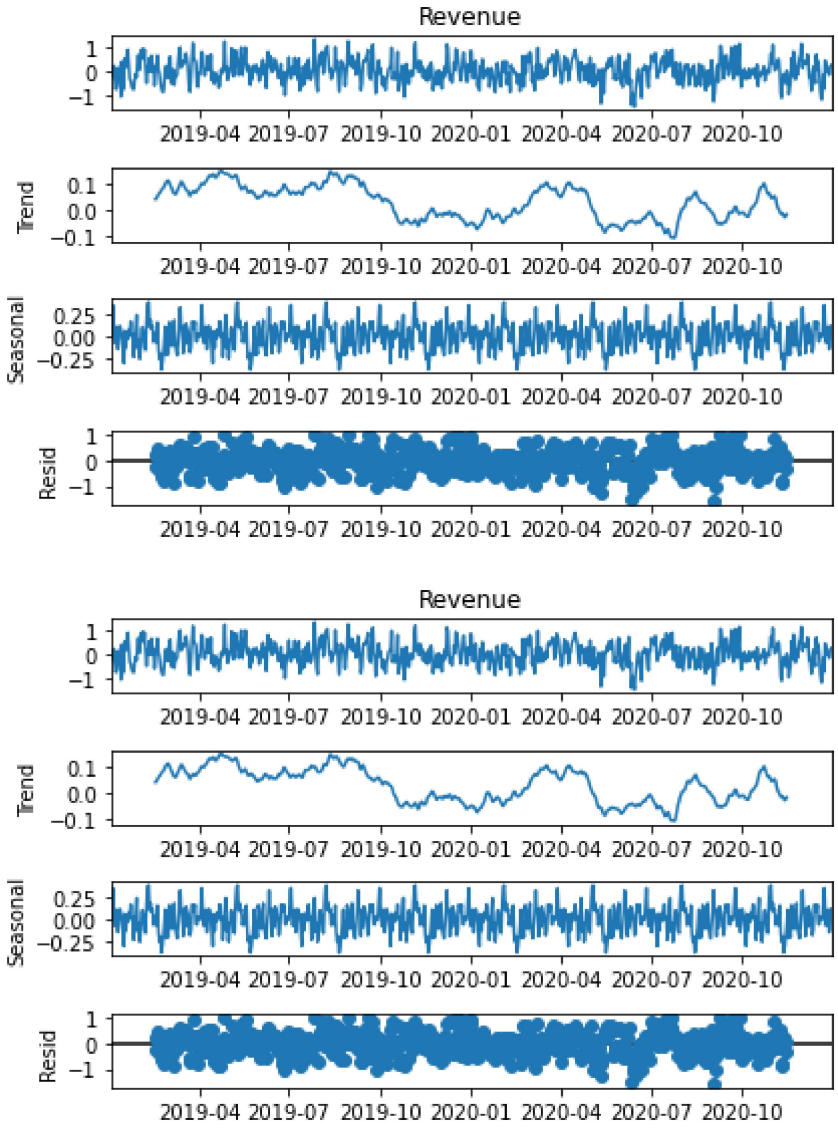
In [26]:
```
df['Revenue'].loc[:'2019-06-30'].plot()
```

Out[26]: <AxesSubplot:xlabel='Date'>



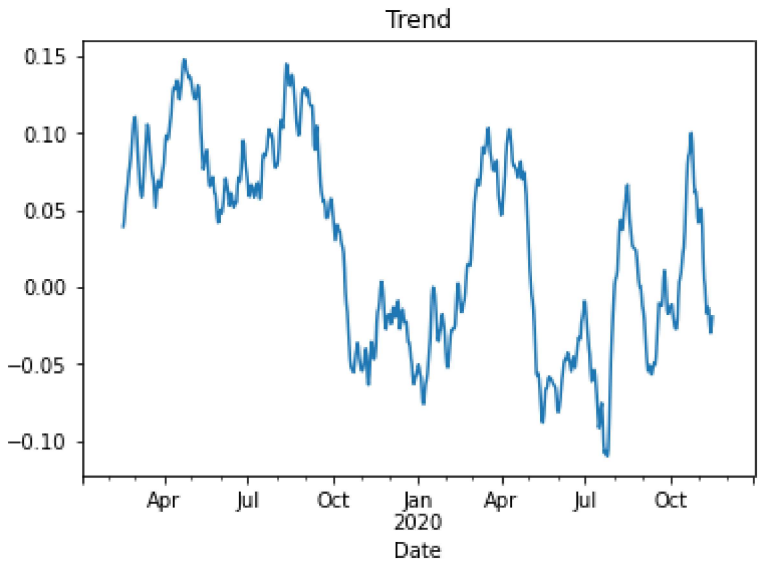
In [27]:
```
#perform decomp

result = seasonal_decompose(df_stationary['Revenue'], period = 90)
result.plot()
```

Out[27]:



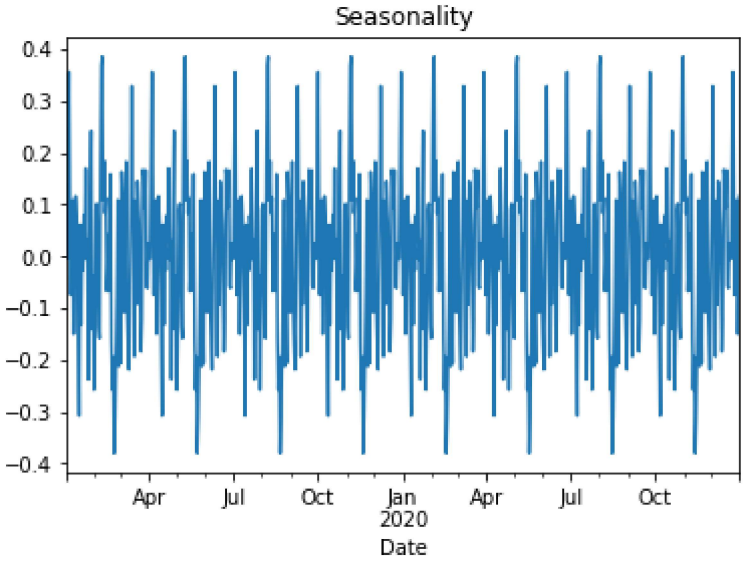
In [28]:
```
#plot trend
plt.title('Trend')
result.trend.plot()
```

Out[28]:
```
<AxesSubplot:title={'center':'Trend'}, xlabel='Date'>
```



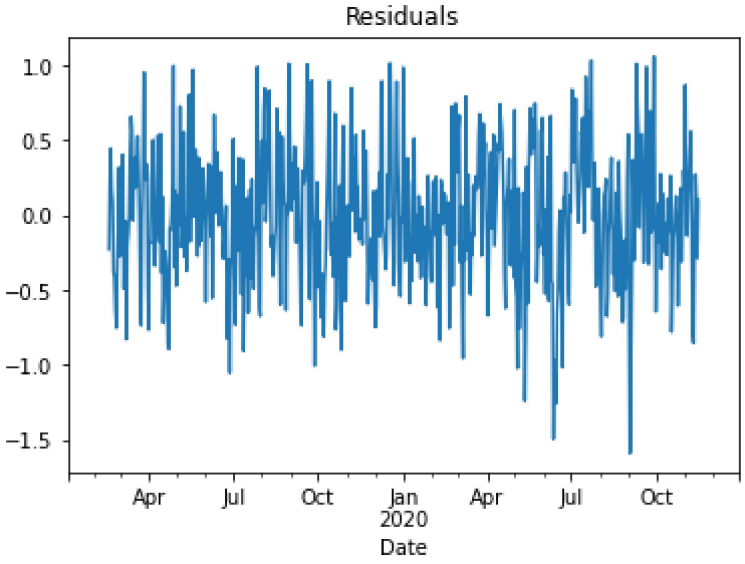
In [29]:
```
plt.title('Seasonality')
result.seasonal.plot()
```

Out[29]: <AxesSubplot:title={'center':'Seasonality'}, xlabel='Date'>

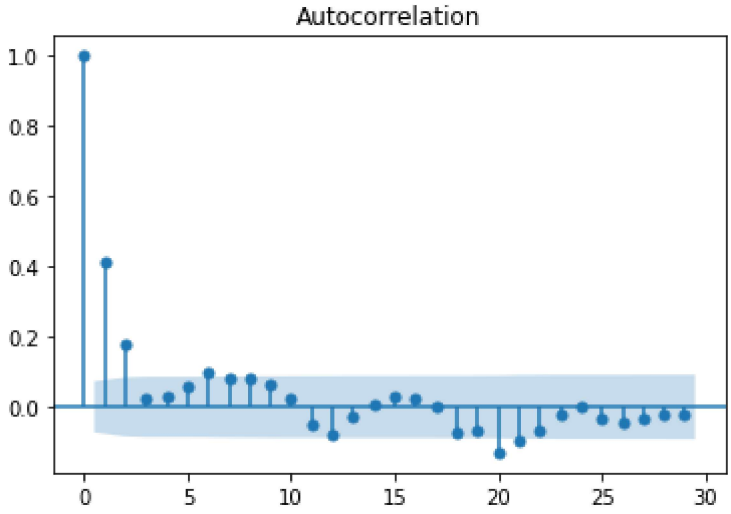


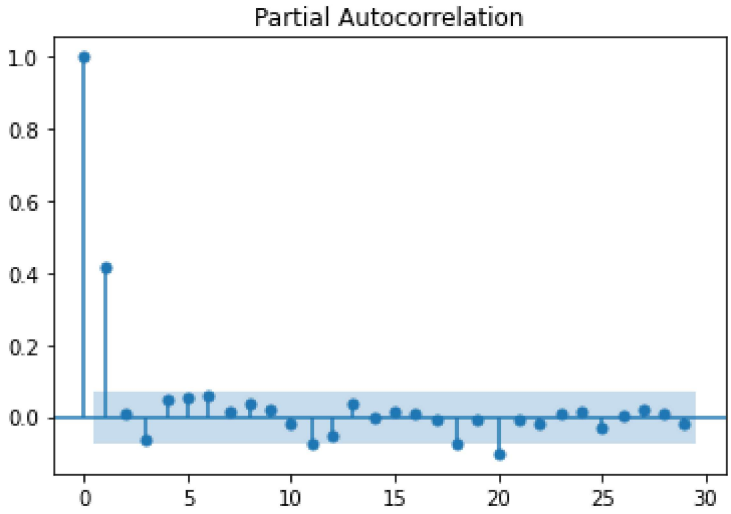
In [30]:
```
#plot trend
plt.title('Residuals')
result.resid.plot()
```

Out[30]: <AxesSubplot:title={'center':'Residuals'}, xlabel='Date'>



In [31]:
```
#plot the ACF and the PACF on residuals no spikes close to white noise
acf_orginal = plot_acf(df_stationary['Revenue'])
pacf_original = plot_pacf(df_stationary['Revenue'])
```





In [32]:
```
import pmdarima as pm
model = pm.auto_arima(df['Revenue'],
                      seasonal = True, n=90,
                      d=1, D=1,
                      start_p=1, start_q=1,
                      max_p=2, max_q=2, # maximum p and q
                      max_P=2, max_Q=2,              # frequency of series
                      trace=True,
                      error_action='ignore',
                      suppress_warnings=True)

print(model.summary())
```

```
Performing stepwise search to minimize aic
 ARIMA(1,1,1)(0,0,0)[0] intercept   : AIC=883.314, Time=0.13 sec
 ARIMA(0,1,0)(0,0,0)[0] intercept   : AIC=1015.972, Time=0.05 sec
 ARIMA(1,1,0)(0,0,0)[0] intercept   : AIC=881.359, Time=0.07 sec
 ARIMA(0,1,1)(0,0,0)[0] intercept   : AIC=906.199, Time=0.09 sec
 ARIMA(0,1,0)(0,0,0)[0]             : AIC=1015.481, Time=0.03 sec
 ARIMA(2,1,0)(0,0,0)[0] intercept   : AIC=883.300, Time=0.11 sec
 ARIMA(2,1,1)(0,0,0)[0] intercept   : AIC=883.348, Time=0.31 sec
 ARIMA(1,1,0)(0,0,0)[0]             : AIC=879.982, Time=0.03 sec
 ARIMA(2,1,0)(0,0,0)[0]             : AIC=881.911, Time=0.06 sec
 ARIMA(1,1,1)(0,0,0)[0]             : AIC=881.927, Time=0.08 sec
 ARIMA(0,1,1)(0,0,0)[0]             : AIC=905.166, Time=0.03 sec
 ARIMA(2,1,1)(0,0,0)[0]             : AIC=881.947, Time=0.19 sec

Best model:  ARIMA(1,1,0)(0,0,0)[0]
Total fit time: 1.196 seconds
                               SARIMAX Results
==========================================================================
=
Dep. Variable:                          y   No. Observations:                 73
1
Model:               SARIMAX(1, 1, 0)   Log Likelihood                -437.99
1
Date:                Thu, 03 Nov 2022   AIC                            879.98
2
Time:                        19:12:08   BIC                            889.16
8
Sample:                             0   HQIC                           883.52
6
                                  - 731
Covariance Type:                  opg
==========================================================================
=
                 coef    std err          z      P>|z|      [0.025      0.97
5]
--------------------------------------------------------------------------
-
ar.L1          0.4142      0.034     12.258      0.000       0.348       0.48
0
sigma2         0.1943      0.011     17.842      0.000       0.173       0.21
6
==========================================================================
======
Ljung-Box (L1) (Q):                   0.02   Jarque-Bera (JB):
```

```
1.92
Prob(Q):                              0.90   Prob(JB):
0.38
Heteroskedasticity (H):               1.00   Skew:
-0.02
Prob(H) (two-sided):                  0.97   Kurtosis:
2.75
================================================================================
======

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

In [38]:
```
model=sm.tsa.statespace.SARIMAX(df['Revenue'],order=(1, 1, 0),seasonal_order=(0,0
results=model.fit()
results.summary()
```

Out[38]:

SARIMAX Results

Dep. Variable:	Revenue	**No. Observations:**	731
Model:	SARIMAX(1, 1, 0)	**Log Likelihood**	-437.991
Date:	Thu, 03 Nov 2022	**AIC**	879.982
Time:	19:22:11	**BIC**	889.168
Sample:	01-01-2019	**HQIC**	883.526
	- 12-31-2020		
Covariance Type:	opg		

	coef	std err	z	P>|z|	[0.025	0.975]
ar.L1	0.4142	0.034	12.258	0.000	0.348	0.480
sigma2	0.1943	0.011	17.842	0.000	0.173	0.216

Ljung-Box (L1) (Q):	0.02	**Jarque-Bera (JB):**	1.92
Prob(Q):	0.90	**Prob(JB):**	0.38
Heteroskedasticity (H):	1.00	**Skew:**	-0.02
Prob(H) (two-sided):	0.97	**Kurtosis:**	2.75

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

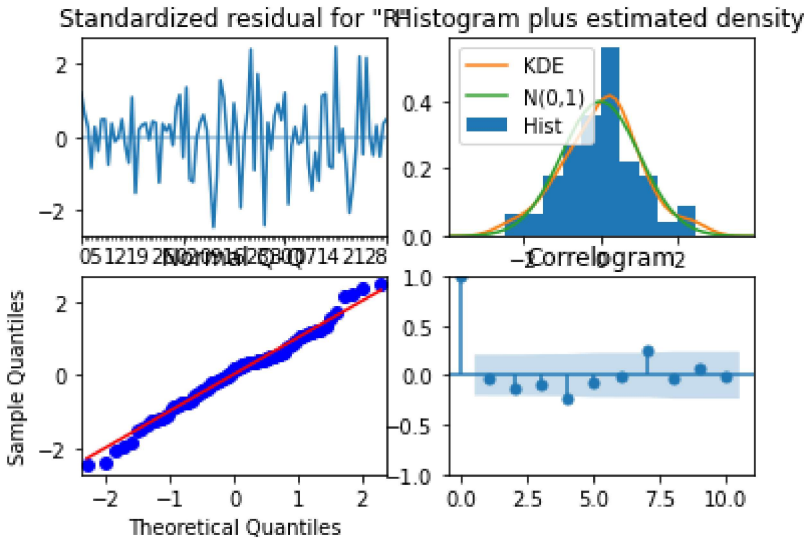
In [40]:
```
mae = np.mean(np.abs(results.resid))
print("Mean absolute error ", mae)
```

```
Mean absolute error  0.3556195422631195
```

In [51]:
```
mae = np.mean(np.abs(x_test['Revenue']))
print("Mean absolute error ", mae)
```

```
Mean absolute error  0.34844199554347843
```

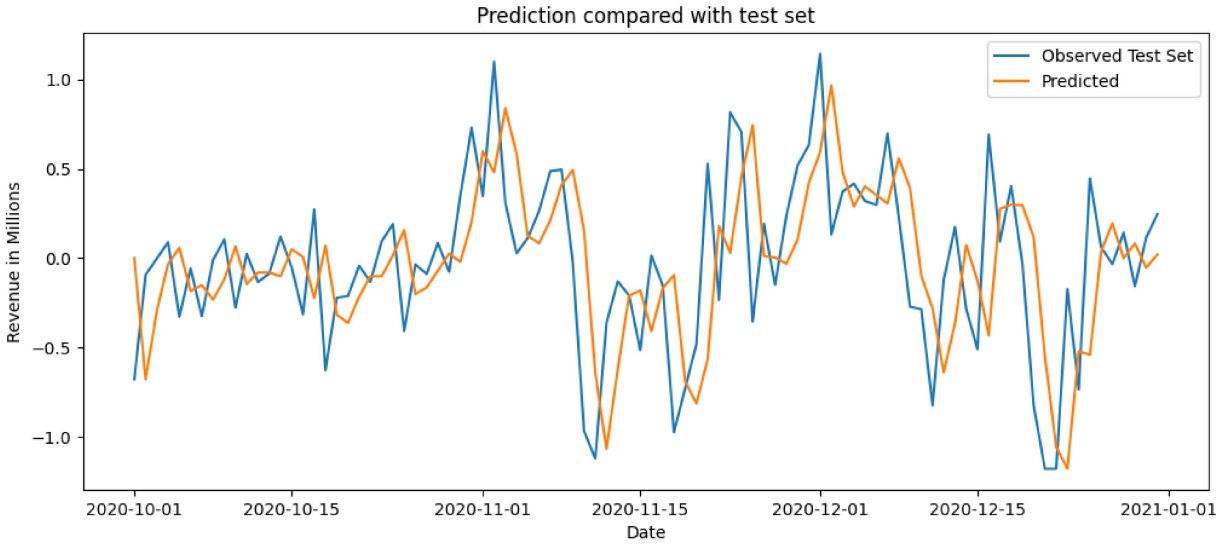
In [101]:
```
results.plot_diagnostics().show()
```



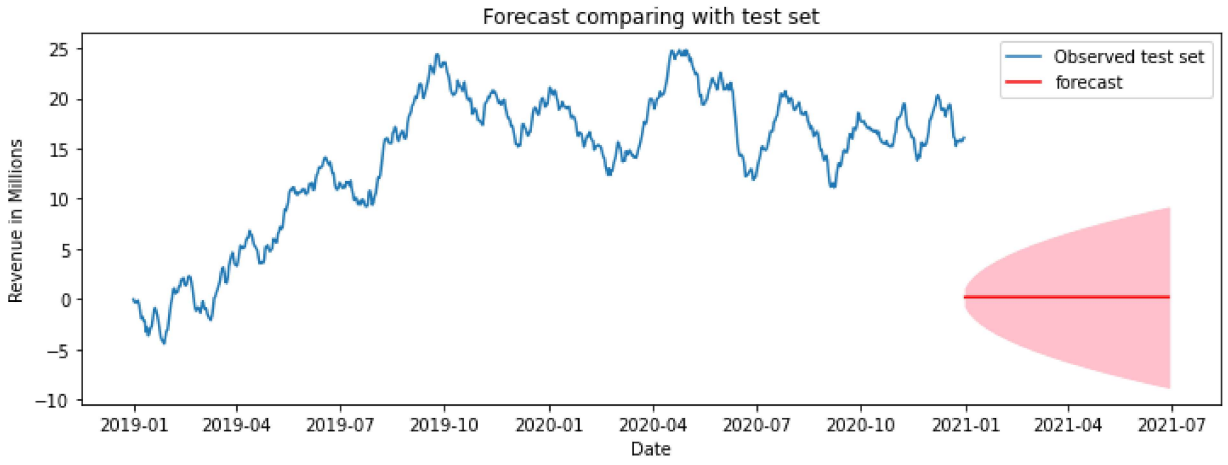
In [109]:
```
# Validate with test set - prediction
pred_dynamic = results.get_prediction(steps = 180, dynamic=True, full_results=Tru
time_forecast = results.get_prediction(steps = 180)
mean_forecast = time_forecast.predicted_mean
confidence_intervals = time_forecast.conf_int()
lower_limits = confidence_intervals.loc[:,'lower Revenue']
upper_limits = confidence_intervals.loc[:,'upper Revenue']
print(mean_forecast)
```

```
2020-10-01    0.000000
2020-10-02   -0.677501
2020-10-03   -0.295152
2020-10-04   -0.033150
2020-10-05    0.057697
                ...
2020-12-27    0.194839
2020-12-28   -0.000107
2020-12-29    0.082740
2020-12-30   -0.052876
2020-12-31    0.020257
Freq: D, Name: predicted_mean, Length: 92, dtype: float64
```

In [110]:
```
# plot predicted vs actual
plt.figure(figsize=(12,5), dpi=100)
plt.plot(x_test['Revenue'], label='Observed Test Set')
plt.plot(mean_forecast, label='Predicted')
plt.title("Prediction compared with test set")
plt.xlabel("Date")
plt.ylabel("Revenue in Millions")
plt.legend()
plt.show()
```



In [98]:
```
#forecast ARIMA model
diff_forecast = results.get_forecast(steps = 180)
mean_forecast = diff_forecast.predicted_mean
confidence_intervals = diff_forecast.conf_int()
lower_limits = confidence_intervals.loc[:,'lower Revenue']
upper_limits = confidence_intervals.loc[:,'upper Revenue']

plt.figure(figsize=(12,4))
plt.plot(df['Revenue'].index, df['Revenue'], label = "Observed test set")
#plot predictions
plt.plot(mean_forecast.index, mean_forecast, color = 'red', label = 'forecast')
#shade group in confidence limits
plt.fill_between(lower_limits.index, lower_limits, upper_limits, color = 'pink')
#set labels and legends
plt.title("Forecast comparing with test set")
plt.xlabel("Date")
plt.ylabel("Revenue in Millions")
plt.legend()
plt.show()
```



In [ ]: