Basic Idea

We will create a semi-automatic algorithm that will alert to buy or sell in real-time.
This will be done in 5 steps:

1) Find a group of assets that move similarly with eachother over the past X periods of time.
2) Find the best pair amongst them.
3) Choose and calculate indicators to buy and sell.
4) Alert buy / sell.
5) Learn and improve the model, and test it.

Basic Concepts

Let's go over some of the concpets we'll use in this project:

1) Cointegration: Similar to correlation. Means that the ratio between two series will vary around a mean.
The two series, Y and X follow the follwing: $Y = \alpha X + e$ where α is the constant ratio and e is white noise
In plain terms, it means that the ratio between the two financial time series will vary around a constant mean
2) Stationarity: A stochastic process whose unconditional joint probability distribution does not change when shifted in time.
In plain terms, not time dependant.
3) Auto-corelation: Similar to the correlation between two different time series, but autocorrelation uses the same time series twice: once in its original form and once lagged one or more time periods. Auto-correlation is some kind of Stationarity.
4) P-value: The probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct.
We will use it to test for conitegration.

Project requirements

In [1]:

```
from asyncio import threads
import seaborn
import numpy as np
import pandas as pd
import statsmodels
import statsmodels.api as sm
import statsmodels.tsa.stattools as ts
from statsmodels.tsa.stattools import coint, adfuller
import yfinance as yf
from yahoo_fin.stock_info import get_data
from datetime import datetime
import matplotlib.pyplot as plt
import plotly.graph_objs as go
from statsmodels.graphics.tsaplots import plot_acf
from matplotlib import pyplot
from itertools import product
```

Data that will be used

We will examine ETFs of tech companies.
Our assumption is that each of them is stationary, and that they will probably be cointegrated, or at least correlated.
We will be looking at the following ETFs:

- VGT
- XLK
- SMH
- SOXX
- IYW

Which are the Top 5 ETFs considering total assets and 5 years look back window profits. (etfdb (https://etfdb.com/etfdb-category/technology-equities/))

Loading data

We use Yahoo Finance as our main data source.
Our calculations we'll be made by the closing price of each ETF.

In [2]:

```
tickers = ['IYW', 'VGT', 'XLK', 'SMH', 'SOXX']

vgt = yf.Ticker('VGT').history(period='max', start=datetime(2016, 10, 27), end=datet
xlk = yf.Ticker('XLK').history(period='max', start=datetime(2016, 10, 27), end=datet
smh = yf.Ticker('SMH').history(period='max', start=datetime(2016, 10, 27), end=datet
soxx = yf.Ticker('SOXX').history(period='max', start=datetime(2016, 10, 27), end=dat
iyw = yf.Ticker('IYW').history(period='max', start=datetime(2016, 10, 27), end=datet

# merge all ETFs data frames to one, by Close tag
vgt_xlk = pd.merge(left = vgt[['Close']], right = xlk[['Close']], left_index = True,
              suffixes = ('_vgt', '_xlk'))
smh_soxx = pd.merge(left = smh[['Close']], right = soxx[['Close']], left_index = Tru
              suffixes = ('_smh', '_soxx'))
combined = pd.merge(left = vgt_xlk, right = smh_soxx, left_index = True, right_index
all = pd.merge(left = iyw[['Close']], right = combined, left_index = True, right_inc
all.rename(columns={'Close': 'Close_iyw'}, inplace=True)

all
```

Out[2]:

Date	Close_iyw	Close_vgt	Close_xlk	Close_smh	Close_soxx
2016-10-26	28.642511	113.432892	44.380337	63.859909	105.450752
2016-10-27	28.505598	112.866890	44.324341	63.673141	104.960930
2016-10-28	28.469574	112.923515	44.277679	63.308914	104.329849
2016-10-31	28.483986	112.932930	44.259003	63.757179	104.895012
2016-11-01	28.246193	112.017929	43.876339	63.168823	103.962471
...	...	...	...	...	...
2022-05-03	91.620003	373.510010	143.820007	238.479996	417.589996
2022-05-04	94.970001	386.459991	148.869995	246.660004	433.790009
2022-05-05	90.150002	366.730011	141.710007	235.080002	412.779999
2022-05-06	89.110001	362.750000	140.570007	232.669998	409.100006
2022-05-09	85.410004	347.130005	135.130005	220.889999	388.269989

1393 rows × 5 columns

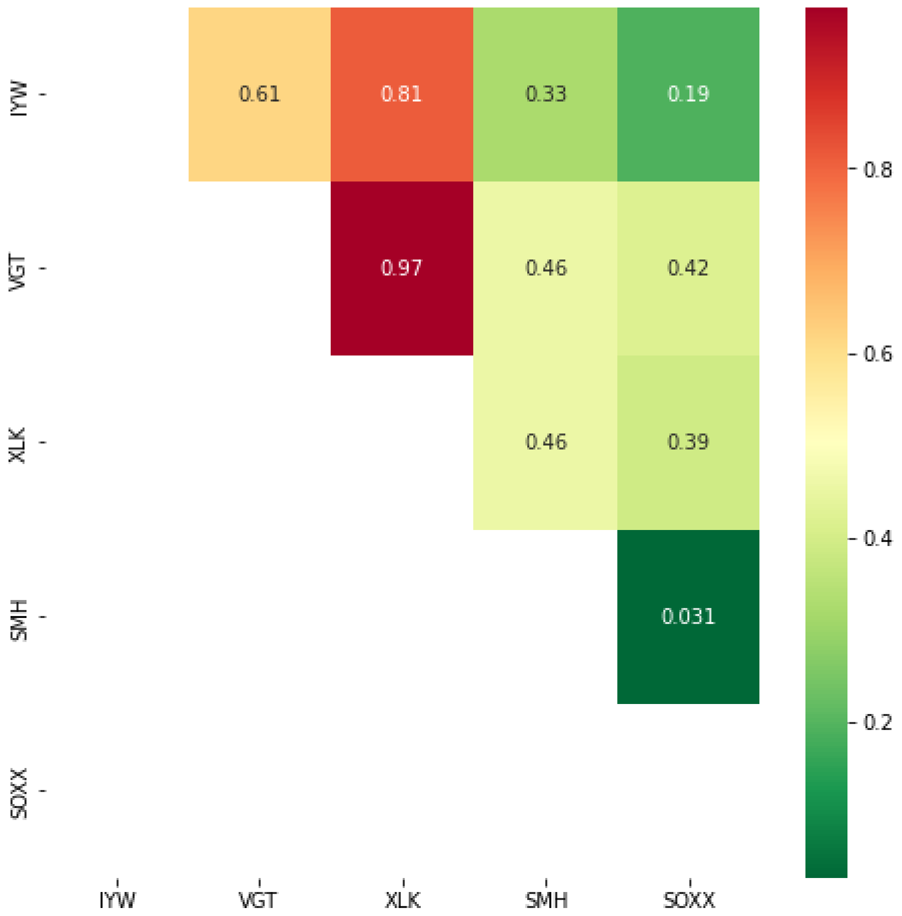
In [3]:

```
def find_cointegrated_pairs(data):
    n = data.shape[1]
    score_matrix = np.zeros((n, n))
    pvalue_matrix = np.ones((n, n))
    keys = data.keys()
    pairs = []
    for i in range(n):
        for j in range(i+1, n):
            S1 = data[keys[i]]
            S2 = data[keys[j]]
            result = coint(S1, S2)
            score = result[0]
            pvalue = result[1]
            score_matrix[i, j] = score
            pvalue_matrix[i, j] = pvalue
            if pvalue < 0.05:
                pairs.append((keys[i], keys[j]))
    return score_matrix, pvalue_matrix, pairs
```

In [7]:

```
# Heatmap to show the p-values of the cointegration test between each pair of
# stocks. Only show the value in the upper-diagonal of the heatmap
scores, pvalues, pairs = find_cointegrated_pairs(all)
fig, ax = plt.subplots(figsize=(8,8))
seaborn.heatmap(pvalues, xticklabels=tickers, yticklabels=tickers, annot=True, cmap=
                , mask = (pvalues >= 0.99)
                )
print(pairs)
```

```
[('Close_smh', 'Close_soxx')]
```



Our algorithm listed one cointegrated pair: SMH/SOXX. We can analyze their price patterns to make sure there is nothing weird going on.

In [12]:

```
S1 = all['Close_smh']
S2 = all['Close_soxx']

score, pvalue, _ = coint(S1, S2)
pvalue
```

Out[12]:

```
0.031230168711730587
```

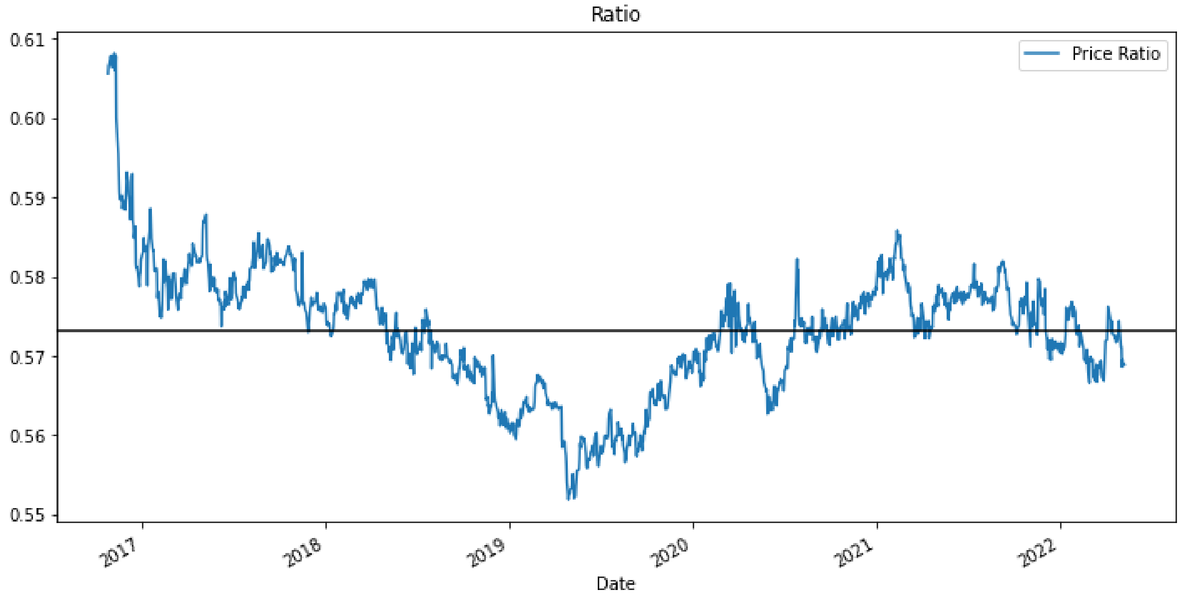
As we can see, the p-value is less than 0.05, which means SMH and SOXX are indeed cointegrated pairs.

Calculating the Ratio

Now we can plot the ratio of the two time series.

In [13]:

```
ratio = S1/S2
ratio.plot(figsize=(12,6))
plt.axhline(ratio.mean(), color='black')
plt.xlim()
plt.title('Ratio')
plt.legend(['Price Ratio']);
```



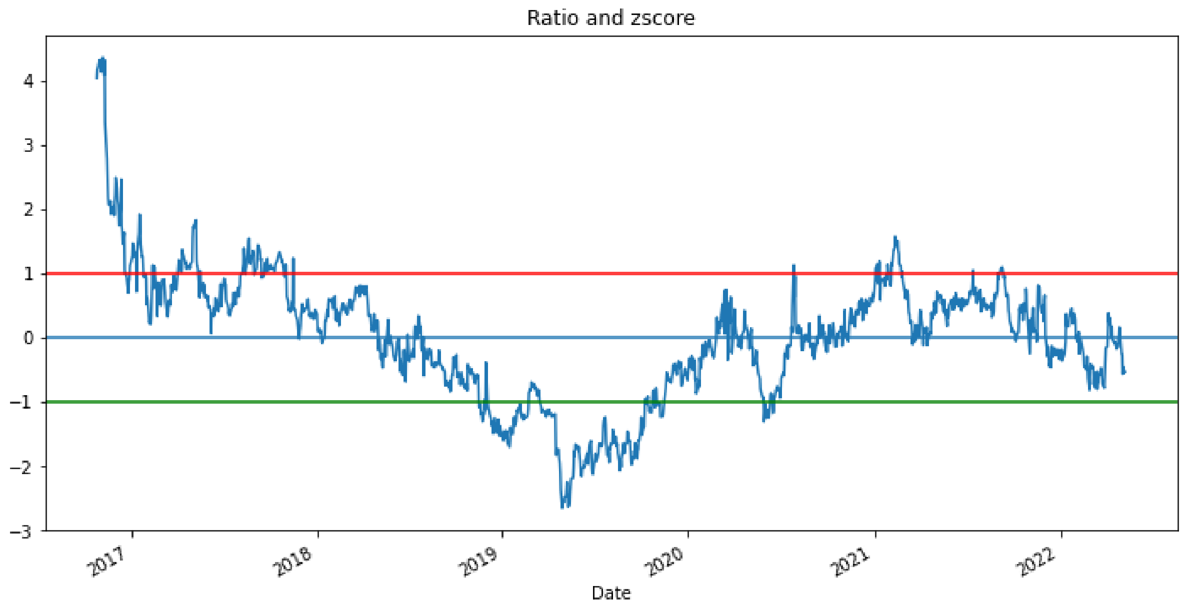
We now need to standardize this ratio because the absolute ratio might not be the most ideal way of analyzing this trend. For this, we need to use z-scores.

A z-score is the number of standard deviations a datapoint is from the mean. More importantly, the nımber of standard deviations above or below the population mean is from the raw score. The z-score is calculated by the follow:

$$z_i = \frac{x_i - \bar{x}}{s}$$

In [27]:

```
def zscore(series):
    return (series - series.mean()) / np.std(series)


zscore(ratio).plot(figsize=(12,6))
plt.axhline(zscore(ratio).mean())
plt.axhline(1.0, color='red')
plt.axhline(-1.0, color='green')
plt.title('Ratio and zscore')
plt.show()
```



By setting two other lines placed at the z-score of 1 and -1, we can clearly see that for the most part, any big divergences from the mean eventually converges back. This is exactly what we want for a pairs trading strategy.

Creating A Model

We want to trade based on the ratio of the two ETFs.
Then, we need to predict the ratio.
This will be done by **Random Forest** algorithm.

But first, we'll check for PoC.

Setup rules

We're going to use the ratio time series that we've created to see if it tells us whether to buy or sell in a particular moment in time. We'll start off by creating a prediction variable Y. If the ratio is positive, it will signal a "buy," otherwise, it will signal a "sell". The prediction model is as follows:

$$Y_t = sign(Ratio_{t+1} - Ratio_t)$$

- When you buy the ratio, you actully buy S1 and sell S2
- When you sell the ratio, you actully sell S1 and buy S2

Rules

- Buy(1) whenever the z-score is below -1, meaning we expect the ratio to increase.
- Sell(-1) whenever the z-score is above 1, meaning we expect the ratio to decrease.

Indicators

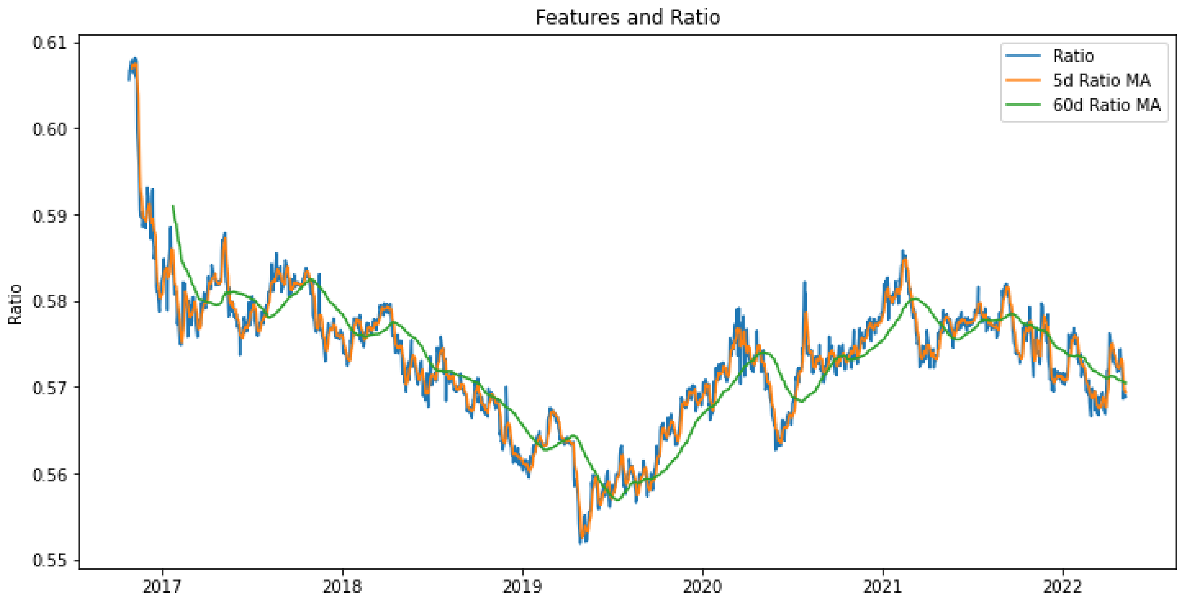
Our main hypothessis is that the prices will return to the mean. Therfore, we will use several indicators and metrics which involve the mean:

Key point: choosing the look back window. Here we choose randomly.

- 60 day Moving Average of Ratio
- 5 day Moving Average of Ratio
- 60 day Standard Deviation
- z score

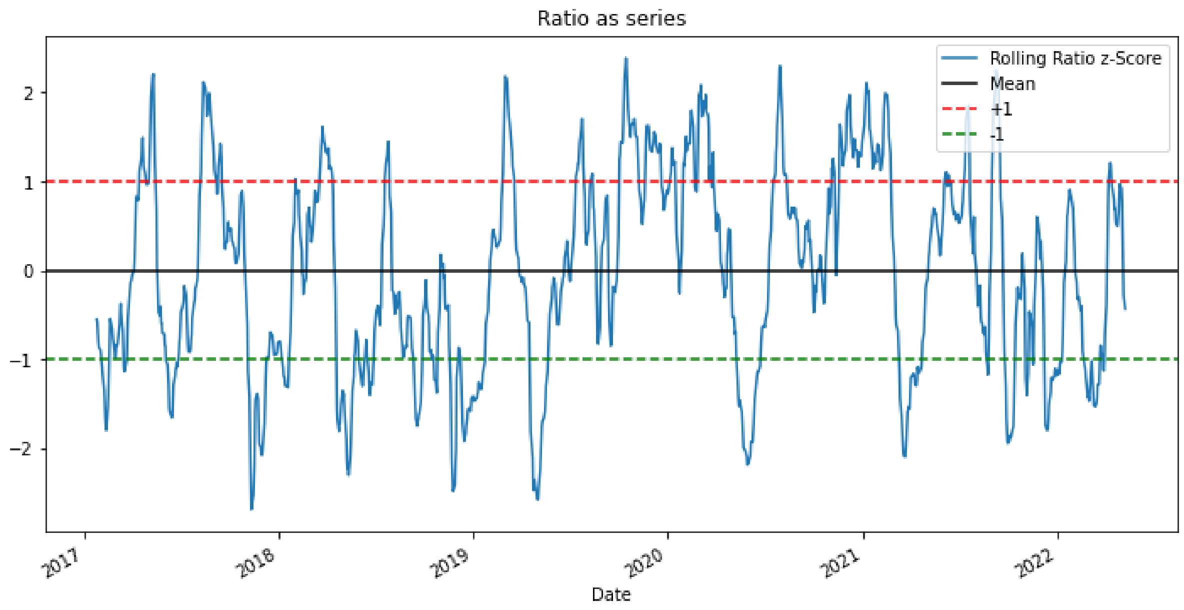
In [15]:

```
ratios_mavg5 = ratio.rolling(window=5, center=False).mean()
ratios_mavg60 = ratio.rolling(window=60, center=False).mean()
std_60 = ratio.rolling(window=60, center=False).std()
zscore_60_5 = (ratios_mavg5 - ratios_mavg60)/std_60
plt.figure(figsize=(12, 6))
plt.plot(ratio.index, ratio.values)
plt.plot(ratios_mavg5.index, ratios_mavg5.values)
plt.plot(ratios_mavg60.index, ratios_mavg60.values)
plt.legend(['Ratio', '5d Ratio MA', '60d Ratio MA'])
plt.title('Features and Ratio')

plt.ylabel('Ratio')
plt.show()
```



In [16]:

```
plt.figure(figsize=(12,6))
zscore_60_5.plot()
plt.axhline(0, color='black')
plt.axhline(1.0, color='red', linestyle='--')
plt.axhline(-1.0, color='green', linestyle='--')
plt.legend(['Rolling Ratio z-Score', 'Mean', '+1', '-1'])
plt.title('Ratio as series')
plt.show()
```



In [17]:

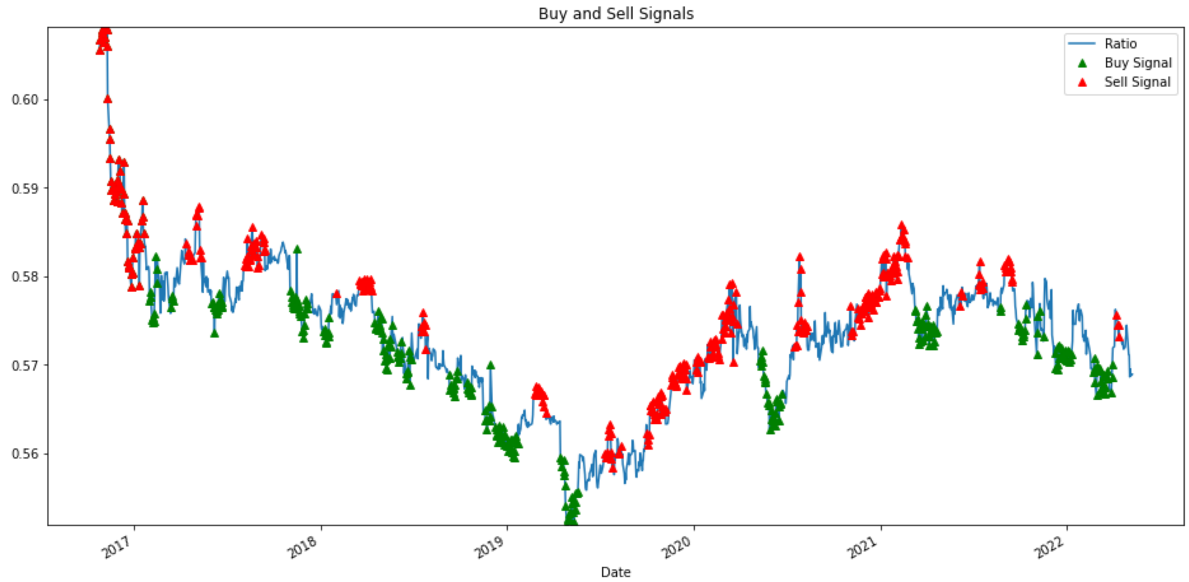
```
# calculate the metrcis
def plot_with_signals(dataframe, lim=False):
    ratios_mavg5 = dataframe.rolling(window=5, center=False).mean()
    ratios_mavg60 = dataframe.rolling(window=60, center=False).mean()
    std_60 = dataframe.rolling(window=60, center=False).std()
    zscore_60_5 = (ratios_mavg5 - ratios_mavg60)/std_60


    plt.figure(figsize=(16,8))

    dataframe.plot()
    buy = dataframe.copy()
    sell = dataframe.copy()
    buy[zscore_60_5>-1] = 0
    sell[zscore_60_5<1] = 0
    buy.plot(color='g', linestyle='None', marker='^')
    sell.plot(color='r', linestyle='None', marker='^')
    x1, x2, y1, y2 = plt.axis()
    plt.axis((x1, x2, dataframe.min(), dataframe.max()))
    plt.title('Buy and Sell Signals')

    if lim:
        plt.xlim(datetime(2021, 1, 1))
    plt.legend(['Ratio', 'Buy Signal', 'Sell Signal'])
    plt.show()
```

PoC

As we can see, we have a graph that signals when to buy and when to sell, by the rules we have set.

In [18]:

```
plot_with_signals(ratio)
```



Benchmark the signals

In [34]:

```
# Trade using a simple strategy
def trade(S1, S2, window1, window2, zscore_sell=1, zscore_buy=None):
    zscore_buy = -1 * zscore_sell if not zscore_buy else zscore_buy

    # check parameters validity.
    if (window1 == 0) or (window2 == 0):
        return 0

    # Compute rolling mean and rolling standard deviation
    ratios = S1/S2
    ma1 = ratios.rolling(window=window1,
                               center=False).mean()
    ma2 = ratios.rolling(window=window2,
                               center=False).mean()
    std = ratios.rolling(window=window2,
                        center=False).std()
    zscore = (ma1 - ma2)/std
#     if zscore_buy != zscore_sell * (-1):
#         zscore = zscore[zscore['Close'].notnull()]

    # Simulate trading
    # Start with no positions
    start_money_const = 100
    money = start_money_const
    countS1 = 0
    countS2 = 0
    for i in range(len(ratios)):
#         print("z-score is", zscore[i])
        # Buy long if the z-score is < zscore_buy
        if zscore[i] < zscore_buy:
            money += S1[i] - S2[i] * ratios[i]
            countS1 -= 1
            countS2 += ratios[i]
#             print('Selling Ratio %s %s %s %s'%(money, ratios[i], countS1,countS2))
        # Sell short if the z-score is > zscore_sell
        elif zscore[i] > zscore_sell:
            money -= S1[i] - S2[i] * ratios[i]
            countS1 += 1
            countS2 -= ratios[i]
#             print('Buying Ratio %s %s %s %s'%(money,ratios[i], countS1,countS2))
        # Clear positions and take profits if the z-score between -0.3 and 0.3
        elif abs(zscore[i]) < 0.3:
            money += S1[i] * countS1 + S2[i] * countS2
            countS1 = 0
            countS2 = 0
#             print('Exit pos %s %s %s %s'%(money,ratios[i], countS1,countS2))


    return money - start_money_const
```

In [35]:

```
trade(all['Close_smh'], all['Close_soxx'], 20, 5)
```

Out[35]:

```
167.52964642616428
```

Seems like we have a proof that trading on the ratio works, and works quite good

We will now use Random Forest to predict the ratio, train our model and test it.

In [36]:

```
s1 = smh[['Open', 'High', 'Low', 'Volume', 'Close']]
s2 = soxx[['Open', 'High', 'Low', 'Volume', 'Close']]
```

In [37]:

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.model_selection import TimeSeriesSplit
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense, Dropout
from keras.utils.vis_utils import plot_model
def build_lstm_model(df):
    scaler = MinMaxScaler()
    feature_transform = scaler.fit_transform(df[['Open', 'High', 'Low', 'Volume']])
    feature_transform= pd.DataFrame(columns=['Open', 'High', 'Low', 'Volume'], data=
    # Process the data for LSTM
    timesplit= TimeSeriesSplit(n_splits=2)
    for train_index, test_index in timesplit.split(feature_transform):
        X_train, X_test = feature_transform[:len(train_index)], feature_transform[le
        y_train, y_test = df[['Close']][:len(train_index)].values.ravel(), df[['Clos
    trainX =np.array(X_train)
    testX =np.array(X_test)
    X_train = trainX.reshape(X_train.shape[0], 1, X_train.shape[1])
    X_test = testX.reshape(X_test.shape[0], 1, X_test.shape[1])
    #Building the LSTM Model
    lstm = Sequential()
    lstm.add(LSTM(32, input_shape=(1, trainX.shape[1]), activation='relu', return_se
    lstm.add(Dense(1))
    lstm.compile(loss='mean_squared_error', optimizer='adam')
#     plot_model(lstm, show_shapes=True, show_layer_names=True)
    return lstm, X_train, X_test, y_train, y_test, train_index
```

In [38]:

```
def predict(lstm, X_train, y_train, X_test):
    history=lstm.fit(X_train, y_train, epochs=100, batch_size=8, verbose=1, shuffle=
    y_pred= lstm.predict(X_test)
    return y_pred
```

In [39]:

```
lstm_s1, X_train1, X_test1, y_train1, y_test1, train_index = build_lstm_model(s1)
lstm_s2, X_train2, X_test2, y_train2, y_test2, _ = build_lstm_model(s2)
train_index = len(train_index)
train_index
```

```
2022-05-24 20:41:41.336090: I tensorflow/core/platform/cpu_feature_gua
rd.cc:193] This TensorFlow binary is optimized with oneAPI Deep Neural
Network Library (oneDNN) to use the following CPU instructions in perf
ormance-critical operations:  AVX2 FMA
To enable them in other operations, rebuild TensorFlow with the approp
riate compiler flags.
```

Out[39]:

```
929
```

In [40]:

```
y_pred1 = predict(lstm_s1, X_train1, y_train1, X_test1)
```

```
117/117 [==============================] - 0s 2ms/step - loss: 569.063
2
Epoch 17/100
117/117 [==============================] - 0s 2ms/step - loss: 388.854
9
Epoch 18/100
117/117 [==============================] - 0s 2ms/step - loss: 271.925
0
Epoch 19/100
117/117 [==============================] - 0s 2ms/step - loss: 200.078
3
Epoch 20/100
117/117 [==============================] - 0s 2ms/step - loss: 158.154
2
Epoch 21/100
117/117 [==============================] - 0s 2ms/step - loss: 134.690
9
Epoch 22/100
117/117 [==============================] - 0s 2ms/step - loss: 121.794
```

In [41]:

```
y_pred2 = predict(lstm_s2, X_train2, y_train2, X_test2)
```

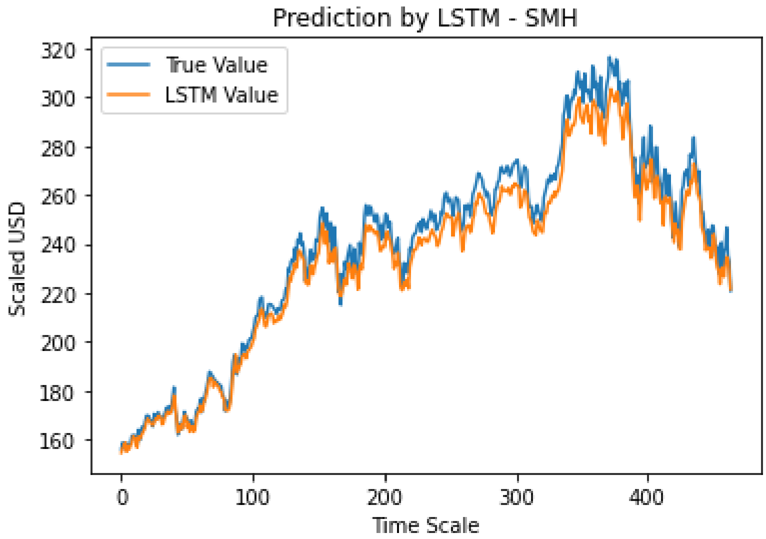
```
Epoch 1/100
117/117 [==============================] - 1s 2ms/step - loss: 32415.4
297
Epoch 2/100
117/117 [==============================] - 0s 2ms/step - loss: 32141.7
988
Epoch 3/100
117/117 [==============================] - 0s 2ms/step - loss: 31504.0
625
Epoch 4/100
117/117 [==============================] - 0s 2ms/step - loss: 30356.9
941
Epoch 5/100
117/117 [==============================] - 0s 2ms/step - loss: 28660.5
898
Epoch 6/100
117/117 [==============================] - 0s 2ms/step - loss: 26480.7
578
Epoch 7/100
```

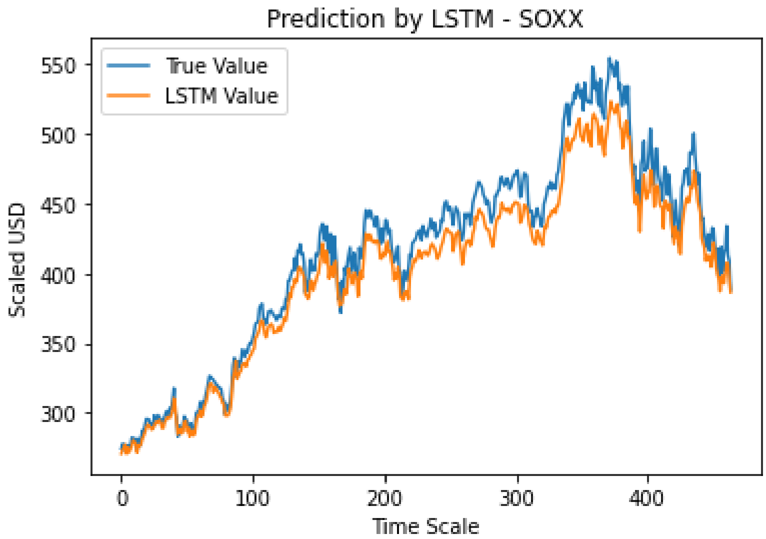
In [42]:

```
def plot_prediction(y_test, y_pred, ticker_name):
    plt.plot(y_test, label='True Value')
    plt.plot(y_pred, label='LSTM Value')
    plt.title(f"Prediction by LSTM - {ticker_name}")
    plt.xlabel('Time Scale')
    plt.ylabel('Scaled USD')
    plt.legend()
    plt.show()
```

In [43]:

```
plot_prediction(y_test1, y_pred1, "SMH")
plot_prediction(y_test2, y_pred2, "SOXX")
```

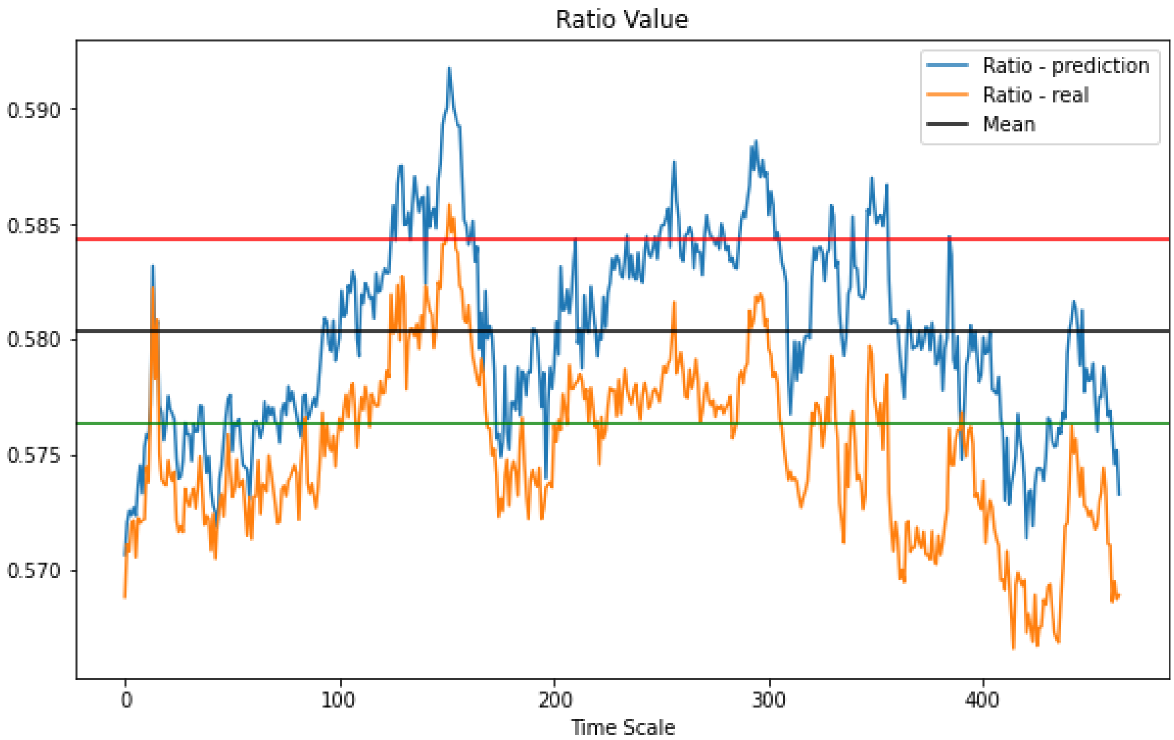




Now that we predicted the prices for these two ETFs, we can calulate the ratio

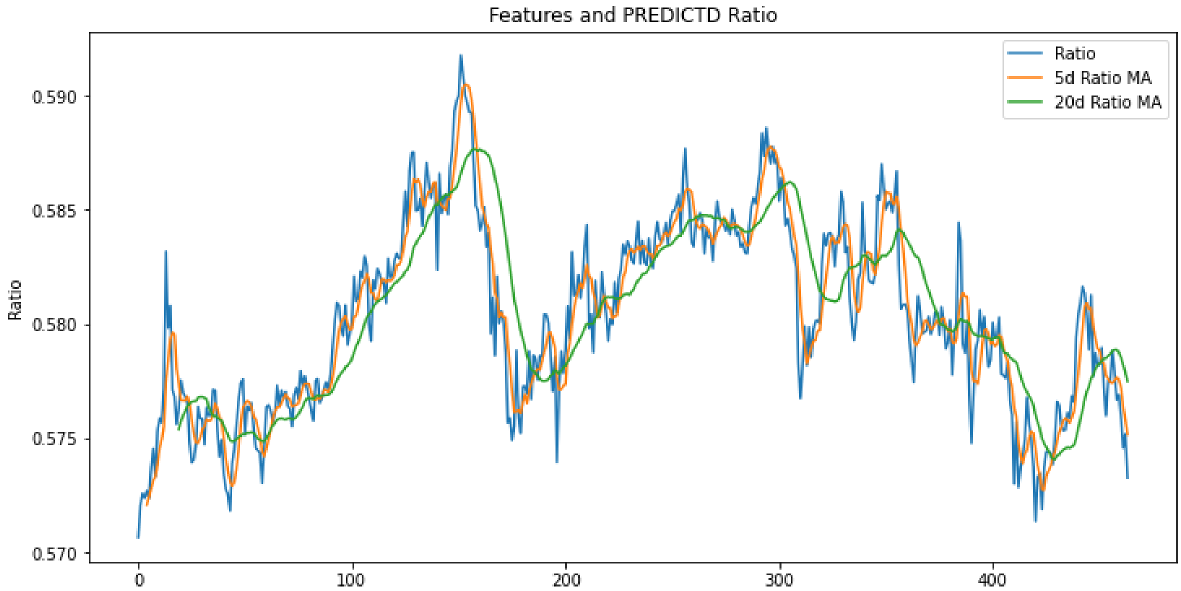
In [44]:

```
ratio_arr = y_pred1 / y_pred2
real_ratio_arr = y_test1 / y_test2
plt.figure(figsize=(10,6))

plt.plot(ratio_arr, label='Ratio - prediction')
plt.plot(real_ratio_arr, label='Ratio - real')
plt.title("Ratio Value")
plt.xlabel('Time Scale')
m = ratio_arr.mean()
plt.axhline(m, color='black')
plt.axhline(m + 0.004, color='red')
plt.axhline(m - 0.004, color='green')
plt.legend(['Ratio - prediction', 'Ratio - real', 'Mean'])
plt.show()
```



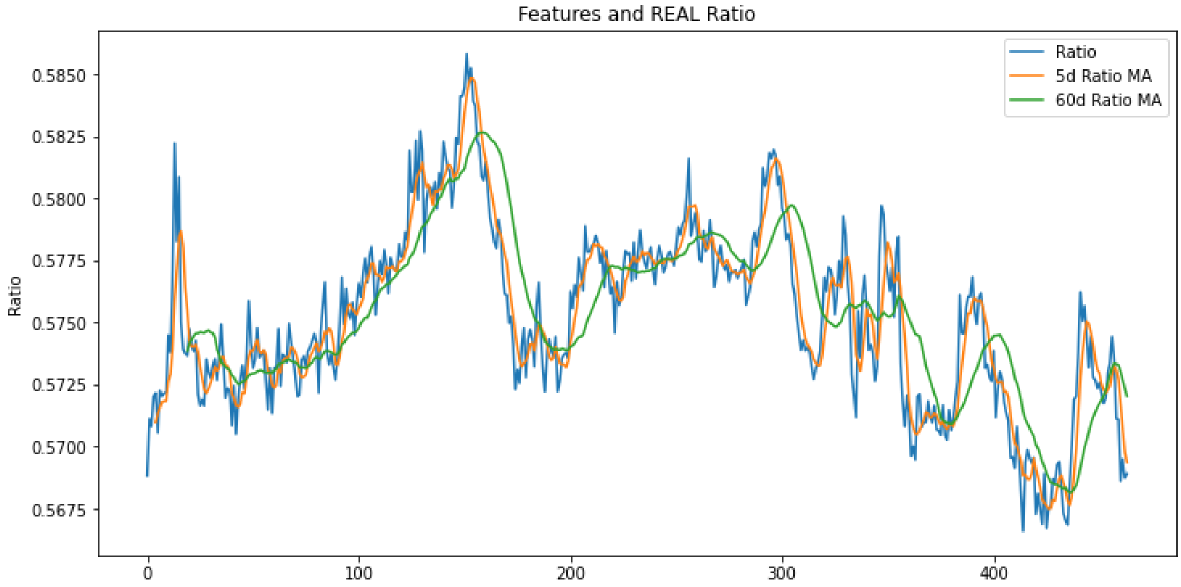
In [45]:

```
r_df = pd.DataFrame(ratio_arr)
ratios_mavg5_ = r_df.rolling(window=5, center=False).mean()
ratios_mavg20_ = r_df.rolling(window=20, center=False).mean()
std_20_ = r_df.rolling(window=20, center=False).std()
zscore_20_5_ = (ratios_mavg5_ - ratios_mavg20_)/std_20_
plt.figure(figsize=(12, 6))
plt.plot(ratio_arr)
plt.plot(ratios_mavg5_)
plt.plot(ratios_mavg20_)
plt.legend(['Ratio', '5d Ratio MA', '20d Ratio MA'])
plt.title('Features and PREDICTD Ratio')

plt.ylabel('Ratio')
plt.show()
```



In [46]:

```
r_df_ = pd.DataFrame(real_ratio_arr)
ratios_mavg5_1 = r_df_.rolling(window=5, center=False).mean()
ratios_mavg60_1 = r_df_.rolling(window=20, center=False).mean()
std_60_1 = r_df_.rolling(window=20, center=False).std()
zscore_60_5_1 = (ratios_mavg5_1 - ratios_mavg60_1)/std_60_1
plt.figure(figsize=(12, 6))
plt.plot(real_ratio_arr)
plt.plot(ratios_mavg5_1)
plt.plot(ratios_mavg60_1)
plt.legend(['Ratio', '5d Ratio MA', '60d Ratio MA'])
plt.title('Features and REAL Ratio')

plt.ylabel('Ratio')
plt.show()
```



Testing

Now that we have the ratio prediction, we can test our model

We will use the 'trade' function that is written above

In [47]:

```
# replace the real closing price in the predicted closing price
smh_close = smh[['Close']]
smh_close_train = smh_close.iloc[:train_index]
smh_close_test = smh_close.iloc[train_index:]
smh_close_test = smh_close_test[['Close']]
smh_close_pred = smh_close_test.copy()
smh_close_pred[['Close']] = y_pred1
smh_close_test

soxx_close = soxx[['Close']]
soxx_close_train = soxx_close.iloc[:train_index]
soxx_close_test = soxx_close.iloc[train_index:]
soxx_close_pred = soxx_close_test.copy()
soxx_close_pred[['Close']] = y_pred2
soxx_close_pred.head()
```

Out[47]:

Date	Close
2020-07-08	270.525146
2020-07-09	273.480896
2020-07-10	274.646423
2020-07-13	277.087128
2020-07-14	270.412628

In [48]:

```
ı = ratio_arr.mean()
es_pred = trade(smh_close_pred['Close'], soxx_close_pred['Close'], 20, 5, zscore_sel
```

In [49]:

```
m = real_ratio_arr.mean()
res_real = trade(smh_close_test['Close'], soxx_close_test['Close'], 20, 5, zscore_se
```

Calculating the error in profits

In [50]:

```
abs(res_pred - res_real)
```

Out[50]:

```
0.00032043457022723487
```

Notice the error is really small!

However, we did not do well regarding the profits themselves:

In [51]:

```
print(f"we made {res_pred} dollars in prediction trading")
print(f"we made {res_real} dollars in real trading")
```

```
we made -0.0003204345703125 dollars in prediction trading
we made -8.526512829121202e-14 dollars in real trading
```

In [52]:

```
ratio_windows = [i for i in range(1, 30)]
std_windows = [i for i in range(1, 30)]
zindex_windows_sell = [round(i,1) for i in np.arange(0.001, 0.01, 0.001)]
zindex_windows_buy = [round(i,1) for i in np.arange(-0.001, -0.01, -0.001)]

combinations = list(list(product(ratio_windows, std_windows, zindex_windows_sell, zi
print(f"number of combinations is: {len(combinations)}")
```

```
number of combinations is: 68121
```

In [53]:

```
def find_best_combination(combinations):
    best_combination = combinations[0]
    scores = []
    best_score = -100000
    tracking = 0
    for ratio_window, std_window, zindex_window_sell, zindex_window_buy in combinati
        score = trade(smh_close_train['Close'], soxx_close_train['Close'], ratio_wir
        if score > best_score:
            best_combination = (ratio_window, std_window, zindex_window_sell, zindex
            best_score = score
        tracking += 1
        if tracking % 1000 == 0:
            print(f"after {tracking} iterations")
        scores.append(score)
    print(f"best combination is: {best_combination} with score: {best_score}")
    return scores
```

In [54]:

```
scores = find_best_combination(combinations)
```

```
after 1000 iterations
after 2000 iterations
after 3000 iterations
after 4000 iterations
after 5000 iterations
after 6000 iterations
after 7000 iterations
after 8000 iterations
after 9000 iterations
after 10000 iterations
after 11000 iterations
after 12000 iterations
after 13000 iterations
after 14000 iterations
after 15000 iterations
after 16000 iterations
after 17000 iterations
after 18000 iterations
after 19000 iterations
after 20000 iterations
after 21000 iterations
after 22000 iterations
after 23000 iterations
after 24000 iterations
after 25000 iterations
after 26000 iterations
after 27000 iterations
after 28000 iterations
after 29000 iterations
after 30000 iterations
after 31000 iterations
after 32000 iterations
after 33000 iterations
after 34000 iterations
after 35000 iterations
after 36000 iterations
after 37000 iterations
after 38000 iterations
after 39000 iterations
after 40000 iterations
after 41000 iterations
after 42000 iterations
after 43000 iterations
after 44000 iterations
after 45000 iterations
after 46000 iterations
after 47000 iterations
after 48000 iterations
after 49000 iterations
after 50000 iterations
after 51000 iterations
after 52000 iterations
after 53000 iterations
after 54000 iterations
after 55000 iterations
after 56000 iterations
after 57000 iterations
```

```
after 58000 iterations
after 59000 iterations
after 60000 iterations
after 61000 iterations
after 62000 iterations
after 63000 iterations
after 64000 iterations
after 65000 iterations
after 66000 iterations
after 67000 iterations
after 68000 iterations
best combination is: (24, 29, 0.0, -0.0) with score: 3.552713678800501
e-13
```

So we found that the best result is

- small_ratio=3 days,
- std & big ratio=5 days,
- zindex=0

with score: 4.689582056016661e-13

In [56]:

```
res_pred = trade(smh_close_pred['Close'], soxx_close_pred['Close'], 24, 29, 0, 0)
res_real = trade(smh_close_test['Close'], soxx_close_test['Close'], 24, 29, 0, 0)
print(f"we made {res_pred} dollars in prediction trading")
print(f"we made {res_real} dollars in real trading")
```

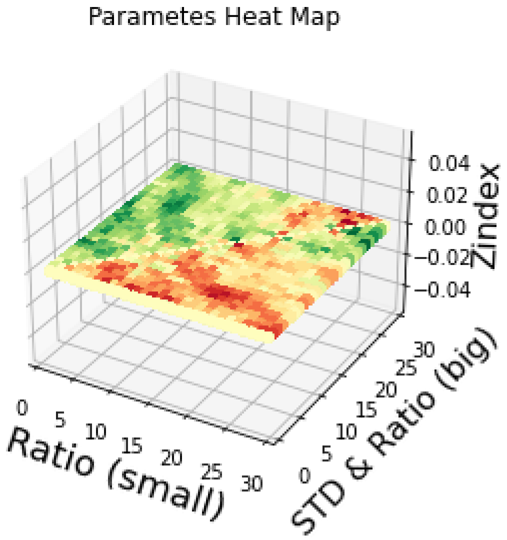
```
we made 7.62939453125e-05 dollars in prediction trading
we made 0.0 dollars in real trading
```

Results

Now we can plot a graph to test the results we have received

In [65]:

```
from mpl_toolkits.mplot3d import Axes3D
# fig = plt.figure()

x = [combination[0] for combination in combinations]
y = [combination[1] for combination in combinations]
z = [combination[2] for combination in combinations]
# ax.scatter(X, Y, Z, c=scores, lw=0, s=20)

ax = plt.axes(projection='3d')
ax.scatter(x, y, z, c=scores, cmap='RdYlGn_r');
ax.set_xlabel('Ratio (small)', fontsize=18)
ax.set_ylabel('STD & Ratio (big)', fontsize=16)
ax.set_zlabel('Zindex', fontsize=16)
# ax.set_zlim(-0.00001,0.00001)
plt.title('Parametes Heat Map')
plt.show()
```



Easy to see that the best parameters are set with high ratio look back window, low STD look back window and pretty much the same for each zindex

Sharp Ratio

In [70]:

```
R = pd.DataFrame(scores)
# mean_log_returns = np.log(returns + 1).mean()
# mean_returns = np.exp(mean_log_returns) - 1
# std = returns.std()
# sharpe_ratio = (mean_returns / std) * np.sqrt(252)
# sharpe_ratio[0]
# r = (R - R.shift(1))/R.shift(1)

# Approach 2
r = R.diff()

sr = r.mean()/r.std() * np.sqrt(252)
sr
```

Out[70]:

```
0    0.0
dtype: float64
```

Sharpe indeed looks good, points on good in-sample results

Conclusion

Our model didn't hold well out-of-sample.

1. We would need to enlarge the number pairs we are using.
2. We would then calculate more accurate results, based on an average between all the pairs.
3. We would then test all the "test" parts.

Improvement

Now we will try to do the same process, but for several pairs. It seems like optimizing one pair just wasn't enough, and that's why we have received bad results.

In [ ]:

In [ ]:

In [ ]: