User manual for Pair Trading

Key Takeaways

Pairs Trading Python (Strategy):

- This is a short-term strategy where we buy and sell two related things at the same time.
- We do it when their prices don't match, and we want to make a profit.

The Law of One Price:

- It says that similar things should cost the same in a fair market.
- But sometimes, things can temporarily deviate.

Data for JPMorgan and Bank of America:

- We're looking at information about two big companies.
- We're checking how their money and prices have changed over time.

Cointegration:

- We're checking if two things are connected for a long time.
- It helps us understand if their prices are related.

The Z-score and Signal:

- We use numbers to figure out if prices are acting strange.
- When the numbers go too high or too low, we know something's up.

Entry and Exit Points:

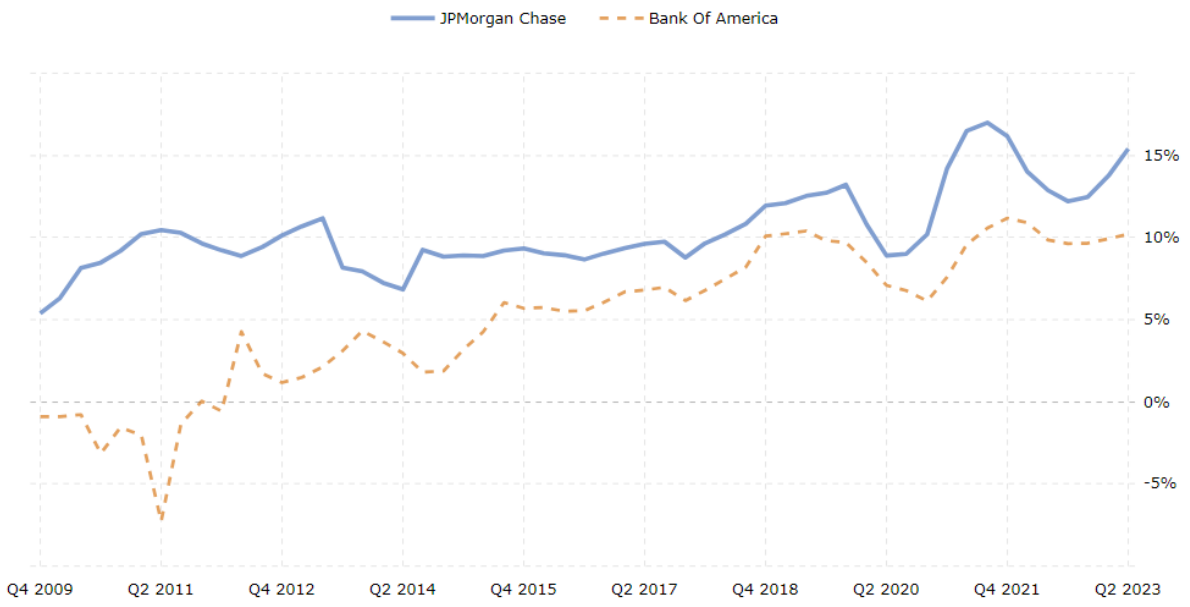
- We decide when to start and stop buying and selling based on those numbers.

Backtesting:

- We look at how well our plan would have worked in the past to see if it's a good idea.

Data Visualization of JPMorgan Chase (JPM) and Bank of America (BAC)

This tutorial will use the financial instruments of JPMorgan Chase (JPM) and Bank of America (BAC). First, let's see the Return On Equity (ROE):



We can see how the ROE between both companies follows a similar historical pattern. These two institutions are between the largest banks in the U.S. and compete in many areas.

1.Let's see their historical price evolution. First, let's import the Python libraries:

```
import statsmodels.api as sm
import pandas as pd
import numpy as np
import yfinance as yf
import matplotlib.pyplot as plt
from statsmodels.tsa.stattools import adfuller
import warnings
```

2. Second, download the data with **yfinance**. The start and end periods are "2015-01-01" and "2023-01-01":

```
#Download data
start = '2015-01-01'
end = "2023-01-01"

stock1 = yf.Ticker('BAC') #
stock1_data = stock1.history(interval='1d', start= start, end= end)

stock2 = yf.Ticker('JPM') #
stock2_data = stock2.history(interval='1d', start=start, end= end)
```

3.Then, we transform the time series , these variables will start from 100. We make this transformation to compare their historical prices

```
23
24  #Charting relative prices
25  stock1_close_relative =stock1_data["Close"]/stock1_data["Close"][0] * 100.
26  stock2_close_relative = stock2_data["Close"]/stock2_data["Close"][0] * 100.
27  plt.plot(stock1_close_relative, label = name_stock1)
28  plt.plot(stock2_close_relative, label = name_stock2)
29  plt.xlabel('Time')
30  plt.ylabel('Relative Close Price')
31  plt.legend()
32  plt.show()
33
```

Pair trading cointegration

implement Dickey-Fuller test

```
55
56  #Dickey Fuller Test
57  dftest = adfuller(errors, maxlag = 1)
58  dfoutput = pd.Series(dftest[0:4],
59                       index=["Test Statistic", "p-value", "#Lags Used", "Number of Observations Used",],
60                       )
61  critical_values = pd.Series(dftest[4].values(), index = dftest[4].keys())
62
63  print(f"Dikey Fuller Result:\n{dfoutput} \n\nDikey Fuller Critical Values:\n{critical_values}")
64
```

Z-score and signal for pair trading

```
66  #z-score
67  spread = errors
68  zscore = (spread - np.mean(spread)) / np.std(spread)
69  zscore.plot(label = "z-score")
70  plt.title(f"z-score {name_stock2} - {name_stock1}")
71  plt.xlabel('Time')
72  plt.ylabel('Values')
73  plt.axhline(y = 1.2, color = 'b', label = '1.2 threshold')
74  plt.axhline(y = -1.2, color = 'b', label = '-1.2 threshold')
75  plt.legend()
76  plt.show()
```

Pair trading rules

```
84  #Short Stocks
85  btest = pd.DataFrame()
86  btest["stock2"] = stock2_data["Close"]
87  btest["stock1"] = stock1_data["Close"]
88  btest["short signal"] = (zscore > signal_entry) & (zscore.shift(1) < signal_entry)
89  btest["short exit"] = (zscore < signal_exit) & (zscore.shift(1) > signal_exit)
90
91  btest["long signal"] = (zscore < -signal_entry) & (zscore.shift(1) > -signal_entry)
92  btest["long exit"] = (zscore > -signal_exit) & (zscore.shift(1) < -signal_exit)
```

Code to generate backtest

```
spread_side = None; counter = -1
backtest_result = []; indicator = 0
for time, signals_stock in btest.iterrows():
    counter+=1
    stock2_, stock1_, short_signal, short_exit, long_signal, long_exit = signals_stock

    if spread_side == None:
        return_stock2 = 0.
        return_stock1 = 0.
        backtest_result.append([time, return_stock2, return_stock1, spread_side])

        if short_signal == True:
            spread_side = "short"
        elif long_signal == True:
            spread_side = "long"

    elif spread_side == "long":
        return_stock2 = btest["stock2"][counter] / btest["stock2"][counter -1] -1.
        return_stock1 = btest["stock1"][counter] / btest["stock1"][counter -1] - 1.
        backtest_result.append([time, return_stock2, -return_stock1, spread_side])

        if long_exit == True:
            spread_side = None

    elif spread_side == "short":
        return_stock2 = btest["stock2"][counter] / btest["stock2"][counter -1] -1.
        return_stock1 = btest["stock1"][counter] / btest["stock1"][counter -1] - 1.
        backtest_result.append([time, -return_stock2, return_stock1, spread_side])

        if short_exit == True:
            spread_side = None
```