In [83]:
```
# numerical analysis
import numpy as np
import pandas as pd
from pandas.plotting import autocorrelation_plot

# import statistics package
import statsmodels
import statsmodels.api as sm
import statsmodels.tsa.api as smt
from statsmodels.tsa.stattools import grangercausalitytests
from statsmodels.stats.diagnostic import het_arch
from statsmodels.compat import lzip
# from arch import arch_model

# visualisation
from matplotlib import pyplot as plt
import matplotlib.mlab as mlab
import seaborn as sns
import plotly.express as px

# machine learning
from sklearn.model_selection import train_test_split
from sklearn.linear_model import  LinearRegression, LogisticRegressi
from sklearn.tree import DecisionTreeClassifier, plot_tree, Decisior
from sklearn.svm import SVC, SVR
from sklearn.ensemble import RandomForestClassifier, GradientBoostir
from sklearn.neural_network import MLPClassifier, MLPRegressor
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score
```

Part 1 - Descriptive Statistics and Visuals

Q1: Describe both banks individually using any appropriate descriptive statistics and visuals, including ARMA processes and unit roots

In [84]:
```
prices = pd.read_csv('IrishBanks.csv', index_col=0)
prices.index=pd.to_datetime(prices.index, infer_datetime_format=Tru
print(prices.head(3))
print(prices.tail(3))

# Data starts 02 January 2018
# Ends 05 February 2021
# Both stock prices have reduced significantly
```

```
                AIB    BoI
Date
2018-01-02  5.460  7.195
2018-01-03  5.435  7.370
2018-01-04  5.450  7.545
                AIB    BoI
Date
2021-02-03  1.542  3.326
2021-02-04  1.554  3.290
2021-02-05  1.584  3.282
```

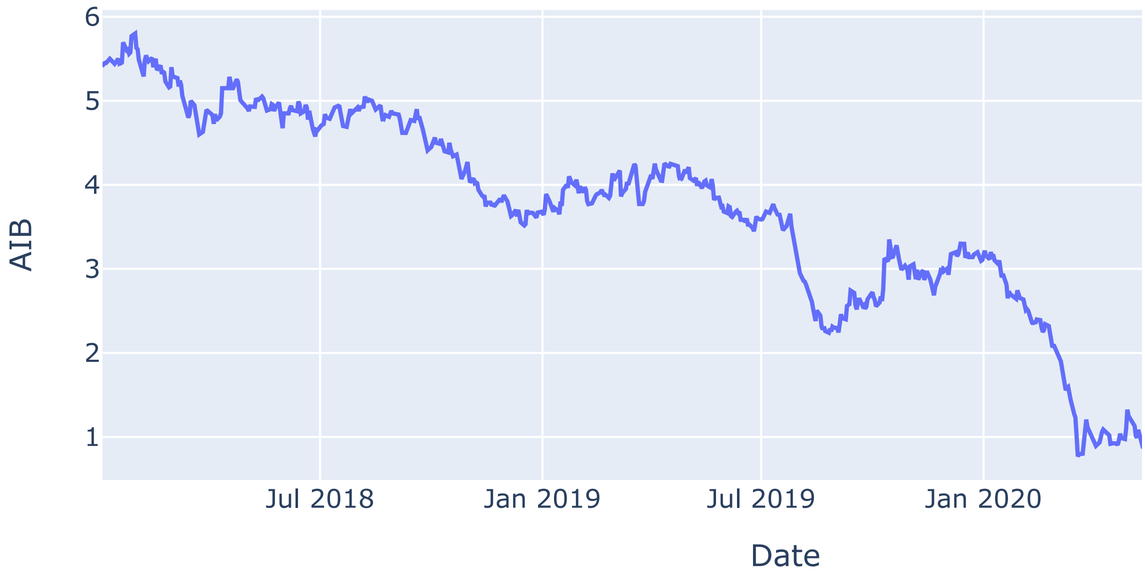
In [85]:
```
prices.describe()

# At its peak, AIB reached €5.80, BoI reached €8.15
# The lowest they fell to was €0.76 and €1.33
```

Out[85]:

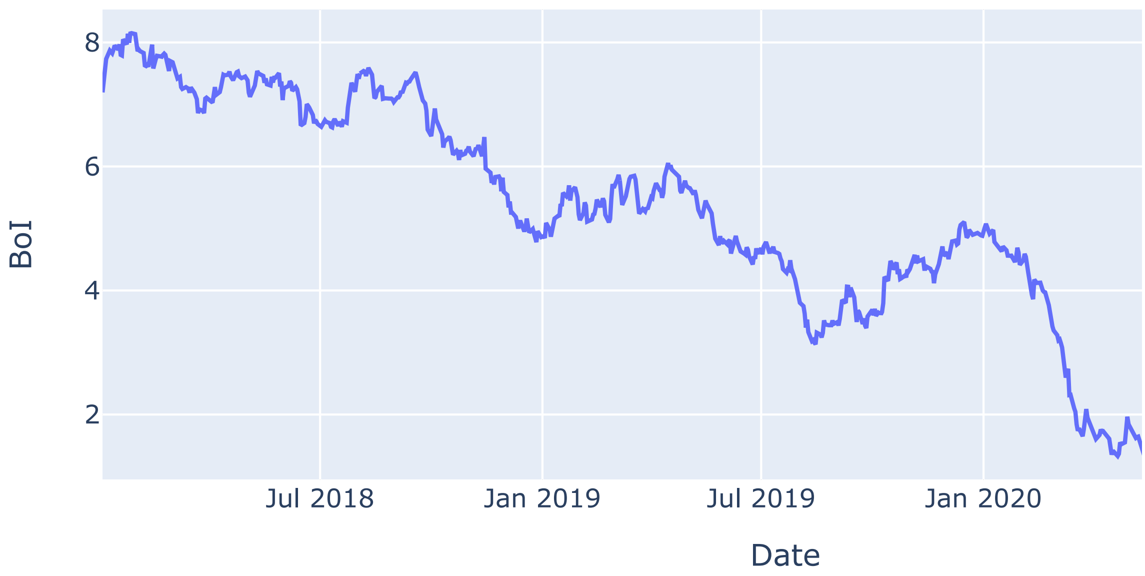
	AIB	BoI
count	789.000000	789.000000
mean	3.132753	4.642466
std	1.497407	2.021505
min	0.768000	1.330000
25%	1.554000	3.020000
50%	3.498000	4.700000
75%	4.424000	6.480000
max	5.800000	8.150000

In [86]:
```
fig1= px.line(prices, x= prices.index, y= "AIB", title = 'AIB stock
fig2= px.line(prices, x= prices.index, y= "BoI", title = 'BoI stock
fig1.show()
fig2.show()

# AIB and BoI prices move in a very similar downward trend
# These series appears to be non-stationary
```

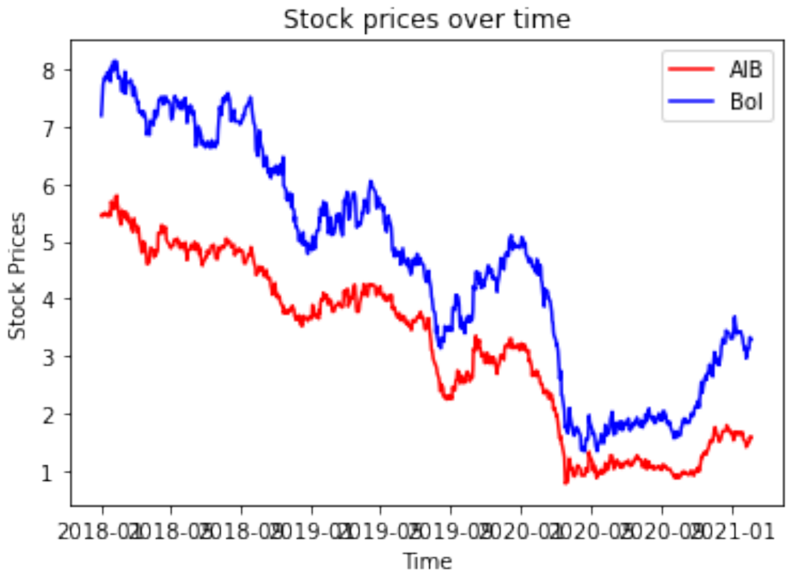
AIB stock price over time



BoI stock price over time



In [87]:
```
plt.plot(prices.index, prices['AIB'], color='red', label = 'AIB')
plt.plot(prices.index, prices['BoI'], color='blue', label = 'BoI')
plt.xlabel('Time')
plt.ylabel('Stock Prices')
plt.title('Stock prices over time')
plt.legend()
plt.show()

# The similarity can be seen clearly below
```



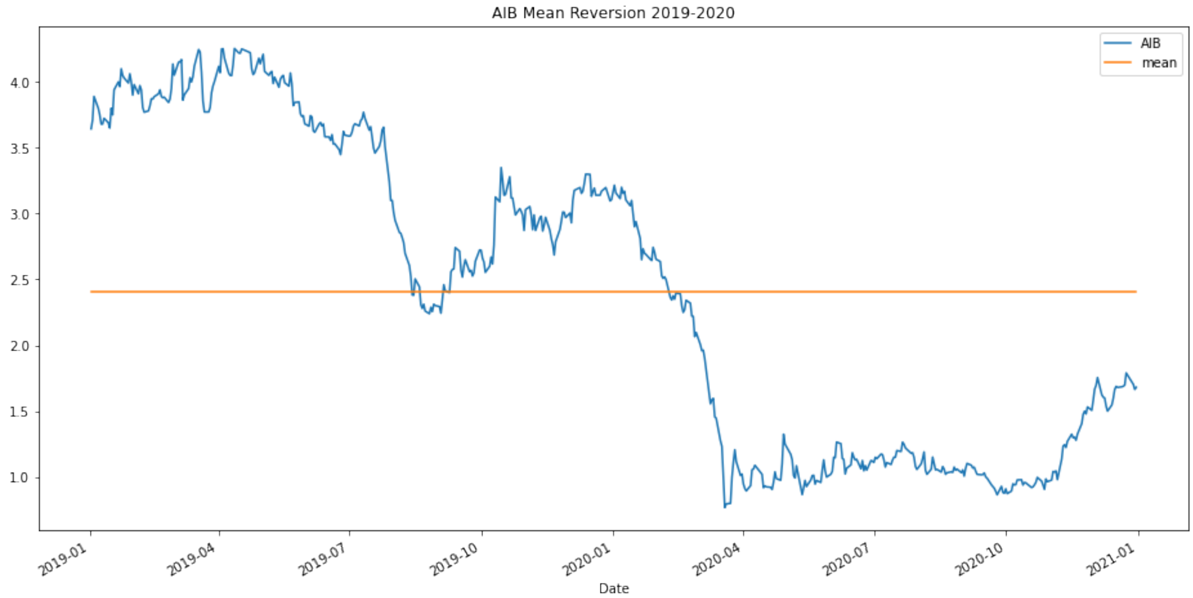
Are the stocks mean reverting?

In [88]:
```
AIB19_20 = prices.loc['2019':'2020'][['AIB']]
```

In [89]:
```
AIB19_20['mean'] = AIB19_20['AIB'].mean()
```

In [90]:
```
AIB19_20.plot(figsize=(16,8), title='AIB Mean Reversion 2019-2020')

# It does not look like AIB's stock price reverts to the mean
```

Out[90]:
```
<AxesSubplot:title={'center':'AIB Mean Reversion 2019-2020'}, xlab
el='Date'>
```

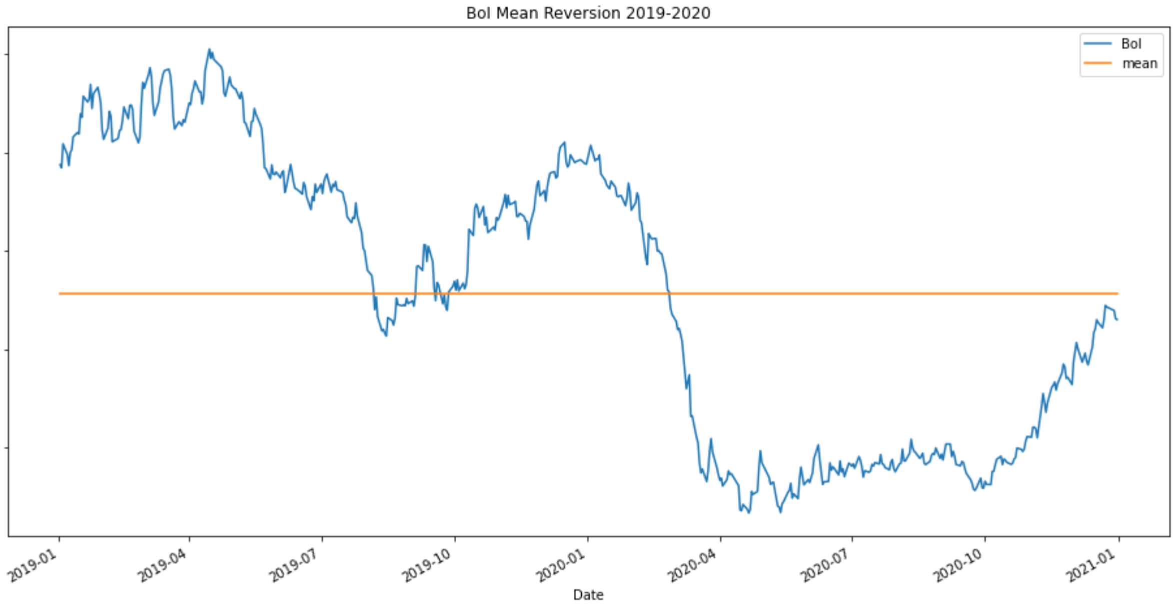


In [91]:
```
# Now test BoI
BoI19_20 = prices.loc['2019':'2020'][['BoI']]
```

In [92]:
```
BoI19_20['mean'] = BoI19_20['BoI'].mean()
```

In [93]:
```
BoI19_20.plot(figsize=(16,8), title='BoI Mean Reversion 2019-2020')

# BoI's stock price looks slightly more mean reverting
```

Out[93]:
```
<AxesSubplot:title={'center':'BoI Mean Reversion 2019-2020'}, xlab
el='Date'>
```

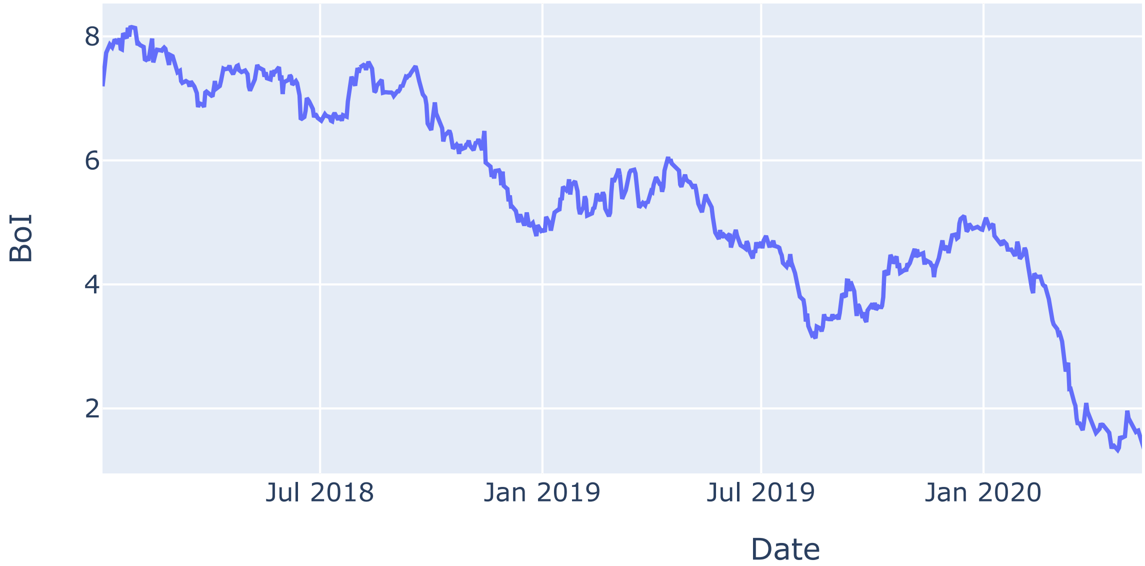


In [94]:
```
# Import the ISEQ as a comparison to the overall Irish stock exchan

ISEQ = pd.read_csv('ISEQ.csv', index_col=0)
ISEQ.index=pd.to_datetime(ISEQ.index, infer_datetime_format=True)
print(ISEQ.head(3))
print(ISEQ.tail(3))
```

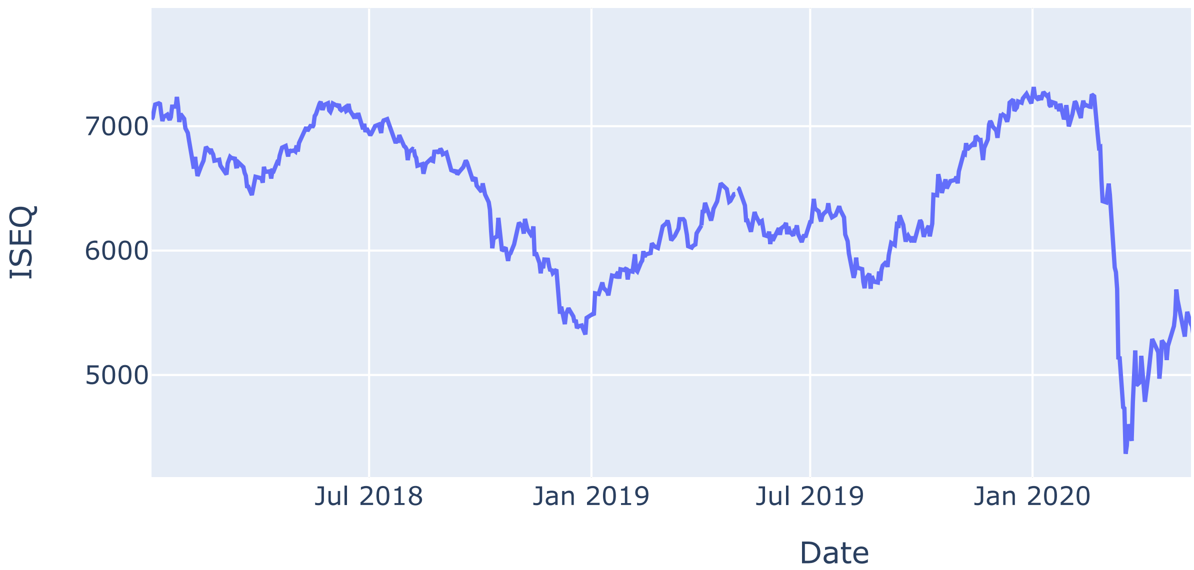
```
                   ISEQ
Date
2018-01-02  7054.549805
2018-01-03  7071.229980
2018-01-04  7126.569824
                   ISEQ
Date
2021-02-03  7337.609863
2021-02-04  7358.020020
2021-02-05  7361.959961
```

In [95]:
```
# How do BoI returns compare to the ISEQ?

fig1= px.line(prices, x= prices.index, y= "BoI", title = 'BoI price
fig2= px.line(ISEQ, x= ISEQ.index, y= "ISEQ", title = 'ISEQ price o
fig1.show()
fig2.show()

# BoI and the ISEQ have somewhat of a similar pattern
```

BoI price over time

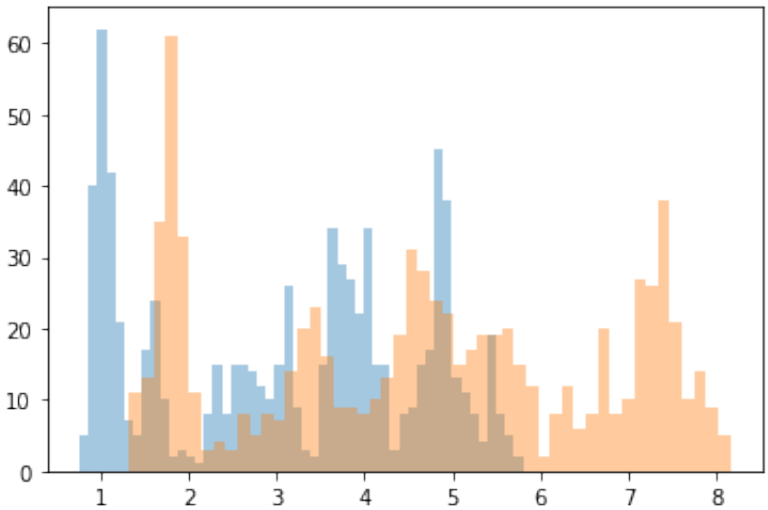


ISEQ price over time



In [96]:
```
# Show the distribution of prices

plt.hist(prices['AIB'], bins=50, alpha = 0.4)
plt.hist(prices['BoI'], bins=50, alpha = 0.4)

# Very similar patterns
```

Out[96]:
```
(array([11., 13., 35., 61., 33., 11.,  3.,  4.,  3.,  8.,  5.,  8.
,  7.,
        14., 20., 23., 16.,  9.,  9.,  8., 10., 13., 19., 31., 28.
, 24.,
        22., 15., 17., 19., 19., 20., 15., 12.,  2.,  8., 12.,  6.
,  8.,
        20.,  8., 10., 27., 26., 38., 21., 10., 14.,  9.,  5.]),
 array([1.33  , 1.4664, 1.6028, 1.7392, 1.8756, 2.012 , 2.1484, 2.
2848,
        2.4212, 2.5576, 2.694 , 2.8304, 2.9668, 3.1032, 3.2396, 3.
376 ,
        3.5124, 3.6488, 3.7852, 3.9216, 4.058 , 4.1944, 4.3308, 4.
4672,
        4.6036, 4.74  , 4.8764, 5.0128, 5.1492, 5.2856, 5.422 , 5.
5584,
        5.6948, 5.8312, 5.9676, 6.104 , 6.2404, 6.3768, 6.5132, 6.
6496,
        6.786 , 6.9224, 7.0588, 7.1952, 7.3316, 7.468 , 7.6044, 7.
7408,
        7.8772, 8.0136, 8.15  ]),
 <BarContainer object of 50 artists>)
```



In [97]:
```
prices.corr()

# 98% correlation betweeb AIB and BoI stock prices
```

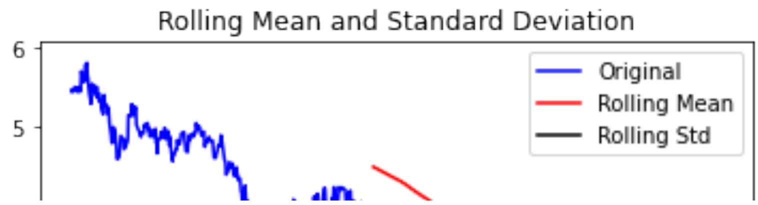
Out[97]:

	AIB	BoI
AIB	1.00000	0.98398
BoI	0.98398	1.00000

Checking Stationarity of Stock Prices

In [98]:
```
test1 = prices.loc['2020-02-05':]
train1 = prices.loc[:'2020-02-04']
```

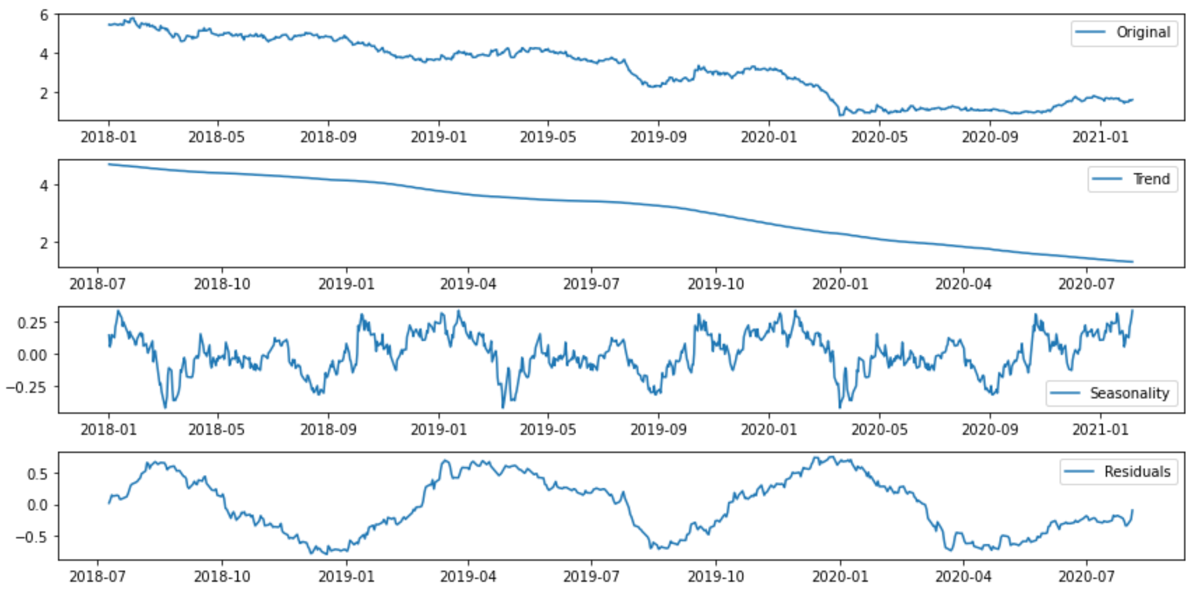
In [99]:
```
# Testing stationarity of AIB Stock Price
from statsmodels.tsa.stattools import adfuller

def test_stationarity(timeseries):
 #Determing rolling statistics
 rolmean = prices['AIB'].rolling(window=365).mean()
 rolstd = prices['AIB'].rolling(window=365).std()
 #Plot rolling statistics:
 plt.plot(prices['AIB'], color='blue', label='Original')
 plt.plot(rolmean, color='red', label='Rolling Mean')
 plt.plot(rolstd, color='black', label = 'Rolling Std')
 plt.legend(loc='best')
 plt.title('Rolling Mean and Standard Deviation')
 plt.show(block=False)

 print("Results of dickey fuller test")
 adft = adfuller(timeseries,autolag='AIC')
 # output for dft will give us without defining what the values are
 #hence we manually write what values does it explains using a for
 output = pd.Series(adft[0:4],index=['Test Statistics','p-value','N
 for key,values in adft[4].items():
   output['critical value (%s)'%key] = values
 print(output)

test_stationarity(train1['AIB'])


# p-value > 0.05, so AIB stock prices are non-stationary
# This was to be expected
# Rolling Mean and Rolling Standard Deviation show the non-stationa
```




```
Results of dickey fuller test
Test Statistics                 -0.974615
p-value                          0.762410
No. of lags used                 0.000000
Number of observations used    531.000000
critical value (1%)             -3.442725
critical value (5%)             -2.866998
critical value (10%)            -2.569677
dtype: float64
```

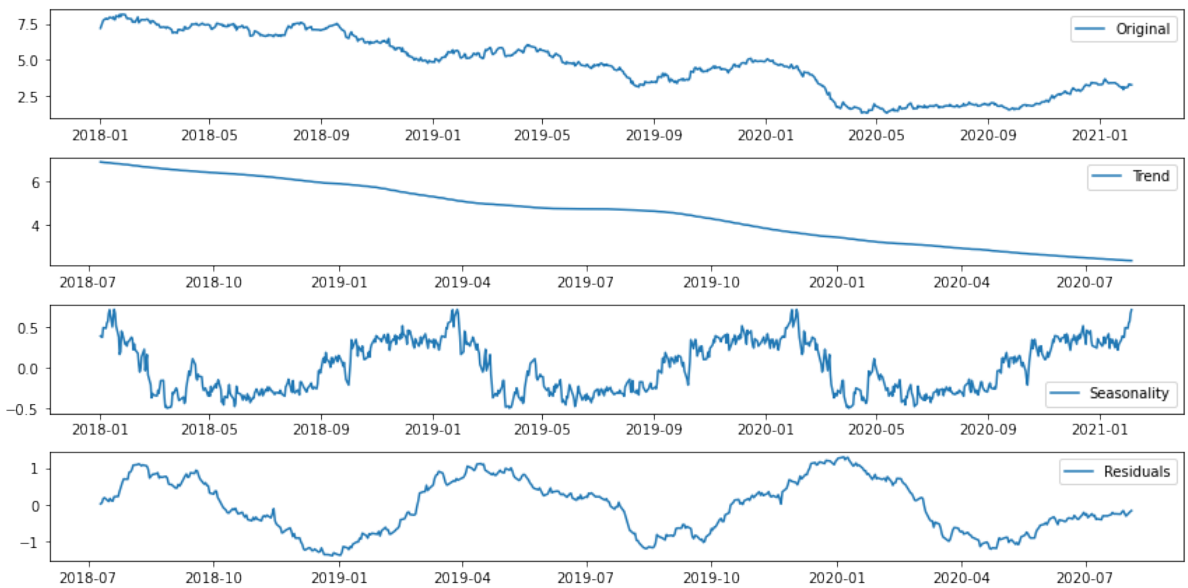
In [100]:
```
# Examine the seasonality of AIB prices
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(prices.AIB, freq = 260) #Avg aro
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.figure(figsize=(12,6))
plt.subplot(411)
plt.plot(prices['AIB'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
plt.show()

# A clear downward trend in stock prices
```

```
/var/folders/mh/j7bgd0wx1mb1jjnzcp0c8dfw0000gn/T/ipykernel_83727/8
42978463.py:3: FutureWarning:

the 'freq'' keyword is deprecated, use 'period' instead
```



In [101]:
```
# Examine the seasonality of BoI
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(prices.BoI, freq = 260) #Avg aro
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.figure(figsize=(12,6))
plt.subplot(411)
plt.plot(prices['BoI'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
plt.show()
```

```
/var/folders/mh/j7bgd0wx1mb1jjnzcp0c8dfw0000gn/T/ipykernel_83727/1
162248132.py:3: FutureWarning:

the 'freq'' keyword is deprecated, use 'period' instead
```

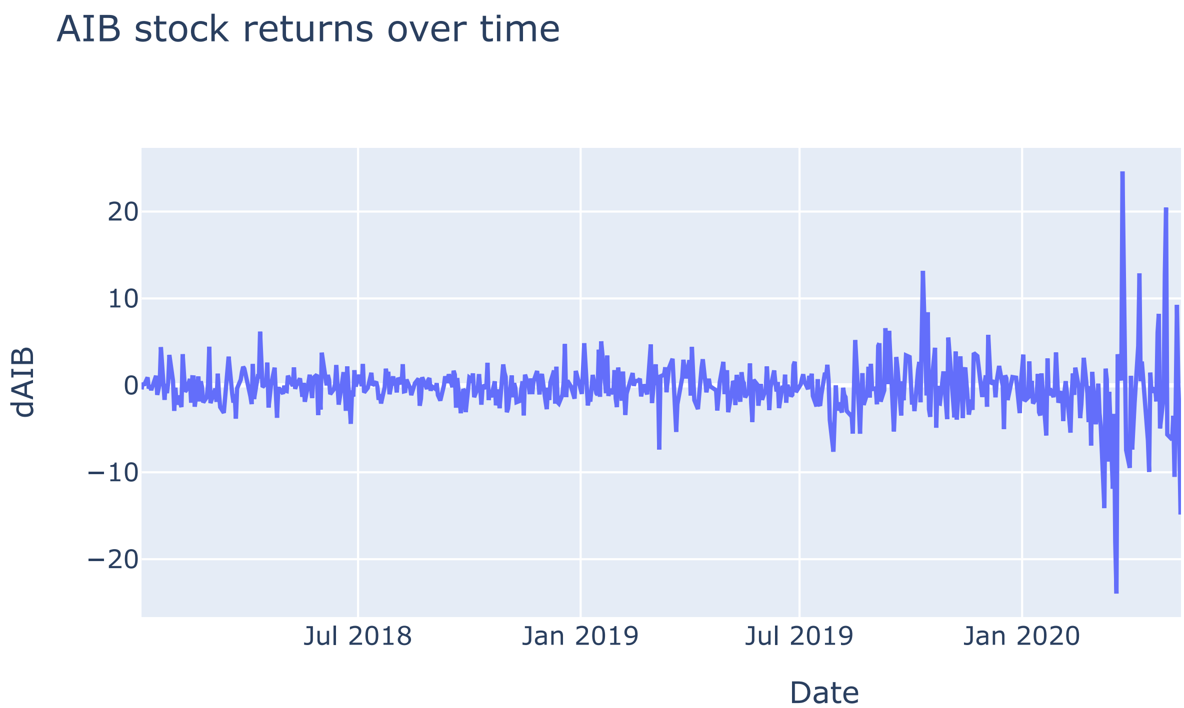


In [102]:
```
# Transform stock prices to returns

prices['dAIB']= prices['AIB'].transform(lambda x : (x - x.shift(1))
prices['dBoI']= prices['BoI'].transform(lambda x : (x - x.shift(1))
prices = prices.dropna()
prices.head(3)

# Transforming to returns may turn the series stationary
```

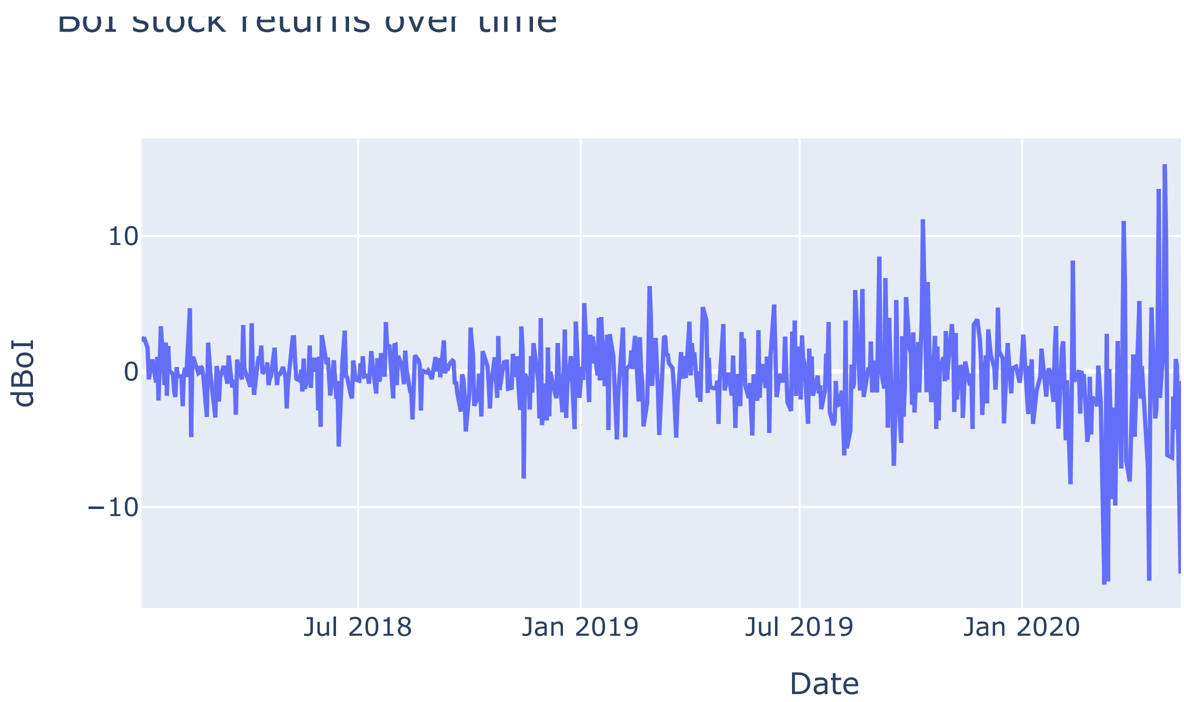
Out[102]:

Date	AIB	BoI	dAIB	dBoI
2018-01-03	5.435	7.370	-0.457875	2.432245
2018-01-04	5.450	7.545	0.275989	2.374491
2018-01-05	5.450	7.735	0.000000	2.518224

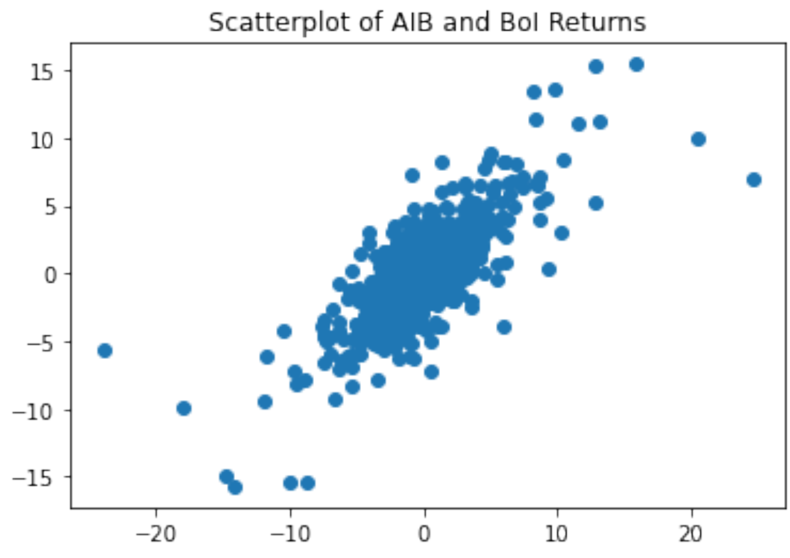
In [103]:
```
fig1= px.line(prices, x= prices.index, y= "dAIB", title = 'AIB stoc
fig2= px.line(prices, x= prices.index, y= "dBoI", title = 'BoI stoc
fig1.show()
fig2.show()

# Both sets of returns look to be stationary, but have some noncons
# There is no evident upward or downward trend over time
```



BoI stock returns over time



In [104]:
```
plt.scatter(prices['dAIB'], prices['dBoI'])
plt.title('Scatterplot of AIB and BoI Returns')
plt.show()

# Positive Correlation is evident
# When one goes up, the other likely follows
```



In [105]:
```
correlation = prices['dAIB'].corr(prices['dBoI'])
print("Correlation of the stock retuns is: ", correlation)
```

```
Correlation of the stock retuns is:  0.7364661575240155
```

In [106]:
```
prices.describe()
```

Out[106]:

	AIB	BoI	dAIB	dBoI
count	788.000000	788.000000	788.000000	788.000000
mean	3.129800	4.639227	-0.093628	-0.045353
std	1.496057	2.020739	3.558539	3.292536
min	0.768000	1.330000	-23.960396	-15.714286
25%	1.552250	3.016000	-1.762296	-1.732064
50%	3.492000	4.698000	-0.181295	-0.144521
75%	4.413500	6.476250	1.250806	1.508710
max	5.800000	8.150000	24.625000	15.458015

Unit Root Tests

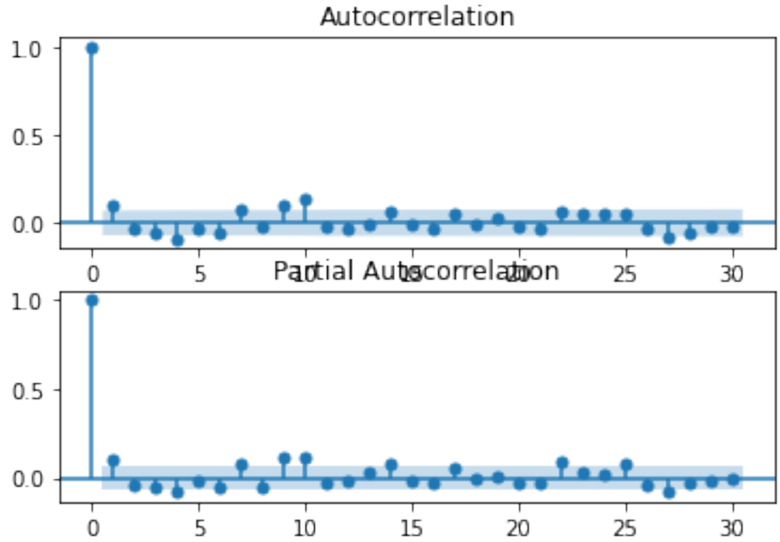
In [107]:
```
acf_AIB = smt.acf(prices['dAIB'], nlags=15)
pacf_AIB = smt.pacf(prices['dAIB'], nlags=15)

acf_BoI = smt.acf(prices['dBoI'], nlags=15)
pacf_BoI = smt.pacf(prices['dBoI'], nlags=15)

correlogram = pd.DataFrame({'acf_AIB':acf_AIB[1:], 'pacf_AIB':pacf_
correlogram

# 15 lags, the returns are daily so 15 days
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/stattoo
ls.py:667: FutureWarning:

fft=True will become the default after the release of the 0.12 rel
ease of statsmodels. To suppress this warning, explicitly set fft=
False.
```

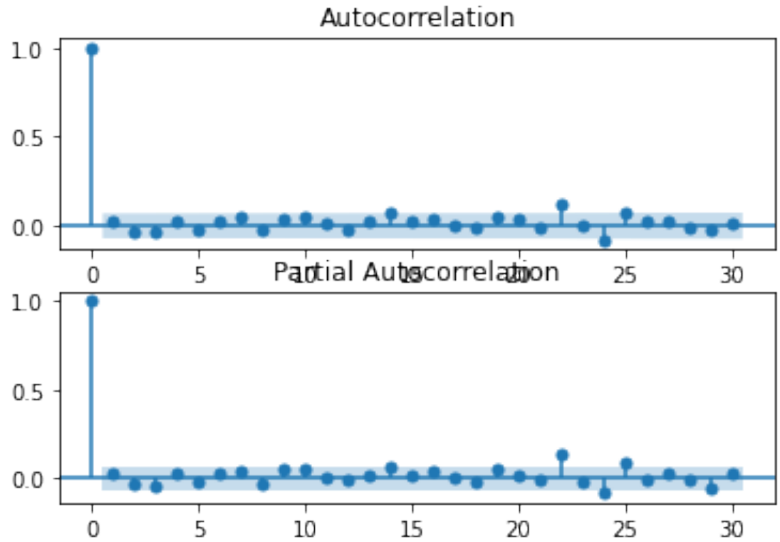
Out[107]:

	acf_AIB	pacf_AIB	acf_BOI	pacf_BOI
0	0.105898	0.106032	0.031277	0.031317
1	-0.028405	-0.040171	-0.030934	-0.032025
2	-0.058914	-0.052411	-0.040280	-0.038506
3	-0.090816	-0.081572	0.031261	0.033043
4	-0.036188	-0.022217	-0.016482	-0.021166
5	-0.053867	-0.057881	0.023390	0.025292
6	0.072869	0.075968	0.043883	0.044364
7	-0.020029	-0.051122	-0.025803	-0.030316
8	0.103871	0.111489	0.040641	0.049289
9	0.139978	0.117690	0.054519	0.052665
10	-0.018961	-0.033325	0.009914	0.004709
11	-0.033642	-0.015353	-0.023111	-0.013881
12	-0.012007	0.029166	0.020283	0.020901
13	0.066896	0.081205	0.070418	0.068023
14	-0.011839	-0.012036	0.021679	0.019680

In [108]:
```
# ACF and PACF of AIB Returns
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

plt.figure()
plt.subplot(211)
plot_acf(prices['dAIB'], ax=plt.gca(), lags=30)
plt.subplot(212)
plot_pacf(prices['dAIB'], ax=plt.gca(), lags=30)
plt.show()
```



In [109]:
```
# ACF and PACF of BoI Returns

plt.figure()
plt.subplot(211)
plot_acf(prices['dBoI'], ax=plt.gca(), lags=30)
plt.subplot(212)
plot_pacf(prices['dBoI'], ax=plt.gca(), lags=30)
plt.show()
```



Autocorrelation Figures

In [110]:
```
autocorrelation_AIB = prices['dAIB'].autocorr()
print("The autocorrelation of daily AIB returns is %4.2f" %(autocor
```

```
The autocorrelation of daily AIB returns is 0.11
```

In [111]:
```
autocorrelation_BoI = prices['dBoI'].autocorr()
print("The autocorrelation of daily BoI returns is %4.2f" %(autocor
```

```
The autocorrelation of daily BoI returns is 0.03
```

In [112]:
```
num_obs = len(prices)
# Number of observations
```

In [113]:
```
from math import sqrt

conf = 1.96/sqrt(num_obs)
print("The approximate confidence interval is +/- %4.2f" %(conf))
```

```
The approximate confidence interval is +/- 0.07
```

ADF Test

An Augmented Dicker Fuller (ADF) Test is a unit test for stationarity. The null hypothesis says alpha=1 in the model equation.

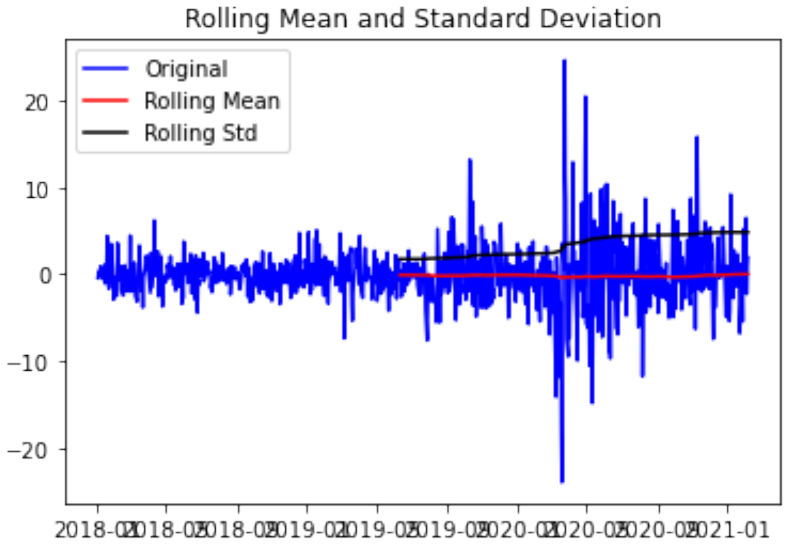
As the null hypothesis assumes the presence of unit root (a=1), the p-value should be less than 0.05 to reject the null hypothesis and infer that the series is stationary. A p-value > 0.05 would infer non-stationarity.

If a series has weak stationarity; the mean, variance and autocorrelation do not depend on time. If a series is not stationary, it will be difficult to model

In [114]:
```
test1 = prices.loc['2020-02-05':]
train1 = prices.loc[:'2020-02-04']
```

In [115]:
```
# Testing stationarity of AIB Stock Returns

def test_stationarity(timeseries):
 #Determing rolling statistics
 rolmean = prices['dAIB'].rolling(window=365).mean()
 rolstd = prices['dAIB'].rolling(window=365).std()

 #Plot rolling statistics:
 plt.plot(prices['dAIB'], color='blue', label='Original')
 plt.plot(rolmean, color='red', label='Rolling Mean')
 plt.plot(rolstd, color='black', label = 'Rolling Std')
```

```
 plt.legend(loc='best')
 plt.title('Rolling Mean and Standard Deviation')
 plt.show(block=False)

 print("Results of dickey fuller test")
 adft = adfuller(timeseries,autolag='AIC')
 # output for dft will give us without defining what the values are
 #hence we manually write what values does it explains using a for
 output = pd.Series(adft[0:4],index=['Test Statistics','p-value','N
 for key,values in adft[4].items():
   output['critical value (%s)'%key] = values
 print(output)

test_stationarity(train1['dAIB'])

# p-value < 0.05, therefore the time series is stationary
# Lack of a trend in the Rolling Mean and Stand Dev shows stationar
```

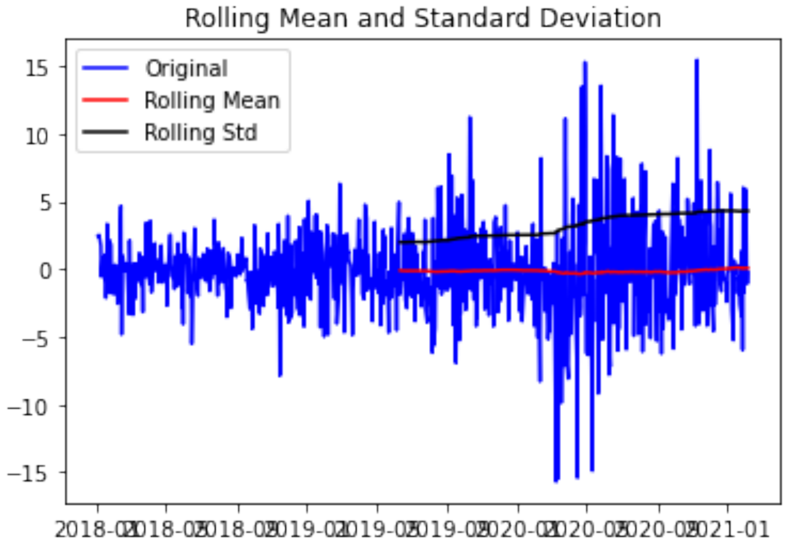



```
Results of dickey fuller test
Test Statistics                 -22.331378
p-value                           0.000000
No. of lags used                  0.000000
Number of observations used     530.000000
critical value (1%)              -3.442749
critical value (5%)              -2.867009
critical value (10%)             -2.569683
dtype: float64
```

In [116]:

```
# Testing stationarity of BoI Stock Returns

def test_stationarity(timeseries):
 #Determing rolling statistics
 rolmean = prices['dBoI'].rolling(window=365).mean()
 rolstd = prices['dBoI'].rolling(window=365).std()

 #Plot rolling statistics:
 plt.plot(prices['dBoI'], color='blue', label='Original')
 plt.plot(rolmean, color='red', label='Rolling Mean')
```

```
    plt.plot(rolmean, color='red', label='Rolling Mean')
    plt.plot(rolstd, color='black', label = 'Rolling Std')
    plt.legend(loc='best')
    plt.title('Rolling Mean and Standard Deviation')
    plt.show(block=False)

    print("Results of dickey fuller test")
    adft = adfuller(timeseries,autolag='AIC')
    # output for dft will give us without defining what the values are
    #hence we manually write what values does it explains using a for
    output = pd.Series(adft[0:4],index=['Test Statistics','p-value','N
    for key,values in adft[4].items():
      output['critical value (%s)'%key] = values
    print(output)

test_stationarity(train1['dBoI'])

# p-value < 0.05, therefore the time series is stationary
# Lack of a trend in the Rolling Mean and Stand Dev shows stationar
```




```
Results of dickey fuller test
Test Statistics                -23.683229
p-value                          0.000000
No. of lags used                 0.000000
Number of observations used    530.000000
critical value (1%)             -3.442749
critical value (5%)             -2.867009
critical value (10%)            -2.569683
dtype: float64
```

In [117]:
```
# Augmented Dickey Fuller Test for stationarity - AIB
from statsmodels.tsa.stattools import adfuller, kpss

result_AIB = adfuller(prices.dAIB.values, autolag='AIC')
print(f'ADF Statistic: {result_AIB[0]}')
print(f'p-value: {result_AIB[1]}')
for key, value in result_AIB[4].items():
  print('Critical Values:')
  print(f'  {key},{value}')

# p-value is much less than significance level of 0.05, again confi
# p-vlaue less than 0.05, therefore we can reject the null hypothes
```

```
ADF Statistic: -7.558246463896617
p-value: 3.0585446331492924e-11
Critical Values:
  1%,-3.438783171038672
Critical Values:
  5%,-2.865262118650577
Critical Values:
  10%,-2.568752018688748
```

In [118]:
```
# Augmented Dickey Fuller Test for stationarity - BoI

result_BoI = adfuller(prices.dBoI.values, autolag='AIC')
print(f'ADF Statistic: {result_BoI[0]}')
print(f'p-value: {result_BoI[1]}')
for key, value in result_BoI[4].items():
  print('Critical Values:')
  print(f'  {key},{value}')

# p-value is much less than significance level of 0.05, again confi
```

```
ADF Statistic: -27.164554340013876
p-value: 0.0
Critical Values:
  1%,-3.438686413400388
Critical Values:
  5%,-2.8652194721349424
Critical Values:
  10%,-2.5687293001910008
```

KPSS Test

Kwiatkowski-Phillips-Schmidt-Shin is another test for stationarity. Key difference from the ADF test is that the null hypothesis here is that the series is actually stationary. Therefore, if the p-value < 0.05 for the KPSS test, the tested series is non-stationary.

In [119]:
```
# KPSS Test for Stationarity - AIB returns
result_AIB2 = kpss(prices.dAIB.values, regression='c')
print('\nKPSS Statistic: %f' % result_AIB2[0])
print('p-value: %f' % result_AIB2[1])
for key, value in result_AIB2[3].items():
  print('Critical Values:')
  print(f'  {key}: {value}')

# p-value > 0.05, so AIB returns are stationary
```

```
KPSS Statistic: 0.108159
p-value: 0.100000
Critical Values:
  10%: 0.347
Critical Values:
  5%: 0.463
Critical Values:
  2.5%: 0.574
Critical Values:
  1%: 0.739

/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/stattoo
ls.py:1875: FutureWarning:

The behavior of using nlags=None will change in release 0.13.Curre
ntly nlags=None is the same as nlags="legacy", and so a sample-siz
e lag length is used. After the next release, the default will cha
nge to be the same as nlags="auto" which uses an automatic lag len
gth selection method. To silence this warning, either use "auto" o
r "legacy"

/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/stattoo
ls.py:1910: InterpolationWarning:

The test statistic is outside of the range of p-values available i
n the
look-up table. The actual p-value is greater than the p-value retu
rned.
```

In [120]:
```
# KPSS Test for Stationarity - BoI returns
result_BoI2 = kpss(prices.dBoI.values, regression='c')
print('\nKPSS Statistic: %f' % result_BoI2[0])
print('p-value: %f' % result_BoI2[1])
for key, value in result_BoI2[3].items():
  print('Critical Values:')
  print(f'  {key}: {value}')

# p-value > 0.05, so the BoI returns are stationary
```

```
KPSS Statistic: 0.156992
p-value: 0.100000
Critical Values:
  10%: 0.347
Critical Values:
  5%: 0.463
Critical Values:
  2.5%: 0.574
Critical Values:
  1%: 0.739

/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/stattoo
ls.py:1910: InterpolationWarning:

The test statistic is outside of the range of p-values available i
n the
look-up table. The actual p-value is greater than the p-value retu
rned.
```

The ADF and KPSS tests have provided consistent results. Both tests show the returns are stationary.

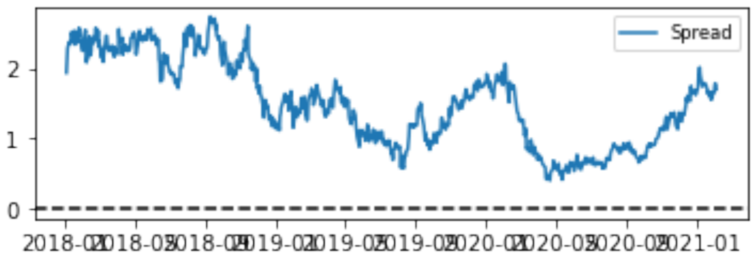
Are the stocks cointegrated?

Cointegration is a long term correlation between two time series

In [121]:
```
# First plot the prices seperately

plt.subplot(2,1,1)
plt.plot(prices['AIB'], label='AIB', color='red')
plt.plot(prices['BoI'], label= 'BoI', color='blue')
plt.legend(loc='best', fontsize='small')
```

Out[121]:
```
<matplotlib.legend.Legend at 0x7fac34316e20>
```

In [122]:
```
# Then plot the spread
# (The difference between the two series)

plt.subplot(2,1,2)
plt.plot(prices['BoI']-prices['AIB'], label='Spread')
plt.legend(loc='best', fontsize='small')
plt.axhline(y=0, linestyle='--', color='k')
plt.show()
```



In [123]:
```
# Reading CSV and transforming to returns again

prices = pd.read_csv('IrishBanks.csv', index_col=0)
prices.index=pd.to_datetime(prices.index, infer_datetime_format=Tru

# Transform stock prices to returns

prices['dAIB']= prices['AIB'].transform(lambda x : (x - x.shift(1))
prices['dBoI']= prices['BoI'].transform(lambda x : (x - x.shift(1))
prices = prices.dropna()
prices.head(3)
```

Out[123]:

Date	AIB	BoI	dAIB	dBoI
2018-01-03	5.435	7.370	-0.457875	2.432245
2018-01-04	5.450	7.545	0.275989	2.374491
2018-01-05	5.450	7.735	0.000000	2.518224

Choosing the Right Model

In [124]:
```
res_grid_AIB = smt.arma_order_select_ic(prices['dAIB'], max_ar=5, m
fit_kw={'method':'css-mle', 'solver':'bfgs'})

print("AIB's AIC")
print(res_grid_AIB.aic)
print("AIB's BIC")
print(res_grid_AIB.bic)

print(res_grid_AIB.aic_min_order)
print(res_grid_AIB.bic_min_order)

# AR on left of grid, MA on top
# Lowest BIC is (0,1)
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
```

```
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)



AIB's AIC
             0            1            2            3            4
\
0  4239.742145  4232.316699  4233.980140  4234.313432  4230.428997
1  4232.862823  4234.115510  4232.776366  4233.156728  4232.325456
2  4233.600092  4231.286373  4226.766873  4223.975109  4231.778120
3  4233.451308  4231.656318  4223.843649  4225.659227  4225.631624
4  4230.268716  4231.513625  4230.135319  4225.357985  4224.278309
5  4231.887147  4233.501176  4232.131124  4226.140746  4222.405513

             5
0  4232.376984
1  4234.311584
2  4233.739614
3  4226.004552
4  4217.088370
5  4222.240087
AIB's BIC
             0            1            2            3            4
\
0  4249.081141  4246.325193  4252.658132  4257.660922  4258.445986
1  4246.871317  4252.793502  4256.123857  4261.173716  4265.011943
2  4252.278084  4254.633863  4254.783861  4256.661595  4269.134104
3  4256.798799  4259.673307  4256.530136  4263.015212  4267.657107
4  4258.285704  4264.200112  4267.491303  4267.383468  4270.973290
5  4264.573634  4270.857161  4274.156607  4272.835726  4273.769992

             5
0  4265.063471
1  4271.667569
2  4275.765096
3  4272.699533
4  4268.452849
5  4278.274064
(4, 5)
(0, 1)
```

In [125]:
```
res_grid_BoI = smt.arma_order_select_ic(prices['dBoI'], max_ar=5, m
```

```
fit_kw={'method':'css-mle', 'solver':'bfgs'})

print("BoI's AIC")
print(res_grid_BoI.aic)
print("BoI's BIC")
print(res_grid_BoI.bic)

print(res_grid_BoI.aic_min_order)
print(res_grid_BoI.bic_min_order)

# Lowest BIC is (0,0)
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.
py:566: ConvergenceWarning:

Maximum Likelihood optimization failed to converge. Check mle_retv
als


BoI's AIC
             0            1            2            3            4
\
0  4117.299756  4118.479644  4119.887847  4120.457913  4121.249869
1  4118.528944  4120.325865  4121.471654  4121.372377  4123.001388
2  4119.726392  4121.330372  4122.889863  4118.576531  4120.830398
```

```
3  4120.566571  4121.664863  4119.295036  4120.344456  4122.042012
4  4121.717677  4123.420077  4113.793553  4120.818948  4122.202824
5  4123.371347  4125.284915  4117.744296  4115.384376  4119.048339

             5
0  4122.964717
1  4121.143434
2  4122.799270
3  4114.636237
4  4120.088931
5  4120.893646
BoI's BIC
             0            1            2            3            4
\
0  4126.638752  4132.488138  4138.565840  4143.805404  4149.266858
1  4132.537438  4139.003857  4144.819144  4149.389366  4155.687875
2  4138.404384  4144.677863  4150.906851  4151.263018  4158.186382
3  4143.914062  4149.681851  4151.981523  4157.700441  4164.067494
4  4149.734666  4156.106564  4151.149538  4162.844431  4168.897805
5  4156.057833  4162.640900  4159.769779  4162.079357  4170.412818

             5
0  4155.651203
1  4158.499419
2  4164.824753
3  4161.331218
4  4171.453410
5  4176.927623
(4, 2)
(0, 0)
```

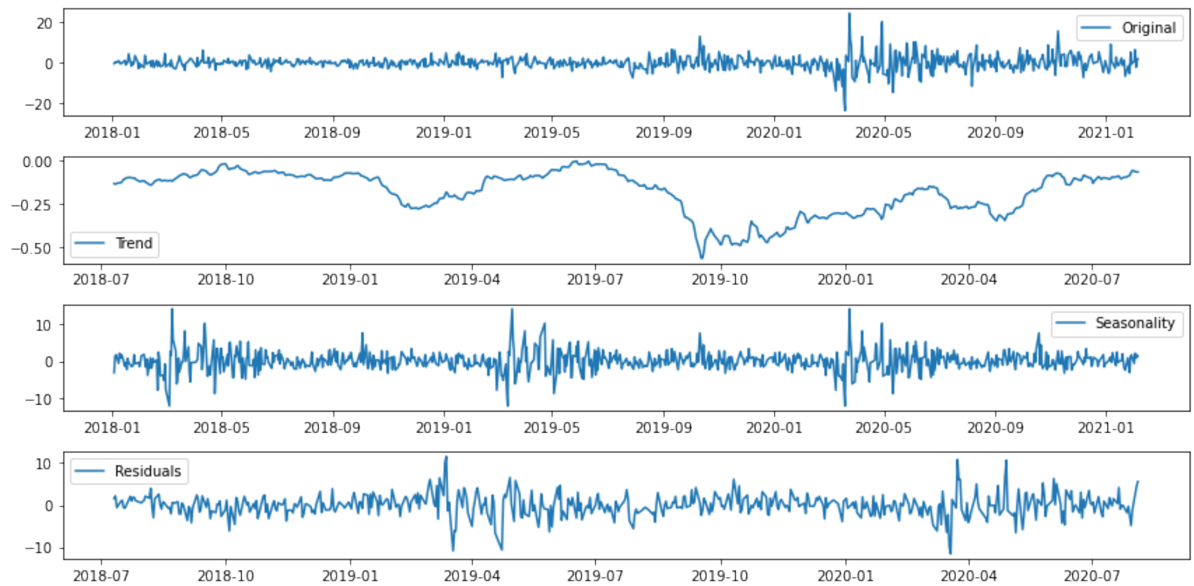
Part 2(a) Forecasting using ARMA Processes

2A: Show how you might forecast both series individually using their ARMA processes

In [126]:

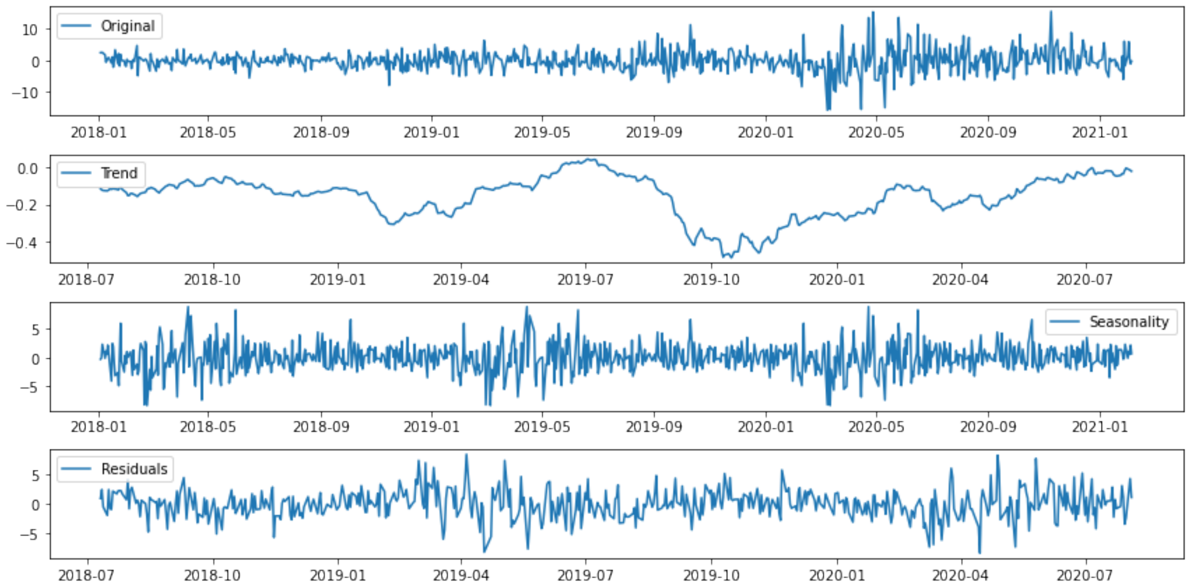
```
# Examine the seasonality of the returns
from statsmodels.tsa.seasonal import seasonal_decompose
decomposition = seasonal_decompose(prices.dAIB, freq = 260) #Was th
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.figure(figsize=(12,6))
plt.subplot(411)
plt.plot(prices['dAIB'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
plt.show()

# Shows stationarity in the returns, no trend
```

```
/var/folders/mh/j7bgd0wx1mb1jjnzcp0c8dfw0000gn/T/ipykernel_83727/2
9824519.py:3: FutureWarning:

the 'freq'' keyword is deprecated, use 'period' instead
```



In [127]:
```
# Examine the seasonality
decomposition = seasonal_decompose(prices.dBoI, freq = 260) #Was th
trend = decomposition.trend
seasonal = decomposition.seasonal
residual = decomposition.resid

plt.figure(figsize=(12,6))
plt.subplot(411)
plt.plot(prices['dBoI'], label='Original')
plt.legend(loc='best')
plt.subplot(412)
plt.plot(trend, label='Trend')
plt.legend(loc='best')
plt.subplot(413)
plt.plot(seasonal,label='Seasonality')
plt.legend(loc='best')
plt.subplot(414)
plt.plot(residual, label='Residuals')
plt.legend(loc='best')
plt.tight_layout()
plt.show()

# Shows stationarity in the returns, no trend
```

```
/var/folders/mh/j7bgd0wx1mb1jjnzcp0c8dfw0000gn/T/ipykernel_83727/2
200272331.py:2: FutureWarning:

the 'freq'' keyword is deprecated, use 'period' instead
```



In [128]:
```
# Make an Auto Regressive Moving Average model for AIB returns
# This function finds the best paramaters for Ar(p) and Ma(q)
from statsmodels.tsa.stattools import ARMA
def best_AR_MA_checker(df,lower,upper):
    from statsmodels.tsa.stattools import ARMA
    from statsmodels.tsa.stattools import adfuller
    arg=np.arange(lower,upper)
    arg1=np.arange(lower,upper)
    best_param_i=0
    best_param_j=0
    temp=12000000
    rs=99
    for i in arg:
        for j in arg1:
            model=ARMA(df, order=(i,0,j))
            result=model.fit(disp=0)
            resid=adfuller(result.resid)
            if (result.aic<temp and  adfuller(result.resid)[1]<0.05
                temp=result.aic
                best_param_i=i
                best_param_j=j
                rs=resid[1]


            print ("AR: %d, MA: %d, AIC: %d; resid stationarity che

    print("the following function prints AIC criteria and finds the
    print("best AR: %d, best MA: %d, best AIC: %d;  resid stationar
best_AR_MA_checker(prices.dAIB,0,4) #For each parameter I want to t
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:
```

In [129]:
```
# Above output might be difficult to read
# It states best ARMA Model is (1,0) for AIB
```

In [130]:
```
# Make an Auto Regressive Moving Average model for BoI returns
# This code finds the best paramaters for Ar(p) and Ma(q)
def best_AR_MA_checker(df,lower,upper):
    from statsmodels.tsa.stattools import ARMA
    from statsmodels.tsa.stattools import adfuller
    arg=np.arange(lower,upper)
    arg1=np.arange(lower,upper)
    best_param_i=0
    best_param_j=0
    temp=12000000
    rs=99
    for i in arg:
        for j in arg1:
            model=ARMA(df, order=(i,0,j))
            result=model.fit(disp=0)
            resid=adfuller(result.resid)
            if (result.aic<temp and  adfuller(result.resid)[1]<0.05
                temp=result.aic
                best_param_i=i
                best_param_j=j
                rs=resid[1]


            print ("AR: %d, MA: %d, AIC: %d; resid stationarity che

    print("the following function prints AIC criteria and finds the
    print("best AR: %d, best MA: %d, best AIC: %d;  resid stationar
best_AR_MA_checker(prices.dBoI,0,4) #For each parameter I want to t
```

```
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
```

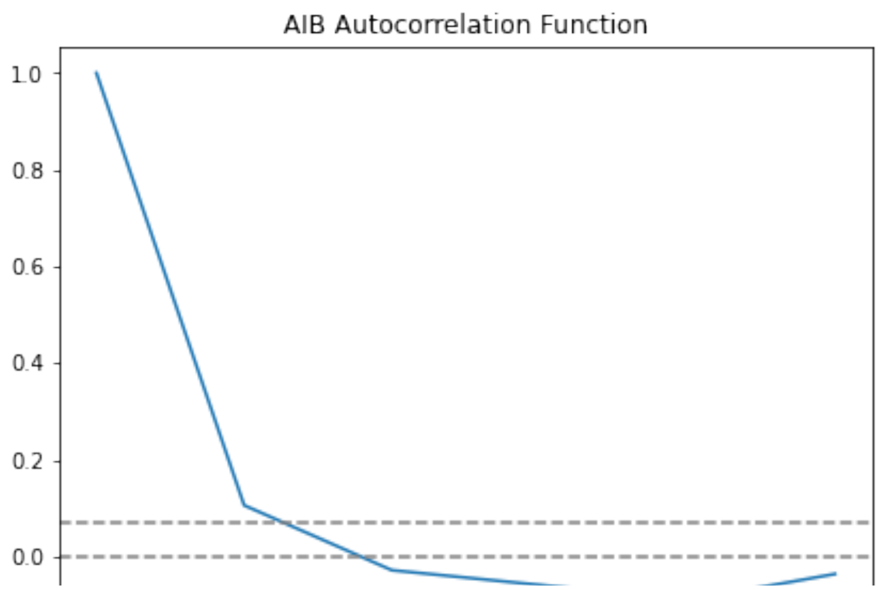
In [131]:
```
# Above says best ARMA model for BoI is (0,0)
# ARMA (1,0) has a very similar AIC
```

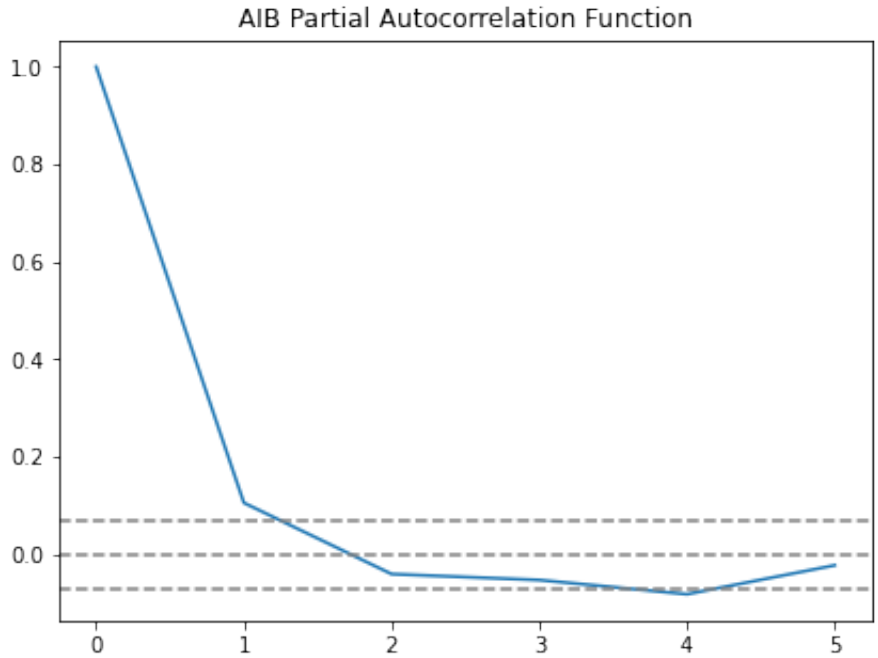
In [132]:
```
from statsmodels.tsa.stattools import acf, pacf
```

In [132]:
```
from statsmodels.tsa.stattools import acf, pacf

lag_acf = acf(prices['dAIB'], nlags=5)
lag_pacf = pacf(prices['dAIB'], nlags=5, method='ols')

# Plot ACF
plt.figure(figsize=(15,5))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0, linestyle='--', color='gray')
plt.axhline(y=-1.96/np.sqrt(len(prices['dAIB'])),linestyle='--', co
plt.axhline(y=1.96/np.sqrt(len(prices['dAIB'])),linestyle='--', col
plt.title('AIB Autocorrelation Function')

# Plot PACF
plt.figure(figsize=(15,5))
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0, linestyle='--', color='gray')
plt.axhline(y=-1.96/np.sqrt(len(prices['dAIB'])),linestyle='--', co
plt.axhline(y=1.96/np.sqrt(len(prices['dAIB'])),linestyle='--', col
plt.title('AIB Partial Autocorrelation Function')

# Look to see where the graph hits zero for the first time
# It cuts it at approximately 2 on Autocorrelation Graph
# Approx 2 on the partial autocorrelation graph
# Therefore the ARMA (p,q) model for AIB might be:
# ARMA (2,2)
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/stattoo
ls.py:667: FutureWarning:

fft=True will become the default after the release of the 0.12 rel
ease of statsmodels. To suppress this warning, explicitly set fft=
False.
```

Out[132]: Text(0.5, 1.0, 'AIB Partial Autocorrelation Function')





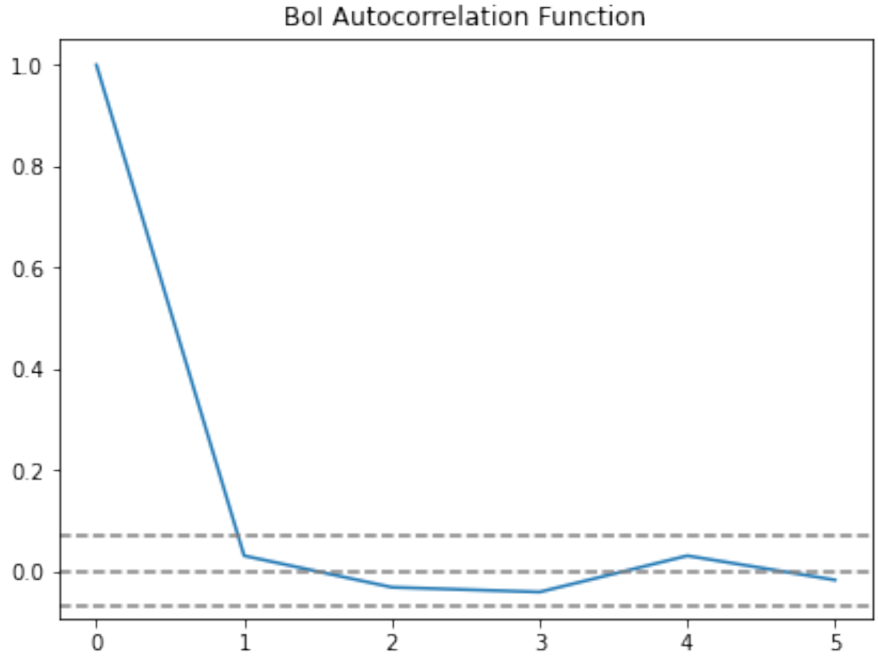
In [133]:

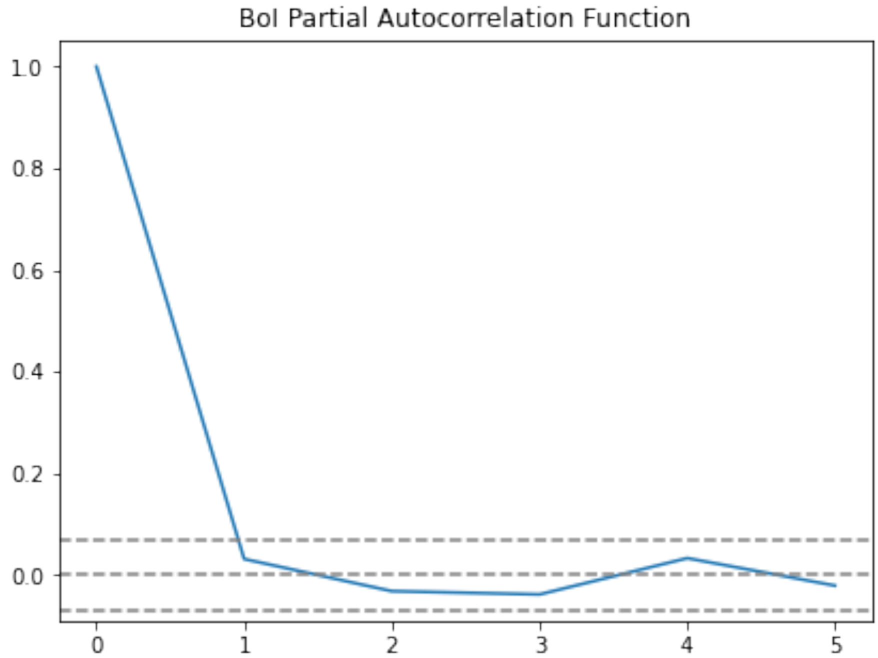
```
lag_acf = acf(prices['dBoI'], nlags=5)
lag_pacf = pacf(prices['dBoI'], nlags=5, method='ols')

# Plot ACF
plt.figure(figsize=(15,5))
plt.subplot(121)
plt.plot(lag_acf)
plt.axhline(y=0, linestyle='--', color='gray')
plt.axhline(y=-1.96/np.sqrt(len(prices['dBoI'])),linestyle='--', co
plt.axhline(y=1.96/np.sqrt(len(prices['dBoI'])),linestyle='--', col
plt.title('BoI Autocorrelation Function')

# Plot PACF
plt.figure(figsize=(15,5))
plt.subplot(122)
plt.plot(lag_pacf)
plt.axhline(y=0, linestyle='--', color='gray')
plt.axhline(y=-1.96/np.sqrt(len(prices['dBoI'])),linestyle='--', co
plt.axhline(y=1.96/np.sqrt(len(prices['dBoI'])),linestyle='--', col
plt.title('BoI Partial Autocorrelation Function')

# Look to see where the graph hits zero for the first time
# It cuts it at approximately 2 on Autocorrelation Graph
# Approx 1 on the partial autocorrelation graph
# Therefor the ARMA (p,q) model for BoI might be:
# ARMA (2,1)
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/stattoo
ls.py:667: FutureWarning:
```

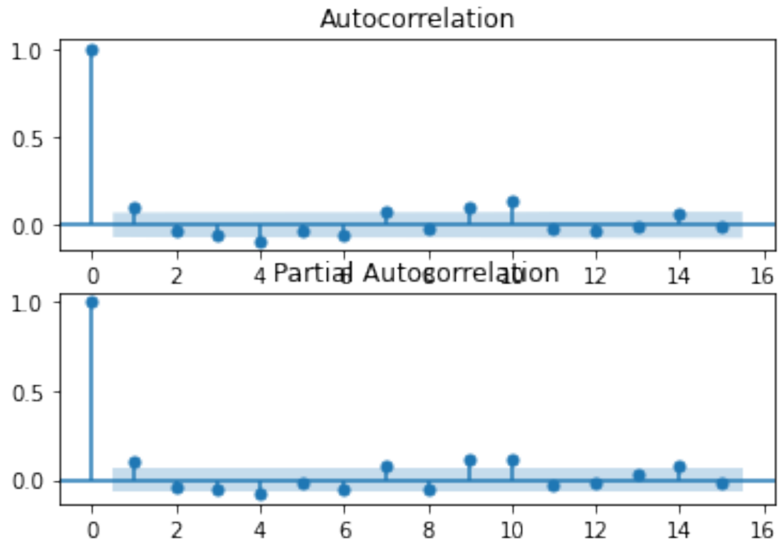
```
fft=True will become the default after the release of the 0.12 rel
ease of statsmodels. To suppress this warning, explicitly set fft=
False.
```

Out[133]: Text(0.5, 1.0, 'BoI Partial Autocorrelation Function')

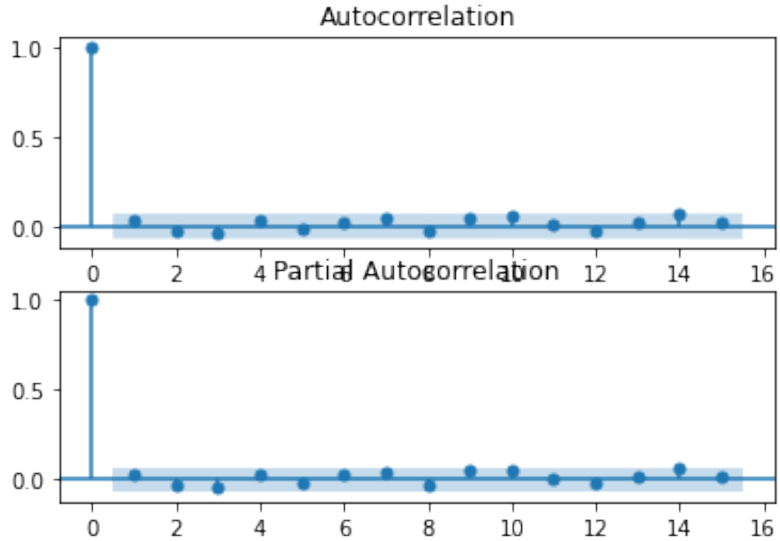




In [134]:
```
from statsmodels.graphics.tsaplots import plot_acf
from statsmodels.graphics.tsaplots import plot_pacf

# Compare ACF and PACF for AIB returns
fig, axes = plt.subplots(2,1)

# Plot ACF
plot_acf(prices['dAIB'], lags = 15, ax=axes[0])
#Plot PACF
plot_pacf(prices['dAIB'], lags=15, ax=axes[1])
plt.show()
```



In [135]:
```
# Compare ACF and PACF for BoI returns
fig, axes = plt.subplots(2,1)

# Plot ACF
plot_acf(prices['dBoI'], lags = 15, ax=axes[0])
#Plot PACF
plot_pacf(prices['dBoI'], lags=15, ax=axes[1])
plt.show()
```



In [136]:
```
# Estimate the AR for AIB returns
# Fit the data to an AR(p) for p = 0,...,6 , and save the BIC

from statsmodels.tsa.arima_model import ARMA

BIC = np.zeros(7)
for p in range(7):
  mod = ARMA(prices['dAIB'], order =(p,0))
  res = mod.fit()
# Save BIC for AR(p)
  BIC[p] = res.bic

# Plot the BIC as a function of p
plt.figure(figsize=(10,5))
plt.plot(range(1,7), BIC[1:7], marker='o')
plt.xlabel('Order of AR Model for AIB Returns')
plt.ylabel('Bayesian Information Criterion')
plt.show()

# Minimum AR at 1
```

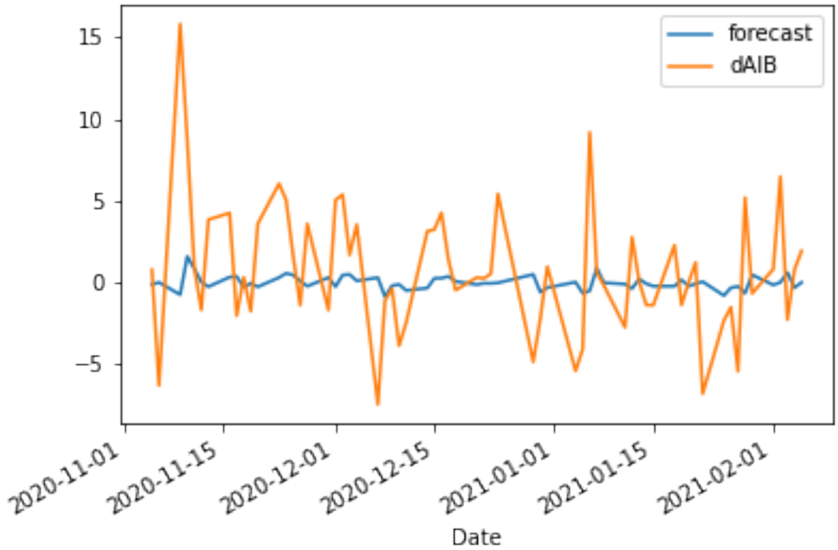
```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:
```

In [137]:
```
from statsmodels.tsa.arima_model import ARMA
# Forecast AIB returns using the first AR(1) model

mod = ARMA(prices['dAIB'], order=(1,0))
res = mod.fit()
res.plot_predict(start ='2020-11-05' ,end ='2021-02-05' )
plt.show()

# Arma model (1,0) used to predict last 6 months prices
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:
```

```
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.68202D+00    |proj g|=  6.64357D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
```

```
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      4       6      1     0     0   0.000D+00   2.682D+00
  F =   2.6820195575481245

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```



In [138]:
```
# Estimate the AR for BoI returns
# Fit the data to an AR(p) for p = 0,...,6 , and save the BIC
BIC = np.zeros(7)
for p in range(7):
  mod = ARMA(prices['dBoI'], order =(p,0))
  res = mod.fit()
# Save BIC for AR(p)
  BIC[p] = res.bic

# Plot the BIC as a function of p
plt.figure(figsize=(10,5))
plt.plot(range(1,7), BIC[1:7], marker='o')
plt.xlabel('Order of AR Model for BoI Returns')
plt.ylabel('Bayesian Information Criterion')
plt.show()

# Minimum AR at 1
```

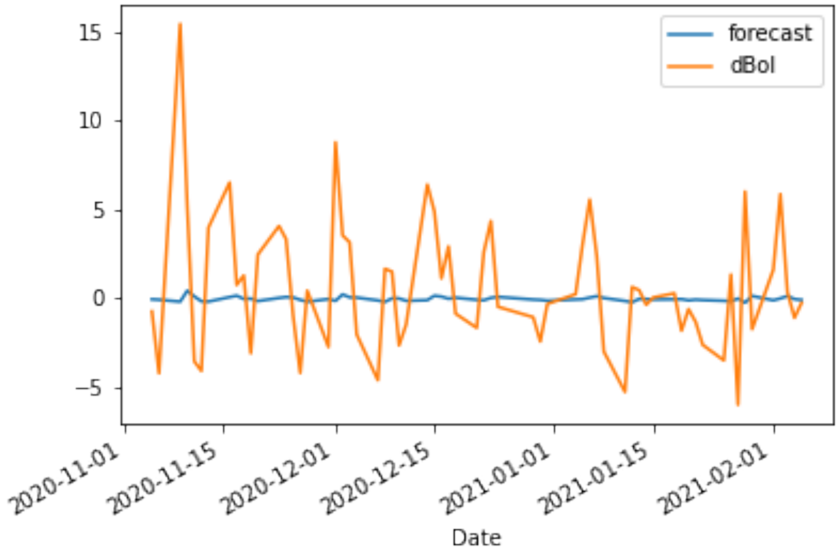
```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:
```

In [139]:
```
# Forecast BoI returns using the first AR(1) model

mod = ARMA(prices['dBoI'], order=(1,0))
res = mod.fit()
res.plot_predict(start ='2020-11-05' ,end ='2021-02-05' )
plt.show()

# Arma model (1,0) used to predict last 6 months prices
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
```

```
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.60947D+00    |proj g|=  2.90878D-04

At iterate    5    f=  2.60947D+00    |proj g|=  0.00000D+00

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value
```

```
           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      5      8      1     0     0   0.000D+00   2.609D+00
  F =   2.6094726799406538

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```

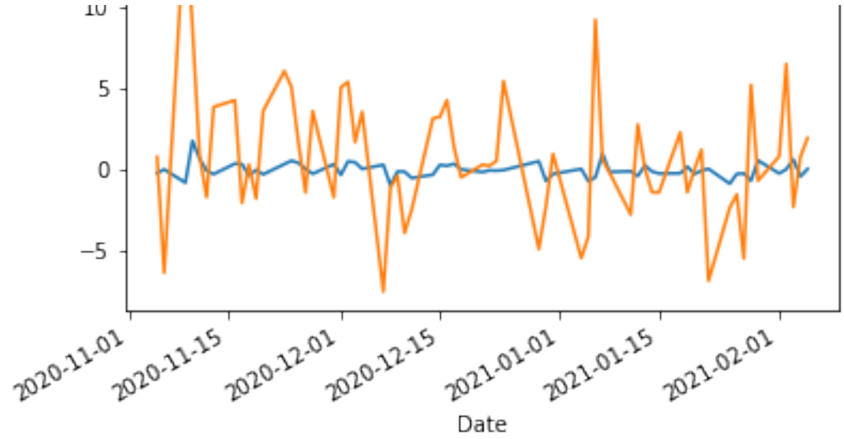


MA Models

In [140]:
```python
# Forecast AIB returns using the first MA(1) model

mod = ARMA(prices['dAIB'], order=(0,1))
res = mod.fit()
res.plot_predict(start ='2020-11-05' ,end ='2021-02-05' )
plt.show()

# Arma model (0,1) used to predict last 6 months prices
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
```

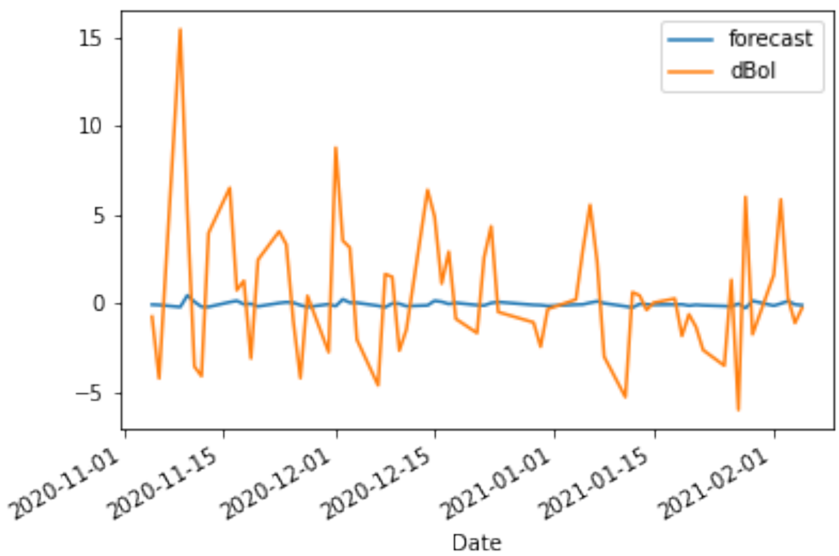
```
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.68167D+00    |proj g|=  6.99441D-05

At iterate    5    f=  2.68167D+00    |proj g|=  0.00000D+00

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      5      8      1     0     0   0.000D+00   2.682D+00
  F =   2.6816730321496633

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```



In [141]:
```
# Forecast BoI returns using the first MA(1) model

mod = ARMA(prices['dBoI'], order=(0,1))
res = mod.fit()
res.plot_predict(start ='2020-11-05' ,end ='2021-02-05' )
plt.show()

# Arma model (0,1) used to predict last 6 months prices
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:
```

```
A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.60944D+00    |proj g|=  1.01696D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      4      6      1     0     0   0.000D+00   2.609D+00
  F =   2.6094413981716196

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```



But Which ARMA Model is Best?

In [142]:
```
# Find Best ARMA Model for AIB

# Fit the data to an AR(1) model and print AIC:
mod_ar1 = ARMA(prices['dAIB'], order=(1, 0))
res_ar1 = mod_ar1.fit()
print("The AIC for an AR(1) is: ", res_ar1.aic)

# Fit the data to an AR(2) model and print AIC:
mod_ar2 = ARMA(prices['dAIB'], order=(2, 0))
res_ar2 = mod_ar2.fit()
print("The AIC for an AR(2) is: ", res_ar2.aic)

# Fit the data to an ARMA(1,1) model and print AIC:
mod_arma11 = ARMA(prices['dAIB'], order = (1,1))
res_arma11 = mod_arma11.fit()
print("The AIC for an ARMA(1,1) is: ", res_arma11.aic)

# AR(1) has the lowest AIC score of the three models - for AIB
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:
```

In [143]:
```
# Find Best ARMA Model for BoI

# Fit the data to an AR(1) model and print AIC:
mod_ar1 = ARMA(prices['dBoI'], order=(1, 0))
res_ar1 = mod_ar1.fit()
print("The AIC for an AR(1) is: ", res_ar1.aic)

# Fit the data to an AR(2) model and print AIC:
mod_ar2 = ARMA(prices['dBoI'], order=(2, 0))
res_ar2 = mod_ar2.fit()
print("The AIC for an AR(2) is: ", res_ar2.aic)

# Fit the data to an ARMA(1,1) model and print AIC:
mod_arma11 = ARMA(prices['dBoI'], order = (2,1))
res_arma11 = mod_arma11.fit()
print("The AIC for an ARMA(2,1) is: ", res_arma11.aic)

# Again AR(1) has the lowest AIC score of the three models for BoI
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.60947D+00    |proj g|=  2.90878D-04

At iterate    5    f=  2.60947D+00    |proj g|=  0.00000D+00

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
```

In [144]:
```
# Forecast AIB Returns using an ARIMA(1,0,0) model
from statsmodels.tsa.arima_model import ARIMA

mod = ARIMA(prices['dAIB'], order=(1,0,0))
res = mod.fit()

# Plot the original series and the forecasted series
res.plot_predict(start='2020-08-05', end='2021-02-05')
plt.show()
```

```
RUNNING THE L-BFGS-B CODE

           * * *
```

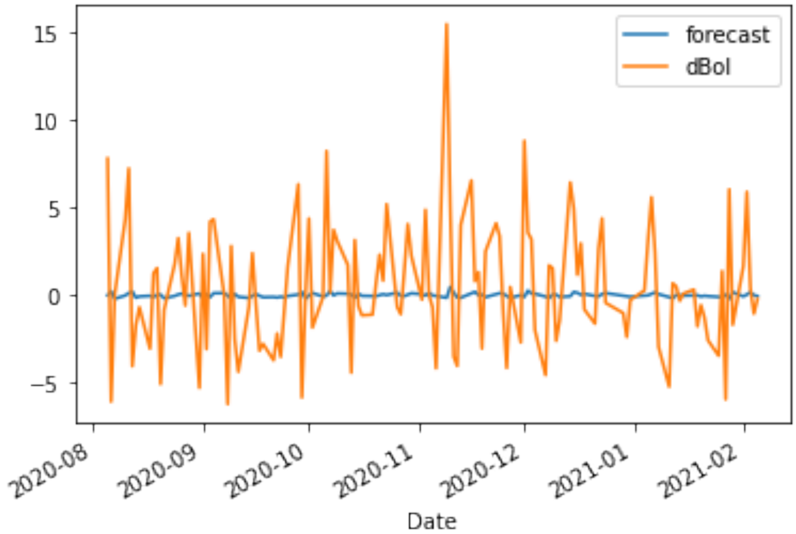
```
Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.68202D+00    |proj g|=  6.64357D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      4      6      1     0     0   0.000D+00   2.682D+00
  F =   2.6820195575481245

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL

/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.
```

In [145]:
```
# Forecast BoI Returns using an ARIMA(1,0,0) model

mod = ARIMA(prices['dBoI'], order=(1,0,0))
res = mod.fit()
```

```
# Plot the original series and the forecasted series
res.plot_predict(start='2020-08-05', end='2021-02-05')
plt.show()
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.60947D+00    |proj g|=  2.90878D-04

At iterate    5    f=  2.60947D+00    |proj g|=  0.00000D+00
```

```
           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      5      8      1     0     0   0.000D+00   2.609D+00
  F =   2.6094726799406538

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```



In [146]:

```
# Model AIB using BIC

model_AIB = smt.ARIMA(prices['dAIB'], order=(1,0,0))
res_AIB = model_AIB.fit()
print(res_AIB.summary())

# No differencing needs to be completed as the returns are stationa
# 1 Lag of AR, 0 Lags of MA
# Lag 1 is significant, p-vlalue < 0.003
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds
```

```
At iterate    0    f=  2.68202D+00    |proj g|=  6.64357D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      4      6      1     0     0   0.000D+00   2.682D+00
  F =   2.6820195575481245

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
                              ARMA Model Results
==================================================================
============
Dep. Variable:                   dAIB   No. Observations:
788
Model:                     ARMA(1, 0)   Log Likelihood
-2113.431
Method:                       css-mle   S.D. of innovations
3.536
Date:                Fri, 19 Nov 2021   AIC
4232.863
Time:                        10:46:32   BIC
4246.871
Sample:                             0   HQIC
4238.248

==================================================================
============
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------
------------
const         -0.0934      0.141     -0.663      0.507     -0.369
0.183
ar.L1.dAIB     0.1058      0.035      2.988      0.003      0.036
0.175
                                    Roots
==================================================================
===========
                  Real          Imaginary           Modulus
Frequency
------------------------------------------------------------------
-----------
AR.1             9.4510           +0.0000j            9.4510
0.0000
```

```
---------------------------------------------------------------------------

/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.
```

In [147]:

```
# Model BoI using BIC

model_BoI = smt.ARIMA(prices['dBoI'], order=(1,0,0))
res_BoI = model_BoI.fit()
print(res_BoI.summary())

# No differencing needs to be completed as the returns are stationa
# 1 Lag of AR, 0 Lags of MA
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:
```

```
statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.60947D+00    |proj g|=  2.90878D-04

At iterate    5    f=  2.60947D+00    |proj g|=  0.00000D+00

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
```

```
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      5      8      1     0     0   0.000D+00   2.609D+00
  F =   2.6094726799406538

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
                              ARMA Model Results
==================================================================
============
Dep. Variable:                    dBoI   No. Observations:
788
Model:                      ARMA(1, 0)   Log Likelihood
-2056.264
Method:                        css-mle   S.D. of innovations
3.289
Date:                 Fri, 19 Nov 2021   AIC
4118.529
Time:                         10:46:32   BIC
4132.537
Sample:                              0   HQIC
4123.914

==================================================================
============
                 coef    std err          z      P>|z|      [0.025
0.975]
------------------------------------------------------------------
------------
const         -0.0453      0.121     -0.374      0.708      -0.282
0.192
ar.L1.dBoI     0.0313      0.036      0.878      0.380      -0.039
0.101
                                    Roots
==================================================================
===========
                  Real          Imaginary           Modulus
Frequency
------------------------------------------------------------------
-----------
AR.1           31.9898           +0.0000j           31.9898
0.0000
------------------------------------------------------------------
-----------
```

In [148]:
```
# Two types of useful forecasts
# - static
# - dynamic

# Dynamic forecasts calculate multi-step forecasts starting from th
# Static forcasts imply a sequence of one-step ahead forecasts, rol
```

In [149]:
```
# Static - AIB

model_all_AIB = smt.ARIMA(prices['dAIB'], order =(1,0,0))
res_all_AIB = model_all_AIB.fit()
res_all_AIB.plot_predict('2020-08-05', '2021-02-05', dynamic=False)

# Last 6 months of prices
# Visually, Static looks like a better model than Dynamic for AIB r
```

```
RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.68202D+00    |proj g|=  6.64357D-05

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg       F
    2      4      6      1     0     0   0.000D+00   2.682D+00
  F =   2.6820195575481245

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL

/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
```

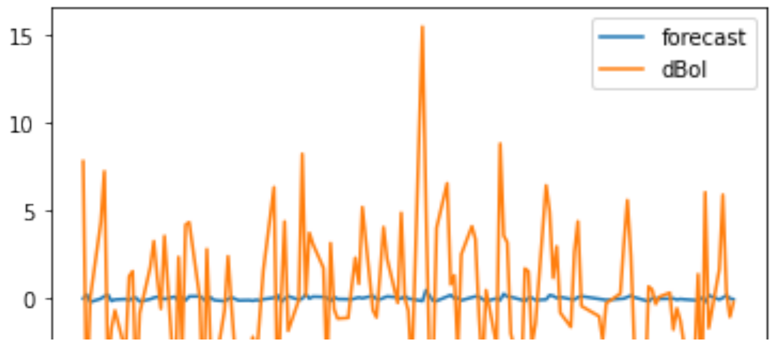
```
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.
```

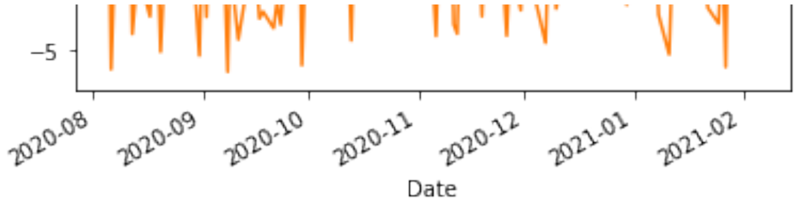
In [150]:
```
# Dynamic - AIB

res_all_AIB.plot_predict('2020-08-05', '2021-02-05', dynamic=True);

# Dynamic is much flatter
```

In [151]:
```
# Static - BoI

model_all_BoI = smt.ARIMA(prices['dBoI'], order =(1,0,0))
res_all_BoI = model_all_BoI.fit()
res_all_BoI.plot_predict('2020-08-05', '2021-02-05', dynamic=False)

# Last 6 months of prices
```

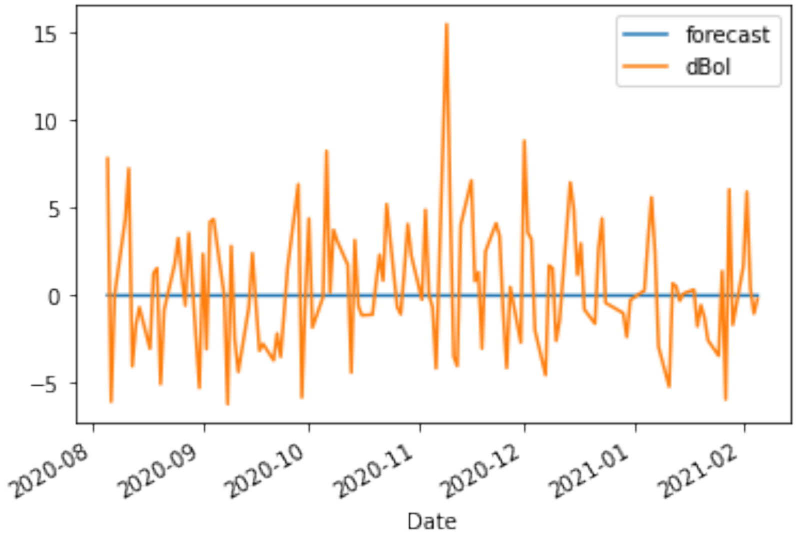
```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
```

```
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/base/ts
a_model.py:581: ValueWarning:

A date index has been provided, but it has no associated frequency
information and so will be ignored when e.g. forecasting.

 This problem is unconstrained.

RUNNING THE L-BFGS-B CODE

           * * *

Machine precision = 2.220D-16
 N =            2     M =           12

At X0         0 variables are exactly at the bounds

At iterate    0    f=  2.60947D+00    |proj g|=  2.90878D-04

At iterate    5    f=  2.60947D+00    |proj g|=  0.00000D+00

           * * *

Tit   = total number of iterations
Tnf   = total number of function evaluations
Tnint = total number of segments explored during Cauchy searches
Skip  = number of BFGS updates skipped
Nact  = number of active bounds at final generalized Cauchy point
Projg = norm of the final projected gradient
F     = final function value

           * * *

   N    Tit     Tnf  Tnint  Skip  Nact     Projg        F
    2      5      8      1     0     0   0.000D+00   2.609D+00
  F =   2.6094726799406538

CONVERGENCE: NORM_OF_PROJECTED_GRADIENT_<=_PGTOL
```





In [152]:
```
# Dynamic - BoI

res_all_BoI.plot_predict('2020-08-05', '2021-02-05', dynamic=True);
```



In [153]:
```
# Check the accuracy of the forecast - AIB

def rmse(pred, target):
  return np.sqrt(((pred - target)**2).mean())

prices_outsample = prices['2019-02-05': '2021-02-05']

pred1= res_all_AIB.predict('2019-02-05', '2021-02-05', dynamic=Fals
stats1= rmse(pred1, prices_outsample['dAIB'])
print('root mean squared error, static: {}'.format(stats1))

pred2= res_all_AIB.predict('2019-02-05', '2021-02-05', dynamic=True
stats2= rmse(pred2, prices_outsample['dAIB'])
print('root mean squared error, dynamic: {}'.format(stats2))

prices_outsample2 = prices['2018-01-03':'2021-02-05']
pred3= res_all_AIB.predict('2018-01-03', '2021-02-05', dynamic=Fals
stats3= rmse(pred3, prices_outsample2['dAIB'])
print('root mean squared error, long (whole period): {}'.format(sta

# The closer to 0 the better
# The static model is a slightly better predictor than the dynamic
# The static graphs did look better than the dynamic
```

```
root mean squared error, static: 4.208232505366341
root mean squared error, dynamic: 4.239276415649806
root mean squared error, long (whole period): 3.536275151420149
```

In [154]:
```
# Check the accuracy of the forecast - BoI

def rmse(pred, target):
  return np.sqrt(((pred - target)**2).mean())

prices_outsample = prices['2019-02-05': '2021-02-05']

pred4= res_all_BoI.predict('2019-02-05', '2021-02-05', dynamic=Fals
stats4= rmse(pred4, prices_outsample['dBoI'])
print('root mean squared error, static: {}'.format(stats4))

pred5= res_all_BoI.predict('2019-02-05', '2021-02-05', dynamic=True
stats5= rmse(pred5, prices_outsample['dBoI'])
print('root mean squared error, dynamic: {}'.format(stats5))

prices_outsample2 = prices['2018-01-03':'2021-02-05']
pred6= res_all_BoI.predict('2018-01-03', '2021-02-05', dynamic=Fals
stats6= rmse(pred3, prices_outsample2['dBoI'])
print('root mean squared error, long (whole period): {}'.format(sta

# The closer to 0 the better
# The static and dynamic models are similar for BoI
```

```
root mean squared error, static: 3.845590536643706
root mean squared error, dynamic: 3.849923383394704
root mean squared error, long (whole period): 3.278440574670801
```

Another ARIMA Method: Forecasting prices

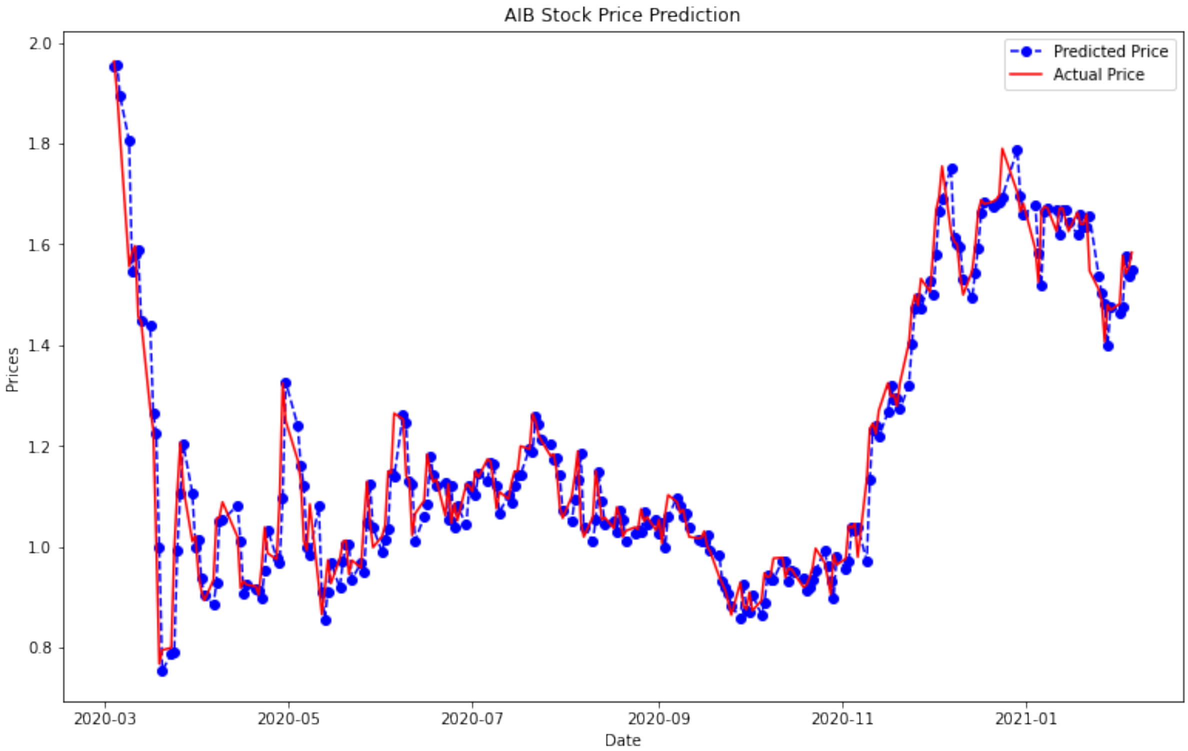
In [155]:
```
from sklearn.metrics import mean_squared_error

# Split AIB Prices into train and test data
train_data, test_data = prices[0:int(len(prices)*0.7)], prices[int(

training_data = train_data['AIB'].values
test_data = test_data['AIB'].values

history = [x for x in training_data]
model_predictions = []
N_test_observations = len(test_data)

# ARIMA (1,1,0) used
# As it is stock prices and not returns, 1 order of differencing ne
for time_point in range(N_test_observations):
  model = ARIMA(history, order =(1,1,0))
  model_fit = model.fit(disp=0)
  output = model_fit.forecast()
  yhat = output[0]
  model_predictions.append(yhat)
  true_test_value = test_data[time_point]
  history.append(true_test_value)
```

```
MSE_error = mean_squared_error(test_data, model_predictions)
print('Testing Mean Squared Error is {}'.format(MSE_error))

# Mean Squared Error is low
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)



Testing Mean Squared Error is 0.004069872225705441
```

In [156]:
```
prices['date']= prices.index
```

In [157]:
```
test_set_range = prices[int(len(prices)*0.7):].index

plt.figure(figsize=(13,8))
plt.plot(test_set_range, model_predictions, color='blue',
         marker='o', linestyle='dashed', label='Predicted Price')
plt.plot(test_set_range, test_data, color='red', label='Actual Pric
plt.title('AIB Stock Price Prediction')
plt.xlabel('Date')
plt.ylabel('Prices')
plt.legend()
plt.show()

# Predicted price maps the actual very well
```



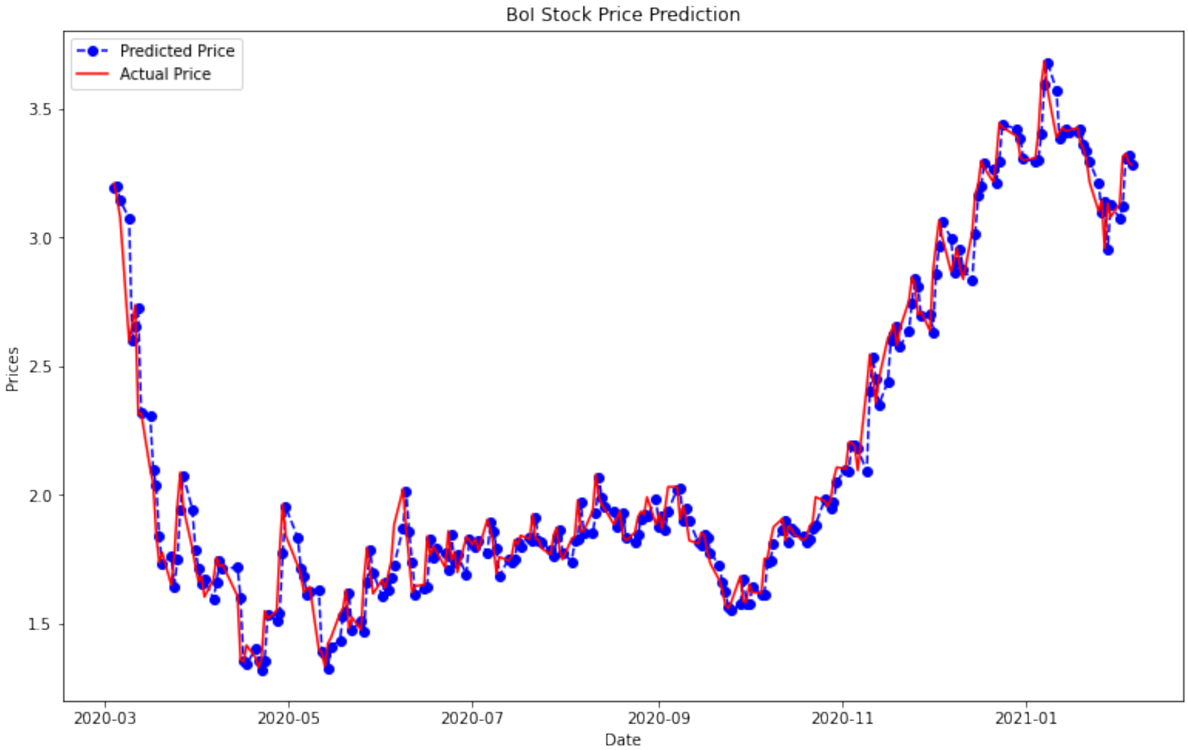
Now for BoI Stock Price

In [158]:
```
# Split BoI Prices into train and test data
train_data, test_data = prices[0:int(len(prices)*0.7)], prices[int(

training_data = train_data['BoI'].values
test_data = test_data['BoI'].values

history = [x for x in training_data]
model_predictions = []
N_test_observations = len(test_data)

# ARIMA (1,1,0) used
# As it is stock prices and not returns, 1 order of differencing ne
for time_point in range(N_test_observations):
  model = ARIMA(history, order =(1,1,0))
```

```
    model_fit = model.fit(disp=0)
    output = model_fit.forecast()
    yhat = output[0]
    model_predictions.append(yhat)
    true_test_value = test_data[time_point]
    history.append(true_test_value)

MSE_error = mean_squared_error(test_data, model_predictions)
print('Testing Mean Squared Error is {}'.format(MSE_error))

# Mean squared error again is low
```

```
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/arima_m
odel.py:472: FutureWarning:


statsmodels.tsa.arima_model.ARMA and statsmodels.tsa.arima_model.A
RIMA have
been deprecated in favor of statsmodels.tsa.arima.model.ARIMA (not
e the .
between arima and model) and
statsmodels.tsa.SARIMAX. These will be removed after the 0.12 rele
ase.

statsmodels.tsa.arima.model.ARIMA makes use of the statespace fram
ework and
is both well tested and maintained.

To silence this warning and continue using ARMA and ARIMA until th
ey are
removed, use:

import warnings
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARM
A',
                        FutureWarning)
warnings.filterwarnings('ignore', 'statsmodels.tsa.arima_model.ARI
MA',
                        FutureWarning)


/opt/anaconda3/lib/python3.9/site-packages/statsmodels/base/model.
py:566: ConvergenceWarning:

Maximum Likelihood optimization failed to converge. Check mle_retv
als


Testing Mean Squared Error is 0.010094015345757164
```

In [159]:
```
test_set_range = prices[int(len(prices)*0.7):].index

plt.figure(figsize=(13,8))
plt.plot(test_set_range, model_predictions, color='blue',
         marker='o', linestyle='dashed', label='Predicted Price')

plt.plot(test_set_range, test_data, color='red', label='Actual Pric

plt.title('BoI Stock Price Prediction')
plt.xlabel('Date')
plt.ylabel('Prices')
plt.legend()
plt.show()

# Again, predicted price maps the actual very well
```



Part 2(b) Analyse Volatility using GARCH

2B: Analyse the volatility of both series individually and demonstrate how a GARCH process can be used to model their volatility

In [160]:
```
from statsmodels.stats.diagnostic import het_arch
from statsmodels.compat import lzip
```

In [161]:
```
# Use a specialist module for ARCH

import sys
!{sys.executable} -m pip install arch
```

```
Collecting arch
  Downloading arch-5.0.1-cp39-cp39-macosx_10_9_x86_64.whl (876 kB)
     |████████████████████████████████| 876 kB 1.9 MB/s eta 0:00:0
1
Collecting property-cached>=1.6.4
  Downloading property_cached-1.6.4-py2.py3-none-any.whl (7.8 kB)
Requirement already satisfied: numpy>=1.17 in /opt/anaconda3/lib/p
ython3.9/site-packages (from arch) (1.20.3)
Requirement already satisfied: statsmodels>=0.11 in /opt/anaconda3
/lib/python3.9/site-packages (from arch) (0.12.2)
Requirement already satisfied: pandas>=1.0 in /opt/anaconda3/lib/p
ython3.9/site-packages (from arch) (1.3.4)
Requirement already satisfied: scipy>=1.3 in /opt/anaconda3/lib/py
thon3.9/site-packages (from arch) (1.7.1)
Requirement already satisfied: python-dateutil>=2.7.3 in /opt/anac
onda3/lib/python3.9/site-packages (from pandas>=1.0->arch) (2.8.2)
Requirement already satisfied: pytz>=2017.3 in /opt/anaconda3/lib/
python3.9/site-packages (from pandas>=1.0->arch) (2021.3)
Requirement already satisfied: six>=1.5 in /opt/anaconda3/lib/pyth
on3.9/site-packages (from python-dateutil>=2.7.3->pandas>=1.0->arc
h) (1.16.0)
Requirement already satisfied: patsy>=0.5 in /opt/anaconda3/lib/py
thon3.9/site-packages (from statsmodels>=0.11->arch) (0.5.2)
Installing collected packages: property-cached, arch
Successfully installed arch-5.0.1 property-cached-1.6.4
```

In [162]:
```
stock_returns = prices[['dAIB','dBoI']].copy()
```

In [163]:
```
# Detect ARCH effects in the data - AIB
data_arch = sm.add_constant(stock_returns['dAIB'])
res_arch = sm.OLS(data_arch['dAIB'], data_arch['const']).fit()
print(res_arch.summary())
```

```
                            OLS Regression Results
==================================================================
============
Dep. Variable:                   dAIB   R-squared:
-0.000
Model:                            OLS   Adj. R-squared:
-0.000
Method:                 Least Squares   F-statistic:
nan
Date:                Fri, 19 Nov 2021   Prob (F-statistic):
nan
Time:                        10:46:57   Log-Likelihood:
-2117.9
No. Observations:                 788   AIC:
4238.
```

```
Df Residuals:                     787   BIC:
4242.
Df Model:                           0
Covariance Type:            nonrobust

==============================================================================
============
                 coef    std err          t      P>|t|      [0.025
0.975]
------------------------------------------------------------------------------
------------
const         -0.0936      0.127     -0.739      0.460      -0.342
0.155
==============================================================================
============
Omnibus:                      153.926   Durbin-Watson:
1.788
Prob(Omnibus):                  0.000   Jarque-Bera (JB):
2722.788
Skew:                           0.313   Prob(JB):
0.00
Kurtosis:                      12.085   Cond. No.
1.00
==============================================================================
============

Notes:
[1] Standard Errors assume that the covariance matrix of the error
s is correctly specified.

/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/tsatool
s.py:142: FutureWarning:

In a future version of pandas all arguments of concat except for t
he argument 'objs' will be keyword-only
```

In [164]:

```
resAIB = het_arch(res_arch.resid,5)
name = ['lm', 'lmpval', 'fval', 'fpval']
lzip(name,resAIB)

# p-value of the Le Grange Multiplier shows the AIB returns definit
```

Out[164]:

```
[('lm', 179.37373729805307),
 ('lmpval', 7.280948968426872e-37),
 ('fval', 46.17870443765159),
 ('fpval', 8.130684806595673e-42)]
```

In [165]:

```
# Detect ARCH effects in the data - BoI
data_arch2 = sm.add_constant(stock_returns['dBoI'])
res_arch2 = sm.OLS(data_arch2['dBoI'], data_arch2['const']).fit()
print(res_arch2.summary())
```

```
                            OLS Regression Results
```

```
                            OLS Regression Results
==============================================================================
Dep. Variable:                   dBoI   R-squared:                       0.000
Model:                            OLS   Adj. R-squared:                  0.000
Method:                 Least Squares   F-statistic:                       nan
Date:                Fri, 19 Nov 2021   Prob (F-statistic):                nan
Time:                        10:46:57   Log-Likelihood:                -2056.6
No. Observations:                 788   AIC:                             4115.
Df Residuals:                     787   BIC:                             4120.
Df Model:                           0
Covariance Type:            nonrobust
==============================================================================
                 coef    std err          t      P>|t|      [0.025      0.975]
------------------------------------------------------------------------------
const         -0.0454      0.117     -0.387      0.699      -0.276       0.185
==============================================================================
Omnibus:                       82.059   Durbin-Watson:                   1.937
Prob(Omnibus):                  0.000   Jarque-Bera (JB):              549.364
Skew:                           0.124   Prob(JB):                   5.09e-120
Kurtosis:                       7.083   Cond. No.                         1.00
==============================================================================

Notes:
[1] Standard Errors assume that the covariance matrix of the errors is correctly specified.
/opt/anaconda3/lib/python3.9/site-packages/statsmodels/tsa/tsatools.py:142: FutureWarning:

In a future version of pandas all arguments of concat except for the argument 'objs' will be keyword-only
```

In [166]:
```
resBoI = het_arch(res_arch2.resid,5)
name = ['lm', 'lmpval', 'fval', 'fpval']
lzip(name,resBoI)

# p-value of the Le Grange Multiplier shows the BoI returns definit
```

Out[166]:
```
[('lm', 85.20678278339041),
 ('lmpval', 6.812577134688816e-17),
 ('fval', 18.97572764802119),
 ('fpval', 7.809618398613308e-18)]
```

In [167]:
```
# GARCH model - AIB
from arch import arch_model

am = arch_model(stock_returns['dAIB'], vol='GARCH')
res = am.fit()
print(res.summary())

# Alpha & beta are both statisitcally significant
# We have come up with a model of volatility that is sufficient
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 4144.5300386
72261
Iteration:      2,   Func. Count:     14,   Neg. LLF: 5641441.8661
56031
Iteration:      3,   Func. Count:     20,   Neg. LLF: 3393.0743669
32486
Iteration:      4,   Func. Count:     26,   Neg. LLF: 1914.2272121
198616
Iteration:      5,   Func. Count:     32,   Neg. LLF: 1903.3433437
556175
Iteration:      6,   Func. Count:     37,   Neg. LLF: 1902.7538937
305376
Iteration:      7,   Func. Count:     42,   Neg. LLF: 2822.2556007
00125
Iteration:      8,   Func. Count:     49,   Neg. LLF: 1904.9451084
146092
Iteration:      9,   Func. Count:     55,   Neg. LLF: 1903.5571551
31903
Iteration:     10,   Func. Count:     61,   Neg. LLF: 1902.5703476
964213
Iteration:     11,   Func. Count:     66,   Neg. LLF: 1902.5702694
25189
Iteration:     12,   Func. Count:     71,   Neg. LLF: 1902.5701781
388946
Iteration:     13,   Func. Count:     76,   Neg. LLF: 1902.5701765
835415
Iteration:     14,   Func. Count:     80,   Neg. LLF: 1902.5701765
84205
Optimization terminated successfully    (Exit mode 0)
            Current function value: 1902.5701765835415
            Iterations: 14
            Function evaluations: 80
```

```
            Gradient evaluations: 14
                         Constant Mean - GARCH Model Results
==================================================================
============
Dep. Variable:                       dAIB   R-squared:
0.000
Mean Model:                 Constant Mean   Adj. R-squared:
0.000
Vol Model:                          GARCH   Log-Likelihood:
-1902.57
Distribution:                      Normal   AIC:
3813.14
Method:                Maximum Likelihood   BIC:
3831.82
                                            No. Observations:
788
Date:                    Fri, Nov 19 2021   Df Residuals:
787
Time:                            10:47:01   Df Model:
1
                               Mean Model
==================================================================
========
                 coef    std err          t      P>|t|    95.0% Co
nf. Int.
------------------------------------------------------------------
--------
mu            -0.0965  7.580e-02     -1.273      0.203 [ -0.245,5.
209e-02]
                             Volatility Model
==================================================================
=========
                 coef    std err          t      P>|t|     95.0% C
onf. Int.
------------------------------------------------------------------
---------
omega          0.1174  8.458e-02      1.388      0.165 [-4.839e-02
,  0.283]
alpha[1]       0.1283  5.921e-02      2.167  3.025e-02  [1.224e-02
,  0.244]
beta[1]        0.8694  5.430e-02     16.012  1.054e-57    [  0.763
,  0.976]
==================================================================
=========

Covariance estimator: robust
```

In [168]:

```python
# E-GARCH - AIB

am_eg = arch_model(stock_returns['dAIB'], vol='EGARCH', o=1)
res_eg = am_eg.fit()
print(res_eg.summary())

# alpha, gamma & beta all statistically significant
```

```
# alpha, gamma & beta all statistically significant
# The gamma term shows that not only for the volatility of AIB retu
# but also some asymmetric affect between positive and negative vol
```

```
Iteration:      1,   Func. Count:      7,   Neg. LLF: 1886586894.1
210165
Iteration:      2,   Func. Count:     17,   Neg. LLF: 4249.0868425
70632
Iteration:      3,   Func. Count:     27,   Neg. LLF: 127087373581
.51004
Iteration:      4,   Func. Count:     37,   Neg. LLF: 12888385617.
320312
Iteration:      5,   Func. Count:     46,   Neg. LLF: 1903.6713332
80587
Iteration:      6,   Func. Count:     53,   Neg. LLF: 425984074.75
55089
Iteration:      7,   Func. Count:     61,   Neg. LLF: 1903.2754498
366771
Iteration:      8,   Func. Count:     68,   Neg. LLF: 1899.6810227
391659
Iteration:      9,   Func. Count:     74,   Neg. LLF: 1899.6686394
735564
Iteration:     10,   Func. Count:     80,   Neg. LLF: 1899.6681482
66571
Iteration:     11,   Func. Count:     86,   Neg. LLF: 1899.6681429
520604
Iteration:     12,   Func. Count:     92,   Neg. LLF: 1899.6681416
80943
Iteration:     13,   Func. Count:     97,   Neg. LLF: 1899.6681416
803965
Optimization terminated successfully    (Exit mode 0)
            Current function value: 1899.668141680943
            Iterations: 13
            Function evaluations: 97
            Gradient evaluations: 13
                     Constant Mean - EGARCH Model Results
==================================================================
============
Dep. Variable:                   dAIB   R-squared:
0.000
Mean Model:             Constant Mean   Adj. R-squared:
0.000
Vol Model:                     EGARCH   Log-Likelihood:
-1899.67
Distribution:                  Normal   AIC:
3809.34
Method:            Maximum Likelihood   BIC:
3832.68
                                        No. Observations:
788
Date:                Fri, Nov 19 2021   Df Residuals:
787
Time:                        10:47:01   Df Model:
1
```

```
                               Mean Model
==================================================================
=========
                 coef    std err          t       P>|t|     95.0% C
onf. Int.
------------------------------------------------------------------
---------
mu            -0.1697  7.845e-02     -2.163  3.054e-02 [ -0.323,-1
.593e-02]
                             Volatility Model
==================================================================
===========
                 coef    std err          t       P>|t|       95.0%
Conf. Int.
------------------------------------------------------------------
-----------
omega          0.0321  2.317e-02      1.385       0.166 [-1.333e-02
,7.751e-02]
alpha[1]       0.1915  9.673e-02      1.980  4.773e-02    [1.913e-
03,  0.381]
gamma[1]      -0.0525  2.492e-02     -2.108  3.500e-02   [ -0.101,
-3.699e-03]
beta[1]        0.9892  8.971e-03    110.271       0.000       [  0.9
72,  1.007]
==================================================================
===========

Covariance estimator: robust
```

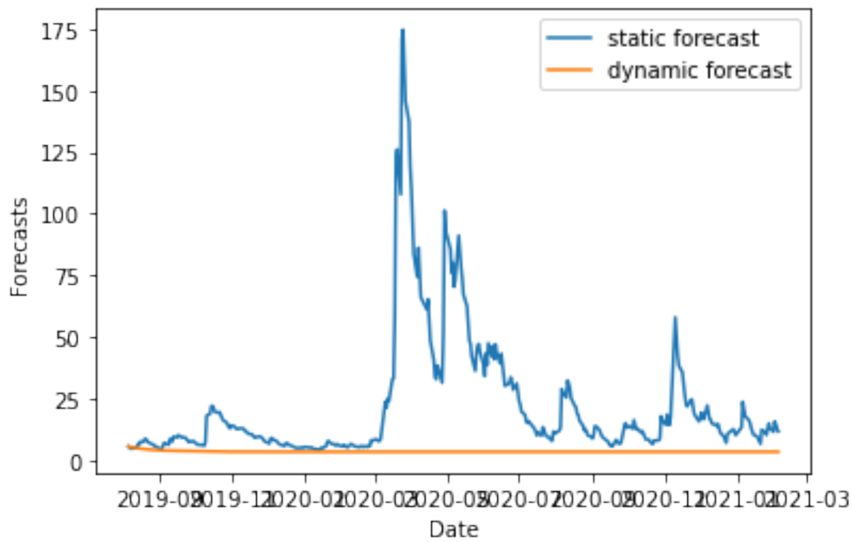
In [169]:

```
# Tne next graph was printing multiple pages of warnings so the fol
# Not recommended in practice, but for this purpose it should be ok

import warnings
warnings.filterwarnings("ignore")
```

In [170]:

```
# Finish by forecasting using GARCH - AIB

# Create two data samples

data_in_the_sample = stock_returns.loc[:'2019-08-05','dAIB']
data_out_of_the_sample = stock_returns.loc['2019-08-06':,'dAIB']

# Static forecasting

am = arch_model(stock_returns['dAIB'], vol='Garch')
cvar_dAIB_stat = {}
for date in data_out_of_the_sample.index:
  res = am.fit(last_obs=date, disp='off')
  forecasts = res.forecast(horizon=1)
  forecasts_res = forecasts.variance.dropna()
  cvar_dAIB_stat[date] = forecasts_res.iloc[1]
cvar_dAIB_stat = pd.DataFrame(cvar_dAIB_stat).T
```

```
# Dynamic Forecasting

res = am.fit(last_obs= '2019-08-06', disp='off')
forecasts = res.forecast(horizon=len(data_out_of_the_sample))
forecasts_res = forecasts.variance.dropna()
cvar_dAIB_dyn = pd.DataFrame(data= forecasts_res.iloc[1].values,\
                             columns=['dynamic forecasting'],\
                             index=data_out_of_the_sample.index)

# Now chart

plt.figure(1)
plt.plot(cvar_dAIB_stat, label = 'static forecast')
plt.plot(cvar_dAIB_dyn, label = 'dynamic forecast')
plt.xlabel('Date')
plt.ylabel('Forecasts')
plt.legend()
plt.show()

warnings.warn("first example of warning!", DeprecationWarning)
```



In [171]:
```
# Static forecast above is much more volatile
# It is not reasonable that we can predict the massive volatility
# Dynamic is probably better for this time series
```

In [172]:
```
# GARCH model - BoI
from arch import arch_model

am2 = arch_model(stock_returns['dBoI'], vol='GARCH')
res2 = am.fit()
print(res2.summary())

# Alpha & beta are both statisitcally significant
# We have come up with a model that is sufficient
```

```
Iteration:      1,   Func. Count:      6,   Neg. LLF: 4144.5300386
```

```
72261
Iteration:      2,   Func. Count:     14,   Neg. LLF: 5641441.8661
56031
Iteration:      3,   Func. Count:     20,   Neg. LLF: 3393.0743669
32486
Iteration:      4,   Func. Count:     26,   Neg. LLF: 1914.2272121
198616
Iteration:      5,   Func. Count:     32,   Neg. LLF: 1903.3433437
556175
Iteration:      6,   Func. Count:     37,   Neg. LLF: 1902.7538937
305376
Iteration:      7,   Func. Count:     42,   Neg. LLF: 2822.2556007
00125
Iteration:      8,   Func. Count:     49,   Neg. LLF: 1904.9451084
146092
Iteration:      9,   Func. Count:     55,   Neg. LLF: 1903.5571551
31903
Iteration:     10,   Func. Count:     61,   Neg. LLF: 1902.5703476
964213
Iteration:     11,   Func. Count:     66,   Neg. LLF: 1902.5702694
25189
Iteration:     12,   Func. Count:     71,   Neg. LLF: 1902.5701781
388946
Iteration:     13,   Func. Count:     76,   Neg. LLF: 1902.5701765
835415
Iteration:     14,   Func. Count:     80,   Neg. LLF: 1902.5701765
84205
Optimization terminated successfully    (Exit mode 0)
            Current function value: 1902.5701765835415
            Iterations: 14
            Function evaluations: 80
            Gradient evaluations: 14
                     Constant Mean - GARCH Model Results
==================================================================
============
Dep. Variable:                     dAIB   R-squared:
0.000
Mean Model:               Constant Mean   Adj. R-squared:
0.000
Vol Model:                        GARCH   Log-Likelihood:
-1902.57
Distribution:                    Normal   AIC:
3813.14
Method:              Maximum Likelihood   BIC:
3831.82
                                          No. Observations:
788
Date:                  Fri, Nov 19 2021   Df Residuals:
787
Time:                          10:47:15   Df Model:
1
                                Mean Model
==================================================================
========
```

```
                    coef    std err          t      P>|t|    95.0% Co
nf. Int.
---------------------------------------------------------------------
--------
mu               -0.0965  7.580e-02     -1.273      0.203 [ -0.245,5.
209e-02]
                              Volatility Model
=====================================================================
=========
                    coef    std err          t      P>|t|     95.0% C
onf. Int.
---------------------------------------------------------------------
---------
omega             0.1174  8.458e-02      1.388      0.165 [-4.839e-02
,  0.283]
alpha[1]          0.1283  5.921e-02      2.167  3.025e-02  [1.224e-02
,  0.244]
beta[1]           0.8694  5.430e-02     16.012  1.054e-57    [  0.763
,  0.976]
=====================================================================
=========

Covariance estimator: robust
```

In [173]:

```python
# E-GARCH - BoI

am_eg2 = arch_model(stock_returns['dBoI'], vol='EGARCH', o=1)
res_eg2 = am_eg2.fit()
print(res_eg2.summary())

# gamma & beta both statistically significant
# The gamma term shows that not only for the volatility of AIB retu
# but also some asymmetric affect between positive and negative vol
```

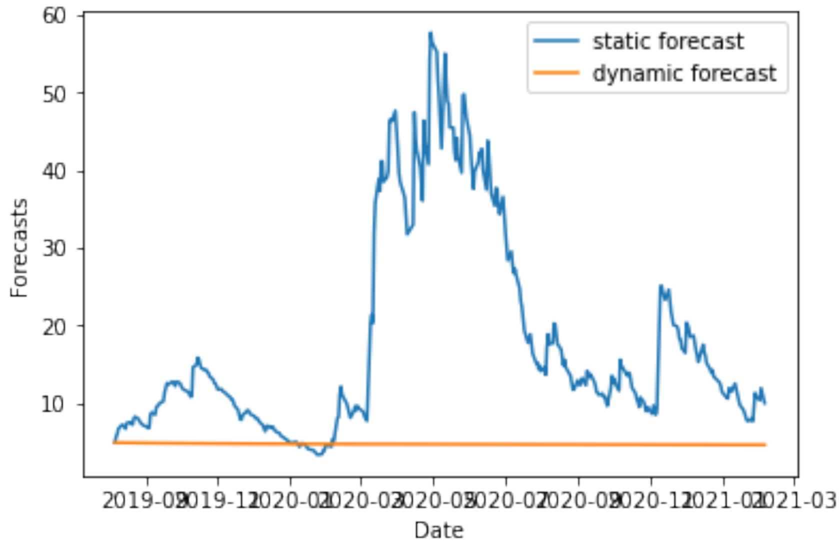
```
Iteration:      1,   Func. Count:      7,   Neg. LLF: 6929.1510965
33769
Iteration:      2,   Func. Count:     18,   Neg. LLF: 6236624667.3
22483
Iteration:      3,   Func. Count:     28,   Neg. LLF: 312014643647
.889
Iteration:      4,   Func. Count:     38,   Neg. LLF: 3284949847.6
14312
Iteration:      5,   Func. Count:     46,   Neg. LLF: 1918.6792505
186056
Iteration:      6,   Func. Count:     52,   Neg. LLF: 1974.5060670
68275
Iteration:      7,   Func. Count:     61,   Neg. LLF: 8914.0224633
8648
Iteration:      8,   Func. Count:     72,   Neg. LLF: 1919.3403510
141159
Iteration:      9,   Func. Count:     79,   Neg. LLF: 437347090.48
57292
Iteration:     10,   Func. Count:     86,   Neg. LLF: 2783616348.8
```

```
1074
Iteration:     11,   Func. Count:     95,   Neg. LLF: 51612633077.
75505
Iteration:     12,   Func. Count:    105,   Neg. LLF: 177886530632
9.2637
Iteration:     13,   Func. Count:    115,   Neg. LLF: 474644985.89
41885
Iteration:     14,   Func. Count:    123,   Neg. LLF: 131999712952
6.8967
Iteration:     15,   Func. Count:    133,   Neg. LLF: 1912.9831260
28099
Iteration:     16,   Func. Count:    140,   Neg. LLF: 3547.9718822
67667
Iteration:     17,   Func. Count:    148,   Neg. LLF: 1906.0359251
761352
Iteration:     18,   Func. Count:    154,   Neg. LLF: 142346695804
4.9404
Iteration:     19,   Func. Count:    165,   Neg. LLF: 517392150974
7.839
Iteration:     20,   Func. Count:    176,   Neg. LLF: 1095157.6992
387641
Iteration:     21,   Func. Count:    183,   Neg. LLF: 1831627.9698
458614
Iteration:     22,   Func. Count:    190,   Neg. LLF: 770127.01794
12183
Iteration:     23,   Func. Count:    197,   Neg. LLF: 775628.61253
27232
Iteration:     24,   Func. Count:    205,   Neg. LLF: 780050.46935
11531
Iteration:     25,   Func. Count:    213,   Neg. LLF: 2935012.4677
572455
Iteration:     26,   Func. Count:    221,   Neg. LLF: 1061935.6016
40977
Iteration:     27,   Func. Count:    237,   Neg. LLF: 2856924.9780
078926
Iteration:     28,   Func. Count:    250,   Neg. LLF: 2522636.0644
91747
Iteration:     29,   Func. Count:    263,   Neg. LLF: 2510985.0695
961746
Iteration:     30,   Func. Count:    279,   Neg. LLF: 2379791.5131
772477
Iteration:     31,   Func. Count:    286,   Neg. LLF: 2425206.2516
00474
Iteration:     32,   Func. Count:    294,   Neg. LLF: 1636262.8747
66716
Iteration:     33,   Func. Count:    310,   Neg. LLF: 1597972.5947
601595
Iteration:     34,   Func. Count:    326,   Neg. LLF: 1597252.0458
914507
Iteration:     35,   Func. Count:    342,   Neg. LLF: 1597110.6325
41253
Iteration:     36,   Func. Count:    358,   Neg. LLF: 1597023.0108
45362
Iteration:     37,   Func. Count:    374,   Neg. LLF: 1601776.9565
```

```
507097
Iteration:     38,   Func. Count:    390,   Neg. LLF: 518761.64510
192344
Iteration:     39,   Func. Count:    406,   Neg. LLF: 145797257287
77.816
Optimization terminated successfully    (Exit mode 0)
            Current function value: 1902.2204114291446
            Iterations: 40
            Function evaluations: 416
            Gradient evaluations: 39
                     Constant Mean - EGARCH Model Results
==================================================================
============
Dep. Variable:                      dBoI   R-squared:
0.000
Mean Model:                Constant Mean   Adj. R-squared:
0.000
Vol Model:                        EGARCH   Log-Likelihood:
-1902.22
Distribution:                     Normal   AIC:
3814.44
Method:               Maximum Likelihood   BIC:
3837.79
                                           No. Observations:
788
Date:                   Fri, Nov 19 2021   Df Residuals:
787
Time:                           10:47:16   Df Model:
1
                                Mean Model
==================================================================
======
                 coef    std err          t      P>|t|  95.0% Conf
. Int.
------------------------------------------------------------------
------
mu            -0.2019  1.447e-04  -1395.082      0.000 [ -0.202, -
0.202]
                                Volatility Model
==================================================================
============
                 coef    std err          t      P>|t|        95.0
% Conf. Int.
------------------------------------------------------------------
------------
omega      6.7477e-03  8.617e-10  7.831e+06      0.000   [6.748e-0
3,6.748e-03]
alpha[1]      -0.0283  6.563e-05   -430.887      0.000 [-2.841e-02
,-2.815e-02]
gamma[1]      -0.0619  6.302e-05   -982.243      0.000 [-6.202e-02
,-6.178e-02]
beta[1]        0.9980  1.812e-09  5.507e+08      0.000        [  0.
998,  0.998]
==================================================================
```

```
============

Covariance estimator: robust
```

In [174]:
```
# Finish by forecasting using GARCH - BoI

# Create two data samples

data_in_the_sample = stock_returns.loc[:'2019-08-05','dBoI']
data_out_of_the_sample = stock_returns.loc['2019-08-06':,'dBoI']

# Static forecasting

am = arch_model(stock_returns['dBoI'], vol='Garch')
cvar_dBoI_stat = {}
for date in data_out_of_the_sample.index:
  res = am.fit(last_obs=date, disp='off')
  forecasts = res.forecast(horizon=1)
  forecasts_res = forecasts.variance.dropna()
  cvar_dBoI_stat[date] = forecasts_res.iloc[1]
cvar_dBoI_stat = pd.DataFrame(cvar_dBoI_stat).T

# Dynamic Forecasting

res = am.fit(last_obs= '2019-08-06', disp='off')
forecasts = res.forecast(horizon=len(data_out_of_the_sample))
forecasts_res = forecasts.variance.dropna()
cvar_dBoI_dyn = pd.DataFrame(data= forecasts_res.iloc[1].values,\
                             columns=['dynamic forecasting'],\
                             index=data_out_of_the_sample.index)

# Now chart

plt.figure(1)
plt.plot(cvar_dBoI_stat, label = 'static forecast')
plt.plot(cvar_dBoI_dyn, label = 'dynamic forecast')
plt.xlabel('Date')
plt.ylabel('Forecasts')
plt.legend()
plt.show()
```



In [175]:
```
# Static forecast above is much more volatile
# Not reasonable that we can predict the massive volatility
# Dynamic is probably better for this time series
```

Part 2(c) VAR Model

2C: Develop a VAR model of the two series and show how one series might influence the other

In [176]:
```
prices.corr()

# 98.4% correlation between the stock prices
# 73.6% correlation between the stock returns
```

Out[176]:

	AIB	BoI	dAIB	dBoI
AIB	1.000000	0.983994	0.004356	-0.007269
BoI	0.983994	1.000000	0.002867	0.006741
dAIB	0.004356	0.002867	1.000000	0.736466
dBoI	-0.007269	0.006741	0.736466	1.000000

In [177]:
```
stock_returns = prices[['dAIB','dBoI']].copy()
```

In [178]:
```
from statsmodels.tsa.vector_ar.vecm import coint_johansen
```

In [179]:
```
# Test for cointegration of the stock returns

def cointegration_test(stock_returns, alpha=0.05):
    """Perform Johanson's Cointegration Test and Report Summary"""
    out = coint_johansen(stock_returns,-1,5)
    d = {'0.90':0, '0.95':1, '0.99':2}
    traces = out.lr1
    cvts = out.cvt[:, d[str(1-alpha)]]
    def adjust(val, length= 6): return str(val).ljust(length)

    # Summary
    print('Name   ::  Test Stat > C(95%)    =>   Signif  \n', '--'*
    for col, trace, cvt in zip(stock_returns.columns, traces, cvts)
        print(adjust(col), ':: ', adjust(round(trace,2), 9), ">", a

cointegration_test(stock_returns)

# Cointegration is evident
```

```
Name   ::  Test Stat > C(95%)    =>   Signif
 ----------------------------------------
dAIB   ::  298.72    > 12.3212   =>   True
dBoI   ::  117.12    > 4.1296    =>   True
```

In [180]:
```
# Split the dataset into train and test data
# VAR model will forecast next year's values

nobs = 260
stock_returns_train, stock_returns_test = stock_returns[1:-nobs], s

# Check size
print(stock_returns_train.shape)
print(stock_returns_test.shape)
```

```
(527, 2)
(260, 2)
```

In [181]:
```
# Selecting Order(p) of the VAR model
# Look for order that gives hte lowest AIC

from statsmodels.tsa.api import VAR

model = VAR(stock_returns)
for i in [1,2,3,4,5,6,7,8,9]:
  result = model.fit(i)
  print('Lag Order =', i)
  print('AIC : ', result.aic)
  print('BIC : ', result.bic)
  print('FPE : ', result.fpe)
  print('HQIC: ', result.hqic, '\n')

  # Lowest AIC at lag 6
```

```
Lag Order = 1
AIC :  4.136117017390967
BIC :  4.1717070675695
FPE :  62.5594366503173
HQIC:  4.14979967347309

Lag Order = 2
AIC :  4.1399434192711375
BIC :  4.199319459887286
FPE :  62.799289675230334
HQIC:  4.162772007128729

Lag Order = 3
AIC :  4.14411857578782
BIC :  4.227328221710054
FPE :  63.06207263553421
HQIC:  4.176112500484777

Lag Order = 4
AIC :  4.129890920347703
BIC :  4.236981943774116
FPE :  62.17126637119178
HQIC:  4.171069652619446

Lag Order = 5
AIC :  4.123336373176197
BIC :  4.2543567043534605
FPE :  61.765198233386876
HQIC:  4.173719449733416

Lag Order = 6
AIC :  4.12180864922691
BIC :  4.276806377173429
FPE :  61.671059921409025
HQIC:  4.181415673061101

Lag Order = 7
AIC :  4.125071632974485
BIC :  4.304095006207763
FPE :  61.87282577708293
HQIC:  4.19392227366527

Lag Order = 8
AIC :  4.13121922085623
BIC :  4.334316648124491
FPE :  62.254637383354215
HQIC:  4.209333214880486

Lag Order = 9
AIC :  4.1308648676184125
BIC :  4.358084918636497
FPE :  62.23292623332048
HQIC:  4.218262018661224
```

In [182]:
```
# Training the VAR model of selected Order(p)

model_fitted = model.fit(6)
model_fitted.summary()
```

Out[182]:
```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Fri, 19, Nov, 2021
Time:                     10:47:45
--------------------------------------------------------------------
--
No. of Equations:         2.00000    BIC:                      4.276
81
Nobs:                     782.000    HQIC:                     4.181
42
Log likelihood:          -3804.85    FPE:                      61.67
11
AIC:                      4.12181    Det(Omega_mle):           59.67
06
--------------------------------------------------------------------
--
Results for equation dAIB
====================================================================
========
             coefficient       std. error           t-stat
prob
--------------------------------------------------------------------
--------
const          -0.120742         0.125740           -0.960
0.337
L1.dAIB         0.067411         0.053538            1.259
0.208
L1.dBoI         0.022445         0.057130            0.393
0.694
L2.dAIB        -0.107096         0.053337           -2.008
0.045
L2.dBoI         0.084193         0.057158            1.473
0.141
L3.dAIB        -0.109968         0.053005           -2.075
0.038
L3.dBoI         0.085928         0.057210            1.502
0.133
L4.dAIB        -0.217026         0.053031           -4.092
0.000
L4.dBoI         0.197156         0.057232            3.445
0.001
L5.dAIB        -0.134342         0.053519           -2.510
0.012
L5.dBoI         0.168451         0.057437            2.933
```

```
0.003
L6.dAIB        -0.076845        0.053476           -1.437
0.151
L6.dBoI         0.020261        0.057546            0.352
0.725
==================================================================
========

Results for equation dBoI
==================================================================
========
             coefficient       std. error           t-stat
prob
------------------------------------------------------------------
--------
const          -0.068848        0.117641           -0.585
0.558
L1.dAIB         0.115824        0.050090            2.312
0.021
L1.dBoI        -0.067220        0.053450           -1.258
0.209
L2.dAIB        -0.022980        0.049901           -0.460
0.645
L2.dBoI        -0.022610        0.053477           -0.423
0.672
L3.dAIB        -0.054121        0.049591           -1.091
0.275
L3.dBoI         0.005093        0.053525            0.095
0.924
L4.dAIB        -0.078010        0.049615           -1.572
0.116
L4.dBoI         0.102353        0.053546            1.911
0.056
L5.dAIB        -0.056160        0.050072           -1.122
0.262
L5.dBoI         0.039339        0.053738            0.732
0.464
L6.dAIB        -0.112538        0.050032           -2.249
0.024
L6.dBoI         0.113613        0.053840            2.110
0.035
==================================================================
========

Correlation matrix of residuals
            dAIB      dBoI
dAIB    1.000000  0.741030
dBoI    0.741030  1.000000
```

Or Alternatively:

In [183]:
```
# Run VAR - for now assume we know the appropriate lag length is 5
```

```
stock_returns = prices[['dAIB','dBoI']].copy()

model = smt.VAR(stock_returns)
res=model.fit(maxlags=5)
print(res.summary())

# For AIB: lag 4 and lag 5 are significant for both
# For BoI: Influenced by AIB's previous day returns
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Fri, 19, Nov, 2021
Time:                     10:47:45
------------------------------------------------------------------
--
No. of Equations:         2.00000    BIC:                    4.254
36
Nobs:                     783.000    HQIC:                   4.173
72
Log likelihood:          -3814.34    FPE:                    61.76
52
AIC:                      4.12334    Det(Omega_mle):         60.06
57
------------------------------------------------------------------
--
Results for equation dAIB
==================================================================
========
             coefficient       std. error           t-stat
prob
------------------------------------------------------------------
--------
const          -0.114026         0.125602           -0.908
0.364
L1.dAIB         0.070008         0.053066            1.319
0.187
L1.dBoI         0.022978         0.056907            0.404
0.686
L2.dAIB        -0.094915         0.052756           -1.799
0.072
L2.dBoI         0.075375         0.056935            1.324
0.186
L3.dAIB        -0.105809         0.052856           -2.002
0.045
L3.dBoI         0.084765         0.057045            1.486
0.137
L4.dAIB        -0.212242         0.052782           -4.021
0.000
L4.dBoI         0.194266         0.056962            3.410
0.001
L5.dAIB        -0.137058         0.053187           -2.577
```

```
0.010
L5.dBoI         0.163066        0.057023           2.860
0.004
===========================================================================
========

Results for equation dBoI
===========================================================================
========
             coefficient       std. error           t-stat
prob
---------------------------------------------------------------------------
--------
const          -0.060463         0.117679          -0.514
0.607
L1.dAIB         0.129913         0.049719           2.613
0.009
L1.dBoI        -0.075804         0.053317          -1.422
0.155
L2.dAIB        -0.006812         0.049428          -0.138
0.890
L2.dBoI        -0.031608         0.053344          -0.593
0.553
L3.dAIB        -0.046207         0.049522          -0.933
0.351
L3.dBoI        -0.001408         0.053447          -0.026
0.979
L4.dAIB        -0.067142         0.049453          -1.358
0.175
L4.dBoI         0.092031         0.053369           1.724
0.085
L5.dAIB        -0.049210         0.049832          -0.988
0.323
L5.dBoI         0.025736         0.053426           0.482
0.630
===========================================================================
========

Correlation matrix of residuals
            dAIB      dBoI
dAIB    1.000000  0.740115
dBoI    0.740115  1.000000
```

In [184]:
```
# Test up to 10 lags to find out the appropriate lag length

res=model.select_order(maxlags=10)
print(res.summary())

# For AIC the min is 10
# For BIC the min is 0
# For FPE the min is 10
# For HQIC the min is 0
```

```
 VAR Order Selection (* highlights the minimums)
==================================================
       AIC         BIC         FPE         HQIC
--------------------------------------------------
0       4.161      4.173*       64.13      4.166*
1       4.153       4.189       63.63       4.167
2       4.156       4.215       63.79       4.179
3       4.159       4.243       64.03       4.192
4       4.143       4.251       62.99       4.184
5       4.134       4.265       62.40       4.184
6       4.130       4.286       62.17       4.190
7       4.131       4.311       62.24       4.200
8       4.135       4.338       62.48       4.213
9       4.133       4.360       62.34       4.220
10     4.126*       4.377      61.91*       4.222
--------------------------------------------------
```

In [185]:
```
# Go with AIC (as HQIC is 0)

model_2 = smt.VAR(stock_returns)
res_2 = model_2.fit(maxlags=10)
print(res_2.summary())

# Lags 4 and 5 are relevant for AIB
# Lags 4, 6 and 10 are relevant for BoI, the return from 4, 6 and 1(
```

```
  Summary of Regression Results
==================================
Model:                         VAR
Method:                        OLS
Date:           Fri, 19, Nov, 2021
Time:                     10:47:45
--------------------------------------------------------------------
--
No. of Equations:         2.00000    BIC:                      4.377
04
Nobs:                     778.000    HQIC:                     4.222
35
Log likelihood:          -3770.75    FPE:                      61.90
96
AIC:                      4.12565    Det(Omega_mle):           58.69
81
--------------------------------------------------------------------
```

```
--
Results for equation dAIB
==================================================================
=========
              coefficient       std. error           t-stat
prob
------------------------------------------------------------------
---------
const           -0.092336         0.124672           -0.741
0.459
L1.dAIB          0.070150         0.053539            1.310
0.190
L1.dBoI          0.018271         0.056756            0.322
0.748
L2.dAIB         -0.119778         0.053523           -2.238
0.025
L2.dBoI          0.090819         0.056904            1.596
0.110
L3.dAIB         -0.113385         0.053706           -2.111
0.035
L3.dBoI          0.082741         0.057134            1.448
0.148
L4.dAIB         -0.229879         0.053663           -4.284
0.000
L4.dBoI          0.222617         0.057312            3.884
0.000
L5.dAIB         -0.122914         0.054319           -2.263
0.024
L5.dBoI          0.166667         0.057637            2.892
0.004
L6.dAIB         -0.069539         0.054107           -1.285
0.199
L6.dBoI          0.018931         0.057893            0.327
0.744
L7.dAIB          0.022045         0.053721            0.410
0.682
L7.dBoI          0.100532         0.058003            1.733
0.083
L8.dAIB         -0.096691         0.053523           -1.807
0.071
L8.dBoI          0.046617         0.057931            0.805
0.421
L9.dAIB          0.093287         0.053429            1.746
0.081
L9.dBoI         -0.001737         0.057707           -0.030
0.976
L10.dAIB         0.168958         0.053236            3.174
0.002
L10.dBoI        -0.067641         0.057364           -1.179
0.238
==================================================================
=========

Results for equation dBoI
```

```
==================================================================
=========
               coefficient       std. error           t-stat
prob
------------------------------------------------------------------
---------
const            -0.053831         0.117997           -0.456
0.648
L1.dAIB           0.116249         0.050672            2.294
0.022
L1.dBoI          -0.068516         0.053717           -1.275
0.202
L2.dAIB          -0.032239         0.050657           -0.636
0.525
L2.dBoI          -0.015288         0.053858           -0.284
0.777
L3.dAIB          -0.052540         0.050831           -1.034
0.301
L3.dBoI          -0.001579         0.054076           -0.029
0.977
L4.dAIB          -0.085050         0.050790           -1.675
0.094
L4.dBoI           0.118405         0.054243            2.183
0.029
L5.dAIB          -0.049229         0.051411           -0.958
0.338
L5.dBoI           0.037166         0.054551            0.681
0.496
L6.dAIB          -0.104196         0.051210           -2.035
0.042
L6.dBoI           0.108366         0.054794            1.978
0.048
L7.dAIB           0.014815         0.050845            0.291
0.771
L7.dBoI           0.062294         0.054897            1.135
0.256
L8.dAIB          -0.042943         0.050658           -0.848
0.397
L8.dBoI          -0.003253         0.054830           -0.059
0.953
L9.dAIB           0.035284         0.050568            0.698
0.485
L9.dBoI           0.014475         0.054618            0.265
0.791
L10.dAIB          0.122223         0.050386            2.426
0.015
L10.dBoI         -0.052077         0.054293           -0.959
0.337
==================================================================
=========

Correlation matrix of residuals
            dAIB      dBoI
dAIB    1.000000  0.736317
```

```
dBoI    0.736317  1.000000
```

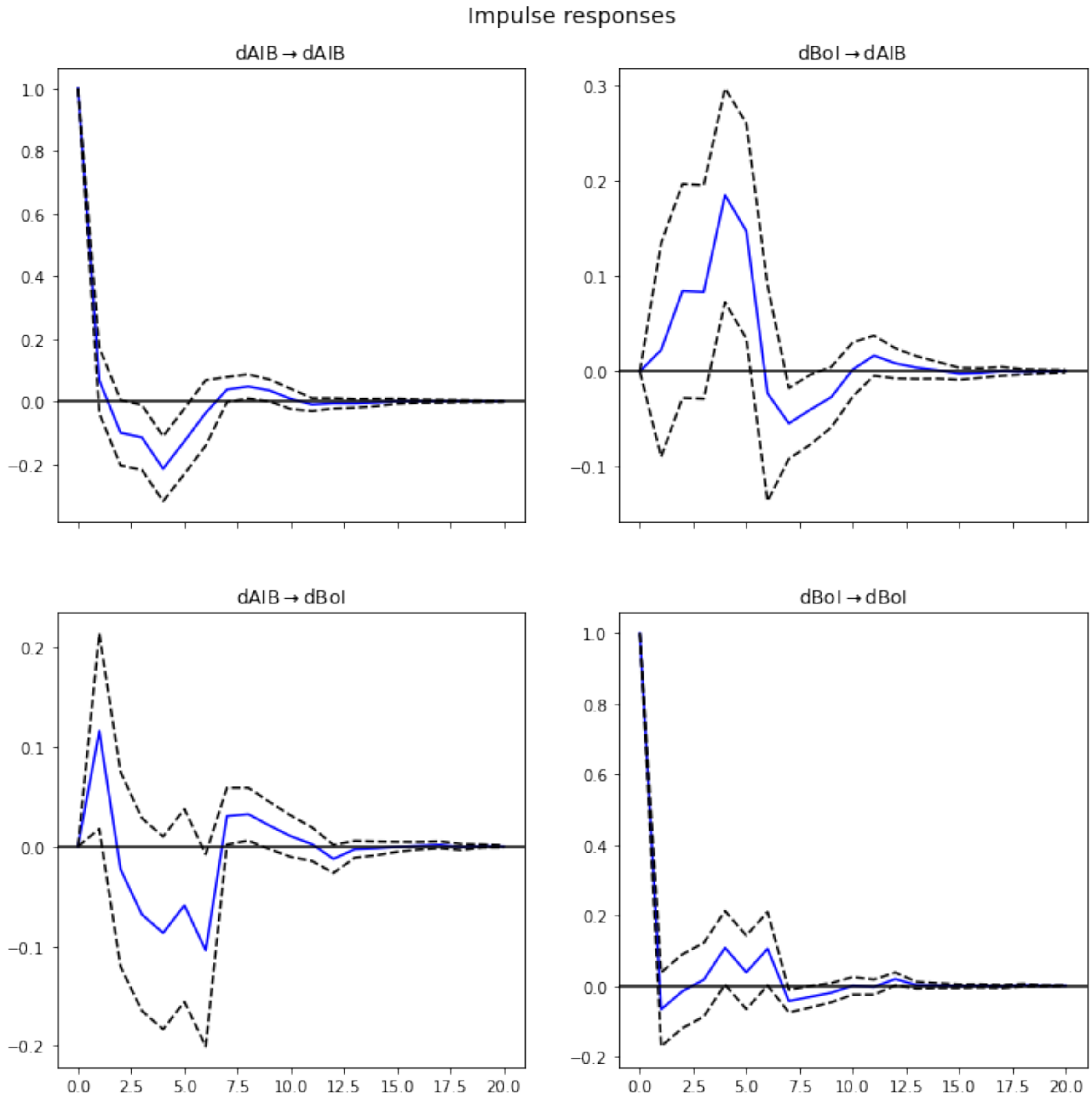
In [186]:

```
# Granger Causality Test

maxlag = 6

test = 'ssr_chi2test'
def grangers_causation_matrix(data, variables, test='ssr_chi2test',
  """Check Granger Causality of all possible combinations of the Ti
  The rows are the response variabel, columns are predictors. The v
  are the P-Values. P-Values lesser than the signnificance level (0
   the Null Hypothesis that the coefficients of the corresponding p
   zero, that is, the x does not cause Y can be rejected.

   data       : pandas dataframe containing the time series variabl
   variables  : list containing names of the time series varaibles.
   """

  df = pd.DataFrame(np.zeros((len(variables), len(variables))), col
  for c in df.columns:
     for r in df.index:
       test_result = grangercausalitytests(data[[r,c]], maxlag=maxl
       p_values = [round(test_result[i+1][0][test][1],4) for i in r
       if verbose: print(f'Y = {r}, X = {c}, P Values = {p_values}'
       min_p_value = np.min(p_values)
       df.loc[r,c] = min_p_value
  df.columns = [var + '_x' for var in variables]
  df.index = [var + '_y' for var in variables]
  return df

gc = grangers_causation_matrix(stock_returns, variables = stock_ret

gc


# If p-value is < significance level of 0.05, then the correspondin

# Both AIB and BoI returns definitely have a significant effect on 
# They cause effects on the other
# This causality makes these time series appropriate for a VAR mode
```

Out[186]:

	dAIB_x	dBoI_x
dAIB_y	1.0000	0.0012
dBoI_y	0.0052	1.0000

In [187]:
```
# Impulse Response Functions

model_ir = smt.VAR(stock_returns)
res_ir = model_ir.fit(maxlags=6)

irf=res_ir.irf(20)
irf.plot();

# A negative shock in AIB, is likely to cause a negative return to 
# And vice-versa
```



Part 3 - Machine Learning House Prices

Q3: Use appropriate machine learning techniques to develop a model to explain house prices

In [188]:
```
houses = pd.read_csv('A1HousePrices.csv')
```

In [189]:
```
houses.head(3)
```

Out[189]:

	id	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floo
0	7129300520	20141013T000000	221900.0	3	1.00	1180	5650	1
1	6414100192	20141209T000000	538000.0	3	2.25	2570	7242	2
2	5631500400	20150225T000000	180000.0	2	1.00	770	10000	1

3 rows × 21 columns

In [190]:
```
houses.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 21 columns):
 #   Column         Non-Null Count  Dtype
---  ------         --------------  -----
 0   id             21613 non-null  int64
 1   date           21613 non-null  object
 2   price          21613 non-null  float64
 3   bedrooms       21613 non-null  int64
 4   bathrooms      21613 non-null  float64
 5   sqft_living    21613 non-null  int64
 6   sqft_lot       21613 non-null  int64
 7   floors         21613 non-null  float64
 8   waterfront     21613 non-null  int64
 9   view           21613 non-null  int64
 10  condition      21613 non-null  int64
 11  grade          21613 non-null  int64
 12  sqft_above     21613 non-null  int64
 13  sqft_basement  21613 non-null  int64
 14  yr_built       21613 non-null  int64
 15  yr_renovated   21613 non-null  int64
 16  zipcode        21613 non-null  int64
 17  lat            21613 non-null  float64
 18  long           21613 non-null  float64
 19  sqft_living15  21613 non-null  int64
 20  sqft_lot15     21613 non-null  int64
dtypes: float64(5), int64(15), object(1)
memory usage: 3.5+ MB
```

In [191]:
```
conv_dates = [1 if values == 2014 else 0 for values in houses.date]
houses['date'] = conv_dates
houses = houses.drop(["id", "sqft_living15", "sqft_lot15", "sqft_ab
houses.head(3)

# Dropping variables that are too unique, have too many missing var
# Sqft_living is a sum of sqft_above and sqft_basement. And basemen
```

Out[191]:

	date	price	bedrooms	bathrooms	sqft_living	sqft_lot	floors	waterfront	view	con
0	0	221900.0	3	1.00	1180	5650	1.0	0	0	
1	0	538000.0	3	2.25	2570	7242	2.0	0	0	
2	0	180000.0	2	1.00	770	10000	1.0	0	0	

In [192]:
```
houses.info()

# No missing values
# No strings
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 21613 entries, 0 to 21612
Data columns (total 15 columns):
 #   Column       Non-Null Count  Dtype
---  ------       --------------  -----
 0   date         21613 non-null  int64
 1   price        21613 non-null  float64
 2   bedrooms     21613 non-null  int64
 3   bathrooms    21613 non-null  float64
 4   sqft_living  21613 non-null  int64
 5   sqft_lot     21613 non-null  int64
 6   floors       21613 non-null  float64
 7   waterfront   21613 non-null  int64
 8   view         21613 non-null  int64
 9   condition    21613 non-null  int64
 10  grade        21613 non-null  int64
 11  yr_built     21613 non-null  int64
 12  zipcode      21613 non-null  int64
 13  lat          21613 non-null  float64
 14  long         21613 non-null  float64
dtypes: float64(5), int64(10)
memory usage: 2.5 MB
```

In [193]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
```

In [194]:
```
reg = LinearRegression()
```

In [195]:
```
# Set the labels as the price column as prices are to be predicted

labels = houses['price']
train1 = houses.drop(['price'],axis=1)
```

In [196]:
```
from sklearn.model_selection import train_test_split
```

In [197]:
```
# 80% train data, 20% test data

x_train , x_test , y_train , y_test = train_test_split(train1 , lab

x_train.shape, y_train.shape, x_test.shape, y_test.shape
```

Out[197]:
```
((17290, 14), (17290,), (4323, 14), (4323,))
```

In [198]:
```
reg.fit(x_train, y_train)
```

Out[198]:
```
LinearRegression()
```

In [199]:
```
# After fitting the model the score is 71.3%

reg.score(x_test, y_test)
```

Out[199]:
```
0.7131150403769968
```

Gradient Boosting

We can try to improve this 71.3% with gradient boosting

In [200]:
```
from sklearn import ensemble
clf = ensemble.GradientBoostingRegressor(n_estimators = 400, max_de
          learning_rate = 0.1, loss = 'ls')
```

In [201]:
```
clf.fit(x_train, y_train)
```

Out[201]:
```
GradientBoostingRegressor(max_depth=5, n_estimators=400)
```

In [202]:
```
clf.score(x_test,y_test)

# Accuracy has improved to 89.2%, a big improvement
# For weak predictions, gradient boosting works really well, even u
```

Out[202]:
```
0.8916183345372595
```

Alternatively, a decision tree could have been used

In [203]:

```
# Decision Tree

dt = DecisionTreeClassifier()

dt.fit(x_train, y_train)
y_pred = dt.predict(x_test)
print("Accuracy is ", accuracy_score(y_test, y_pred)*100)
```

```
Accuracy is  0.9715475364330326
```

In [204]:

```
# Visualise the decision tree

plt.figure(figsize=(16,16))
plot_tree(dt, max_depth=3, fontsize=10, feature_names=x_train.colum
plt.show()

# First step: Is the latitude of the house less than or equal to 47
# Then is the sqft_living less than or equal to 1935?
# And so on
# So, location and then living area were the first two questions
```