Project Part 3

Kai Wen Tay

Table of Content:

Introduction

Pairs Trading is one of the most fundamental market-neutral strategies used by hedge funds. It is particularly fun because it focuses on relative pricing. By buying a relatively undervalued security and selling a relatively overvalued one, a profit can be made upon the pair's price convergence. Here, we aim to get some clarity on pair trading strategies by retrieving the data for two stocks, calculating how well they "trend" with each other, and try to potentially profit off their spread differentials. Specifically, here, we concentrate on the two ETFs, SOXL and SOXX, with the idea being that long term fluctuations eventually return to the mean value for a leveraged ETF and its parent.

Market Data Retrieval

As can be seen on our Get Data ipynb file, we use the get_bars function from the alpaca_trade_api to set up our dataset for the two ETFs. Specifically, we use a fetch_data function, as documented, to fetch our data.

```
def fetch_data(symbol, start_date, end_date):
    try:
        data = api.get_bars(symbol, TimeFrame.Minute, start_date, end_date, limit=1000000).df
        return data
    except Exception as e:
        print(f"Error fetching data for {symbol}: {e}")
        return pd.DataFrame()
```

Then, for each symbol, we run the fetch data function to get the required amount of data for our timeframe (in this case, 6 months):

```
# Define the symbols and the date range for 3 months
symbols = ['SOXL', 'SOXX']
end_date = '2023-12-01'  # Current date
start_date = '2022-6-01'  # 6 months ago from the end date

# Fetch and store minute data for each symbol
for symbol in symbols:
    print(f"Fetching minute data for {symbol}...")
    minute_data = fetch_data(symbol, start_date, end_date)
    if not minute_data.empty:
        # Save to CSV
        csv_path = f"{path}/{symbol}_minute_data.csv"
        minute_data.to_csv(csv_path)
        print(f"Data saved to {csv_path}")
    else:
        print(f"No data fetched for {symbol}")
```

We then store our data locally (in this case, we kept our data on GitHub), in csv format.

```
# combine the data into one dataframe (taking only close, and noting that timestamps might be different)
df = pd.DataFrame()
for symbol in symbols:
    csv_path = f"{path}/{symbol}_minute_data.csv"
    df_temp = pd.read_csv(csv_path, index_col='timestamp', parse_dates=True)
    df_temp = df_temp[['close']]
    df_temp.columns = [symbol]

    # Forward fill missing values
    df_temp = df_temp.ffill()

    df = pd.concat([df, df_temp], axis=1)

# Forward fill any remaining missing values in the combined DataFrame
df = df.ffill()
df.dropna(inplace = True)

# save the combined data
df.to_csv(f"{path}/minute_data.csv")
```

Trading Strategy Development

Using our minute by minute data, we want to backtest our trading strategy to find the optimal pair allocation to trade between the two. As per our introduction, given that SOXL is the 3X Leveraged version of SOXX, we assume that the difference between the two ETFs to be fixed, since the leveraged version should leverage the up or down moves to end up at some mean value similar to some scaled version of SOXX. Our backtest aims to address to key issue: (i) are the two pairs even cointegrated?, and (ii) what is the optimal allocation for our trading strategy.

To start, we took 6 months of data from our two ETFs and compared them:

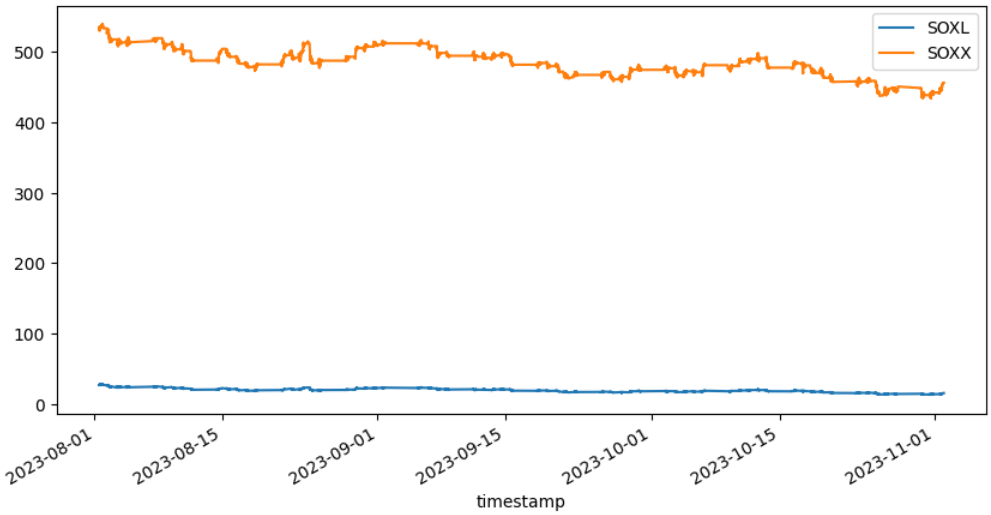

Figure 1: Price Graph of SOXL & SOXX

First, using statsmodel on Python, we regress the two underlyings against each other, with X being SOXL and Y being SOXX. Here, using that package, we arrive at a beta of 7.077. Visualising the spread, where:

$$Spread = Y - \beta X$$

We see that there appears to be some kind of bound our spread value sticks to. However, further tests will be conducted

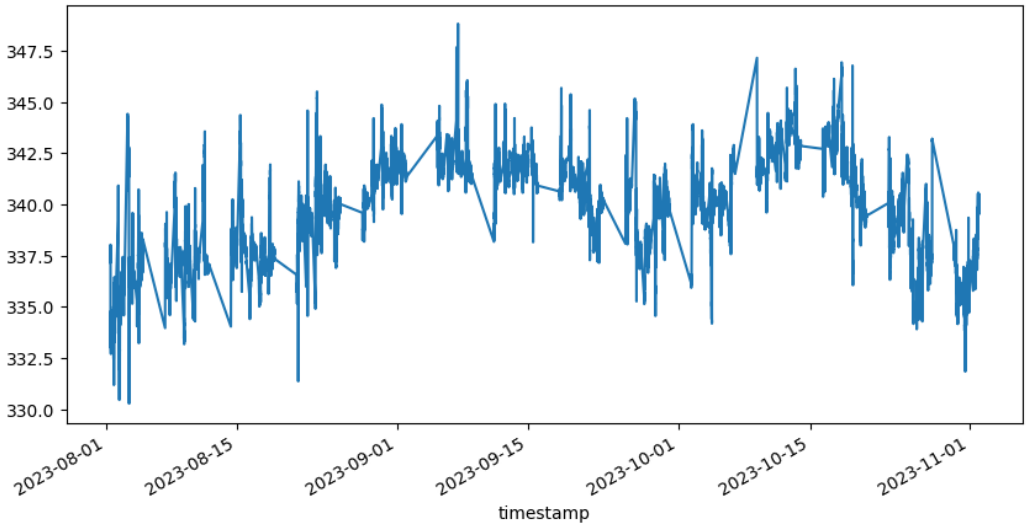

Figure 2: Spread Over Time of our Underlyings

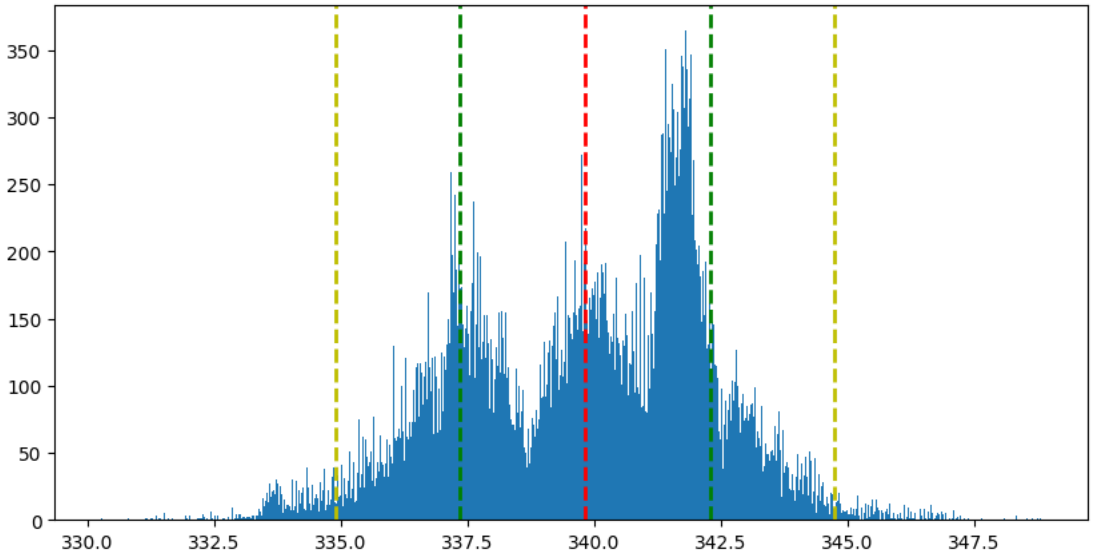

Figure 3: Distribution of Spread Values

Here, we apply the Augmented Dickey Fuller Test. It tests the null hypothesis that a unit root is present in a time series sample. In particular, our test statistic in this case is:

$$DF = \frac{\gamma}{SE(\gamma)}$$

Where γ is just the coefficient to the previous period's spread.

Specifically we want our test statistic, DF, to be as negative as possible. In this case, for our beta value of 7.077, we get a ADF test statistic of -11.53. For posterity, we plotted our DF statistics against beta values as well. We can see that our values minimise somewhere at 7.

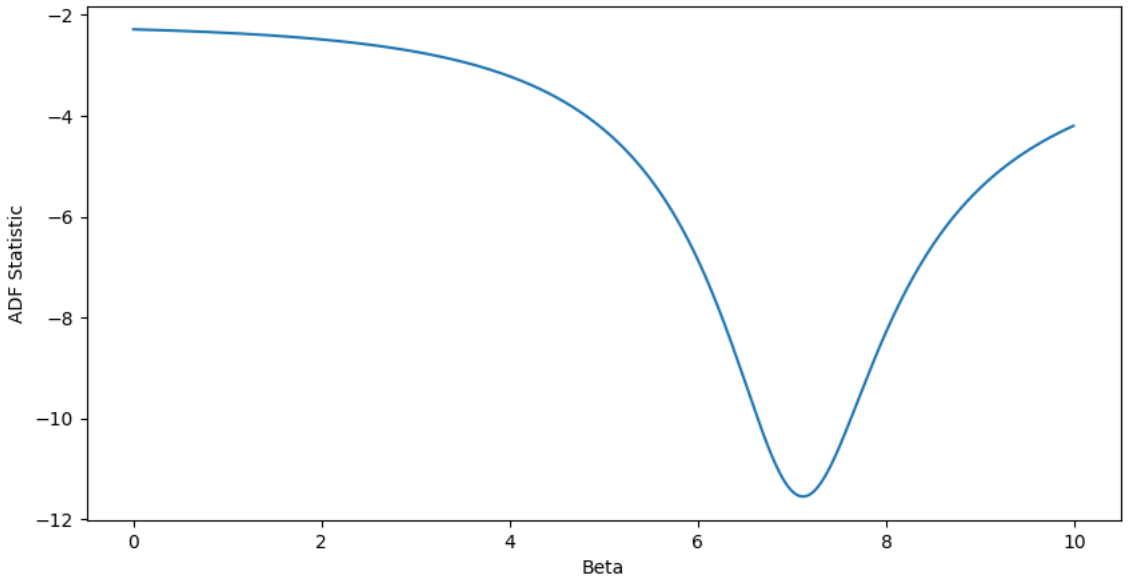

Figure 4: ADF Statistic vs Beta

In fact, our min ADF Statistic & Beta are -11.55 and 7.12 respectively. This beta value gives us a clear indication of a strong cointegration for this pair.

Code Explanation

We built 2 functions available in our Alpaca_Trading_Backtester file:

1. A trading strategy function to test our trading strategy and output our orderbook:
   This function does exactly like we outlined earlier. If we breach the spread, we store the position and append the latest order into our order book. Then we observe to see if the position should be closed. We also return a wealth time series for us to then map our performance, and our mean spread and standard deviation to do any additional analysis.

2. A performance summary function to return the summary statistics of our performance:
   This function is very simplistic. It returns performance statistics for our given set of returns, such as Sharpe, Volatility, VaR.

Testing & Optimisation

Taking our optimized beta value, we want to test it through our timeframe of data. Specifically, we want implement a strategy that:

- Long β units of X and short Y when the spread becomes too large
- Short β units of X and long Y when the spread is too small

We then close the positions whenever the spread has reverted to their mean.

Some pointers to note:

1. For ease of risk management, we stick to only one open position at any one point in time and keep at it until we close the last opened position
2. We size our position to some value of our wealth, specifically:

$$Multiplier = \frac{\text{Max Portfolio Threshold}}{|Y - \beta X|}$$

   This helps us overspend our money. Of course, we can set the Max Portfolio Threshold to what we like but in this case we set it at 50% of the last know portfolio value
3. Our threshold is half a standard deviation (i.e. we trade whenever we are above or below the 0.5 standard deviation threshold). We tested a few values and did not find marginal increase in benefits to increase our standard deviation threshold.

Using a rounded $\beta = 7$ for ease of units sizing, we test based on the timeframe of data we have using a wealth starting at USD100,000, we can see that:

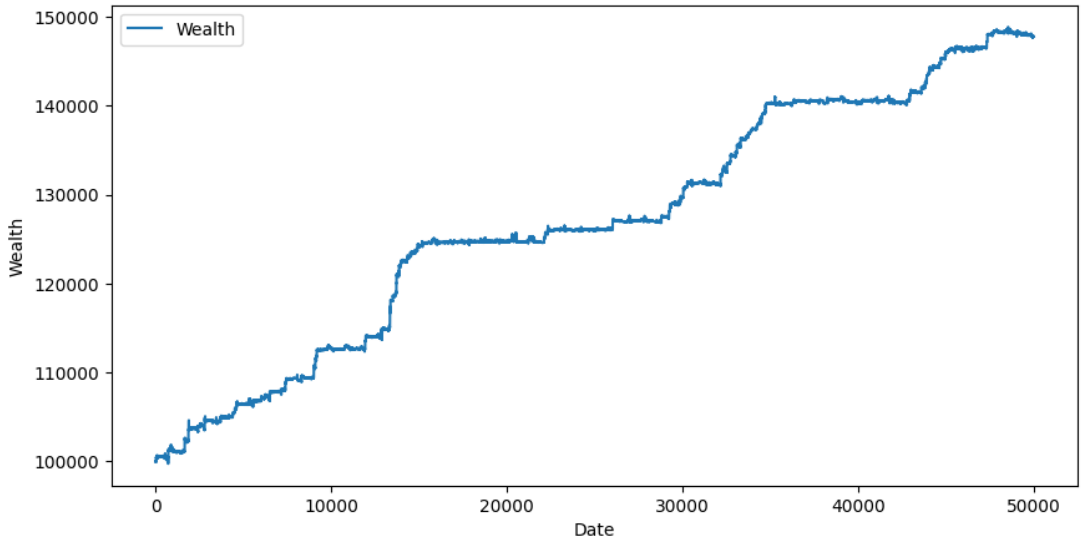

Figure 5: Wealth Over Trading Time

Analysing the returns annualized, we get:

Mean	Volatility	Sharpe Ratio	Skewness	Excess Kurtosis
77.48%	11.12%	6.93	9.08	323.06

Automation & Scheduling

This is the breezy part. Given our strategy's fixed beta and trading values, we just need to maintain an orderbook and a mark to market to know how many units we should hit. In our code's case, we reuse the tradeapi module, and took reference from the originally provided code to implement our trading strategy. We also took reference from our backtester code. We did not really have to maintain anything else other than an orderbook. All the code is available in Alpaca_Trading_Executor. For our orders, we implement limit orders that are 1% away from our intended prices so we avoid buying into sudden falls or spikes.

Results & Lessons Learnt

Pair trading is a wonderful strategy to mitigate risk and trade on asymmetric pricings. However, there are some key risks to take note of. We have too short a timeframe of testing and should ideally lengthen it if possible. Also, we should not restrict ourselves to one specific pair. An idea strategy would be to test a portfolio of pairs, allocating a specific statistic to our pairs, and allocating a specific portfolio weight to each pair based on the pair. This would promote diversification while also allowing us to put on more positions, potentially giving us a less skewed result. The other consideration here is the need to have better ways to store the data if our strategy does come to pass.

Compliance & Legal Considerations

The key consideration here is if we are trying "manipulating the market" by proxy, and forcing the market to stick a specific spread, although it has the wherewithal to move past this spread level.

Conclusion

This has honestly been the most fun and engaging part of the course. I chose a slightly harder and statistical method and struggled to implement the trading strategy slightly and backtest it, primarily due to the need to juggle contrasting positions.