This is how you do Pairs Trading: Statistical Arbitrage

How you can do algorithimic trading?

ONEPAGECODE
27 OCT 2023 • PAID


```
# Import Dependencies
from cointegration_analysis import
estimate_long_run_short_run_relationships,
engle_granger_two_step_cointegration_test
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
%matplotlib inline
```

An analysis of cointegration can be performed in Python using the libraries and modules imported in this code. This module imports the estimate_long_run_short_run_relationships and engle_granger_two_step_cointegration_test functions from the cointegration_analysis module, along with pandas, numpy, and matplotlib libraries for data manipulation. Inline plotting is made possible with the "%matplotlib inline" command in Jupyter notebooks.

Download the source code from the link in comment sections!

Data Analysis

This section visualizes our data and helps to understand, do the cointegration_analysis and display the results as well as make a selection of pairs that are cointegrated to further explore in the Algorithm section.

```
def read_data(filename):
    '''
    This function reads the .csv stored at the 'filename'
location and returns a DataFrame
    with two levels of column names. The first level column
contains the Stock Name and the
    second contains the type of market data, e.g. bid/ask,
price/volume.
    '''
    df = pd.read_csv(filename, index_col=0)
    df.columns = [df.columns.str[-2:], df.columns.str[:-3]]

    return df

market_data = read_data('Pairs Trading.csv')

# Show the First 5 Rows
display(market_data.head())

# Show the Stocks
stock_names =
list(market_data.columns.get_level_values(0).unique())
print('The stocks available are',stock_names)
```

	AA				BB				CC		...	MM		NN			
	BidPrice	BidVolume	AskPrice	AskVolume	BidPrice	BidVolume	AskPrice	AskVolume	BidPrice	BidVolume	...	AskPrice	AskVolume	BidPrice	BidVolume	AskPrice	AskVolume
01/01/2018 00:05	80.00	135	80.25	135	100.55	105	100.85	105	97.85	112	...	95.20	140	137.20	122	137.60	122
01/01/2018 00:10	79.95	124	80.20	145	100.50	104	100.80	113	97.80	109	...	95.20	144	137.20	111	137.55	110
01/01/2018 00:15	79.95	124	80.15	155	100.65	100	100.95	107	97.85	110	...	95.25	156	137.25	117	137.65	106
01/01/2018 00:20	79.95	130	80.10	154	100.75	112	101.05	109	97.95	100	...	95.25	153	137.25	107	137.65	97
01/01/2018 00:25	79.90	121	80.10	138	100.60	126	100.85	124	97.95	98	...	95.20	153	137.15	111	137.60	109

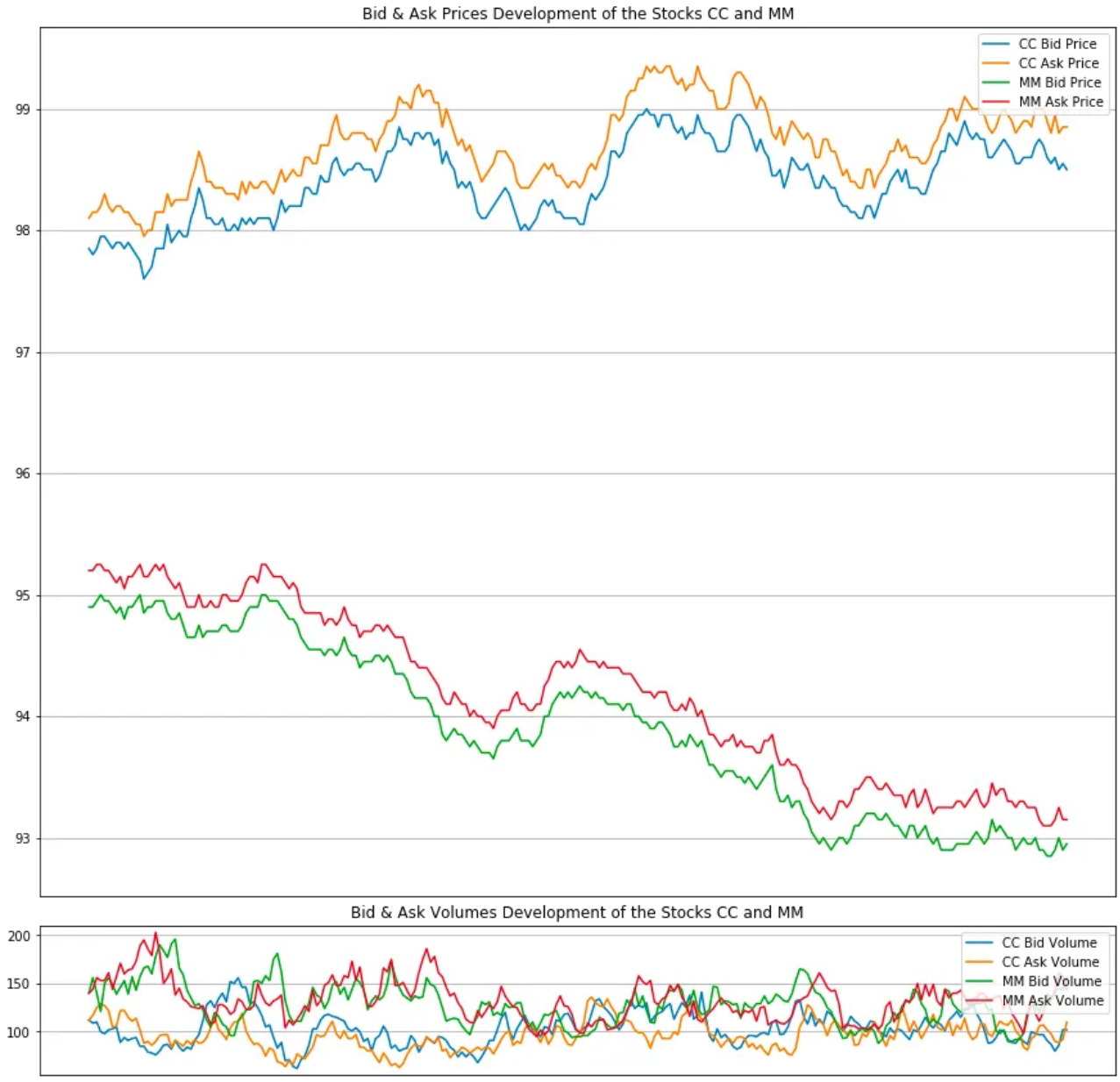
Python code uses two levels of column names to generate a DataFrame from a .csv file. Firstly, the stocks' names are displayed, and secondly, the market data, such as bid/ask and price/volume, is shown. read_data reads the .csv file and converts it into

a DataFrame. A DataFrame created by the read_data function is stored in the market_data variable. In this example, the dataframe is displayed with the first 5 rows using the display function. As part of the DataFrame, a variable called stock_names is created to store the names of the stocks. A final step in the printing process is getting the unique values of the column names in the first level of the DataFrame and displaying the available stocks.

Onepagecode is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

```
# Specify variable used for Plotting
market_data_segmented = market_data[:250]

# Defining Plots
def bid_ask_price_plot(stock1, stock2):
    '''
    This function creates a subplot with a specified gridsize
to be able to
    effectively match it with a different subplot while still
maintaining
    it's independency of being able to just show this plot.
    '''
    ax1 = plt.subplot2grid((6, 1), (0, 0), rowspan=5,
colspan=1)
    plt.title('Bid & Ask Prices Development of the Stocks ' +
stock1 + " and " + stock2)
    plt.grid()

    ax1.plot(market_data_segmented.index,
             market_data_segmented[stock1, 'BidPrice'])
    ax1.plot(market_data_segmented.index,
             market_data_segmented[stock1, 'AskPrice'])

    ax1.plot(market_data_segmented.index,
             market_data_segmented[stock2, 'BidPrice'])
    ax1.plot(market_data_segmented.index,
             market_data_segmented[stock2, 'AskPrice'])

    # We don't want to see all the timestamps
    ax1.axes.get_xaxis().set_visible(False)

    ax1.legend([stock1 + " Bid Price", stock1 + " Ask Price",
stock2 + " Bid Price", stock2 + " Ask Price"], loc='upper
right')
```

```
def bid_ask_volume_plot(stock1, stock2):
    '''
    This function is very similar to above's function with the
exception
    of creating a smaller subplot and using different data.
This function
    is meant for displaying volumes.
    '''
    ax2 = plt.subplot2grid((6, 1), (5, 0), rowspan=1,
colspan=1)
    plt.title('Bid & Ask Volumes Development of the Stocks ' +
stock1 + " and " + stock2)
    plt.grid()

    ax2.plot(market_data_segmented.index,
             market_data_segmented[stock1, 'BidVolume'])
    ax2.plot(market_data_segmented.index,
             market_data_segmented[stock1, 'AskVolume'])

    ax2.plot(market_data_segmented.index,
             market_data_segmented[stock2, 'BidVolume'])
    ax2.plot(market_data_segmented.index,
             market_data_segmented[stock2, 'AskVolume'])

    # We don't want to see all the timestamps
    ax2.axes.get_xaxis().set_visible(False)

    ax2.legend([stock1 + " Bid Volume", stock1 + " Ask Volume",
stock2 + " Bid Volume", stock2 + " Ask Volume"], loc='upper
right')

# Show Plot
plt.figure(figsize=(15, 15))
plt.show(bid_ask_price_plot("CC", "MM"),
bid_ask_volume_plot("CC", "MM"))
```



bid_ask_price_plot and bid_ask_volume_plot plot bid and ask prices and volumes for two stocks specified in the code. First, a plot is created with a title and four lines representing the bid and ask prices for the two stocks. Second, a smaller plot is created, with a title and four lines representing the bid and ask volumes for both stocks. The plots can be matched with each other using a specified grid size in both functions. Plots from both functions are shown together using the plt.show function.

```
# Calculate mid-prices of each stock and add them to the
DataFrame
for stock in stock_names:
    market_data[stock,'MidPrice'] =
(market_data[stock,'BidPrice'] + market_data[stock,'AskPrice'])
/ 2
```

```
    market_data = market_data.sort_index(axis=1)

market_data.head()
```

	AA					BB					...	NN					OO
	AskPrice	AskVolume	BidPrice	BidVolume	MidPrice	AskPrice	AskVolume	BidPrice	BidVolume	MidPrice	...	AskPrice	AskVolume	BidPrice	BidVolume	MidPrice	AskPrice
01/01/2018 00:05	80.25	135	80.00	135	80.125	100.85	105	100.55	105	100.700	...	137.60	122	137.20	122	137.400	89.05
01/01/2018 00:10	80.20	145	79.95	124	80.075	100.80	113	100.50	104	100.650	...	137.55	110	137.20	111	137.375	89.00
01/01/2018 00:15	80.15	155	79.95	124	80.050	100.95	107	100.65	100	100.800	...	137.65	106	137.25	117	137.450	89.00
01/01/2018 00:20	80.10	154	79.95	130	80.025	101.05	109	100.75	112	100.900	...	137.65	97	137.25	107	137.450	89.00
01/01/2018 00:25	80.10	138	79.90	121	80.000	100.85	124	100.60	126	100.725	...	137.60	109	137.15	111	137.375	89.00

Mid-price is calculated for each stock by adding the bid price and ask price, then dividing by two. In the market_data dataframe, these mid-prices are stored in a column named MidPrice.. With the .head() function, the market_data dataframe is sorted by column indexes and displayed. By displaying the mid-prices in a sorted dataframe for a list of stocks, this code calculates the mid-prices.

```
def mid_price_check(stock):
    '''
    Function that checks for different stocks if the MidPrice
    is correctly specified.
    '''
    plt.figure(figsize=(20, 5))
    plt.plot(market_data[stock,'AskPrice'][:100])
    plt.plot(market_data[stock,'MidPrice'][:100])
    plt.plot(market_data[stock,'BidPrice'][:100])

    plt.xticks([]) # Timestamp is not Important
    plt.title('Ask, Bid and Mid Price Development of Stock ' +
stock)
    plt.legend(["Ask Price", "Mid Price", "Bid Price"],
loc='lower left')
    plt.show()

mid_price_check('MM')
```

Using the code, a function called mid_price_check plots the mid price of various stocks. stock refers to the stock to be checked, and is taken in as a parameter. This function creates a 20x5 figure using plt.figure(). The plt.plot() function then plots the stock's ask price, mid price, and bid price using the stock data. A no-ticks limit is set with plt.xticks([]). PLT.title() is used to create a title for the figure, and Plt.legend() is used to label the graph lines. In conclusion, the graph is presented using the plt.show() function. With this function, you can see the mid price for a particular stock, compare it to the ask price and bid price, and determine which is the better price.

```
# Obtain the statistical parameters for each and every pair
data_analysis = {'Pairs': [],
                 'Constant': [],
                 'Gamma': [],
                 'Alpha': [],
                 'P-Value': []}

data_zvalues = {}

for stock1 in stock_names:
    for stock2 in stock_names:
        if stock1 != stock2:
            if (stock2, stock1) in data_analysis['Pairs']:
                continue

            pairs = stock1, stock2
            constant =
estimate_long_run_short_run_relationships(np.log(
                market_data[stock1, 'MidPrice']),
np.log(market_data[stock2, 'MidPrice']))[0]
            gamma =
estimate_long_run_short_run_relationships(np.log(
                market_data[stock1, 'MidPrice']),
np.log(market_data[stock2, 'MidPrice']))[1]
```

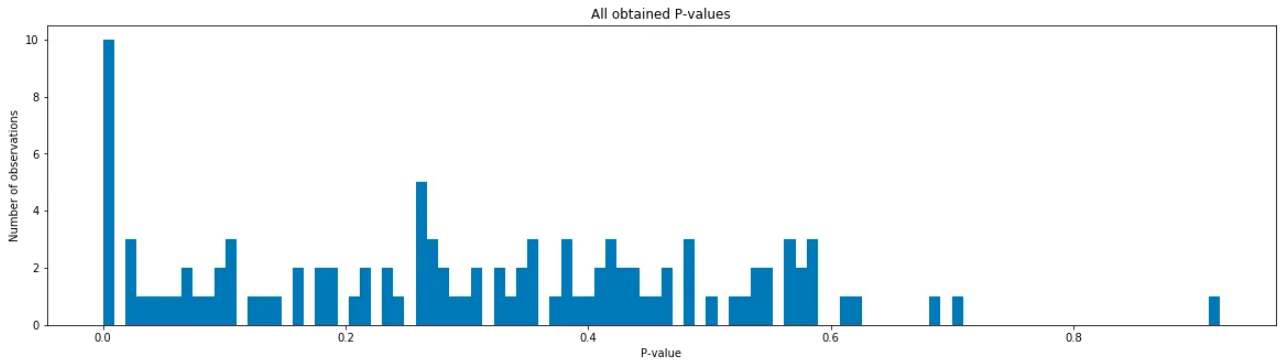
```
            alpha =
estimate_long_run_short_run_relationships(np.log(
                market_data[stock1, 'MidPrice']),
np.log(market_data[stock2, 'MidPrice']))[2]
            pvalue =
engle_granger_two_step_cointegration_test(np.log(
                market_data[stock1, 'MidPrice']),
np.log(market_data[stock2, 'MidPrice']))[1]
            zvalue =
estimate_long_run_short_run_relationships(np.log(
                market_data[stock1, 'MidPrice']),
np.log(market_data[stock2, 'MidPrice']))[3]

            data_analysis['Pairs'].append(pairs)
            data_analysis['Constant'].append(constant)
            data_analysis['Gamma'].append(gamma)
            data_analysis['Alpha'].append(alpha)
            data_analysis['P-Value'].append(pvalue)

            data_zvalues[pairs] = zvalue

data_analysis =
round(pd.DataFrame(data_analysis),4).set_index('Pairs')
```

In this code, we analyze market data for multiple stock names. Data will be stored in two variables defined in the first line. In the next line, we define a dictionary for storing z-values, which will be calculated later. A for loop is used to iterate through each stock name in the stock_names list. A code that does this twice indicates that it is looking for relationships between different pairs of stocks. It checks if the two stocks being compared are the same first. As long as they are, the code moves on to the next iteration. When the pair is not the same, the code checks if it has already been added to the ‘Pairs’ list. A code iteration is also carried out if it is positive. For the two stock names being compared, the following lines use the estimate_long_run_short_run_relationships function to calculate the constant, gamma, and alpha values. Long-term and short-term relationship between two stocks is represented in these values. In the following line, the function engle_granger_two_step_cointegration_test is used to test for cointegration between two stock prices. This test is then used to calculate the p-value. Calculating the z-value requires the same function used to determine the constant, gamma, and alpha values previously. After all pairs and constants have been added, you will need to append the gamma, alpha, and p-value to the data_analysis

dictionary. The z-value is also added to the data_zvalues dictionary with the pairs as keys. Before being set as the index, the data_analysis dictionary is converted into a dataframe and rounded to four decimal places.

Onepagecode is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

```
# Visualize the P-values
def plot_pvalues():
    """
    This function plots all obtained P-values.
    """
    plt.figure(figsize=(20, 5))
    plt.hist(data_analysis['P-Value'], bins=100)
    plt.xlabel('P-value')
    plt.ylabel('Number of observations')
    plt.title('All obtained P-values')
    plt.show()

plot_pvalues()
```



An analysis function like this one operates on a set of data called data_analysis and plots the distribution of P-values. A histogram's x-axis represents P-values, and its y-axis represents observations.

```
# Show Top 10 and Bottom 10
display(data_analysis.sort_values('P-Value')[:10])
display(data_analysis.sort_values('P-Value')[-10:])
```

Pairs	Constant	Gamma	Alpha	P-Value
(FF, NN)	-2.6104	1.4881	-0.0022	0.0000
(FF, MM)	0.3061	0.9676	-0.0338	0.0000
(BB, DD)	-2.4182	1.4758	-0.0075	0.0000
(DD, HH)	-1.6797	1.2787	-0.0040	0.0000
(BB, JJ)	-2.4809	1.6206	-0.0000	0.0000
(DD, JJ)	0.2277	1.0356	0.0002	0.0000
(MM, NN)	-2.9034	1.5153	-0.0002	0.0000
(BB, HH)	-4.8181	1.8712	-0.0024	0.0009
(HH, JJ)	1.9492	0.7041	-0.0002	0.0010
(AA, II)	0.0445	0.9944	-0.0002	0.0031

Pairs	Constant	Gamma	Alpha	P-Value
(AA, BB)	4.2082	0.0571	-0.0001	0.5739
(EE, JJ)	5.0700	-0.0369	-0.0002	0.5765
(AA, MM)	10.2964	-1.2938	-0.0001	0.5812
(AA, KK)	9.7238	-1.1233	-0.0001	0.5826
(AA, JJ)	4.1324	0.0773	-0.0001	0.5860
(AA, DD)	4.2125	0.0540	-0.0001	0.6103
(CC, EE)	1.7955	0.5551	-0.0002	0.6199
(CC, OO)	8.4182	-0.8627	-0.0002	0.6888
(CC, GG)	2.8739	0.3184	-0.0002	0.7055

(AA, CC)	6.4143	-0.4308	-0.0001	0.9207

Iterates over a dataset and sort it according to the column ‘P-Value’. It returns the first 10 rows of the dataset in ascending order, while it returns the last 10 rows in descending order. Based on their p-values, we can view the 10 most and least significant data points. The probability of a null hypothesis being true can be assessed using P-values. Low p-values indicate higher significance, while high p-values indicate lower significance. Sorting the dataset by p-values allows us to quickly identify the most and least important data points. Data analysis using this code can help us understand what variables have the greatest impact on our data.

```
# Selecting tradable pairs where P-Value < 0.01 and create a
seperate DataFrame containing these pairs
tradable_pairs_analysis = data_analysis[data_analysis['P-
Value'] < 0.01].sort_values('P-Value')

tradable_pairs_analysis
```

Pairs	Constant	Gamma	Alpha	P-Value
(BB, DD)	-2.4182	1.4758	-0.0075	0.0000
(BB, JJ)	-2.4809	1.6206	-0.0000	0.0000
(DD, HH)	-1.6797	1.2787	-0.0040	0.0000
(DD, JJ)	0.2277	1.0356	0.0002	0.0000
(FF, MM)	0.3061	0.9676	-0.0338	0.0000
(FF, NN)	-2.6104	1.4881	-0.0022	0.0000
(MM, NN)	-2.9034	1.5153	-0.0002	0.0000
(BB, HH)	-4.8181	1.8712	-0.0024	0.0009
(HH, JJ)	1.9492	0.7041	-0.0002	0.0010
(AA, II)	0.0445	0.9944	-0.0002	0.0031

The code below analyzes data_analysis as a dataset. In the first step of the process, any row with a P-Value value greater than 0.05 is removed. The remaining rows are then sorted ascendingly based on the values in the P-Value column. This dataset is stored in a variable called tradable_pairs_analysis, which can be analyzed further. A trading opportunity is identified and prioritized based on the most significant or relevant data points in the dataset with this code.

```
# Get all the tradable stock pairs into a list
stock_pairs =
list(tradable_pairs_analysis.index.values.tolist())

# Show the Pairs
stock_pairs
```

```
[('BB', 'DD'),
 ('BB', 'JJ'),
 ('DD', 'HH'),
 ('DD', 'JJ'),
 ('FF', 'MM'),
 ('FF', 'NN'),
 ('MM', 'NN'),
 ('BB', 'HH'),
 ('HH', 'JJ'),
 ('AA', 'II')]
```

It creates an array called stock_pairs which includes every possible combination of stock pairs, which is a list is called stock_pairs. Following that, it creates another list called stock_pairs_executed, which stores lists of pairs formed from iterating over stock_pairs. Each pair contains one stock from the top position in the stock_pairs list and another stock from a lower position, providing all possible combinations. It can be used to analyze different stock pairs and execute trades.

Onepagecode is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

Algorithm

Within this section we build further on the Data Analysis section by zooming in onto the pairs and calculate additional data on the error correction terms.

```
# Create a list of unique tradable stocks
list_stock1 = [stock[0] for stock in stock_pairs]
list_stock2 = [stock[1] for stock in stock_pairs]

for stock in list_stock2:
    list_stock1.append(stock)

unique_stock_list = list(set(list_stock1))
```

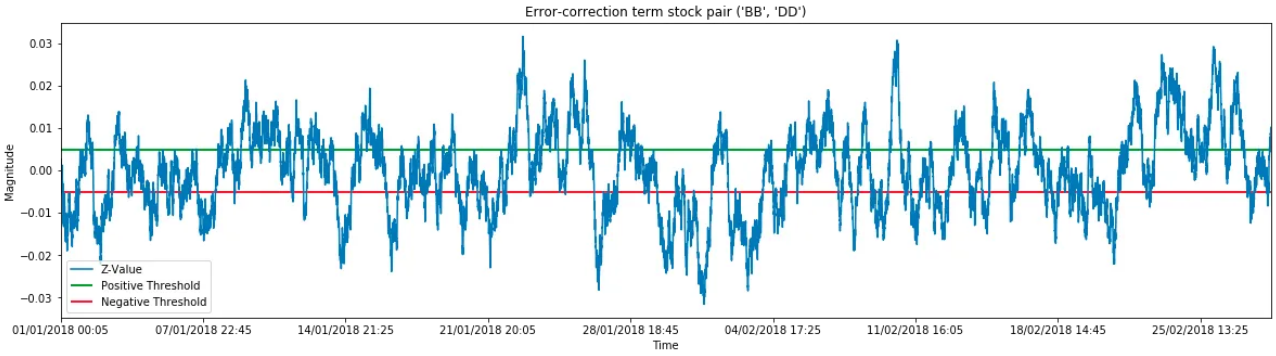
```
# Create a new DataFrame containing all market information for
the tradable pairs
tradable_pairs_data = market_data[unique_stock_list]
tradable_pairs_data.head()
```

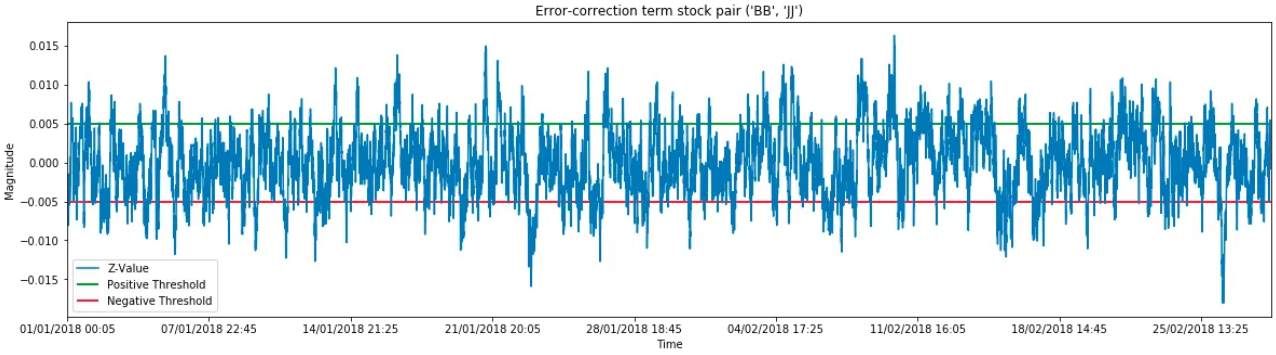
	JJ					AA					...	DD					BB
	AskPrice	AskVolume	BidPrice	BidVolume	MidPrice	AskPrice	AskVolume	BidPrice	BidVolume	MidPrice	...	AskPrice	AskVolume	BidPrice	BidVolume	MidPrice	AskPrice
01/01/2018 00:05	80.05	156	79.80	156	79.925	80.25	135	80.00	135	80.125	...	117.30	102	116.95	102	117.125	100.85
01/01/2018 00:10	80.00	161	79.75	157	79.875	80.20	145	79.95	124	80.075	...	117.25	103	116.95	98	117.100	100.80
01/01/2018 00:15	79.90	145	79.65	152	79.775	80.15	155	79.95	124	80.050	...	117.20	113	116.90	109	117.050	100.95
01/01/2018 00:20	79.95	140	79.65	154	79.800	80.10	154	79.95	130	80.025	...	117.30	117	116.95	100	117.125	101.05
01/01/2018 00:25	79.95	137	79.65	158	79.800	80.10	138	79.90	121	80.000	...	117.30	129	116.95	106	117.125	100.85

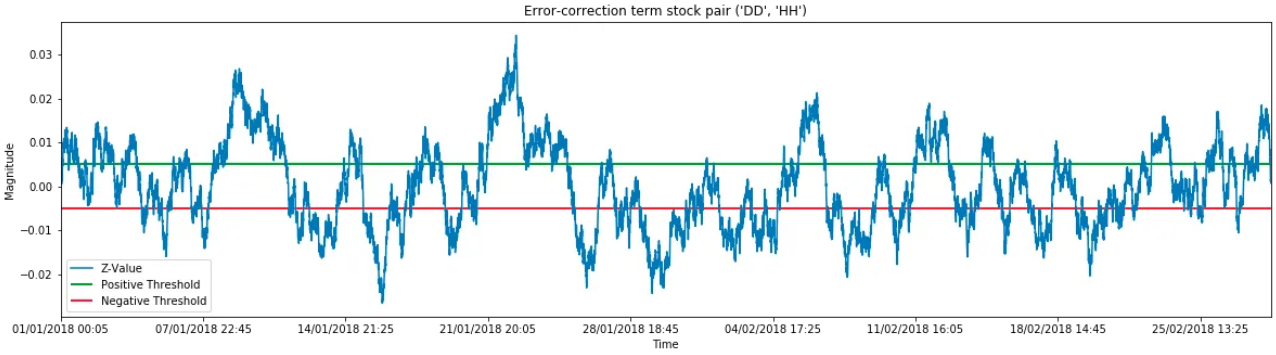
Using this code, we create a list called list_stock1 that contains the first element from every pair in stock_pairs. Additionally, list_stock2 is created, which contains the second element of every pair in stock_pairs. After that, it iterates through list_stock2 and adds every element. Then it creates a new list called unique_stock_list that only contains unique elements from list_stock1. In addition, this list will be used to filter the market_data to create a new dataframe called tradable_pairs_data that includes only the stocks in the unique_stock_list.

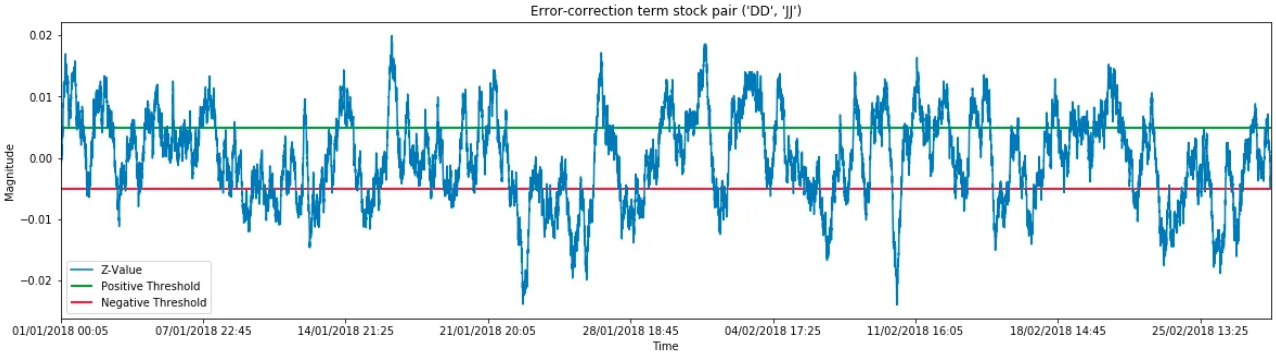

```
def Plot_Tradable_Z():
    """
    This function plots the z-values of all pairs based on
    the data_zvalues dataframe.
    """
    for pair in stock_pairs:
        zvalue = data_zvalues[pair]
        plt.figure(figsize=(20,5))
        plt.title('Error-correction term stock pair
```

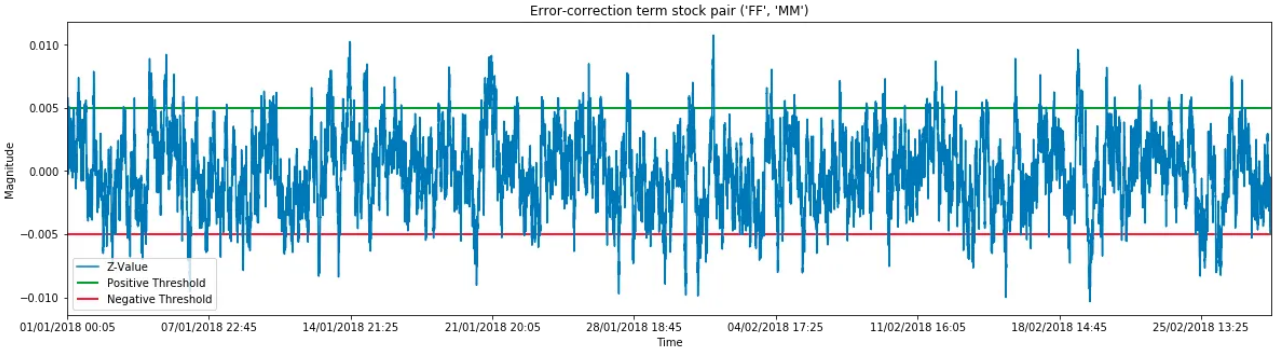
```
{}'.format(pair))
        zvalue.plot()
        plt.xlabel('Time')
        plt.ylabel('Magnitude')

        xmin = 0
        xmax = len(zvalue)
        plt.hlines(0.005, xmin, xmax, 'g') # Note 0.005 is
randomly chosen
        plt.hlines(-0.005, xmin, xmax, 'r') # Note -0.005 is
randomly chosen

        plt.legend(['Z-Value', 'Positive Threshold', 'Negative
Threshold'], loc='lower left')

        plt.show()

Plot_Tradable_Z()
```

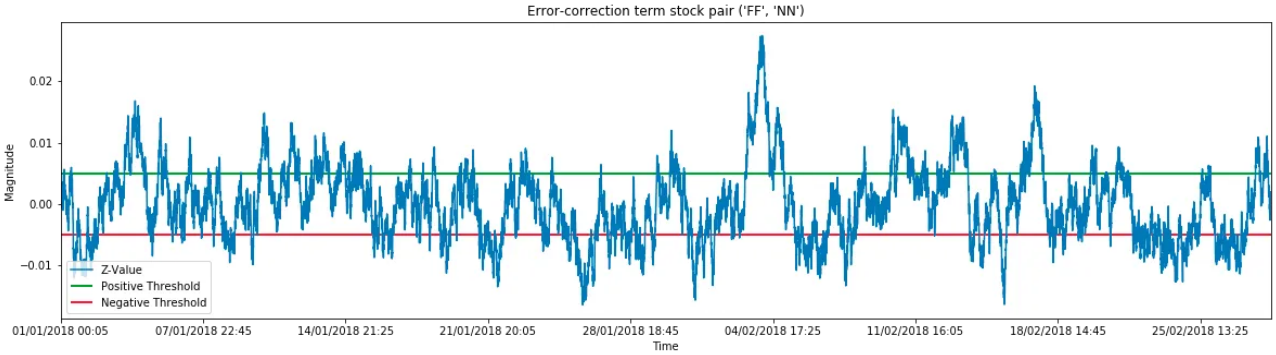


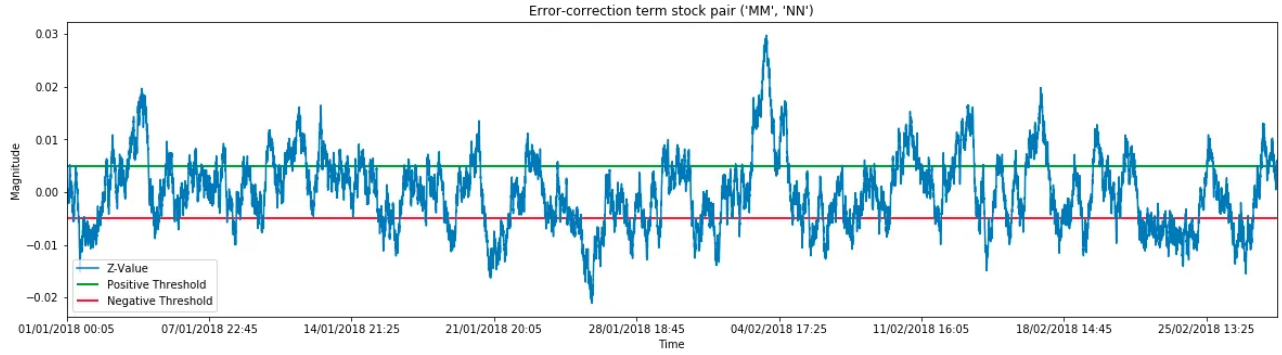


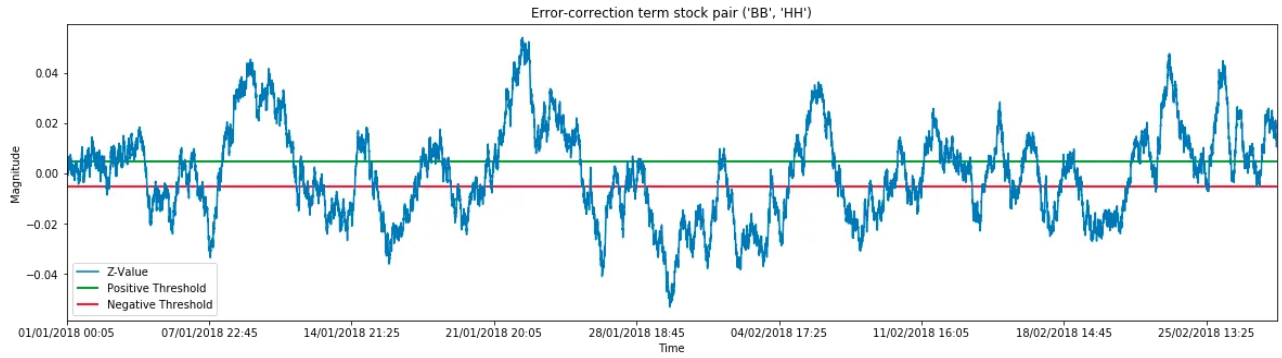

Error-correction term stock pair ('DD', 'HH')
Magnitude
Time
Z-Value
Positive Threshold
Negative Threshold
01/01/2018 00:05
07/01/2018 22:45
14/01/2018 21:25
21/01/2018 20:05
28/01/2018 18:45
04/02/2018 17:25
11/02/2018 16:05
18/02/2018 14:45
25/02/2018 13:25

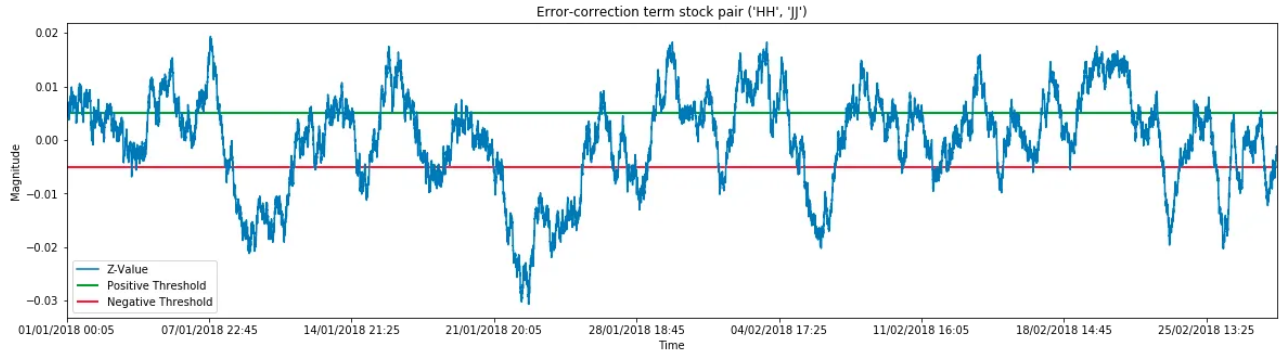

Error-correction term stock pair ('DD', 'JJ')
Magnitude
Time
Z-Value
Positive Threshold
Negative Threshold
01/01/2018 00:05
07/01/2018 22:45
14/01/2018 21:25
21/01/2018 20:05
28/01/2018 18:45
04/02/2018 17:25
11/02/2018 16:05
18/02/2018 14:45
25/02/2018 13:25

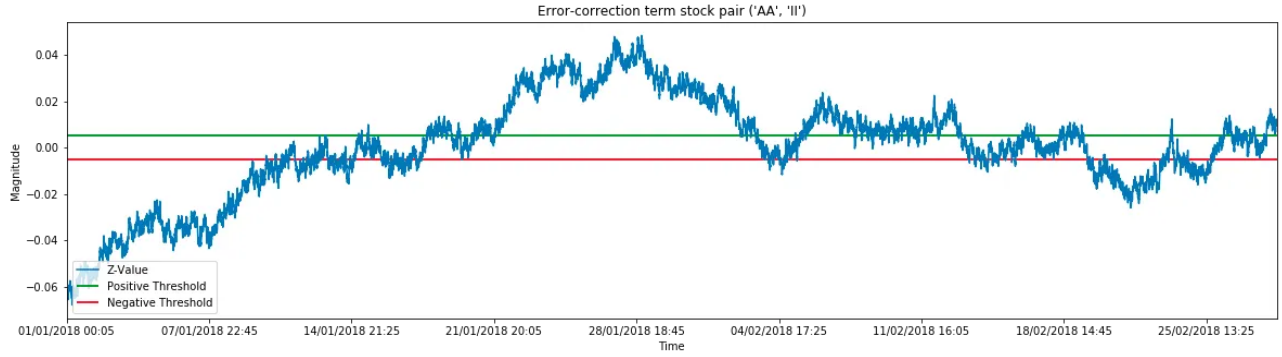

Error-correction term stock pair ('FF', 'MM')
Magnitude
Time
Z-Value
Positive Threshold
Negative Threshold
01/01/2018 00:05
07/01/2018 22:45
14/01/2018 21:25
21/01/2018 20:05
28/01/2018 18:45
04/02/2018 17:25
11/02/2018 16:05
18/02/2018 14:45
25/02/2018 13:25


Error-correction term stock pair ('FF', 'NN')
Magnitude
Time
Z-Value
Positive Threshold
Negative Threshold
01/01/2018 00:05
07/01/2018 22:45
14/01/2018 21:25
21/01/2018 20:05
28/01/2018 18:45
04/02/2018 17:25
11/02/2018 16:05
18/02/2018 14:45
25/02/2018 13:25





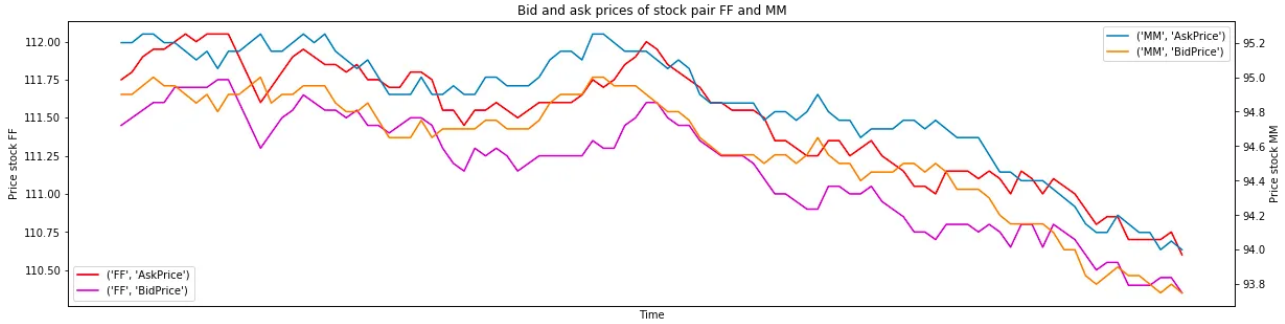




A function called Plot_Tradable_Z plots the z-values of all pairs based on the data_zvalues dataframe. Each pair in the stock_pairs list is looped through using a for loop. Afterward, it plots the z-values of each pair. Plots usually have a title that indicates what stock pair they represent as well as a x-axis and y-axis showing time and magnitude. Also added are horizontal lines at 0.005 and -0.005 threshold

values, randomly chosen. Z-value, positive threshold, and negative threshold are displayed in the plot legend. Using Plot_Tradable_Z() at the end of the code executes the function and displays the plots after it has been defined. The z-values of different stock pairs can be visualized and analyzed with this function.

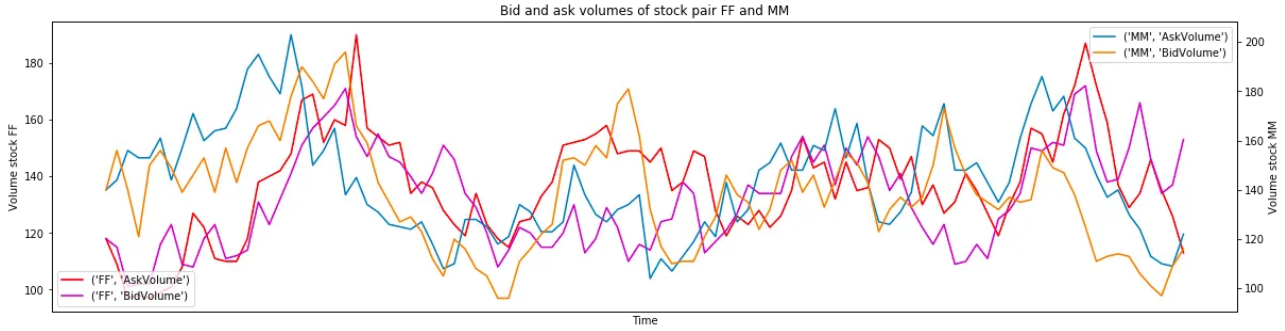
```
# Select randomly chosen pair from the tradable stock and
visualize bid and ask prices, bid and ask volumes, and the z-
values
import random

# Choose random stock
random_pair = random.choice(stock_pairs)

# Create a plot showing the bid and ask prices of a randomly
chosen stock
def Plot_RandomPair_BidAskPrices():
    """
    This function plots the bid and ask price of a randomly
chosen tradable pair.
    """
    plt.figure(figsize=(20,5))
    plt.title('Bid and ask prices of stock pair {} and
{}'.format(random_pair[0], random_pair[1]))

    plt.plot(tradable_pairs_data[random_pair[0],
'AskPrice'].iloc[:100], 'r')
    plt.plot(tradable_pairs_data[random_pair[0],
'BidPrice'].iloc[:100], 'm')
    plt.xlabel('Time')
    plt.ylabel('Price stock {}'.format(random_pair[0]))
    plt.legend(loc='lower left')

    plt.twinx()
    plt.plot(tradable_pairs_data[random_pair[1],
'AskPrice'].iloc[:100])
    plt.plot(tradable_pairs_data[random_pair[1],
'BidPrice'].iloc[:100])
    plt.xticks([])
    plt.ylabel('Price stock {}'.format(random_pair[1]))
    plt.legend(loc='upper right')

    plt.show()

Plot_RandomPair_BidAskPrices()
```



From a group of tradable pairs, this code randomly selects two stocks using the random library. Next, it defines a function named Plot_RandomPair_BidAskPrices, which plots the bid and ask prices. In this function, the randomly chosen pairs are used to plot the bid and ask prices for each stock. A title is also given to the graph including the names of the two stocks. Each stock's bid and ask prices are plotted using different colors, and the legend explains what each color means. In addition, the graph is displayed by using the plt library.

```
# Create a plot showing the bid and ask volumes of a randomly
chosen stock
def Plot_RandomPair_BidAskVolumes(): # Plot not really
clarifying, maybe other kind of plot?
    """
    This function plots the bid and ask volumes of a randomly
chosen tradable pair.
    """
    plt.figure(figsize=(20,5))
    plt.title('Bid and ask volumes of stock pair {} and
{}'.format(random_pair[0],random_pair[1]))

    plt.plot(tradable_pairs_data[random_pair[0],
'AskVolume'].iloc[:100], 'r')
    plt.plot(tradable_pairs_data[random_pair[0],
'BidVolume'].iloc[:100], 'm')
```

```
    plt.xlabel('Time')
    plt.ylabel('Volume stock {}'.format(random_pair[0]))
    plt.legend(loc='lower left')

    plt.twinx()
    plt.plot(tradable_pairs_data[random_pair[1],
'AskVolume'].iloc[:100])
    plt.plot(tradable_pairs_data[random_pair[1],
'BidVolume'].iloc[:100])
    plt.xticks([])
    plt.ylabel('Volume stock {}'.format(random_pair[1]))
    plt.legend(loc='upper right')

    plt.show()

Plot_RandomPair_BidAskVolumes()
```



It takes no arguments and returns nothing. Using this function, a figure with a size of 20x5 and a title indicating the two stocks is created. There are red and magenta lines representing the bid and ask volumes of the first stock, respectively. The volume of the second stock is plotted on the secondary y-axis. On the x-axis, time is represented while on the y-axis, stock volume is represented. Next, the function plots the ask and bid volumes for the second stock on the same plot, using different colors this time. It shows how the bid and ask volumes for the two randomly chosen stocks compare over time. It is useful for analyzing trading patterns and trends between different stocks.

Onepagecode is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

```
# Create a Dataframe containing information about the error-
correction term of each pair
data_error_correction_term = {'Pair': [],
                              'CountZeroCrossings': [],
                              'TradingPeriod': [],
                              'LongRunMean': [],
                              'Std': []}

for pair in stock_pairs:
    zvalue = data_zvalues[pair]
    my_array = np.array(zvalue)
    count = ((my_array[:-1] * my_array[1:]) < 0).sum()
    trading_period = 1 / count
    long_run_mean = zvalue.mean()
    std = zvalue.std()

    data_error_correction_term['Pair'].append(pair)

data_error_correction_term['CountZeroCrossings'].append(count)

data_error_correction_term['TradingPeriod'].append(trading_peri
od)

data_error_correction_term['LongRunMean'].append(round(long_run
_mean, 4))
    data_error_correction_term['Std'].append(round(std, 4))

data_error_correction_term =
pd.DataFrame(data_error_correction_term).set_index('Pair')

data_error_correction_term
```

Pair	CountZeroCrossings	TradingPeriod	LongRunMean	Std
(BB, DD)	660	0.001515	-0.0	0.0100
(BB, JJ)	1475	0.000678	-0.0	0.0044
(DD, HH)	396	0.002525	-0.0	0.0097
(DD, JJ)	592	0.001689	0.0	0.0072
(FF, MM)	1413	0.000708	0.0	0.0031
(FF, NN)	705	0.001418	-0.0	0.0061
(MM, NN)	627	0.001595	0.0	0.0068
(BB, HH)	334	0.002994	0.0	0.0181
(HH, JJ)	363	0.002755	-0.0	0.0090
(AA, II)	363	0.002755	0.0	0.0214

Each loop of this code is designed to correct an error in a dataset containing stock trading information. The relevant data is stored in a dictionary that is initialized with the appropriate keys. The code then loops through each stock pair in the stock_pairs variable and calculates a z-value based on the data_zvalues. A numpy array is formed from the z-value array and the number of zero crossings is calculated by comparing each element with the next. Using this count, we calculate a trading period as well as long run means and standard deviations for that stock pair. In the empty dictionary, the resulting values are added to the respective keys. This is followed by the conversion of the dictionary into a Pandas dataframe, where the index is set to have the column ‘Pair’. The error correction data can be easily visualized and analyzed in this way.

Threshold Analysis for All Stock Pairs

This section is for discovering what treshold would maximize profits for a given pair. We did many loops over different thresholds to find the ideal level. In the next section, the pairs with the highest profits those that are unique (prevent having a stock as ‘BB’ be traded twice) will be picked to determine the final profit

```
# Create a new column within the earlier defined DataFrame with
Z-Values of all stock pairs
for pair in stock_pairs:
```

```
    stock1 = pair[0]
    stock2 = pair[1]

    tradable_pairs_data[stock1+stock2, 'Z-Value'] =
data_zvalues[stock1,stock2]
```


Iterating through a list of stock names called stock_pairs is done with a for loop. First, the first stock name is assigned to stock1 and the second stock name is assigned to stock2. Then, a new entry is created in a dictionary named tradable_pairs_data. This entry's key is a combination of stock names (stock1+stock2) and the string 'Z-Value'. This entry is derived from another dictionary called data_zvalues, which uses the names of stocks as keys. Using pairs of stock names as keys, this code takes information from a dictionary and organizes it in another dictionary in a new way.

```
# Create a Dictionary that saves all Gamma values of each pair
gamma_dictionary = {}

for pair, value in tradable_pairs_analysis.iterrows():
    gamma_dictionary[pair]= value['Gamma']

gamma_dictionary
```

```
{('BB', 'DD'): 1.4758,
 ('BB', 'JJ'): 1.6206,
 ('DD', 'HH'): 1.2787,
 ('DD', 'JJ'): 1.0356,
 ('FF', 'MM'): 0.9676,
 ('FF', 'NN'): 1.4881,
 ('MM', 'NN'): 1.5153,
 ('BB', 'HH'): 1.8712,
 ('HH', 'JJ'): 0.7041,
 ('AA', 'II'): 0.9944}
```

This code creates the empty dictionary gamma_dictionary as the first line of code. A dictionary stores key-value pairs, rather than storing values in a specific order like a list. In the second line of code, a for loop is started. Throughout this loop, each row in the tradable_pairs_analysis table is analyzed. tradeable_pairs_analysis consists of a table that contains the values for each row and the pairs and values for each pair. The values in each row are easy to access this way. It adds a new pair of key-value pairs to the gamma_dictionary dictionary. The key variable is the pair variable, and the value variable is the Gamma value. As a result, this code loops through each row in a table and adds the values of pair and gamma to an empty dictionary.

```
# Create a Dictionary that saves all Standard Deviation values
of each pair
std_dictionary = {}

for pair, value in data_error_correction_term.iterrows():
    std_dictionary[pair]= value['Std']

std_dictionary
```

```
{('BB', 'DD'): 0.01,
 ('BB', 'JJ'): 0.0044,
 ('DD', 'HH'): 0.0097,
 ('DD', 'JJ'): 0.0072,
 ('FF', 'MM'): 0.0031,
 ('FF', 'NN'): 0.0061,
 ('MM', 'NN'): 0.0068,
 ('BB', 'HH'): 0.0181,
 ('HH', 'JJ'): 0.009,
 ('AA', 'II'): 0.0214}
```

A dictionary filled with pairs and corresponding values is added for each row of the data_error_correction_term data set. The pairs are used as keys in the dictionary, with their corresponding values. Additionally, it creates another key, time, which is assigned the std value from the data set. By creating a dictionary, this code will allow us to store and access standardized data values from the data_error_correction_term data set.

This is our Algorithm for finding the correct thresholds that are able to generate the greatest amount of profit. We find it important to not maximize the profit because what holds for historic data is not guaranteed to hold for future data. We therefore specify a limited selection of thresholds with a linspace.

```
positions = {}
limit = 100

for pair in stock_pairs:
    stock1 = pair[0]
    stock2 = pair[1]

    gamma = gamma_dictionary[stock1,stock2]

    for i in np.linspace(0.05, 1.0, 10):
        threshold = i * std_dictionary[stock1,stock2]
```

```
        current_position_stock1 = 0
        current_position_stock2 = 0

        column_name_stock1 = stock1 + ' Pos - Thres: ' +
str(threshold)

        BidPrice_Stock1 =
tradable_pairs_data[stock1,'BidVolume'][0]
        AskPrice_Stock1 =
tradable_pairs_data[stock1,'AskVolume'][0]
        BidPrice_Stock2 =
tradable_pairs_data[stock2,'BidVolume'][0]
        AskPrice_Stock2 = tradable_pairs_data[stock1
```



```
,'AskVolume'][0]

        positions[column_name_stock1] = []

        for time, data_at_time in
tradable_pairs_data.iterrows():

            BidVolume_Stock1 = data_at_time[stock1,
'BidVolume']
            AskVolume_Stock1 = data_at_time[stock1,
'AskVolume']
            BidVolume_Stock2 = data_at_time[stock2,
'BidVolume']
            AskVolume_Stock2 = data_at_time[stock2,
'AskVolume']

            zvalue = data_at_time[stock1+stock2,'Z-Value']
```

```
            # If the zvalues of (BB,DD) are high the spread
diverges, i.e. sell BB (=stock1=y) and buy DD (=stock2=x)
            if zvalue >= threshold:
                hedge_ratio = gamma * (BidPrice_Stock1 /
AskPrice_Stock2)

                if hedge_ratio >= 1:

                    max_order_stock1 = current_position_stock1
+ limit
                    max_order_stock2 = max_order_stock1 /
hedge_ratio

                    trade = np.floor(min((BidVolume_Stock1 /
hedge_ratio), AskVolume_Stock2, max_order_stock1,
max_order_stock2))

                    positions[column_name_stock1].append((-
trade * hedge_ratio) + current_position_stock1)

                    current_position_stock1 = ((- trade *
hedge_ratio) + current_position_stock1)

                elif hedge_ratio < 1:

                    max_order_stock1 = current_position_stock1
+ limit
                    max_order_stock2 = max_order_stock1 *
hedge_ratio

                    trade = np.floor(min((BidVolume_Stock1 *
hedge_ratio), AskVolume_Stock2, max_order_stock1,
max_order_stock2))

                    positions[column_name_stock1].append((-
trade / hedge_ratio) + current_position_stock1)

                    current_position_stock1 = ((- trade /
hedge_ratio) + current_position_stock1)

            elif zvalue <= -threshold:
                hedge_ratio = gamma * (AskPrice_Stock1 /
BidPrice_Stock2)

                if hedge_ratio >= 1:

                    max_order_stock1 =
```

```
abs(current_position_stock1 - limit)
                    max_order_stock2 = max_order_stock1 /
hedge_ratio

                    trade = np.floor(min((AskVolume_Stock1 /
hedge_ratio), BidVolume_Stock2, max_order_stock1,
max_order_stock2))

                    positions[column_name_stock1].append((+
trade * hedge_ratio) + current_position_stock1)

                    current_position_stock1 = (+ trade *
hedge_ratio) + current_position_stock1

                elif hedge_ratio < 1:

                    max_order_stock1 =
abs(current_position_stock1 - limit)
                    max_order_stock2 = max_order_stock1 *
hedge_ratio

                    trade = np.floor(min((AskVolume_Stock1 *
hedge_ratio), BidVolume_Stock2, max_order_stock1,
max_order_stock2))

                    positions[column_name_stock1].append((+
trade / hedge_ratio) + current_position_stock1)

                    current_position_stock1 = (+ trade /
hedge_ratio) + current_position_stock1

                BidPrice_Stock1 = data_at_time[stock1,
'BidPrice']
                AskPrice_Stock1 = data_at_time[stock1,
'AskPrice']
                BidPrice_Stock2 = data_at_time[stock2,
'BidPrice']
                AskPrice_Stock2 = data_at_time[stock2,
'AskPrice']

            else:
```

```
positions[column_name_stock1].append(current_position_stock1)

        column_name_stock2 = stock2 + ' Pos - Thres: ' +
str(threshold)

        if hedge_ratio >= 1:
            positions[column_name_stock2] =
positions[column_name_stock1] / hedge_ratio * -1

        elif hedge_ratio < 1:
            positions[column_name_stock2] =
positions[column_name_stock1] / (1 / hedge_ratio) * -1
```

Based on gamma dictionaries and standard deviation dictionaries, this code calculates trading positions for two stocks. Code finds gamma (a measure of price correlation between two stocks) and creates thresholds for each pair of stock pairs. As a result, the code iterates through values between 0.05 and 1.0 and calculates each stock's current position based on the bid and ask prices. A positions dictionary stores these positions. Using the bid and ask volumes and the maximum order size, based upon the hedge ratio (calculated using the gamma value), the code calculates and executes trades if the z-value is above the threshold. The code does the same if the z-value is below threshold, but it uses a different hedge ratio. Based on the hedge ratio, the code calculates the positions for the other stock in the pair. The positions dictionary also contains thresholds for each pair, along with the positions.

```
positions_final = np.ceil(pd.DataFrame(positions))
positions_final['Timestamp'] = tradable_pairs_data.
```

```
index
positions_final = positions_final.set_index('Timestamp')
```

An array of positions is converted to a pandas dataframe in this code. Positions_final is created by rounding up the numerical values in the DataFrame. The DataFrame is then created with a new column called Timestamp and the index values are assigned from a data source called tradable_pairs_data. The Timestamp column is then assigned an index in the DataFrame. The code converts a list of position data into a DataFrame and adds a column indexed by timestamp data. The data can be more easily manipulated and analyzed as a result.

```
# The difference between the positions
positions_diff = positions_final.diff()[1:]

# Positions_diff first rows
positions_diff.head()

# OPTIONAL to Excel to Save the Amount of Trades
# positions_diff[(positions_diff !=
0)].count().to_excel('Thresholds.xlsx')
```

	BB Pos - Thres: 0.0005	DD Pos - Thres: 0.0005	BB Pos - Thres: 0.0015555555555555557	DD Pos - Thres: 0.0015555555555555557	BB Pos - Thres: 0.0026111111111111114	DD Pos - Thres: 0.0026111111111111114	BB Pos - Thres: 0.0036666666666666666	DD Pos - Thres: 0.0036666666666666666
Timestamp								
01/01/2018 00:10	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
01/01/2018 00:15	0.0	0.0	-98.0	80.0	-98.0	80.0	0.0	0.0
01/01/2018 00:20	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
01/01/2018 00:25	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
01/01/2018 00:30	132.0	-106.0	0.0	0.0	0.0	0.0	0.0	0.0

It begins by creating a variable called positions_diff, and then assigning it the result of applying the diff() function to a variable called positions_final. When the function is called, it excludes the first row of data by calling [1:]. Afterward, the head() function displays the first few rows. This code calculates the difference between each position and the previous one, and displays the first few rows. Data can be analyzed over time in order to identify patterns or trends.

This method is used to value our last position by the correct market value. This to ensure that in the time between our last trade and the last timestamp does not hold any secrets (a market crash for example) that are not calculated in the PnL. One could say the profit is for example €50.000 while it is actually far lower because our positions are worth next to nothing due to a market crash.

```
positions_diff[-1:] = -positions_final[-1:]
```

Using this Python code, two lists are involved: positions_diff and positions_final. It takes the last element of the second list and changes its sign to negative. After this is done, it replaces the last element of the first list with the updated element. This is accomplished by copying the value of the last element in the second list and changing it to its negative counterpart. It then updates the last element in the first list.

To determine which threshold is the most profitable, we determine the PnL of each combination of pair and threshold.

```
pnl_dataframe = pd.DataFrame()

for pair in stock_pairs:
    stock1 = pair[0]
    stock2 = pair[1]

    Stock1_AskPrice = tradable_pairs_data[stock1, 'AskPrice']
[1:]
    Stock1_BidPrice = tradable_pairs_data[stock1, 'BidPrice']
[1:]
    Stock2_AskPrice = tradable_pairs_data[stock2, 'AskPrice']
[1:]
    Stock2_BidPrice = tradable_pairs_data[stock2, 'BidPrice']
[1:]

    for i in np.linspace(0.05, 1.0, 10):
        threshold = i * std_dictionary[stock1, stock2]

        column_name_1 = stock1 + ' Pos - Thres: ' +
str(threshold)
```

```
        column_name_2 = stock2 + ' Pos - Thres: ' +
str(threshold)
```

Onepagecode is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

```
        pnl_dataframe[stock1 + str(threshold)] =
np.where(positions_diff[column_name_1] > 0,

positions_diff[column_name_1] * -Stock1_BidPrice,
positions_diff[column_name_1] * -Stock1_AskPrice)
        pnl_dataframe[stock2 + str(threshold)] =
np.where(positions_diff[column_name_2] > 0,

positions_diff[column_name_2] * -Stock2_BidPrice,
positions_diff[column_name_2] * -Stock2_AskPrice)

pnl_dataframe.head()
```

	BB0.0005	DD0.0005	BB0.0015555555555555557	DD0.0015555555555555557	BB0.0026111111111111114	DD0.0026111111111111114	BB0.0036666666666666666	DD0.0036666666666666666
0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
1	-0.0	-0.0	9893.1	-9352.0	9893.1	-9352.0	-0.0	-0.0
2	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
3	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0
4	-13259.4	12439.1	-0.0	-0.0	-0.0	-0.0	-0.0	-0.0

pnl_dataframe is a dataframe filled with statistics about stock pairs created by this code. A list of stock pairs is first read and each pair's first and second stock are stored as variables. The variables ask and bid prices are then retrieved from a dataset called tradable_pairs_data and added to the dataset. Next, a for loop iterates through values from 0.05 to 1.0, with increments of 0.05. Calculates a threshold for each value by multiplying it by the stock pair's standard deviation. Based on the stock names and threshold values, two column names are generated. As the code proceeds, each stock's threshold value is added to the dataframe in the last two lines. By multiplying the position difference by the negative ask or bid price, it calculates the profit or loss using the np.where function if the positions_diff column is greater than 0. Following that, the information is stored in the appropriate column of the dataframe.

```
# Create Columns for the pnl_threshold dataframe
pairs = []
thresholds = []

for pair in stock_pairs:
    stock1 = pair[0]
    stock2 = pair[1]
```



```
    for i in np.linspace(0.05, 1.0, 10):
        threshold = i * std_dictionary[stock1, stock2]
        pair = stock1, stock2
        pairs.append(pair)
        thresholds.append(threshold)
```

A variable stock1 is assigned to every pair in the list stock_pairs, and a variable stock2 is assigned to every pair in the list. Using numpy, a range, from 0.05 to 1.0, with 10 evenly-spaced intervals, is then generated. It assigns a variable called threshold to each number in this range, multiplied by the standard deviation dictionary of the pair (stock1 and stock2). In order to include this threshold in the thresholds list, it is added. Lastly, stock1 and stock2 are added to the pairs list. Each pair in the stock_pairs list is then processed, resulting in the pairs and thresholds lists, respectively, being filled with related pairs and thresholds.

```
# Include columns and append PnLs
pnl_threshold = {'Pairs' : pairs,
```

```
                    'Thresholds': thresholds,
                    'PnLs' : []}

for pair in stock_pairs:
    stock1 = pair[0]
    stock2 = pair[1]

    for i in np.linspace(0.05, 1.0, 10):
        threshold = i * std_dictionary[stock1,stock2]
        pnl_threshold['PnLs'].append(pnl_dataframe[stock1 +
str(threshold)].sum() + pnl_dataframe[stock2 +
str(threshold)].sum())

pnl_threshold = pd.DataFrame(pnl_threshold)
pnl_threshold = pnl_threshold.set_index('Pairs')
# pnl_threshold.to_excel('Thresholds.xlsx')
```

There are three keys in the dictionary pnl_threshold: Pairs, Thresholds, and Pairs-threshold. Stock pairs are represented by the ‘Pairs’ key, thresholds are represented by the ‘Thresholds’ key, and PnLs is an empty list. The dictionary is then looped through for each pair in stock_pairs. Throughout this loop, stock1 and stock2 are assigned to the first and second elements of each pair. Within the first loop, a new loop iterates through a sequence of numbers between 0.05 and 1.0. The threshold is calculated each time by multiplying the current number by a value from std_dictionary based on stock1 and stock2 indices. After this threshold has been determined, it is used to access values from a data frame called pnl_dataframe and add them to the ‘PnLs’ list in pnl_threshold. A dataframe is created from the dictionary and indexed by the ‘Pairs’ column after the loops finish. Thus, the dataframe contains columns for thresholds and PnLs, with rows representing stock pairs and their corresponding values.

```
# Find Highest PnLs
highest_pnls = pnl_threshold.groupby(by='Pairs').agg({'PnLs' :
max})
highest_pnls.sort_values('PnLs', ascending=False)
```

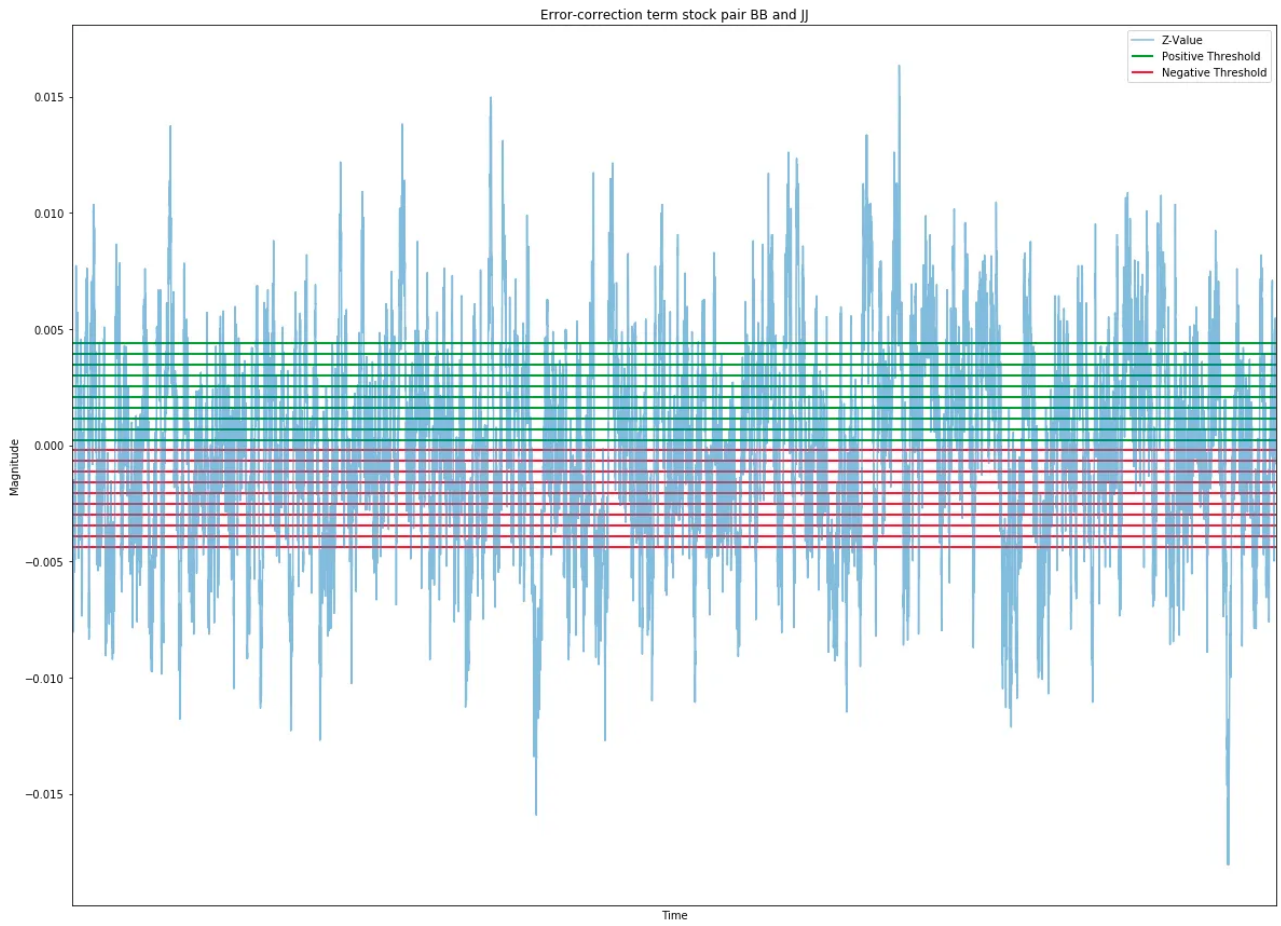
Onepagecode is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

	PnLs
Pairs	
(FF, MM)	59186.40
(BB, JJ)	46740.30
(FF, NN)	29441.50
(BB, DD)	26140.55
(MM, NN)	21508.00
(DD, JJ)	19887.50
(DD, HH)	19541.65
(BB, HH)	16306.90
(AA, II)	10433.60
(HH, JJ)	5390.10

Data in the pnl_threshold variable is grouped by the Pairs column using the groupby function. To calculate the maximum (max) value of each PnLs column, we use the agg function. Highest_pnls holds the resulting data. By using the sort_values function, we sort the data in the highest_pnls variable descendingly according to PnLs values. To sort the data from highest to lowest, this code finds the highest profit and loss (PnL) value for each pair.

```
# Plot error-correction term (z-value) to observe what the
spread looks like (see slide for comparison plot cointegrated
pair)
def Plot_Thresholds(stock1, stock2):
    zvalue = tradable_pairs_data[stock1+stock2,'Z-Value']
    plt.figure(figsize=(20,15))
    plt.xticks([])
    plt.title('Error-correction term stock pair ' + stock1 + '
and ' + stock2)
```

```
    zvalue.plot(alpha=0.5)
    plt.xlabel('Time')
    plt.ylabel('Magnitude')
    xmin = 0
    xmax = len(zvalue)

    # Boundries chosen to give an approximate good fit
    plt.hlines(pnl_threshold['Thresholds'][10:20], xmin, xmax,
'g')
    plt.hlines(-pnl_threshold['Thresholds'][10:20], xmin, xmax,
'r')

    plt.legend(['Z-Value', 'Positive Threshold', 'Negative
Threshold'])
    plt.show()

Plot_Thresholds('BB','JJ')
```



Plot_Thresholds is a function that takes two stock names as arguments, stock1 and stock2. In this function, the trading pairs are retrieved from a variable called tradable_pairs_data by adding a + symbol between the stock names and then

retrieving the Z-Value. This results in a figure with a size of 20 by 15 and the ticks on the x-axis are hidden. These two stocks are referred to in the figure's title. Eventually, the Z-Value data is plotted on the figure with an alpha value of 0.5 (to partially make it transparent). Afterwards, the function adds y-labels of Magnitude and Time to the figure. Moreover, it sets the minimum and maximum values based on the length of the Z-Value data on the x-axis. After adding two horizontal lines, the function checks positive threshold values and negative threshold values, using the pnl_threshold variable. In the legend of the figure, the Z-Value line and the two threshold lines are identified. In the end, the function displays the chart using the plt.show() command, passing in the names of the two stocks as arguments. In this case, BB and JJ are used as stock names. To help analyze the performance of two stocks, this code plots the Z-Value data for each stock along with positive and negative threshold lines.

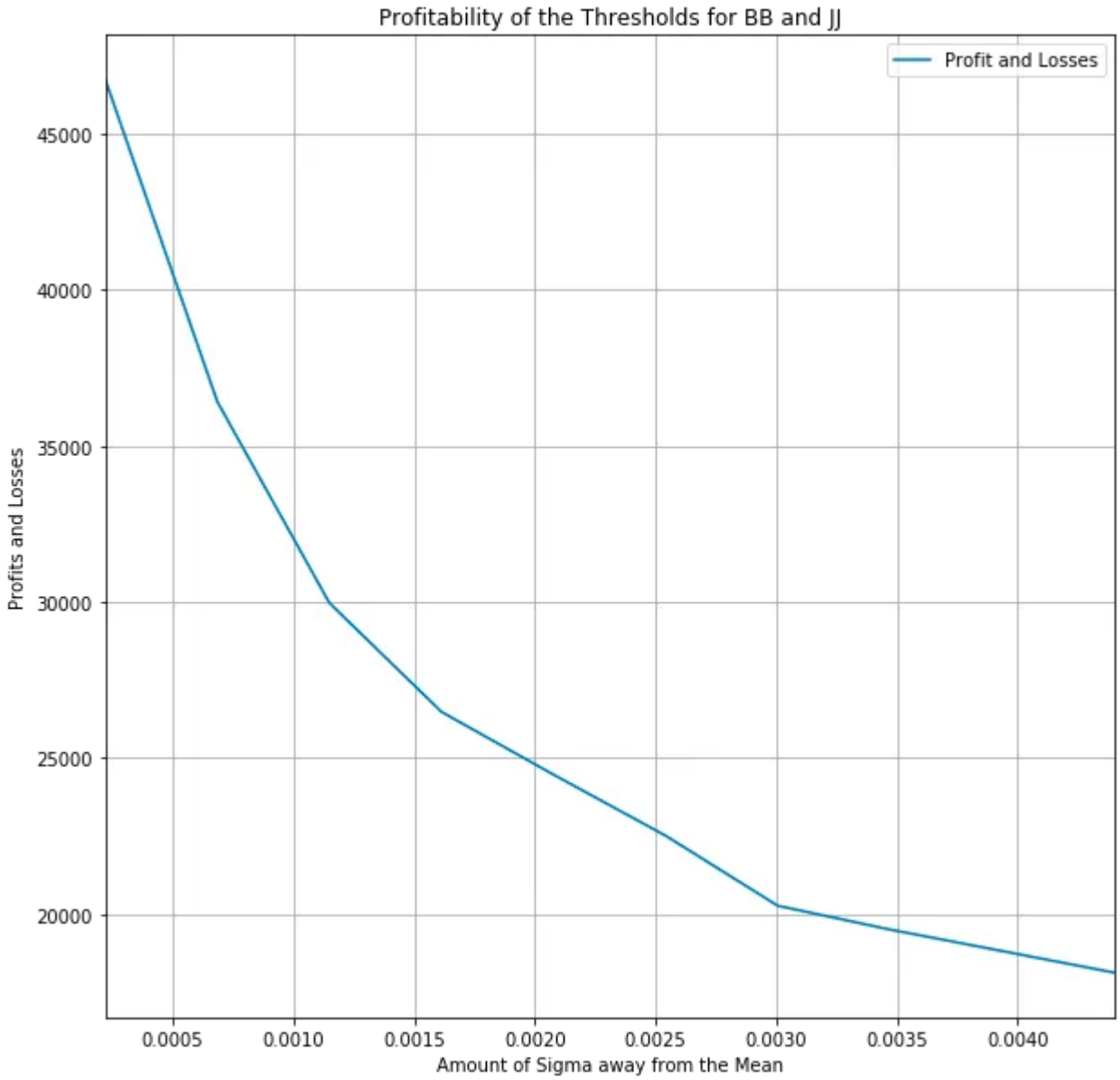
```
# Create a Plot that displays the Profitability of the
Thresholds

def profitability_of_the_thresholds(stock1, stock2):
```



```
    pnl_threshold[(pnl_threshold.index == (stock1,
stock2))].plot(x='Thresholds', y='PnLs', figsize=(10,10))
    plt.title('Profitability of the Thresholds for ' + stock1 +
' and '
```

Onepagecode is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

```
    + stock2)
       plt.xlabel('Amount of Sigma away from the Mean')
       plt.ylabel('Profits and Losses')
       plt.legend(['Profit and Losses'])
       plt.grid()

profitability_of_the_thresholds('BB','JJ')
```



In the following code, profitability_of_the_thresholds is defined as a function that takes two parameters: stock1 and stock2. Using a pandas data frame called

pnl_threshold, this function plots the profitability of the two stocks (stock1 and stock2). Standard deviations represent the distance from the average value of the stocks at these thresholds. In the following code, the plot is titled, has an x-axis label, a y-axis label, and a legend, and a grid is displayed. The function is then called with two specific stocks, BB and JJ, and the plot shows their profitability at various thresholds. Different thresholds are visualized for two stocks in order to figure out the profitability for each.

Algorithm Strategy 1

Making use of the previous analysis to determine which pairs should be traded. Based on that the algorithm, with slight modifications, is ran again to calculate the final profits.

```
# Determine the treshold, manually chosen based on
pnl_threshold and ensuring no overlap.
threshold_dictionary = {('BB', 'JJ'): 0.000220,
                        ('FF', 'MM'): 0.000155,
                        ('DD', 'HH'): 0.000485,
                        ('AA', 'II'): 0.001070}

threshold_dictionary
```

```
{('BB', 'JJ'): 0.00022,
 ('FF', 'MM'): 0.000155,
 ('DD', 'HH'): 0.000485,
 ('AA', 'II'): 0.00107}
```

Dictionary entries are stored in key-value pairs. A pair consists of two tuples, where the first item represents a category and the second item represents a different category. There are two letters associated with each of these categories. There is a decimal number associated with each pair. Comparing two categories is done using a threshold. In a similar category, a pair has a value greater than an input. Otherwise,

they are considered separate. Using this dictionary, you can categorize items according to their similarities and differences.

```
# Selection of the final pairs for this trading strategy
stock_pairs_final = [('BB', 'JJ'),
                     ('FF', 'MM'),
                     ('DD', 'HH'),
```

Onepagecode is a reader-supported publication. To receive new posts and support my work, consider becoming a free or paid subscriber.

```
                     ('AA', 'II')]

stock_pairs_final
```

```
[('BB', 'JJ'), ('FF', 'MM'), ('DD', 'HH'), ('AA', 'II')]
```

('BB', 'JJ'), ('FF', 'MM')] The following code creates a list called stock_pairs_final and assigns it with two stock pair tuples. There are four tuples in the first line, each representing a pair of stocks. In each tuple, stock is indicated by two capital letters. Assigned to the list stock_pairs_final are only the first two tuples from the original four. The code removes the last two stock pairs from the list and keeps only the first two. In some cases, it may be preferable to only consider a limited number of stock pairs.

This algorithm is a slight modification as the previous one used. In this algorithm we incorporate the chosen pairs, with the corresponding tresholds, to determine the most optimal positions.

```
positions_strategy_1 = {}
limit = 100

for pair in stock_pairs_final:
    stock1 = pair[0]
```

```
    stock2 = pair[1]

    gamma = gamma_dictionary[stock1,stock2]

    threshold = threshold_dictionary[stock1,stock2]

    current_position_stock1 = 0
    current_position_stock2 = 0

    positions_strategy_1[stock1] = []

    for time, data_at_time in tradable_pairs_data.iterrows():

        BidPrice_Stock1 = data_at_time[stock1, 'BidPrice']
        AskPrice_Stock1 = data_at_time[stock1, 'AskPrice']
        BidPrice_Stock2 = data_at_time[stock2, 'BidPrice']
        AskPrice_Stock2 = data_at_time[stock2, 'AskPrice']
```



```
        BidVolume_Stock1 = data_at_time[stock1, 'BidVolume']
        AskVolume_Stock1 = data_at_time[stock1, 'AskVolume']
        BidVolume_Stock2 = data_at_time[stock2, 'BidVolume']
        AskVolume_Stock2 = data_at_time[stock2, 'AskVolume']

        zvalue = data_at_time[stock1+stock2,'Z-Value']

        if zvalue >= threshold:
            hedge_ratio = gamma * (BidPrice_Stock1 /
AskPrice_Stock2)

            if hedge_ratio >= 1:

                max_order_stock1 = current_position_stock1 +
limit
                max_order_stock2 = max_order_stock1 /
hedge_ratio

                trade = np.floor(min((BidVolume_Stock1 /
hedge_ratio), AskVolume_Stock2, max_order_stock1,
max_order_stock2))
```

```
                positions_strategy_1[stock1].append((- trade *
hedge_ratio) + current_position_stock1)

                current_position_stock1 = ((- trade *
hedge_ratio) + current_position_stock1)

            elif hedge_ratio < 1:

                max_order_stock1 = current_position_stock1 +
limit
                max_order_stock2 = max_order_stock1 *
hedge_ratio

                trade = np.floor(min((BidVolume_Stock1 *
hedge_ratio), AskVolume_Stock2, max_order_stock1,
max_order_stock2))

                positions_strategy_1[stock1].append((- trade /
hedge_ratio) + current_position_stock1)

                current_position_stock1 = ((- trade /
hedge_ratio) + current_position_stock1)

        elif zvalue <= -threshold:
            hedge_ratio = gamma * (AskPrice_Stock1 /
BidPrice_Stock2)

            if hedge_ratio >= 1:

                max_order_stock1 = abs(current_position_stock1
- limit)
                max_order_stock2 = max_order_stock1 /
hedge_ratio

                trade = np.floor(min((AskVolume_Stock1 /
hedge_ratio), BidVolume_Stock2, max_order_stock1,
max_order_stock2))

                positions_strategy_1[stock1].append((+ trade *
hedge_ratio) + current_position_stock1)

                current_position_stock1 = (+ trade *
hedge_ratio) + current_position_stock1

            elif hedge_ratio < 1:
```

```
                max_order_stock1 = abs(current_position_stock1
- limit)
                max_order_stock2 = max_order_stock1 *
hedge_ratio

                trade = np.floor(min((AskVolume_Stock1 *
hedge_ratio), BidVolume_Stock2, max_order_stock1,
max_order_stock2))

                positions_strategy_1[stock1].append((+ trade /
hedge_ratio) + current_position_stock1)

                current_position_stock1 = (+ trade /
hedge_ratio) + current_position_stock1

        else:


positions_strategy_1[stock1].append(current_position_stock1)

    if hedge_ratio >= 1:
        positions_strategy_1[stock2] =
positions_strategy_1[stock1] / hedge_ratio * -1

    elif hedge_ratio < 1:
        positions_strategy_1[stock2] =
positions_strategy_1[stock1] / (1 / hedge_ratio) * -1
```

Using this code, a pair of stocks is traded according to a trading strategy. Positions_strategy_1 is created, and 100 is set as the upper limit for trading. Following that, it performs an iterative calculation of gamma and threshold for a list of stock pairs. Following that, variables are initialized for each stock's current position and an empty list is created for stock 1. After that, the code enters a for loop for each time and data point in the tradable_pairs_data array. A for loop calculates the BidPrice, AskPrice, BidVolume, AskVolume, and z-value of each stock within it. Following that, if the z-value exceeds the threshold, the operation is repeated. Using the gamma and each stock's bid price and ask price, the hedge ratio is calculated. In the presence of a hedge ratio greater than 1 or equal to 1, the code calculates the maximum order quantity for each stock in accordance with the position and limit. To calculate the trade, it takes the minimum of the BID Volume for stock 1 divided by the hedge ratio, the ASK Volume for stock 2, and the maximum order quantity for each stock. A position_strategy_1 dictionary is then created and the stock1 position

is updated accordingly. It follows the same steps but multiplies BidVolume for stock1 by the hedge ratio if the hedge ratio is less than 1. Alternatively, the code calculates the hedge ratio based on the asking price and bid price of stock1 and stock2, respectively, if the z-value is below the negative threshold. Besides checking if the hedge ratio is greater than or less than 1, the code performs the necessary calculations. Lastly, if the z-value for stock1 is not above or below the threshold, the code stores the stock1 position in the positions_strategy_1 dictionary. For the second position, the code divides the position for stock1 by 1/hedge ratio, and multiplies by -1 to indicate a short position, and checks the hedge ratio again.

```
# Set Ceiling (to prevent positions with not enough volume
available) as well as define the timestamp
positions_strategy_1 =
np.ceil(pd.DataFrame(positions_strategy_1))
positions_strategy_1['Timestamp'] = tradable_pairs_data.index
positions_strategy_1 =
positions_strategy_1.set_index('Timestamp')
```

An array of integers is created from a pandas DataFrame called positions_strategy_1 and then rounded to the nearest whole number. A new DataFrame called tradable_pairs_data is added to the DataFrame using the index values from a new DataFrame called Timestamp. Lastly, it adds a column called Timestamp to positions_strategy_1. As a whole, it is likely that the code is being used to organize and manipulate data in a particular format for use in a trading strategy.

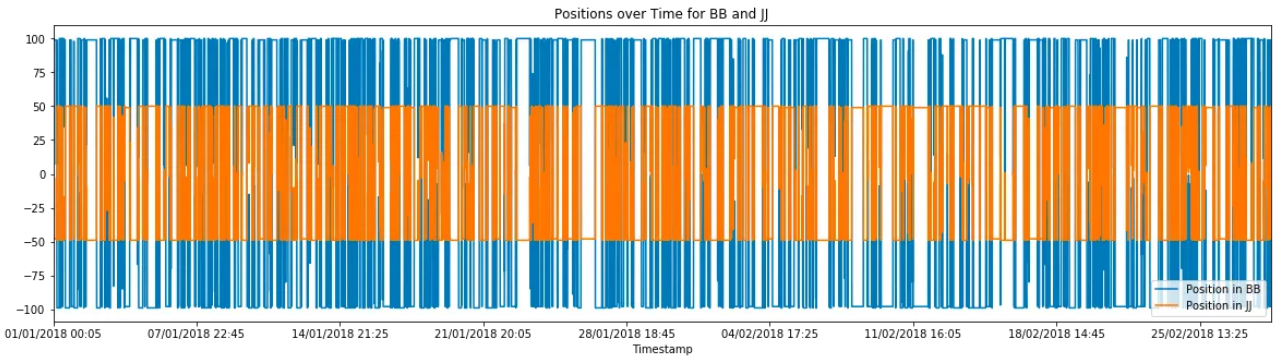
```
# The difference between the positions
positions_diff_strategy_1 = positions_strategy_1.diff()[1:]
```

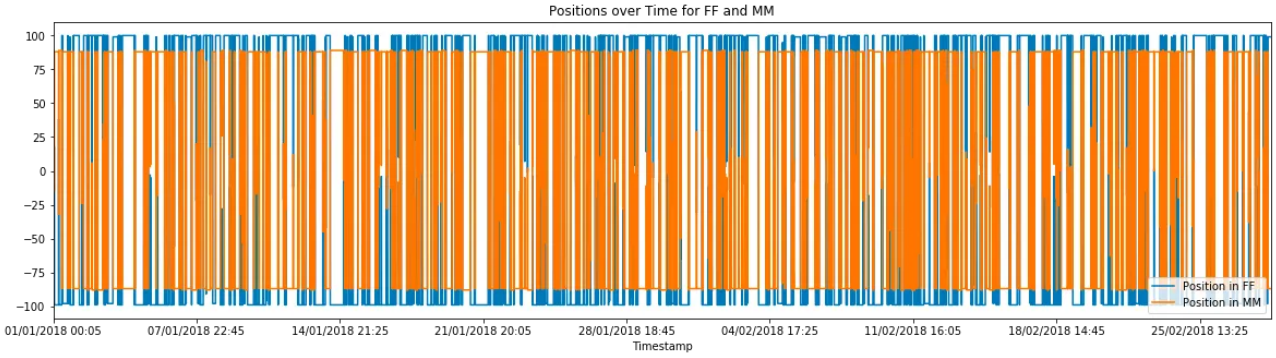
By using the .diff() function, this code takes in the variable ‘positions_strategy_1’ and calculates the difference between each data point. The first data point is then sliced off ([1:]) and saved in the variable ‘positions_diff_strategy_1’. Analyzing trends or changes in data is possible with this technique, as it allows comparison of consecutive data points.

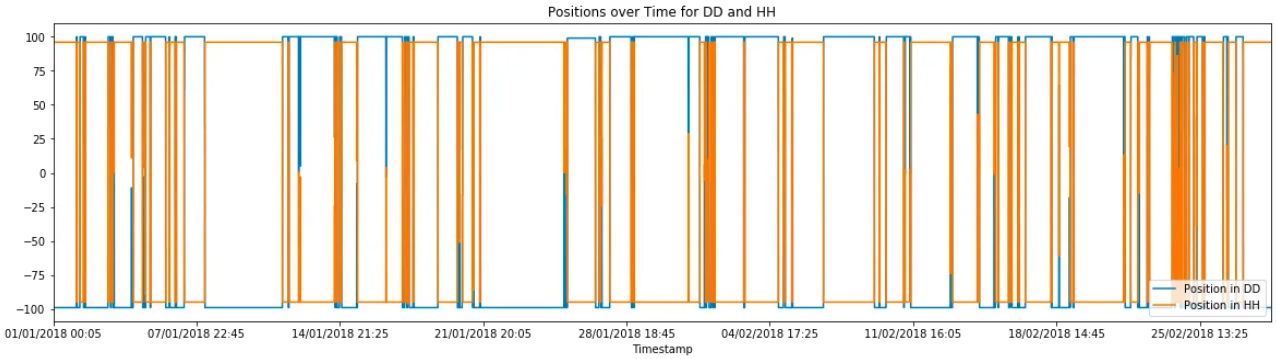
```
#Used as mentioned earlier.
positions_diff_strategy_1[-1:] = -positions_strategy_1[-1:]
```

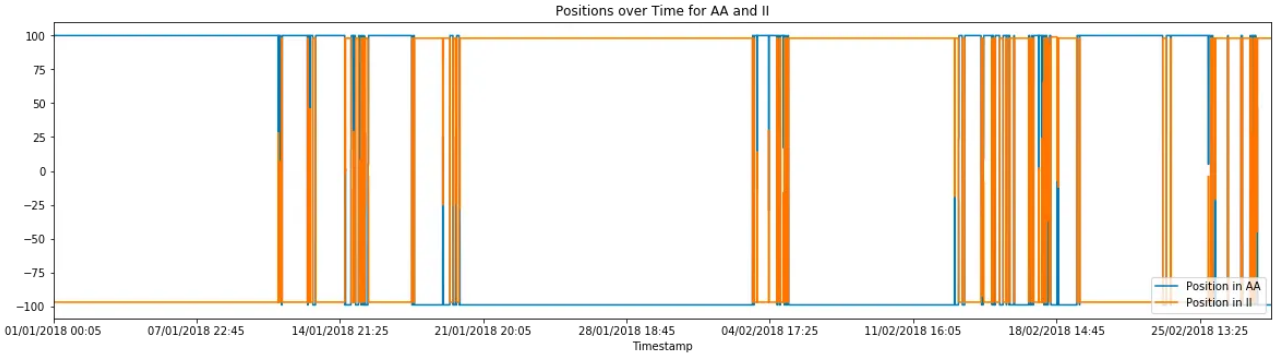
By assigning the final value of position_strategy_1 to the final value of position_diff_strategy_1, this code creates the differences between the two. It replaces the last item in positions_diff_strategy_1 with the last item in positions_strategy_1. In situations where you need to match the last value in two lists, this could be useful for updating or synchronizing information.

```
# Show Positions over Time
for pairs in stock_pairs_final:
    stock1 = pairs[0]
    stock2 = pairs[1]

    plt.figure(figsize=(20,5))

    positions_strategy_1[stock1].plot()
    positions_strategy_1[stock2].plot()

    plt.title('Positions over Time for ' + stock1 + ' and ' +
stock2)
    plt.legend(["Position in " + stock1,"Position in " +
stock2], loc='lower right')

    plt.show()
```









It loops through a collection of stocks, retrieving the first and second items for each pair. Furthermore, it creates a 20 by 5 figure. In addition, it plots the positions of the first and second stock over time in the pair. It then adds a title to the figure with the names of the two stocks and a legend indicating which line represents which stock. Lastly, it displays the figure. Each pair of stocks in the collection goes through this process.

Re-run the PnL forloop to determine the profits. This could also be manually calculated with pnl_threshold values.

```
pnl_dataframe_strategy_1 = pd.DataFrame()

for pair in stock_pairs_final:
```

```
    stock1 = pair[0]
    stock2 = pair[1]

    Stock1_AskPrice = tradable_pairs_data[stock1,'AskPrice']
[1:]
    Stock1_BidPrice = tradable_pairs_data[stock1,'BidPrice']
[1:]
    Stock2_AskPrice = tradable_pairs_data[stock2,'AskPrice']
[1:]
    Stock2_BidPrice = tradable_pairs_data[stock2,'BidPrice']
[1:]

    pnl_dataframe_strategy_1[stock1] =
np.where(positions_diff_strategy_1[stock1] > 0,
positions_diff_strategy_1[stock1] * -Stock1_BidPrice,
positions_diff_strategy_1[stock1] * -Stock1_AskPrice)
    pnl_dataframe_strategy_1[stock2] =
np.where(positions_diff_strategy_1[stock2] > 0,
positions_diff_strategy_1[stock2] * -Stock2_BidPrice,
positions_diff_strategy_1[stock2] * -Stock2_AskPrice)

print("The total profit is:
€",round(pnl_dataframe_strategy_1.sum().sum()))
```

```
The total profit is: € 135848.0
```

With this code, a dataframe called pnl_dataframe_strategy_1 will be created and populated with profit data for multiple stock pairs. The script loops through a list of stock pairs and retrieves ask and bid prices for each stock. A dataframe is created using these prices and other data to calculate each stock's profit. Lastly, the line prints the total profit in euros for all stock pairs.

```
pnl_dataframe_strategy_1['Timestamp'] =
tradable_pairs_data.index[1:]
pnl_dataframe_strategy_1 =
pnl_dataframe_strategy_1.set_index('Timestamp')

pnl_dataframe_strategy_1['PnL'] =
pnl_dataframe_strategy_1.sum(axis=1)
pnl_dataframe_strategy_1['Cum PnL'] =
pnl_dataframe_strategy_1['PnL'].cumsum()
```

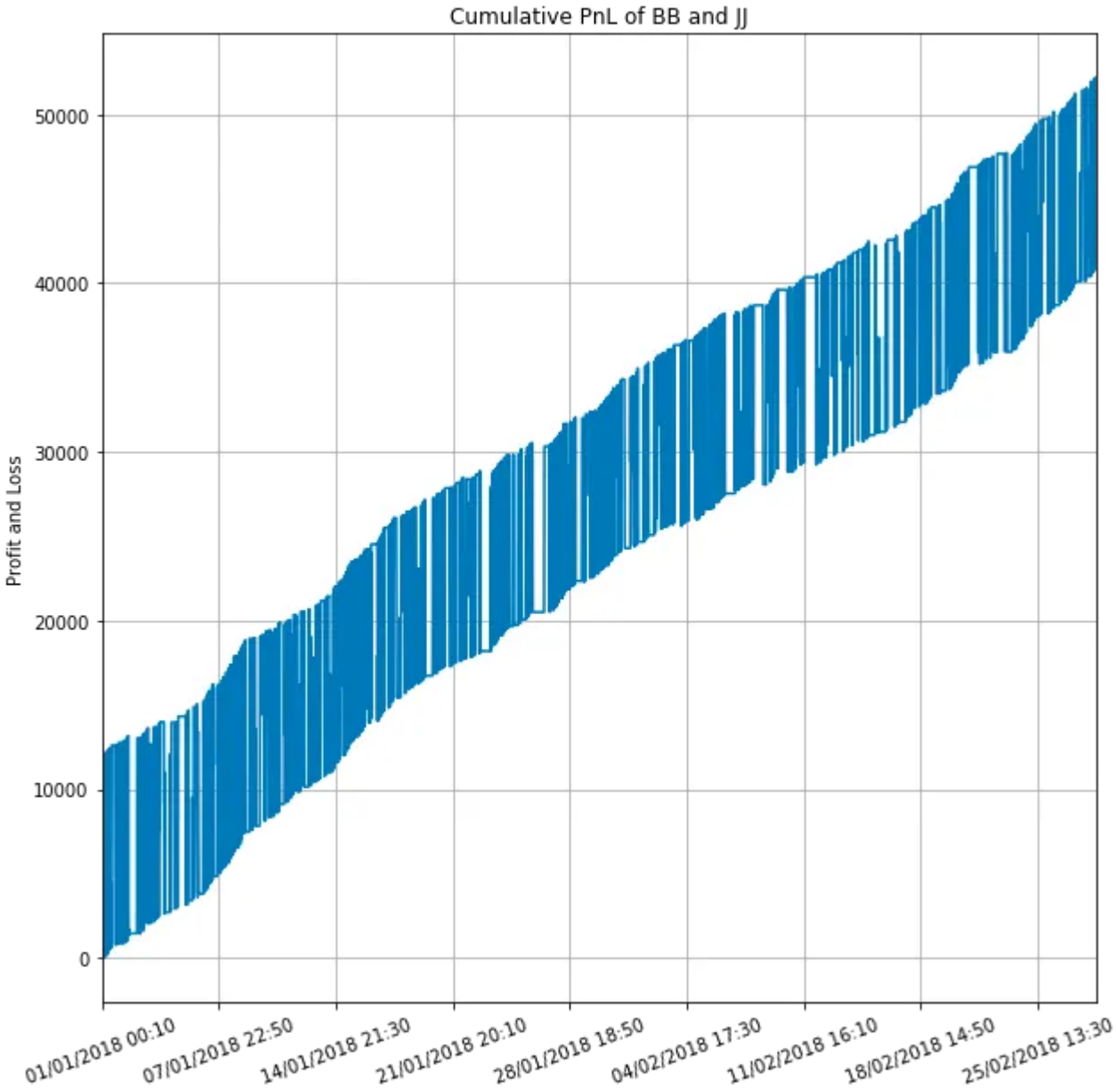
```
for pair in stock_pairs_final:
    stock1 = pair[0]
    stock2 = pair[1]

    pnl_dataframe_strategy_1[stock1+stock2 + ' PnL'] =
pnl_dataframe_strategy_1[stock1] +
pnl_dataframe_strategy_1[stock2]
    pnl_dataframe_strategy_1[stock1+stock2 + ' Cum PnL'] =
pnl_dataframe_strategy_1[stock1+stock2 + ' PnL'].cumsum()

pnl_dataframe_strategy_1.tail()
```

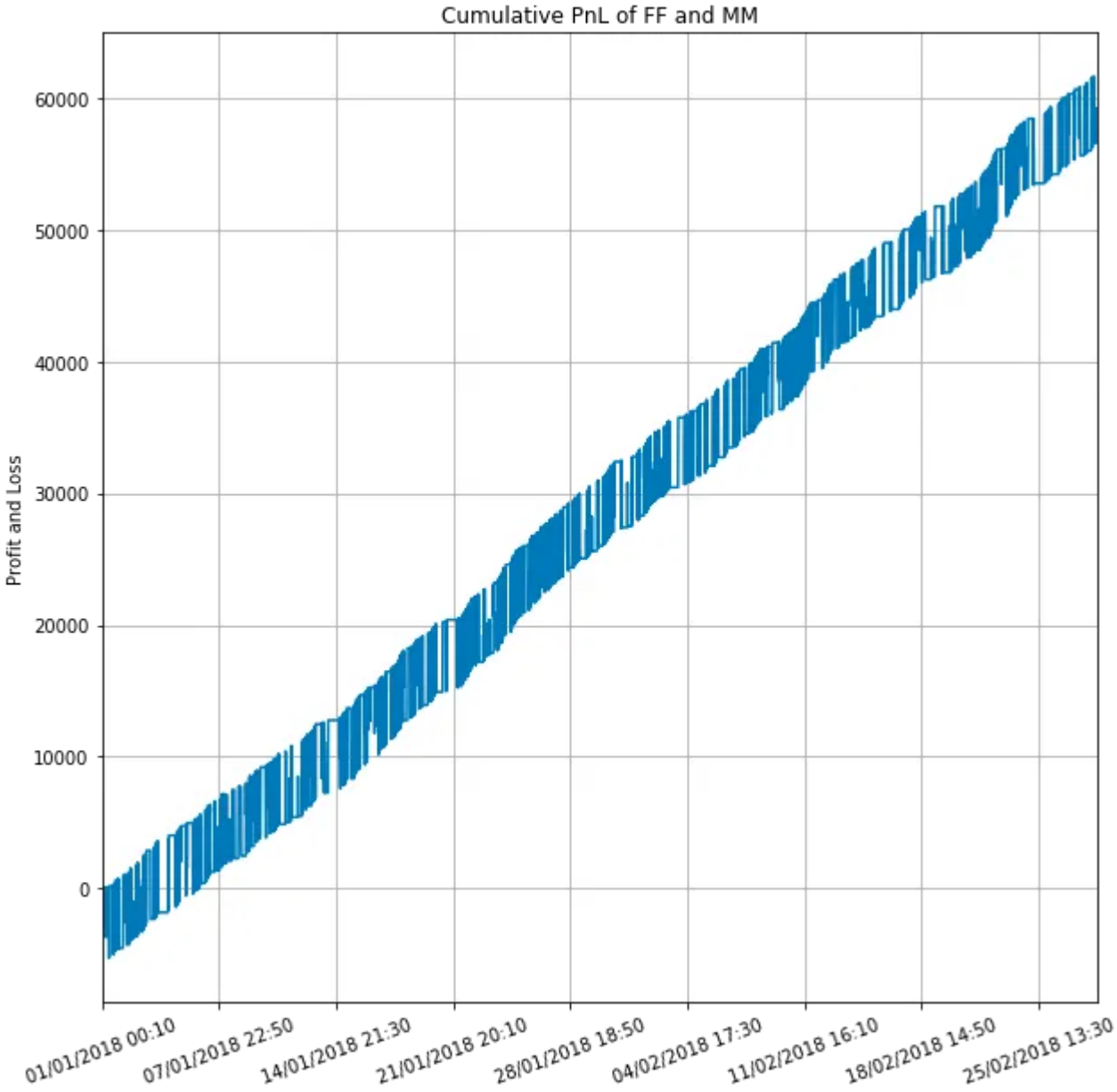
	BB	JJ	FF	MM	DD	HH	AA	II	PnL	Cum PnL	BBJJ PnL	BBJJ Cum PnL	FFMM PnL	FFMM Cum PnL	DDHH PnL	DDHH Cum PnL	AAII PnL	AAII Cum PnL
Timestamp																		
28/02/2018 23:35	-7181.25	2861.95	-0.0	-0.00	-0.0	-0.0	-0.0	-0.0	-4319.30	124462.35	-4319.30	40830.20	-0.00	56565.05	-0.0	16309.3	-0.0	10757.8
28/02/2018 23:40	14880.00	-5932.85	-0.0	-0.00	-0.0	-0.0	-0.0	-0.0	8947.15	133409.50	8947.15	49777.35	-0.00	56565.05	-0.0	16309.3	-0.0	10757.8
28/02/2018 23:45	4032.00	-1615.95	-0.0	-0.00	-0.0	-0.0	-0.0	-0.0	2416.05	135825.55	2416.05	52193.40	-0.00	56565.05	-0.0	16309.3	-0.0	10757.8
28/02/2018 23:50	-0.00	-0.00	-0.0	-0.00	-0.0	-0.0	-0.0	-0.0	0.00	135825.55	-0.00	52193.40	-0.00	56565.05	-0.0	16309.3	-0.0	10757.8
28/02/2018 23:55	-9383.50	3862.50	10474.2	-7851.75	-11127.6	14371.2	-9187.2	8864.1	21.95	135847.50	-5521.00	46672.40	2622.45	59187.50	3243.6	19552.9	-323.1	10434.7

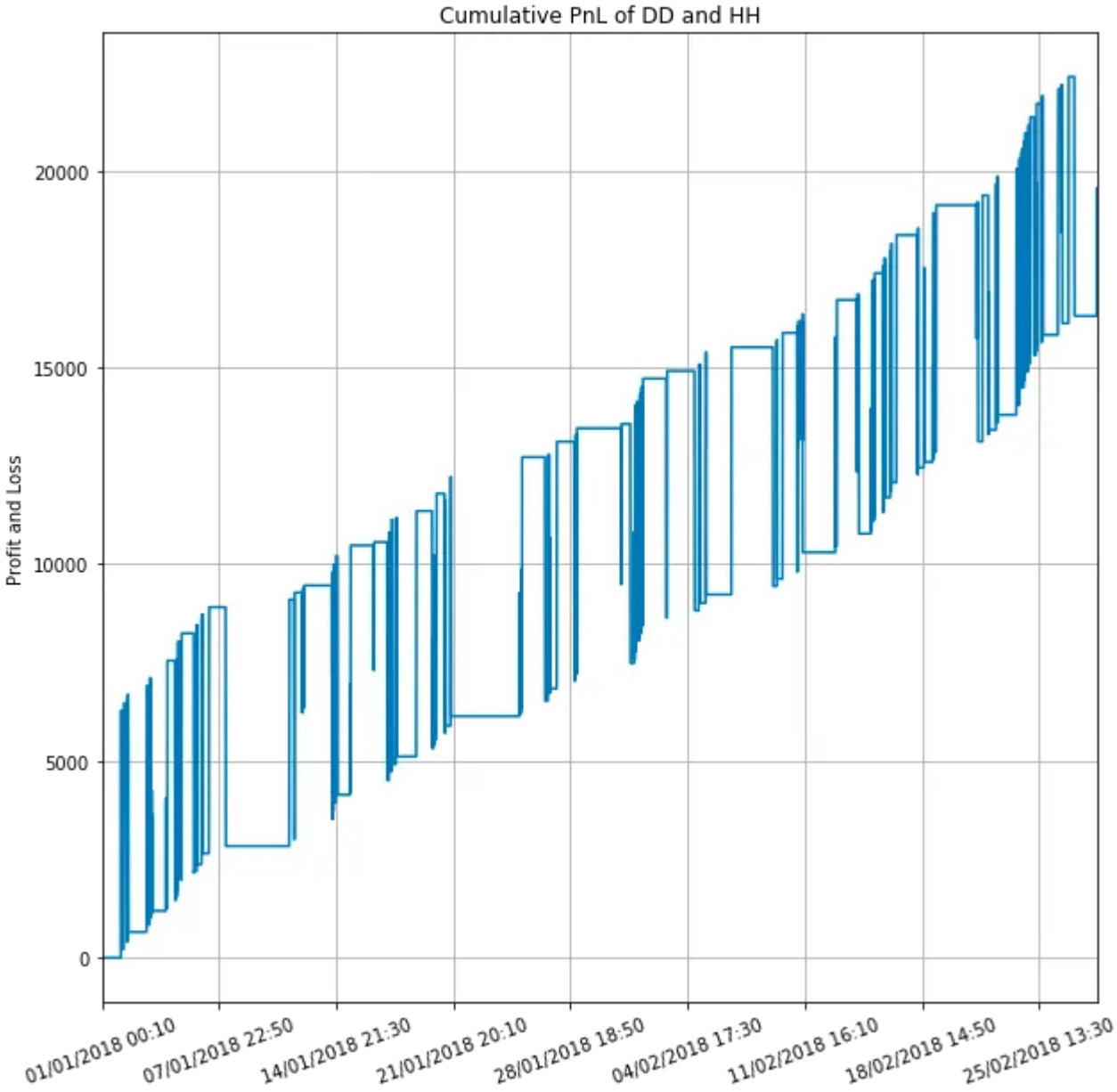
In this we create a profit-and-loss dataframe for a pair trading strategy. First, it extracts the timestamp from the tradable pairs data and uses it as the dataframe's index. A strategy's total PnL is calculated by summing the PnLs of each pair. Additionally, it calculates the cumulative PnL. Following that, it sums the PnL of each individual stock in each pair to calculate the PnL for each pair. Afterwards, it calculates the pair-wise cumulative PnL. In the last few rows of the PnL dataframe, it displays the last few rows of the dataframe. The code aims to track the profitability of a trading strategy involving pairs of stocks.

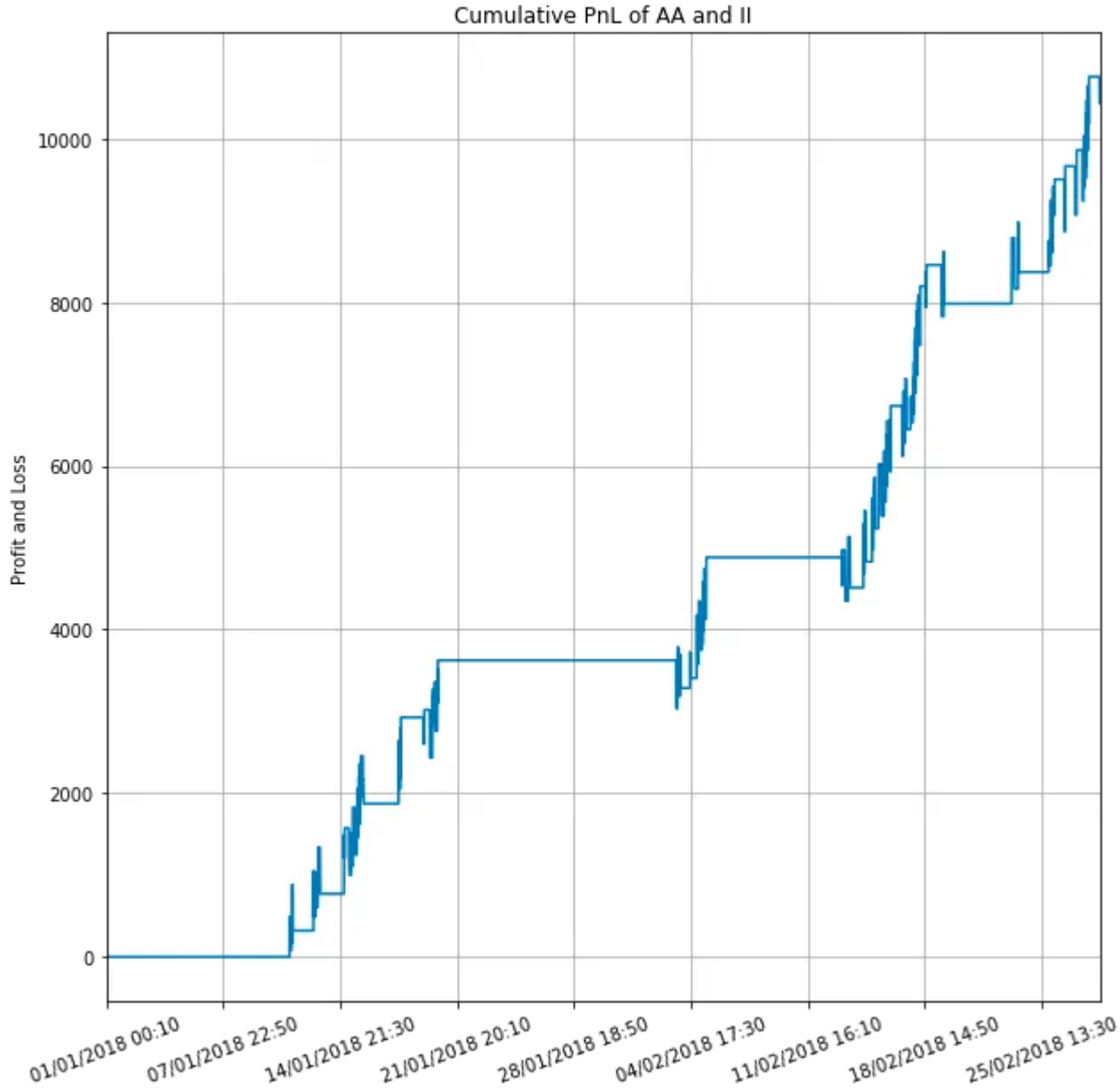
```
# All Pairs's PnL

for pair in stock_pairs_final:
    stock1 = pair[0]
    stock2 = pair[1]

    pnl_dataframe_strategy_1[stock1+stock2 + ' Cum
PnL'].plot(figsize=(10,10))
    plt.title('Cumulative PnL of ' + stock1 + ' and ' + stock2)
    plt.ylabel('Profit and Loss')
    plt.xlabel("")
    plt.grid()
```

```
plt.xticks(rotation=20)
plt.show()
```




Cumulative PnL of FF and MM
Profit and Loss
60000
50000
40000
30000
20000
10000
0
01/01/2018 00:10
07/01/2018 22:50
14/01/2018 21:30
21/01/2018 20:10
28/01/2018 18:50
04/02/2018 17:30
11/02/2018 16:10
18/02/2018 14:50
25/02/2018 13:30

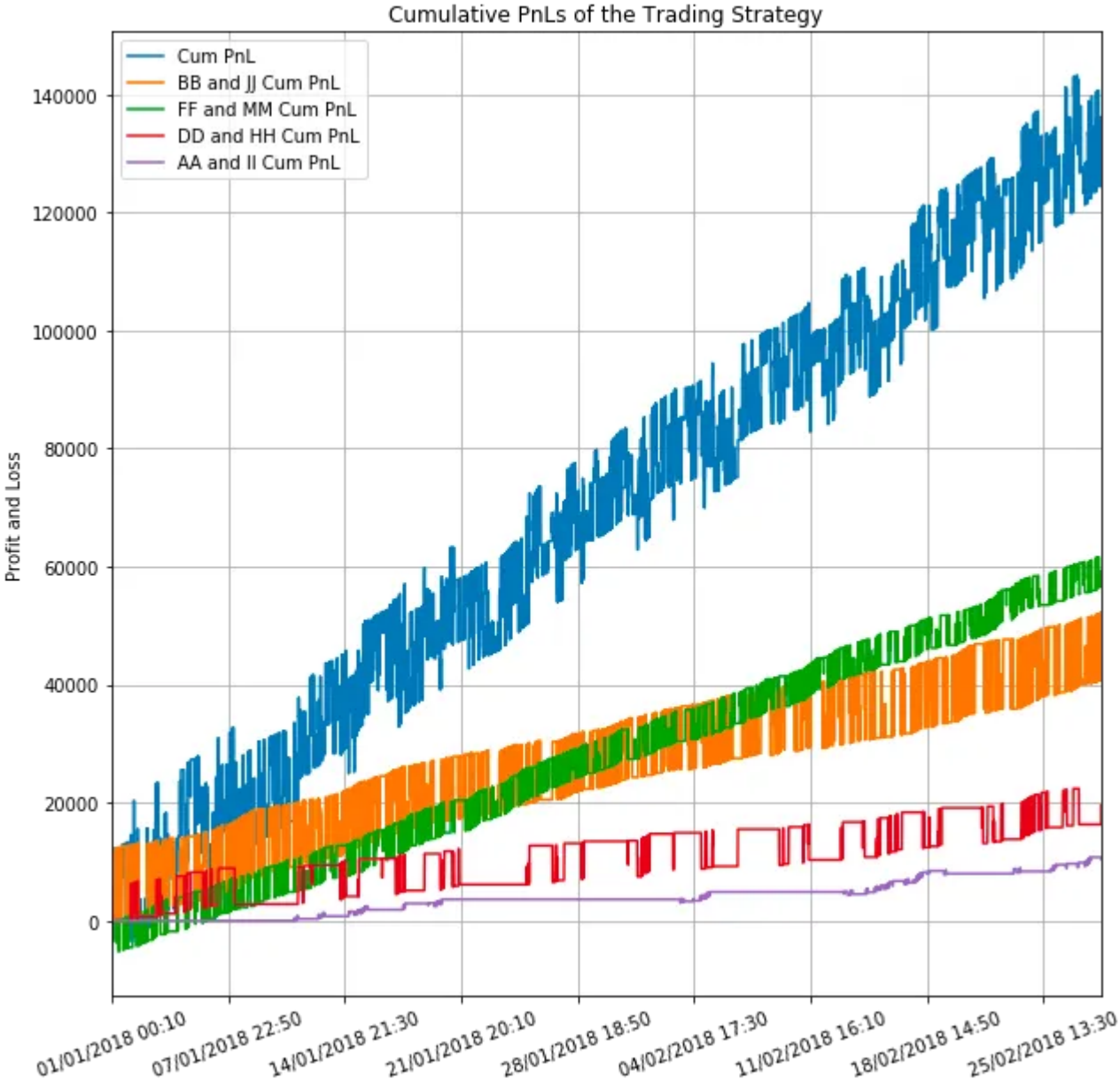

Cumulative PnL of DD and HH
Profit and Loss
20000
15000
10000
5000
0
01/01/2018 00:10
07/01/2018 22:50
14/01/2018 21:30
21/01/2018 20:10
28/01/2018 18:50
04/02/2018 17:30
11/02/2018 16:10
18/02/2018 14:50
25/02/2018 13:30



The code plots a cumulative profit and loss graph of stock pairs, with the x-axis representing the time frame and the y-axis representing the profit and loss. Looping through a list of stock pair combinations and assigning the first stock to the stock1 variable and the second stock to the stock2 variable. Using these variables, the graph title and axis labels will be generated. It is easier to read the labels on the x-axis with the grid added for better visual representation. The graph is shown at the end.

```
# All Pairs's PnLs (including total) in one graph

pnl_dataframe_strategy_1['Cum PnL'].plot()

for pair in stock_pairs_final:
```

```
    stock1 = pair[0]
    stock2 = pair[1]

    pnl_dataframe_strategy_1[stock1+stock2 + ' Cum
PnL'].plot(figsize=(10,10))
    plt.legend(['Cum PnL', 'BB and JJ Cum PnL', 'FF and MM Cum
PnL', 'DD and HH Cum PnL','AA and II Cum PnL'])
    plt.title('Cumulative PnLs of the Trading Strategy')
    plt.ylabel('Profit and Loss')
    plt.xlabel("")
    plt.grid()
    plt.xticks(rotation=20)
```



Each pair of stocks is assigned to a stock1 and stock2 variable within the loop. After calculating these variables, it plots the cumulative PnL for each stock pair. It is given a title and axes labels, and it is visualized with a grid. Rotating the x-axis labels improves readability.

That's it folks see you In the next one.

7 Comments

Write a comment...

Onepagecode Oct 27, 2023 **Author**

Download the code from here: http://onepagecode.s3-website-us-east-1.amazonaws.com/

Download the jupyter notebook from here: http://onepagecode.s3-website-us-east-1.amazonaws.com/Datasets/Pairs-trading-27-october-2023-8-41-pm.ipynb

LIKE REPLY SHARE

5 replies

Leia Dec 12, 2023

Where's the "Pairs Trading.csv"? and where's the "cointegration_analysis" module?

LIKE REPLY SHARE

5 more comments...