Implementation: Cryptocurrency

Group 1

Yichen Song | Yu Sheng | Weiqi Tong | Yating Liu

Cryptocurrency is a new form of money created around the year 2009 through a white-paper introduced by Satoshi Nakamoto to digitize and store 'value' through the internet, known as Bitcoin. Since then, thousands of other cryptocurrencies have been created.

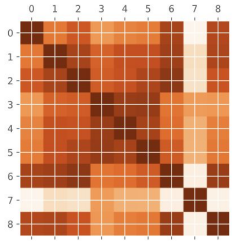
Cryptocurrency uses cryptography for security. Many cryptocurrencies are decentralized systems based on blockchain technology, a distributed ledger enforced by a disparate network of computers. It is not issued by any central authority, rendering it theoretically immune to government interference or manipulation. Since prices are based on supply and demand, the rate at which a cryptocurrency can be exchanged for another currency can fluctuate widely.

The current market valuation for all cryptocurrencies is about $400 billion USD. Evidenced by the drastic increase in cryptocurrency prices and market caps in 2017, it presents a tremendous opportunity for investment. On the opposite side of this opportunity, the cryptocurrency market contains high volatility and numerous risks. Thus we choose cross-cryptocurrency arbitrage strategy and trade on the spreads of cryptocurrency pairs to hedge the risk. This arbitrage is based on mean-reverting effects of spread.

I. Specification

A. Universe

In this implementation, we choose to use the top nine most traded cryptocurrencies (listed in https://www.blockchain.com/markets) including bitcoin and eight altcoins: eos, ripple, ethereum, bitcash, litecoin, stellar, tether, tron. We choose nine cryptocurrencies because we want to have a decent number of pairs to construct the portfolio and hedge the risk while still having enough historical data for backtesting. The biggest limit of cryptocurrency is the lack of historical data. Many cryptocurrencies are recently offered. For example, one of the most traded cryptocurrency Bitcoin SV starts trading from November 2018.



This graph shows the correlation between 9 cryptocurrencies.

B. Date Range

Since the price data for tron(one of the top 10 cryptocurrency) is only available after August 2017, our data range starts from August 2017 to December 2018. Following the rule of 7:3 train-test ratio, which is commonly used in machine learning algorithms, we choose data from August 2017 to July 2018 for in-sample test and from July 2018 to present for out-of-sample test.

C. Data Sources

We download our data(price of cryptocurrency in USD) from https://coinmetrics.io/. Because we want to focus on cross-currency arbitrage and ignore the effect of rate differences in different exchanges, we try to find data that reflects the average price of cryptocurrencies across different exchanges. The methodology this website used to calculate the price of cryptocurrency is consistent with our goal. We also check the validity of the datasets by comparing it with the data found in https://www.blockchain.com/, and the difference is not significant.

D. Signal Generation

For each pair of cryptocurrency, we use deviations of spread from mean as the signal. For example, one of the pairs we choose is ripple and bitcoin. First for the training period, we run linear regression with ripple as Y variable and bitcoin as X variable, and calculate prediction residuals. Regression model: Y=beta*X+c, where Y, X represents the price of ripple and bitcoin respectively, beta is the regression coefficient.

Prediction residual = realY - predictY = real Y-beta* realX-c

Then we find the mean and standard deviation of these residuals. Our signal for testing data is (residuals for testing set - mean of training set)/std of training set

E. Portfolio Construction

Based on our universe of 9 cryptocurrencies, we first perform cointegration tests on every possible cryptocurrency pairs using training dataset. We construct the portfolio using 7 pairs that pass the cointegration test, which shows a statistically significant connection between two cryptocurrencies. Moreover, we assume that the spread between two cryptocurrencies is mean reverting. After that, we perform linear regression on each selected pair and trade each pair using the signals from the linear regression model.

For the basic implementation, we use equal weighting for each pair. That is, we trade the spread with 1/7 of our money for each pair.

The time horizon for our strategy is one week, which means we will close the trade if it has last for one week. This time horizon is set for the liquidity purpose.

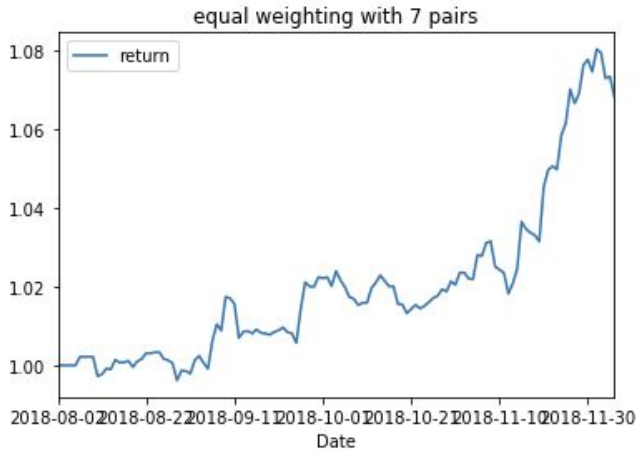
Also, we would open a margin account. When we don't trade certain spread (signal not large enough), we would deposit that money into an account, which will make risk free profit.

F. Execution

We start with initial investment of $1,000,000,000.
Every time when we find a signal for long position, we divide current total asset (including initial investment and cumulative PnL so far) into seven equal parts A1, to A7. Based on signal, we short the spread when the spread goes down and crosses over mean+z*std. By shorting the spread, we short Y cryptocurrency, and long X cryptocurrency. We long the spread when the spread goes up and crosses over mean-z*std. More detaily, take one pair, bitcoin and bitcoin cash, for example, when the spread crosses over mean+z*std, we will short A1*beta /(1+beta) dollars of bitcoin and long A1/(1+beta) dollars of bitcoin cash, where beta is the regression coefficient of bitcoin against bitcoin cash.
Trading frequencies mainly depends on the signals, but also restricted by time horizon one week.

II. Implementation

A. PnL Graph



We can see from the graph there is a continuous upward trend of return with small fluctuations, especially there is a sharp increase in December. Notice that there exists a downward trend after November, which may provide some uncertainty for the future.

B. Stats

Annualized return: 26.3% | Sharpe ratio: 2.229| Maximum drawdown: 1.295% | Volatility: 2.1%

The return is much higher than we expected with respect to the large volatility and uncertainty in cryptocurrency market. This indicates trading on spreads is a good decision and hedge the risk well. Sharpe ratio is 2.587, bigger than 2 which indicates this is a good systematic strategy. Maximum drawdown and volatility are small shows that this strategy has a low risk. However, transaction cost is not included in this graph thus conclusion above need to be evaluated further.

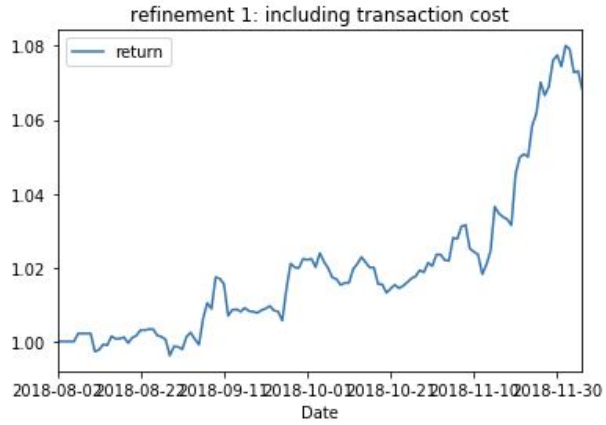
C. Difficulties

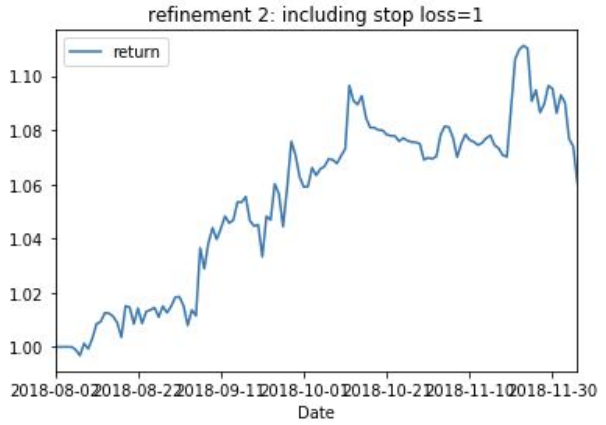
At first we were not sure what is a reliable source for cryptocurrency data. So we go online and compare several data sources and choose the one we are using right now.
Because some cryptocurrencies are newly offered, like Bitcoin SV which starts trading from 2018-11-09, it limits the data we could use. Moreover, data for different currencies start from different dates. Thus we have to select the same starting date, and to have a reasonably long training and testing period, we decide to disregard Bitcoin SV and only include the other 9 currencies from the top 10 most traded cryptocurrencies.
We implement the refinements separately by writing separate functions written by different team members. After we figure out what refinements give a better result, it is hard to combine these features into a single function since different people have different coding style. It took us some time to achieve this goal.
Besides, we have many refinement ideas. However, the most thrilling ones about signal weighting and net positions are complicated. We tried to implement, but our codes return many bugs that we are unable to fix. Therefore, we can only mention it in the proposal.

III. Refinements [This section details and shows results for any follow up research.]

A. Implemented

(1) Transaction cost:
From our research we found that the transaction cost is usually a fixed price per unit of currency. To convert it to a percentage, we divide it by the historical minimum price. When trading with each pair, the total transaction cost is calculated as a weighted average of transaction costs of two cryptocurrencies(using linear regression coefficient beta). From the PnL graph, we can see that the result is rather close to the basic trade when we did not include transaction costs.

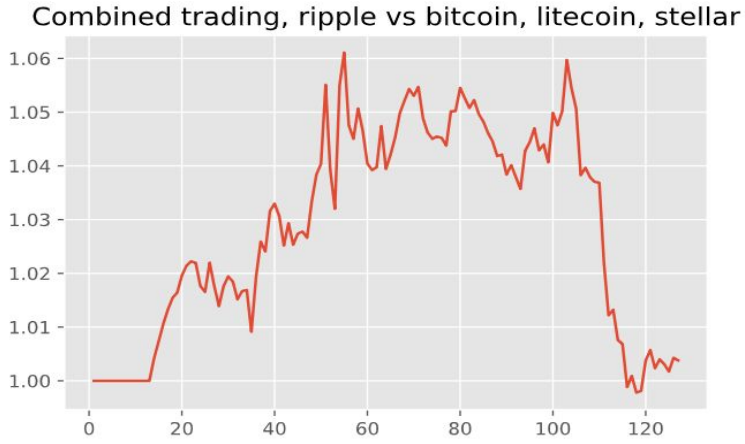




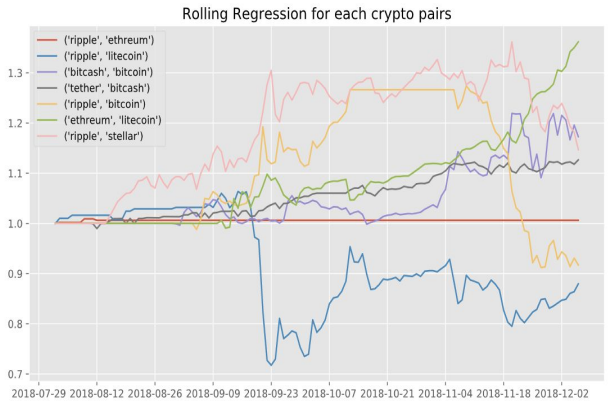
(2)Stop loss:

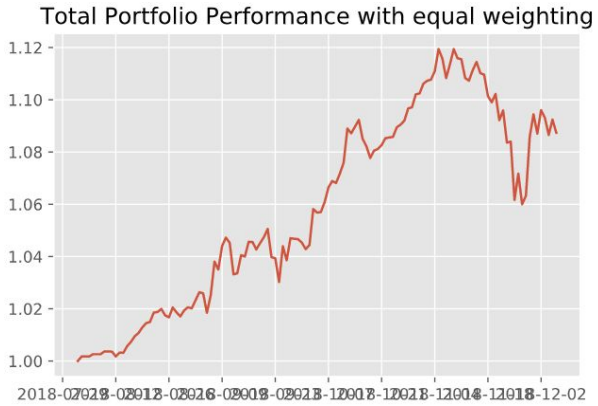
For this refinement, we include stop loss=1, and the z value for each pair is the same as the best z value we found from the basic trade. It gives a surprisingly good result, except in the last month when there is a significant drawdown. We guess the reason might be that during the last month although the spread deviates from the mean by a large amount, it eventually reverts back, but with stop loss we clear our position before we gain the profit.

(3) Combine pairs with common cryptocurrency into the same linear regression model:
Since it is observed that ripple is cointegrated with bitcoin, litecoin and stellar, we are interested in whether creating a combined portfolio for these four currencies may result in good performance. In this case, we set ripple as y and the other three coins as X in the linear regression. However, it seems the performance is not quite satisfactory as shown below. The reason may be that the regression coefficients of stellar is much larger compared to the others, so the performance is quite risky in some periods of time.



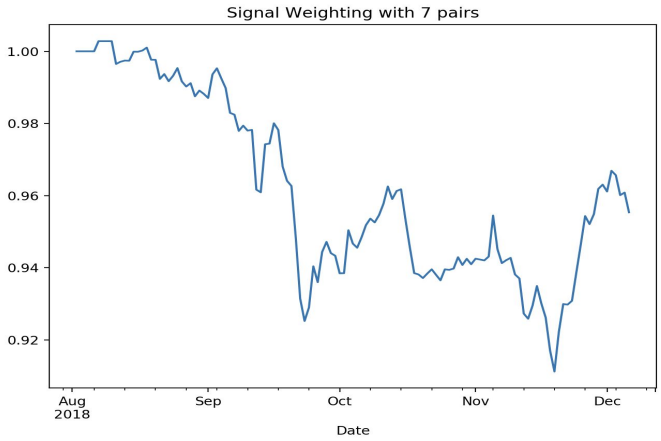
(4) Change linear regression to rolling regression:
This tends to be a meaningful refinement. The linear regression model in the original implementation is based on a fixed time period which is our training period and remains unchanged through the backtest. However, we can retrain the regression every day and calculate the signal for the next one time point. This can make the regression model most updated and capture recent patterns in price movements. Note the regression coefficient is changing day by day now. To reduce transaction cost, if we already enter the market, we wouldn't adjust our position. We will only use the latest regression to calculate spread. That means only when we clear our position and reenter the market, will we use the latest regression coefficient to determine our position on different components of the same pair. The left figure below shows the performance of each pair, and the right shows the total portfolio performance.





(5) Use correlation as weighting :

We believe that in linear regression model R square indicates how close two cryptocurrencies are related. Therefore, we would like to include the in sample R square to construct our portfolio. The higher the in sample R square of two cryptocurrencies is, the more money we would invest in this pair. Although we think this refinement is reasonable, the result seems to be disappointing,



B. Proposals

In this portfolio, we focus on trading spreads, neglecting the actual components of the spread(two cryptocurrencies that compose spread). However, you may already notice that there are actually duplicate components in different pairs in our portfolio. For instance, litecoin appears in both (ripple, litecoin) and (ethereum, litecoin). Therefore there may exist some trading in both directions of litecoin; we may be long litecoin in trading (ripple, litecoin) and short litecoin in trading in (ethereum, litecoin). Therefore, instead of trading on spreads separately, we may construct a portfolio merging the position of same cryptocurrency to reduce the transaction cost.

For the money management part, it is actually more intuitive to use z score of each spread (price of spread-mean divided by volatility sigma) for signal weighting. Since z score represents the strongness of the signal, we should trade more on the spread with strong signals. However, this

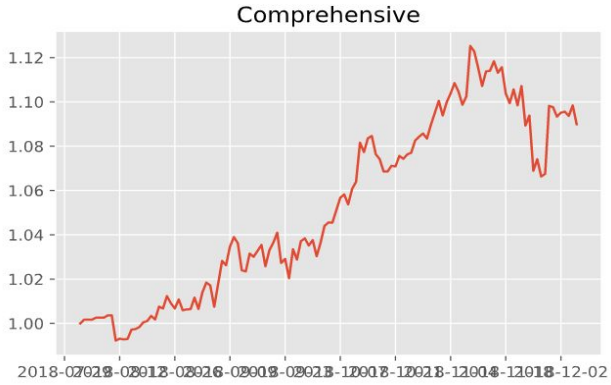
may lead to a problem of a high rebalance frequency, since signal is changing all the time. Ultimately, this weighting strategy may lead to a high transaction cost. The possible way to solve it is by combining the net position strategy as mentioned above, which we have not yet explored.

Last but not least, as mentioned in the refinement, we do not include the latency cost (bid-ask spread) in the transaction cost since we do not have the price data in minutes. If we have data of higher frequency, then the trading price should be the price of cryptocurrency when the transaction is officially completed (normally around 30 minutes). Latency cost plays an important role in whether we can make the money in practice, many fund lost their profit because of it. We would like to conduct further research on it.

IV. Conclusion

A. Out of Sample Test(s)

From the above refinements, we found that including transaction costs, stop loss and using rolling regression to calculate spread gives a more reasonable and better pnl results. Thus, for the comprehensive out of sample test, we include these three features and get the following graph:



Annualized return: 29.4% | Sharpe ratio: 2.587| Maximum drawdown: 5.35% | Volatility: 3.79%

We can observe that starting from October the portfolio suffers some loss. It is reasonable since all the cryptocurrencies have been falling with high volatility. Before this period, our performance is very stable. Note that since we only use a testing period of four months, the pattern may not have a good level of generalization. Therefore, the annualized return calculated may not be very meaningful. A more robust testing requires longer time periods.

B. Trading Recommendation

However, although we have a positive return from our out-of-sample test, since our out of sample dataset is relatively small, the result may not be reliable. In addition, the waiting time, could be extremely long. It can take up to 16 hour in extreme cases for one transaction to be confirmed and cryptocurrencies actually appear in our wallet. If we want to decrease waiting time, we have to pay higher transaction fee.
The cryptocurrency market has been volatile as ever over the last 6 months. In addition, the cryptocurrency is in bear market now. Unless you are a skilled trader, it is harder to make money in a bear market than in a bull market – and we have been in a bear market for some time now. Therefore we would only recommend this strategy to a skillful trader.

V. Appendix

```
import pandas as pd
import numpy as np
from statsmodels.tsa.stattools import coint
from sklearn.linear_model import LinearRegression
import matplotlib.pyplot as plt

# from __future__ import division


# three cryptocurrencies: ripple, EOS, ethreum, bitcoin

# bitcoin starts from to
bit = pd.read_csv('/Users/Yating/Desktop/MATH5300 Hedge fund/implementation2/btc.csv', skiprows=1,
                  names=['date', '1', '2', '3', '4', 'price', '5', '6', '7', '8',
                         '9', '10', '11', '12', '13', '14'])
bit.index = pd.to_datetime(bit['date'])
bit = bit['2017-08-28':'2018-12-06']

# eos starts from 2017-07-02 to 2018-12-06
eos = pd.read_csv('/Users/Yating/Desktop/MATH5300 Hedge fund/implementation2/eos.csv', skiprows=1,
                  names=['date', '1', '2', '3', 'price', '4', '5', '6', '7', '8'])
eos.index = pd.to_datetime(eos['date'])
eos = eos['2017-08-28':'2018-12-06']

# ripple starts from 2013-01-02 to 2018-12-06
ripple = pd.read_csv('/Users/Yating/Desktop/MATH5300 Hedge fund/implementation2/xrp.csv', skiprows=1,
                     names=['date', '1', '2', '3', '4', '5', '6', '7', '8', 'price', '9'])
ripple.index = pd.to_datetime(ripple['date'])
ripple = ripple['2017-08-28':'2018-12-06']

# ethreum starts from 2015-08-07 to 2018-12-06
ethreum = pd.read_csv('/Users/Yating/Desktop/MATH5300 Hedge fund/implementation2/eth.csv', skiprows=9,
                      names=['date', 'txVolume', 'adjustedTxVolume',
                             'txCount', 'Marketcap', 'price', 'exchangeVolume', 'generatedCoins',
                             'fees', 'activeAddresses', 'medianTxValue', 'medianFee', 'averageDifficulty',
```

```
                          'paymentCount', 'blockSize', 'blockCount'])
ethreum.index = pd.to_datetime(ethreum['date'])
ethreum = ethreum['2017-08-28':'2018-12-06']

bitcash = pd.read_csv('/Users/Yating/Desktop/MATH5300 Hedge fund/implementation2/bch.csv', skiprows=1,
                      names=['date', '1', '2', '3', '4', 'price', '5', '6',
                             '7', '8', '9', '10', '11', '12', '13', '14', '15'])
bitcash.index = pd.to_datetime(bitcash['date'])
bitcash = bitcash['2017-08-28':'2018-12-06']

litecoin = pd.read_csv('/Users/Yating/Desktop/MATH5300 Hedge fund/implementation2/ltc.csv', skiprows=1,
                       names=['date', '1', '2', '3', '4', 'price', '5', '6',
                              '7', '8', '9', '10', '11', '12', '13', '14', '15'])
litecoin.index = pd.to_datetime(litecoin['date'])
litecoin = litecoin['2017-08-28':'2018-12-06']

stellar = pd.read_csv('/Users/Yating/Desktop/MATH5300 Hedge fund/implementation2/xlm.csv', skiprows=1,
                      names=['date', '1', '2', '3', 'price', '4', '5', '6'])
stellar.index = pd.to_datetime(stellar['date'])
stellar = stellar['2017-08-28':'2018-12-06']

tether = pd.read_csv('/Users/Yating/Desktop/MATH5300 Hedge fund/implementation2/usdt.csv', skiprows=1,
                     names=['date', '1', '2', '3', 'price', '4', '5', '6', '7'])
tether.index = pd.to_datetime(tether['date'])
tether = tether['2017-08-28':'2018-12-06']

tron = pd.read_csv('/Users/Yating/Desktop/MATH5300 Hedge fund/implementation2/trx.csv', skiprows=1,
                   names=['date', '1', '2', '3', '4', '5', '6', 'price', '7'])
tron.index = pd.to_datetime(tron['date'])
tron = tron['2017-08-28':'2018-12-06']

t = '2018-08-01'

# calculate correlation:
df = pd.DataFrame()
df['eos'] = eos['price'][:t]
df['ripple'] = ripple['price'][:t]
df['ethreum'] = ethreum['price'][:t]
df['bitcoin'] = bit['price'][:t]
df['bitcash'] = bitcash['price'][:t]
df['litecoin'] = litecoin['price'][:t]
df['stellar'] = stellar['price'][:t]
df['tether'] = tether['price'][:t]
df['tron'] = tron['price'][:t]

df.corr().abs().unstack().drop_duplicates()
'''
eos       eos         1.000000
          ripple      0.597055
          ethreum     0.695612
```

```
        bitcoin     0.498773
        bitcash     0.570974
        litecoin    0.590499
        stellar     0.815531
        tether      0.065561
        tron        0.812631
ripple  ethreum     0.783339
        bitcoin     0.684217
        bitcash     0.752264
        litecoin    0.750291
        stellar     0.835912
        tether      0.185649
        tron        0.822196
ethreum bitcoin     0.733027
        bitcash     0.810370
        litecoin    0.861859
        stellar     0.758151
        tether      0.314995
        tron        0.726742
bitcoin bitcash     0.879286
        litecoin    0.884730
        stellar     0.576233
        tether      0.478593
        tron        0.513922
bitcash litecoin    0.875709
        stellar     0.575844
        tether      0.437779
        tron        0.602385
litecoin stellar    0.630572
        tether      0.439228
        tron        0.617143
stellar tether      0.028404
        tron        0.850220
tether  tron        0.100769
dtype: float64
'''


# choose ripple, ethereum and litecoin


# cointegration:
def get_pairs(train):
    s = train.corr().abs().unstack()
    # .drop_duplicates()
    corr = s.sort_values()
    pairs = list(corr.index)
    ans = []
    for pair in pairs:
        _, p_value, _ = coint(df[[pair[0]]], df[[pair[1]]])
```

```
        if p_value < 0.05:
            ans.append(pair)
    selected = corr[ans]
    return list(selected.index)


print(get_pairs(df))


# [('ripple', 'litecoin'), ('litecoin', 'ethreum'), ('ripple', 'ethreum')]

# [('tether', 'bitcash'), ('ripple', 'bitcoin'), ('ripple', 'litecoin'), ('ethreum', 'litecoin'),
# ('ripple', 'ethreum'), ('ripple', 'stellar'), ('bitcash', 'bitcoin')]


##########################################################################################
# choose seven pairs: 1) ripple and bitcoin   2) ripple and ethreum


def basic_pair_trade(c1, c2):  # two dataframes for currency, c1 is X, c2 is Y
    X = pd.DataFrame()
    X['X'] = c1['price']
    X.index = pd.to_datetime(c1['date'])
    ols = LinearRegression().fit(X[:t], c2['price'][:t])
    train_spread = c2['price'][:t] - ols.predict(X[:t])
    mean = np.mean(np.array(train_spread))
    std = np.std(np.array(train_spread))

    test_spread = c2['price'][t:] - ols.predict(X[t:])

    sig = (test_spread - mean) / std
    df1 = pd.DataFrame()
    df1['signal'] = sig
    df1['spread'] = test_spread
    df1['beta'] = ols.coef_[0]
    df1['X'] = c1[t:]['price']
    df1['Y'] = c2[t:]['price']
    df1['Date'] = c1[t:]['date']

    zEntries = np.arange(start=0.5, stop=4, step=0.1)
    exitZscore = 0
    cum_rets = pd.DataFrame()

    for z in zEntries:

        df1['long entry'] = ((sig < -1.0 * z) & (sig.shift(1) > -1.0 * z))
        df1['long exit'] = ((sig > - exitZscore) & (sig.shift(1) < - exitZscore))
        df1['num units long'] = np.nan
        df1.loc[df1['long entry'], 'num units long'] = 1
        df1.loc[df1['long exit'], 'num units long'] = 0
```

```
    df1['num units long'][0] = 0
    df1['num units long'] = df1['num units long'].fillna(method='pad')

    # set up num units short
    df1['short entry'] = ((sig > z) & (sig.shift(1) < z))
    df1['short exit'] = ((sig < exitZscore) & (sig.shift(1) > exitZscore))
    df1.loc[df1['short entry'], 'num units short'] = -1
    df1.loc[df1['short exit'], 'num units short'] = 0
    df1['num units short'][0] = 0
    df1['num units short'] = df1['num units short'].fillna(method='pad')

    df1['numUnits'] = df1['num units long'] + df1['num units short']

    df1['spread pct ch'] = (df1['spread'] - df1['spread'].shift(1)) / (
        df1['X'] * np.abs(df1['beta']) + df1['Y'])

    df1['port rets'] = df1['spread pct ch'] * df1['numUnits'].shift(1)

    df1['cum rets'] = df1['port rets'].cumsum()
    df1['cum rets'] = df1['cum rets'] + 1

    if cum_rets.empty:
        cum_rets = pd.DataFrame(df1['cum rets'])
        cum_rets.rename(columns={'cum rets': 'cumRets_{}'.format(round(z, 2))}, inplace=True)
    else:
        cum_rets = cum_rets.join(pd.DataFrame(df1['cum rets']), how="outer")
        cum_rets.rename(columns={'cum rets': 'cumRets_{}'.format(round(z, 2))}, inplace=True)

print("\nBACKTEST Y against X:\n")
try:
    for i in cum_rets.columns:
        print(
            "Z-Score Entry {}: ".format(i[8:11]),
            str((cum_rets[i][len(cum_rets) - 1] - cum_rets[i][1]) * 100)[0:4]
        )
    maxCAGR = 0.0
    bestZ = ""
    for i in cum_rets.columns:
        if float((cum_rets[i][len(cum_rets) - 1] - cum_rets[i][1]) * 100) > float(maxCAGR):
            maxCAGR = (cum_rets[i][len(cum_rets) - 1] - cum_rets[i][1]) * 100
            bestZ = i
            print("\nBest 1-yr return {} with {} ZScore Entry".format(
                str((cum_rets[bestZ][len(cum_rets) - 1] - cum_rets[bestZ][1]) * 100)[0:4] + "%",
                bestZ[8:11])
            )
            print("1-yr Return {} with 1.0 ZScore Entry".format(
                str((cum_rets["cumRets_1.0"][len(cum_rets) - 1] - cum_rets["cumRets_1.0"][1]) * 100)[0:4] + "%"
            ))
except KeyError:
    print("KeyError")
```

```
    cum_rets.index = pd.to_datetime(df1['Date'])
    cumRets_1 = cum_rets['cumRets_1.0']
    cum_rets.drop(labels=[i for i in cum_rets.columns if i != bestZ], inplace=True, axis=1)

    if bestZ != "cumRets_1.0":
        cum_rets = cum_rets.join(pd.DataFrame(cumRets_1))
        cum_rets = cum_rets.join(df1['Date'])

    return cum_rets


# [('tether', 'bitcash'), ('ripple', 'bitcoin'), ('ripple', 'litecoin'), ('ethreum', 'litecoin'),
# ('ripple', 'ethreum'), ('ripple', 'stellar'), ('bitcash', 'bitcoin')]
rets1 = basic_pair_trade(tether, bitcash)
rets2 = basic_pair_trade(ripple, bit)
rets3 = basic_pair_trade(ripple, litecoin)
rets4 = basic_pair_trade(ethreum, litecoin)
rets5 = basic_pair_trade(ripple, ethreum)
rets6 = basic_pair_trade(ripple, stellar)
rets7 = basic_pair_trade(bitcash, bit)

cum_rets = pd.DataFrame()
cum_rets['Date'] = bit['date']['2018-08-02':]
cum_rets.index = cum_rets['Date']
rets = []
# rets.append(0)
for i in range(1, len(rets1)):  # skip the first entry which is nan
    mean = (rets1.iloc[i, 0] + rets2.iloc[i, 0] + rets3.iloc[i, 0] + rets4.iloc[i, 0] + rets5.iloc[i, 0] + rets6.iloc[
        i, 0] + rets7.iloc[i, 0]) / 7
    rets.append(mean)
cum_rets['return'] = rets

cum_rets.plot()
plt.title("equal weighting with 7 pairs")

# stats:
port_ret = []
port_ret.append(0)
for i in range(len(rets) - 1):
    port_ret.append(rets[i + 1] - rets[i])
sharpe = np.mean(port_ret) / np.std(port_ret) * (252 ** 0.5)
print('sharpe is: ', sharpe)

D = len(rets)
ann_ret = rets.prod() ** (12 / D) - 1
print('annualized return is: ', ann_ret)

vol = np.std(cum_rets['return'])
```

```
def max_drawdown(X):
    mdd = 0
    peak = X[0]
    for x in X:
        if x > peak:
            peak = x
        dd = (peak - x) / peak
        if dd > mdd:
            mdd = dd
    return mdd


maxd = max_drawdown(cum_rets['return'])
print('volatility is: ', vol)
print('maximum drawdown is: ', maxd)

#################################################################################################
# refinement 1: include transaction cost:
mB = np.min(bit['price'])
mEthreum = np.min(ethreum['price'])
mLite = np.min(litecoin['price'])
mBcash = np.min(bitcash['price'])
mRipple = np.min(ripple['price'])
mStellar = np.min(stellar['price'])
cost = {'bit': 1.184 / mB, 'ethreum': 0.347 / mEthreum, 'litecoin': 0.198 / mLite, 'bitcash': 0.097 / mBcash,
        'ripple': 0.0037 / mRipple, 'stellar': 0.00001 / mStellar, 'tether': 0.0025}

bit.name = 'bit'
ethreum.name = 'ethreum'
litecoin.name = 'litecoin'
bitcash.name = 'bitcash'
ripple.name = 'ripple'
stellar.name = 'stellar'
tether.name = 'tether'


def transaction_cost(c1, c2):  # c1 is X and c2 is Y
    cost1 = cost[c1.name]
    cost2 = cost[c2.name]
    X = pd.DataFrame()
    X['X'] = c1['price']
    X.index = pd.to_datetime(c1['date'])
    ols = LinearRegression().fit(X[:t], c2['price'][:t])
    train_spread = c2['price'][:t] - ols.predict(X[:t])
    mean = np.mean(np.array(train_spread))
    std = np.std(np.array(train_spread))

    test_spread = c2['price'][t:] - ols.predict(X[t:])
```

```
  sig = (test_spread - mean) / std
  df1 = pd.DataFrame()
  df1['signal'] = sig
  df1['spread'] = test_spread
  df1['beta'] = ols.coef_[0]
  df1['X'] = c1[t:]['price']
  df1['Y'] = c2[t:]['price']
  df1['Date'] = c1[t:]['date']
  beta = ols.coef_[0]
  transaction = 1 / (1 + beta) * cost2 + beta / (1 + beta) * cost1

  zEntries = np.arange(start=0.5, stop=4, step=0.1)
  exitZscore = 0
  cum_rets = pd.DataFrame()

  for z in zEntries:

      df1['long entry'] = ((sig < -1.0 * z) & (sig.shift(1) > -1.0 * z))
      df1['long exit'] = ((sig > - exitZscore) & (sig.shift(1) < - exitZscore))
      df1['num units long'] = np.nan
      df1.loc[df1['long entry'], 'num units long'] = 1
      df1.loc[df1['long exit'], 'num units long'] = 0
      df1['num units long'][0] = 0
      df1['num units long'] = df1['num units long'].fillna(method='pad')

      # set up num units short
      df1['short entry'] = ((sig > z) & (sig.shift(1) < z))
      df1['short exit'] = ((sig < exitZscore) & (sig.shift(1) > exitZscore))
      df1.loc[df1['short entry'], 'num units short'] = -1
      df1.loc[df1['short exit'], 'num units short'] = 0
      df1['num units short'][0] = 0
      df1['num units short'] = df1['num units short'].fillna(method='pad')

      df1['numUnits'] = df1['num units long'] + df1['num units short']

      df1['spread pct ch'] = (df1['spread'] - df1['spread'].shift(1)) / (
          df1['X'] * np.abs(df1['beta']) + df1['Y'])

      df1['port rets'] = df1['spread pct ch'] * df1['numUnits'].shift(1) * (1 - transaction)

      df1['cum rets'] = df1['port rets'].cumsum()
      df1['cum rets'] = df1['cum rets'] + 1

      if cum_rets.empty:
          cum_rets = pd.DataFrame(df1['cum rets'])
          cum_rets.rename(columns={'cum rets': 'cumRets_{}'.format(round(z, 2))}, inplace=True)
      else:
          cum_rets = cum_rets.join(pd.DataFrame(df1['cum rets']), how="outer")
          cum_rets.rename(columns={'cum rets': 'cumRets_{}'.format(round(z, 2))}, inplace=True)
```

```
    print("\nBACKTEST Y against X:\n")
    try:
        for i in cum_rets.columns:
            print(
                "Z-Score Entry {}: ".format(i[8:11]),
                str((cum_rets[i][len(cum_rets) - 1] - cum_rets[i][1]) * 100)[0:4]
            )
        maxCAGR = 0.0
        bestZ = ""
        for i in cum_rets.columns:
            if float((cum_rets[i][len(cum_rets) - 1] - cum_rets[i][1]) * 100) > float(maxCAGR):
                maxCAGR = (cum_rets[i][len(cum_rets) - 1] - cum_rets[i][1]) * 100
                bestZ = i
                print("\nBest 1-yr return {} with {} ZScore Entry".format(
                    str((cum_rets[bestZ][len(cum_rets) - 1] - cum_rets[bestZ][1]) * 100)[0:4] + "%",
                    bestZ[8:11])
                )
                print("1-yr Return {} with 1.0 ZScore Entry".format(
                    str((cum_rets["cumRets_1.0"][len(cum_rets) - 1] - cum_rets["cumRets_1.0"][1]) * 100)[0:4] + "%"
                ))
    except KeyError:
        print("KeyError")

    cum_rets.index = pd.to_datetime(df1['Date'])
    cumRets_1 = cum_rets['cumRets_1.0']
    cum_rets.drop(labels=[i for i in cum_rets.columns if i != bestZ], inplace=True, axis=1)

    if bestZ != "cumRets_1.0":
        cum_rets = cum_rets.join(pd.DataFrame(cumRets_1))
        cum_rets = cum_rets.join(df1['Date'])
    return cum_rets


rets1 = transaction_cost(tether, bitcash)
rets2 = transaction_cost(ripple, bit)
rets3 = transaction_cost(ripple, litecoin)
rets4 = transaction_cost(ethreum, litecoin)
rets5 = transaction_cost(ripple, ethreum)
rets6 = transaction_cost(ripple, stellar)
rets7 = transaction_cost(bitcash, bit)

cum_rets = pd.DataFrame()
cum_rets['Date'] = bit['date']['2018-08-02':]
cum_rets.index = cum_rets['Date']
rets = []
for i in range(1, len(rets1)):  # skip the first entry which is nan
    mean = (rets1.iloc[i, 0] + rets2.iloc[i, 0] + rets3.iloc[i, 0] + rets4.iloc[i, 0] + rets5.iloc[i, 0] + rets6.iloc[
        i, 0] + rets7.iloc[i, 0]) / 7
    rets.append(mean)
cum_rets['return'] = rets
```

```
cum_rets.plot()
plt.title("refinement 1: including transaction cost")

# stats:
port_ret = []
port_ret.append(0)
for i in range(len(rets) - 1):
    port_ret.append(rets[i + 1] - rets[i])
sharpe = np.mean(port_ret) / np.std(port_ret) * (252 ** 0.5)
print('sharpe is: ', sharpe)

D = len(rets)
rets = pd.Series(rets)
ann_ret = rets.prod() ** (12 / D) - 1
print('annualized return is: ', ann_ret)

vol = np.std(cum_rets['return'])


def max_drawdown(X):
    mdd = 0
    peak = X[0]
    for x in X:
        if x > peak:
            peak = x
        dd = (peak - x) / peak
        if dd > mdd:
            mdd = dd
    return mdd


maxd = max_drawdown(cum_rets['return'])
print('volatility is: ', vol)
print('maximum drawdown is: ', maxd)


#########################################################################################################
# refinement 2: include stop loss:

def stop_loss(c1, c2, z, stop_loss):  # c1 is X, c2 is Y

    signal = pd.DataFrame(index=c1.index)  # should have 6 columns: spread, mean, std, prev_spread,
prev_mean, prev_std
    X = pd.DataFrame()
    X['X'] = c1['price']
    X.index = pd.to_datetime(c1['date'])
    ols = LinearRegression().fit(X[:t], c2['price'][:t])
    train_spread = c2['price'][:t] - ols.predict(X[:t])
    mean = np.mean(np.array(train_spread))
```

```
    std = np.std(np.array(train_spread))
    test_spread = c2['price'][t:] - ols.predict(X[t:])
    sig = (test_spread - mean) / std
    signal['spread'] = sig
    signal['mean'] = np.mean(sig)
    signal['std'] = np.std(sig)
    signal['prev_spread'] = sig.shift(1)
    signal['prev_mean'] = np.mean(sig)
    signal['prev_std'] = np.std(sig)

    flag = 0  # indicating if we hold positions, 0: no position, 1: short spread, -1: long spread
    positions = pd.DataFrame(index=signal.index, columns=[c1.name, c2.name])
    positions.iloc[0] = [0, 0]  # first row, first day, no position for all of the three futures
    cum_position = {c1.name: 0, c2.name: 0}
    pnl = pd.DataFrame(index=signal.index, columns=['portfolio'])
    pnl.iloc[0]['portfolio'] = 10000  # initial asset
    for i, (spread, mean, std, prev_spread, prev_mean, prev_std) in enumerate(signal.values):
        if i == 0:
            continue

        # short spread
        if flag == 0 and spread > mean + z * std and prev_spread < prev_mean + z * prev_std:
            # soybean - soymeal- soyoil
            # soybean is Y, soymeal and soyoil is X
            # c1 is X, c2 is Y
            positions.iloc[i][c2.name] = -1 / 2 * pnl.iloc[i - 1]['portfolio'] / c2.iloc[i]['price']
            positions.iloc[i][c1.name] = 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c1.iloc[i]['price']

            cum_position[c2.name] -= 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c2.iloc[i]['price']
            cum_position[c1.name] += 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c1.iloc[i]['price']

            flag = 1

            pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio']

        # long spread
        elif flag == 0 and spread < mean - z * std and prev_spread > prev_mean - z * prev_std:
            positions.iloc[i][c2.name] = 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c2.iloc[i]['price']
            positions.iloc[i][c1.name] = -1 / 2 * pnl.iloc[i - 1]['portfolio'] / c1.iloc[i]['price']

            cum_position[c2.name] += 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c2.iloc[i]['price']
            cum_position[c1.name] -= 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c1.iloc[i]['price']

            flag = -1

            pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio']

        # clear position or stop loss
        elif flag == 1 and ((spread < mean and prev_spread > prev_mean) or (
                spread > mean + stop_loss * std and prev_spread < prev_mean + stop_loss * prev_std)):
```

```
            positions.iloc[i][c2.name] = -cum_position[c2.name]
            positions.iloc[i][c1.name] = -cum_position[c1.name]

            cum_position[c1.name] = 0
            cum_position[c2.name] = 0

            pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio'] - positions.iloc[i][c2.name] * (
            c2.iloc[i]['price'] - c2.iloc[i - 1]['price'])
            + positions.iloc[i][c1.name] * (c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])

            flag = 0

        # clear position or stop loss
        elif flag == -1 and ((spread > mean and prev_spread > prev_mean) or (
                spread < mean - stop_loss * std and prev_spread > prev_mean - stop_loss * prev_std)):
            positions.iloc[i][c2.name] = -cum_position[c2.name]
            positions.iloc[i][c1.name] = -cum_position[c1.name]

            cum_position[c1.name] = 0
            cum_position[c2.name] = 0

            pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio'] + positions.iloc[i][c2.name] * (
            c2.iloc[i]['price'] - c2.iloc[i - 1]['price'])
            - positions.iloc[i][c1.name] * (c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])

            flag = 0

        else:
            positions.iloc[i] = positions.iloc[i - 1]
            if positions.iloc[i][c2.name] > 0:
                pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio'] + positions.iloc[i][c2.name] * (
                c2.iloc[i]['price'] - c2.iloc[i - 1]['price'])
                - positions.iloc[i][c1.name] * (c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])

            elif positions.iloc[i][c2.name] < 0:
                pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio'] - positions.iloc[i][c1.name] * (
                c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])
                + positions.iloc[i][c1.name] * (c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])

            else:
                pnl.iloc[i] = pnl.iloc[i - 1]

    ret = pnl / pnl.iloc[0]
    return ret[t:]


rets1 = stop_loss(tether, bitcash, z=1, stop_loss=1)
rets2 = stop_loss(ripple, bit, z=1, stop_loss=1)
rets3 = stop_loss(ripple, litecoin, z=1.6, stop_loss=1)
rets4 = stop_loss(ethreum, litecoin, z=1, stop_loss=1)
```

```
rets5 = stop_loss(ripple, ethreum, z=2.4, stop_loss=1)
rets6 = stop_loss(ripple, stellar, z=1, stop_loss=1)
rets7 = stop_loss(bitcash, bit, z=0.5, stop_loss=1)

cum_rets = pd.DataFrame()
cum_rets['Date'] = bit['date']['2018-08-02':]
cum_rets.index = cum_rets['Date']
rets = []
for i in range(1, len(rets1)):  # skip the first entry which is nan
    mean = (rets1.iloc[i, 0] + rets2.iloc[i, 0] + rets3.iloc[i, 0] + rets4.iloc[i, 0] + rets5.iloc[i, 0] + rets6.iloc[
        i, 0] + rets7.iloc[i, 0]) / 7
    rets.append(mean)
cum_rets['return'] = rets

cum_rets.plot()
plt.title("out of sample test: including refinements 1,2,4")

plt.title("refinement 2: including stop loss=1")

# stats:
port_ret = []
port_ret.append(0)
for i in range(len(rets) - 1):
    port_ret.append(rets[i + 1] - rets[i])
sharpe = np.mean(port_ret) / np.std(port_ret) * (252 ** 0.5)
print('sharpe is: ', sharpe)

D = len(rets)
rets = pd.Series(rets)
ann_ret = rets.prod() ** (12 / D) - 1
print('annualized return is: ', ann_ret)

vol = np.std(cum_rets['return'])


def max_drawdown(X):
    mdd = 0
    peak = X[0]
    for x in X:
        if x > peak:
            peak = x
        dd = (peak - x) / peak
        if dd > mdd:
            mdd = dd
    return mdd


maxd = max_drawdown(cum_rets['return'])
print('volatility is: ', vol)
print('maximum drawdown is: ', maxd)
```

```
##############################################################################################
# refinement 5: signal weighting: use correlation

t = '2018-08-01'
r2 = dict()
rets = pd.DataFrame()
df['eos'] = eos['price'][:t]
df['ripple'] = ripple['price'][:t]
df['ethreum'] = ethreum['price'][:t]
df['bitcoin'] = bit['price'][:t]
df['bitcash'] = bitcash['price'][:t]
df['litecoin'] = litecoin['price'][:t]
df['stellar'] = stellar['price'][:t]
df['tether'] = tether['price'][:t]
df['tron'] = tron['price'][:t]
pairs = [('tether', 'bitcash'), ('ripple', 'bitcoin'), ('ripple', 'litecoin'), ('ethreum', 'litecoin'),
         ('ripple', 'ethreum'), ('ripple', 'stellar'), ('bitcash', 'bitcoin')]
for pair in pairs:
    ols = LinearRegression().fit(df[[pair[0]]], df[[pair[1]]])
    r2[' '.join(pair)] = ols.score(df[[pair[0]]], df[[pair[1]]])
rets = pd.DataFrame()
rets['tether bitcash'] = basic_pair_trade(tether, bitcash)['cumRets_1.0']
rets['ripple bitcoin'] = basic_pair_trade(ripple, bit)['cumRets_1.0']
rets['ripple litecoin'] = basic_pair_trade(ripple, litecoin)['cumRets_1.0']
rets['ethreum litecoin'] = basic_pair_trade(ethreum, litecoin)['cumRets_1.0']
rets['ripple ethreum'] = basic_pair_trade(ripple, ethreum)['cumRets_1.0']
rets['ripple stellar'] = basic_pair_trade(ripple, stellar)['cumRets_1.0']
rets['bitcash bitcoin'] = basic_pair_trade(bitcash, bit)['cumRets_1.0']
rets['t'] = rets[
    ['tether bitcash', 'ripple bitcoin', 'ripple litecoin', 'ethreum litecoin', 'ripple ethreum', 'ripple stellar',
     'bitcash bitcoin']].mean(axis=1)
rets['total'] = rets.multiply(r2).sum(axis=1) / sum(r2.values())
rets.dropna(inplace=True)
rets['total'].plot()
plt.title('Signal Weighting with 7 pairs')
plt.show()

rets['t'].plot()
plt.title('Signal Weighting with 7 pairs')
plt.show()


##########################################################

# comprehensive out-of-sample test
def OOS(c1, c2, z, stop_loss):  # c1 is X, c2 is Y
    signal = pd.DataFrame(index=c1.index, columns={"spread", "mean", "std"}).iloc[
             t:]  # should have 6 columns: spread, mean, std, prev_spread, prev_mean, prev_std
    X = pd.DataFrame()
```

```
  X['X'] = c1['price']
  X.index = pd.to_datetime(c1['date'])
  for i in range(t, X.shape[0]):
      X_train = X.iloc[i - t:i]
      y = c2['price']
      y_train = y.iloc[i - t:i]
      ols = LinearRegression().fit(X_train, y_train)
      train_spread = y_train - ols.predict(X_train)
      mean = np.mean(np.array(train_spread))
      std = np.std(np.array(train_spread))

      test_spread = y.iloc[i] - ols.predict(np.array(X.iloc[i]).reshape(1, -1))
      sig = (test_spread - mean) / std
      idx = i - t
      signal.iloc[idx]['spread'] = sig[0]
      signal.iloc[idx]['mean'] = mean
      signal.iloc[idx]['std'] = std
  signal['prev_spread'] = signal.shift(1)["spread"]
  signal['prev_mean'] = signal.shift(1)["mean"]
  signal['prev_std'] = signal.shift(1)["std"]

  flag = 0  # indicating if we hold positions, 0: no position, 1: short spread, -1: long spread
  positions = pd.DataFrame(index=signal.index, columns=[c1.name, c2.name])
  positions.iloc[0] = [0, 0]  # first row, first day, no position for all of the three futures
  cum_position = {c1.name: 0, c2.name: 0}
  pnl = pd.DataFrame(index=signal.index, columns=['portfolio'])
  pnl.iloc[0]['portfolio'] = 10000  # initial asset
  for i, (spread, mean, std, prev_spread, prev_mean, prev_std) in enumerate(signal.values):
      if i == 0:
          continue

      # short spread
      if flag == 0 and spread > mean + z * std and prev_spread < prev_mean + z * prev_std:
          # soybean - soymeal- soyoil
          # soybean is Y, soymeal and soyoil is X
          # c1 is X, c2 is Y
          positions.iloc[i][c2.name] = -1 / 2 * pnl.iloc[i - 1]['portfolio'] / c2.iloc[i]['price']
          positions.iloc[i][c1.name] = 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c1.iloc[i]['price']

          cum_position[c2.name] -= 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c2.iloc[i]['price']
          cum_position[c1.name] += 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c1.iloc[i]['price']

          flag = 1

          pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio']

      # long spread
      elif flag == 0 and spread < mean - z * std and prev_spread > prev_mean - z * prev_std:
          positions.iloc[i][c2.name] = 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c2.iloc[i]['price']
          positions.iloc[i][c1.name] = -1 / 2 * pnl.iloc[i - 1]['portfolio'] / c1.iloc[i]['price']
```

```
            cum_position[c2.name] += 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c2.iloc[i]['price']
            cum_position[c1.name] -= 1 / 2 * pnl.iloc[i - 1]['portfolio'] / c1.iloc[i]['price']

            flag = -1

            pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio']

        # clear position or stop loss
        elif flag == 1 and ((spread < mean and prev_spread > prev_mean) or (
                spread > mean + stop_loss * std and prev_spread < prev_mean + stop_loss * prev_std)):
            positions.iloc[i][c2.name] = -cum_position[c2.name]
            positions.iloc[i][c1.name] = -cum_position[c1.name]

            cum_position[c1.name] = 0
            cum_position[c2.name] = 0

            pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio'] - positions.iloc[i][c2.name] * (
            c2.iloc[i]['price'] - c2.iloc[i - 1]['price'])
            + positions.iloc[i][c1.name] * (c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])

            flag = 0

        # clear position or stop loss
        elif flag == -1 and ((spread > mean and prev_spread > prev_mean) or (
                spread < mean - stop_loss * std and prev_spread > prev_mean - stop_loss * prev_std)):
            positions.iloc[i][c2.name] = -cum_position[c2.name]
            positions.iloc[i][c1.name] = -cum_position[c1.name]

            cum_position[c1.name] = 0
            cum_position[c2.name] = 0

            pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio'] + positions.iloc[i][c2.name] * (
            c2.iloc[i]['price'] - c2.iloc[i - 1]['price'])
            - positions.iloc[i][c1.name] * (c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])

            flag = 0

        else:
            positions.iloc[i] = positions.iloc[i - 1]
            if positions.iloc[i][c2.name] > 0:
                pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio'] + positions.iloc[i][c2.name] * (
                c2.iloc[i]['price'] - c2.iloc[i - 1]['price'])
                - positions.iloc[i][c1.name] * (c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])

            elif positions.iloc[i][c2.name] < 0:
                pnl.iloc[i]['portfolio'] = pnl.iloc[i - 1]['portfolio'] - positions.iloc[i][c1.name] * (
                c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])
                + positions.iloc[i][c1.name] * (c1.iloc[i]['price'] - c1.iloc[i - 1]['price'])
```

```
        else:
            pnl.iloc[i] = pnl.iloc[i - 1]

    ret = pnl / pnl.iloc[0]

    return ret[t:]


rets1 = OOS(tether, bitcash, z=1, stop_loss=1)
rets2 = OOS(ripple, bit, z=1, stop_loss=1)
rets3 = stop_loss(ripple, litecoin, z=1.6, stop_loss=1)
rets4 = stop_loss(ethreum, litecoin, z=1, stop_loss=1)
rets5 = stop_loss(ripple, ethreum, z=2.4, stop_loss=1)
rets6 = stop_loss(ripple, stellar, z=1, stop_loss=1)
rets7 = stop_loss(bitcash, bit, z=0.5, stop_loss=1)

cum_rets = pd.DataFrame()
cum_rets['Date'] = bit['date']['2018-08-02':]
cum_rets.index = cum_rets['Date']
rets = []
for i in range(1, len(rets1)):  # skip the first entry which is nan
    mean = (rets1.iloc[i, 0] + rets2.iloc[i, 0] + rets3.iloc[i, 0] + rets4.iloc[i, 0] + rets5.iloc[i, 0] + rets6.iloc[
        i, 0] + rets7.iloc[i, 0]) / 7
    rets.append(mean)
cum_rets['return'] = rets

cum_rets.plot()
plt.title("out of sample test: including refinements 1,2,4")

plt.title("refinement 2: including stop loss=1")

# stats:
port_ret = []
port_ret.append(0)
for i in range(len(rets) - 1):
    port_ret.append(rets[i + 1] - rets[i])
sharpe = np.mean(port_ret) / np.std(port_ret) * (252 ** 0.5)
print('sharpe is: ', sharpe)

D = len(rets)
rets = pd.Series(rets)
ann_ret = rets.prod() ** (12 / D) - 1
print('annualized return is: ', ann_ret)

vol = np.std(cum_rets['return'])


def max_drawdown(X):
    mdd = 0
    peak = X[0]
```

```
    for x in X:
        if x > peak:
            peak = x
        dd = (peak - x) / peak
        if dd > mdd:
            mdd = dd
    return mdd


maxd = max_drawdown(cum_rets['return'])
print('volatility is: ', vol)
print('maximum drawdown is: ', maxd)
```