Statistical Arbitrage on US Equities

Niranjan Jahagirdar, Shravan Tummala, Ziyang Zeng

July 29, 2024

1 Problem Statement

This project aims to study and implement the statistical arbitrage strategy proposed by Marco Avellaneda and Jeong-Hyun Lee in their 2008 paper, "Statistical Arbitrage in the U.S. Equities Market." Like Avellaneda and Lee, we hypothesized that stock prices (and returns) are influenced by market or sector factors, represented by Exchange-Traded Funds (ETFs), and that deviations from an equilibrium level can be modeled as a mean-reverting stationary process known as the spread. Our approach involved:

- Identifying cointegrated stock-ETF groups and estimating their spread over a specified trailing window, while statistically testing for stationarity.
- Fitting a mean-reverting model to the spread process for each stock to determine reversion speeds and equilibrium levels.
- Developing systematic trading signals to quantify deviations from equilibrium in the spread.

Then, by carefully timing our exit from these positions, we can profit from the reversion of the spread.

Ultimately, our project culminated in a strategy which attained the following results in the set of 2020-2023: Annual return = 13.44%, Sharpe ratio = 1.9754785548941465, max-

imum drawdown = -6.6%. Our inclusion of a stationarity test for the spreads, correlation filter and bucketing of z scores helped us improve our signal.

2 Basics of Statistical Arbitrage

2.1 Intuition

Statistical arbitrage is a well-known concept in finance, involving the exploitation of price discrepancies between related financial instruments. The basic intuition behind this strategy can be illustrated with a simple example: consider a hypothetical universe with only one stock, S, issued by an oil and gas company. The price of this stock is closely tied to a single factor, F, which represents oil prices. When oil prices rise due to increased global demand, investors generally expect the stock price S to also increase, reflecting the anticipated growth in shareholder equity and the enhanced perceived value of each share. However, there may be instances where the stock price S does not move in line with the change in oil prices, rising much less or more than expected. This discrepancy can occur because stock prices are influenced not only by quantifiable factors like oil prices but also by other unobservable or idiosyncratic factors.

In situations where the stock price deviates from what is predicted by observable factors, the difference can often be attributed to idiosyncratic residuals, which are the part of stock returns not explained by common factors. If we assume that stock prices eventually revert to some average or mean value, these deviations present a trading opportunity. The fundamental idea is to exploit the gap between the current stock price and its fair value, which should theoretically be influenced by factors such as oil prices. By going long on stocks that are underpriced (buying shares when the price is lower than expected) or short on stocks that are overpriced (selling shares when the price is higher than expected), traders can potentially profit from the eventual convergence of the stock price to its mean value.

In a more complex scenario involving multiple stocks, the goal remains to earn a profit from these spreads. In this context, a "pair" consists of a stock and its relevant factors,

and the objective is to maximize earnings from the difference between each stock's actual price and the price forecasted by its factors. To implement this strategy, one first identifies stocks with quick mean reversion times, using this characteristic as an initial filter. This is followed by calculating a dimensionless "signal" that scales with deviations from the mean. A large signal magnitude indicates that the stock price has significantly deviated from its expected value, and, according to the mean reversion theory, the price should eventually return to this mean.

The trading strategy involves using signal cutoffs to decide when to enter or exit positions. Specifically, these signals determine whether to open a long position (buy the stock) or a short position (sell the stock), depending on whether the stock is undervalued or overvalued relative to its predicted fair value. The process also includes criteria for exiting these positions, either by selling the stock in a long position or buying back the stock in a short position. This systematic approach allows traders to capitalize on the reversion of stock prices to their mean, exploiting temporary inefficiencies in the market to generate profits.

2.2 Pairs Trading

Pairs trading is a market-neutral strategy that exploits the historical price relationship between two correlated assets, such as stocks within the same industry. The core idea is based on the assumption that while the prices of these assets may move together, temporary deviations from their typical relationship can occur. For example, if two stocks that usually move in tandem suddenly diverge in price—due to, say, one outperforming the other without any fundamental reason—this divergence creates an opportunity. A trader can go long (buy) the underperforming stock and short (sell) the outperforming one, betting that the prices will eventually converge back to their historical norm. This strategy profits from the relative price movement rather than the direction of the market, making it resilient in both rising and falling markets. The success of pairs trading relies on identifying pairs that have a strong historical correlation and are likely to revert to their mean price relationship.

2.3 Co-integration

Co-integration is a more refined statistical technique that is particularly useful in pairs trading, as it helps identify pairs of financial instruments whose price movements are not just correlated, but also maintain a stable long-term relationship. While correlation measures the degree to which two securities move together, co-integration specifically focuses on the stability of their price relationship over time.

Technically, two time series X_t and Y_t are said to be co-integrated if there exists a linear combination of these series that is stationary, even if the individual series themselves are non-stationary. The concept of stationarity implies that the mean and variance of the series do not change over time, and there is no trend. In pairs trading, this stationary linear combination is often referred to as the "spread" between the two assets, calculated as $Z_t = X_t - \beta Y_t$, where β is a constant co-integration coefficient.

The statistical tools used to test for co-integration include the Engle-Granger two-step method and the Johansen test. The Engle-Granger method first tests the residuals from a regression of X_t on Y_t for stationarity, using tests like the Augmented Dickey-Fuller (ADF) test. If the residuals are found to be stationary, the series are considered co-integrated. The Johansen test, on the other hand, uses a maximum likelihood approach to test for co-integration and can identify multiple co-integrating relationships if they exist.

In practice, a co-integrated pair is ideal for pairs trading because the stationarity of the spread implies that any divergence from the equilibrium is temporary, making it a predictable source of profit as the prices revert. This predictability enhances the robustness of the pairs trading strategy, providing a statistical basis for expecting mean reversion and thus making the strategy more reliable.

3 Avellanada's Analysis

3.1 ETF Approach

The paper utilizes a set of sector ETFs to represent various industry sectors in the U.S. market. These ETFs serve as benchmarks against which individual stock returns are regressed. For each stock, the return is regressed against the return of the corresponding sector ETF:

$$R_i = \alpha + \beta R_{\text{ETF}} + \epsilon_i,$$

where R_i is the return of stock i, R_{ETF} is the return of the sector ETF, α and β are regression coefficients, and ϵ_i represents the residual or idiosyncratic component.

The residual ϵ_i is analyzed for mean-reversion characteristics. Stocks are considered for trading when their residuals deviate significantly from the mean, signaling potential mispricing relative to the sector. Trading signals are generated based on the residuals.

For example, if a stock's residual is high, indicating overpricing relative to the sector, a short position may be taken on the stock. Conversely, a low residual may signal an opportunity to go long. To maintain market neutrality, the strategy often involves hedging the portfolio against overall market movements, typically using the S&P 500 index or a broad market ETF.

The paper reports that ETF-based strategies achieved an average annual Sharpe ratio of around 1.1 from 1997 to 2007, with a noticeable decline in performance after 2002. The introduction of trading volume information improved the Sharpe ratio, suggesting that volume-based adjustments could enhance the performance of ETF-based statistical arbitrage.

3.2 PCA Approach

The method starts with a matrix of historical returns for a large number of stocks. The data is standardized to account for differences in volatility across stocks. A correlation matrix is constructed from the standardized returns, capturing the co-movement of stocks.

The correlation matrix is decomposed into eigenvalues and eigenvectors. The eigenvalues represent the amount of variance explained by each principal component, while the eigenvectors indicate the composition of each component in terms of individual stocks. A certain number of principal components (factors) are selected based on their explanatory power. The top factors, which explain the most variance, are retained for further analysis. Each stock's return is decomposed into a portion explained by the selected factors and a residual component.

The residuals are then analyzed for mean-reversion behavior. Similar to the ETF approach, trading signals are generated based on the deviations of the residuals from their historical mean. This approach can involve going long or short on stocks based on their residuals.

The PCA-based strategies were found to have an average annual Sharpe ratio of 1.44 over the period 1997 to 2007. However, there was a marked decline in performance post-2003, with the Sharpe ratio dropping to around 0.9. PCA strategies outperformed ETF-based strategies, particularly in volatile market conditions. This is likely due to PCA's ability to capture a broader range of systematic factors, providing a more nuanced understanding of the residuals. The variability in the number of significant eigenvalues over time suggests that the market's underlying factor structure can change, which has implications for the robustness and adaptability of PCA-based strategies.

3.3 Residual Analysis

In statistical arbitrage, residual modelling involves isolating the idiosyncratic component of a stock's return, which is not explained by the systematic factors. This residual component is crucial as it forms the basis for identifying potential trading opportunities. The Ornstein-Uhlenbeck (O-U) process is commonly used to model these residuals due to its mean-reverting properties.

The O-U process is a type of stochastic differential equation (SDE) that describes the evolution of a variable X_t over time. The general form of the O-U process is given by:

$$dX_t = \kappa(\mu - X_t)dt + \sigma dW_t,$$

where:

- X_t is the state variable representing the residual at time t,
- κ is the rate of mean reversion,
- μ is the long-term mean of the process,
- σ is the volatility parameter, and
- W_t is a Wiener process or Brownian motion.

The term $\kappa(\mu - X_t)$ represents the tendency of X_t to revert to its mean μ, with the speed of reversion governed by κ. The stochastic term σdW_t introduces randomness into the process, capturing the unpredictable fluctuations.

To apply the O-U process in residual modelling, it is necessary to estimate the parameters κ, μ, and σ. This is typically done using historical data, where the residuals X_t are calculated from the deviations of the stock's return from the predicted return based on systematic factors.

The discretized form of the O-U process can be represented as:

$$X_{t+1} = X_t e^{-\kappa \Delta t} + \mu(1 - e^{-\kappa \Delta t}) + \sigma \sqrt{1 - e^{-2\kappa \Delta t}}\, \epsilon_{t+1},$$

where Δt is the time increment, and ϵ_{t+1} is a standard normal random variable.

Using maximum likelihood estimation or least squares, the parameters can be estimated by fitting the model to the observed data. The conditional mean and variance of the process provide insights into the expected movement of the residuals, aiding in the formulation of trading strategies.

3.4 Signal Generation

Signal generation in statistical arbitrage involves identifying entry and exit points for trades based on the analysis of the residuals. The signals are derived from the deviations of the residuals from their expected values, leveraging the mean-reversion property of the O-U process.

A common approach is to standardize the residuals, converting them into a dimensionless z-score, which measures the number of standard deviations a residual is from its mean. The z-score Z_t for the residual X_t is calculated as:

$$Z_t = \frac{X_t - \mu}{\sigma_{\text{eq}}},$$

where $\sigma_{\text{eq}} = \frac{\sigma}{\sqrt{2\kappa}}$ is the equilibrium standard deviation of the O-U process.

The z-score provides a normalized measure of deviation, making it easier to set consistent thresholds across different stocks. For instance, a high positive z-score might indicate an overbought condition, suggesting a short position, while a high negative z-score might indicate an oversold condition, suggesting a long position.

Trading rules are based on the z-score thresholds. The specific thresholds used are:

- **Enter Long Position:** If $Z_t < -1.25$
- **Enter Short Position:** If $Z_t > 1.25$
- **Exit Long Position:** If $Z_t \geq -0.5$
- **Exit Short Position:** If $Z_t \leq 0.75$

These entry and exit thresholds are typically determined through backtesting and optimization, aiming to balance the frequency of trades with the profitability of each trade.

The strategy is designed to be market-neutral, with positions in individual stocks balanced by opposite positions in either the corresponding sector ETF (for the ETF approach) or synthetic hedges based on PCA-derived factors (for the PCA approach). This ensures

that the portfolio's overall exposure to market-wide movements is minimized, isolating the returns from the idiosyncratic movements captured by the residuals.

Advanced signal generation techniques might also incorporate trading volume or other market microstructure variables. For instance, signals might be weighted more heavily when volume is low, indicating less noise, or adjusted to account for liquidity constraints.

3.5 Performance Metrics

Cumulative Returns: Cumulative returns represent the total return generated by the strategy over the backtesting period. It is calculated by compounding the daily returns over the entire period. The formula is:

$$\text{Cumulative Return} = \left(\frac{\text{Ending Portfolio Value}}{\text{Starting Portfolio Value}} \right) - 1$$

Sharpe Ratio: The Sharpe Ratio measures the risk-adjusted return of the strategy. It is defined as the ratio of the average excess return over the risk-free rate to the standard deviation of the excess returns:

$$\text{Sharpe Ratio} = \frac{\mathbb{E}[R_p - R_f]}{\sigma_p}$$

where $\mathbb{E}[R_p]$ is the expected portfolio return, R_f is the risk-free rate, and σ_p is the standard deviation of the portfolio returns.

Information Ratio: The Information Ratio compares the strategy's excess return over a benchmark to the volatility of that excess return. It is calculated as:

$$\text{Information Ratio} = \frac{\mathbb{E}[R_p - R_b]}{\sigma_{R_p - R_b}}$$

where R_b is the return of the benchmark, and $\sigma_{R_p - R_b}$ is the tracking error or the standard deviation of the excess returns over the benchmark.

Maximum Drawdown: Maximum Drawdown (MDD) measures the largest peak-to-trough decline in the portfolio's value during the backtesting period. It is defined as:

$$\text{Maximum Drawdown} = \max_{t \in (0,T)} \left(\frac{P_t - P_{\text{peak}}}{P_{\text{peak}}} \right)$$

where P_t is the portfolio value at time t and P_{peak} is the maximum portfolio value observed prior to time t.

4 Data

4.1 Data Sources

1. **Individual Stocks in S&P 500:**
    - **Source:** Bloomberg
    - **Type of Data:** Daily price data, including Open, High, Low, Close, Adjusted Close, and Volume for each stock in the S&P 500.
2. **ETF Returns:**
    - **Source:** Same financial data providers.
    - **Type of Data:** Daily returns for various ETFs representing different sectors - IYR US Equity, IYT US Equity, OIH US Equity, RKH US Equity, RTH US Equity, SMH US Equity, VPU US Equity, XLE US Equity, XLF US Equity, XLI US Equity, XLK US Equity, XLP US Equity, XLV US Equity, XLY US Equity, XWEB US Equity

4.2 Data Preporcessing

Steps Taken to Clean and Preprocess the Data

1. **Mapping Equities to ETFs:**
    - Each underlying equity in the S&P 500 is mapped to its corresponding sector ETF to facilitate sector-based analysis.

2. **Data Cleaning:**

   - **Removing Duplicates:** Checked for and removed any duplicate entries to ensure the dataset is free of redundant data.
   - **Renaming Columns:** Standardized column names to maintain consistency across datasets from different sources.

3. **Z-score Transformation:**

   - Performed z-score transformation on the data to normalize the values. This transformation helps in future analysis by converting the data into a standard normal distribution (mean = 0, standard deviation = 1).

4. **Handling Missing Data:**

   - Forward fill method was used to handle missing data points. This involves filling in missing values with the last available data point to maintain continuity in the historical price data.

5 Implementation and Results

5.1 Methodology

In line with our belief that prices (and log returns) will eventually revert to their means, we can profit if we correctly identify periods where an empirical price deviates from its forecast, then take the appropriate position in the stock such that we will profit when the spread goes to zero. Note that for each stock, the forecast is made not for the purpose of profiting from its price in and of itself, but rather to compare the actual price to "what it should be" and thus profit off the gap.

But what should that forecasted stock price be? To answer this question, on every day T we run a regression for each stock i using a trailing 60-day data window:

$$R_{t,i} = \beta d_t + \sum_{j=1}^{m} \phi_{i,j} F_{t,j} + dX_t$$

where $R_{t,i}$ is the simple day-t return of the stock and $F_{t,j}$ the simple day-t return of the jth ETF, computed with respect to the day-(t-1) prices. The $\phi_{i,j}$'s are the coefficients for each ETF, representing the relative strength and direction of the relationship between the stock price and that ETF.

This regression decomposes a stock return into a drift term βd_t, some components correlated with various ETF returns, as well as a residual dX_t. Empirically, the returns of most stocks seem to have negligible drift, corroborating our belief in mean-stationarity and justifying our inattention towards βd_t. It is the residual which interests us, since it captures the stock's idiosyncratic fluctuations.

Hence, if we study dX_t's behavior, we can better understand the evolution of X_t, the stock's spread away from its equilibrium value.

We hypothesize, as many earlier researchers have, that for most stocks, the spread X_t indeed fluctuates around some equilibrium level, and any deviations are bounded. These are the key characteristics of stationary processes, so one natural model choice is the stationary, mean-reverting Ornstein-Uhlenbeck (OU) process, which has the differential form:

$$dX_t = \theta(m - X_t)dt + \sigma dW_t$$

By considering a transformation $f(X_t; t) = X_t e^{\theta t}$ and applying Itô's Lemma, the above stochastic differential equation can be transformed into:

$$df(X_t; t) = \left(\theta X_t e^{\theta t} + e^{\theta t}\theta(m - X_t) + \frac{1}{2}\sigma^2\right) dt + e^{\theta t}\sigma dW_t$$

$$= \left(\theta m e^{\theta t}\right) dt + \left(\sigma e^{\theta t}\right) dW_t$$

Given an initial condition $f(X_{t_0}; t_0) = X_{t_0} e^{\theta t_0}$, the above stochastic differential equation for df can be integrated from $t = t_0$ to T to arrive at an analytical expression for $f(X_T; T) =$

$X_T e^{\theta T}$ at any time T:

$$X_T e^{\theta T} = X_{t_0} e^{\theta t_0} + \int_{t_0}^{T} \theta m e^{\theta t} dt + \int_{t_0}^{T} \sigma e^{\theta t} dW_t$$

Setting $T = t_0 + \Delta t$, we arrive at:

$$X_{t_0+\Delta t} = a + bX_{t_0} + \epsilon$$

where $a = m(1 - e^{-\theta \Delta t})$, $b = e^{-\theta \Delta t}$, and $\epsilon \sim \text{Norm}\left(0, \sigma^2 \frac{1-e^{-2\theta \Delta t}}{2\theta}\right)$.

Clearly, this is an order-1 autoregression with parameters a and b. We can estimate a and b by regressing X_t onto its lag-1 series X_{t-1}. Then, using a and b, we can estimate the parameters of our mean-reverting spread model:

$$\theta = -\ln(b)/\Delta t$$

$$m = \frac{a}{1-b}$$

$$\sigma = \sqrt{V(\epsilon) \cdot 2\theta/(1-b^2)}$$

After we estimate θ, we selected stocks with $\theta > 252/30$ (indicating stocks whose mean reversion time constant $\tau = 1/\theta$ is predicted to not exceed 30 days) to estimate their signal at the current time-step (i.e. at day $n = 60$ of the estimation window) as a normalized deviation of the spread from its long-run mean:

$$s = \frac{X_n - E(X)_{eqm}}{V(X)_{eqm}} = \frac{X_n - m}{\sigma/\sqrt{2\theta}}$$

The mean reversion time filter ensures that we avoid having to hold positions for lengthy periods, which serves to manage the risk of our OU model assumptions and parameter estimation procedure becoming invalid before we close our positions. We also adopted Avellaneda's approach to correct for the finite-sample bias in the estimated long-run mean,

m_i, of each stock i by subtracting from each m_i the cross-sectional average of m over all stocks fitted to the OU model.

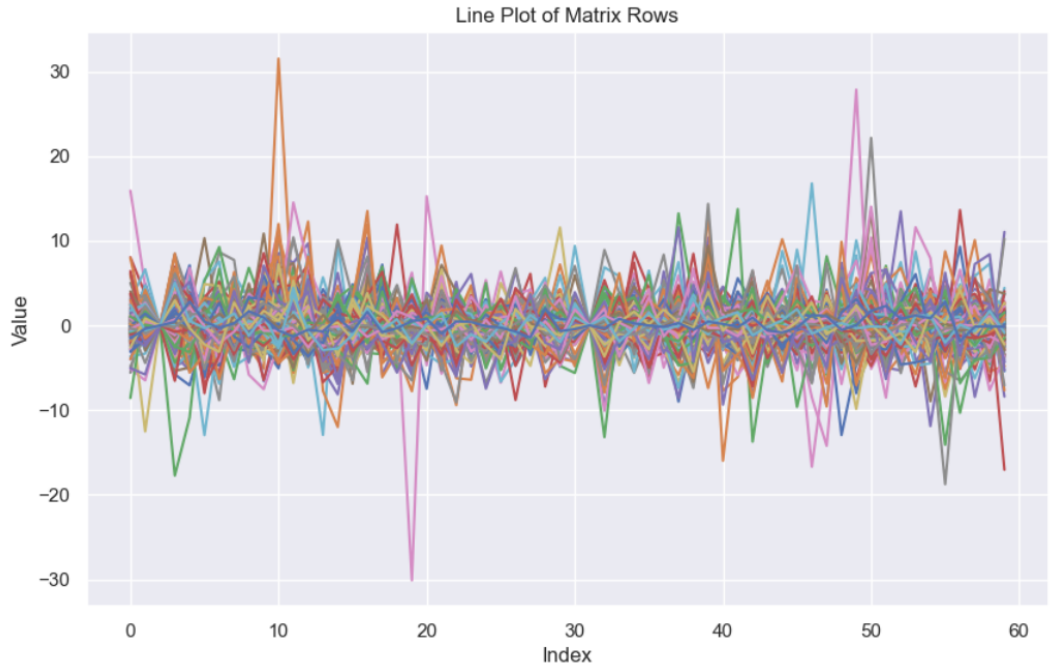


Figure 1: Residual Plots

5.2 Initial Attempt

To determine whether to take long or short positions on stocks, we calculated the normalized residual value, taking equal and opposite positions on the corresponding sector ETF. The following thresholds were applied to open or close positions:

- **Open Long Position on Stock**: $z\_score < -1.25$
- **Close Long Position on Stock**: $z\_score > -0.5$
- **Open Short Position on Stock**: $z\_score > 1.25$
- **Close Short Position on Stock**: $z\_score < 0.75$

We implemented a daily rebalancing strategy, where we checked and took equal-weighted long or short positions based on the signals.

Results: Using a daily rebalance frequency and daily returns, we ran a backtest for 3 years from 2021 to the end of 2023, with an estimation window of 60 days. The initial results over this 3-year period showed a return of -0.60% with a Sharpe ratio of -0.25, indicating suboptimal performance.

5.3 Improved Attempt

5.3.1 Filter Out Stocks

To ensure the robustness and reliability of our trading strategy, we applied several filtering criteria to the stocks under consideration:

1. **Remove Stocks with Missing Price Data**: Stocks with incomplete data can lead to inaccuracies in analysis and decision-making. We excluded any stocks that had gaps in their daily price records during the specified period, ensuring that our dataset contained only those stocks with consistent and complete data.

2. **Remove Stocks with Average Daily Liquidity Below 100 Million**: Liquidity is crucial for executing trades efficiently and at desired prices. We filtered out stocks whose average daily trading volume was below 100 million USD, as low liquidity could result in larger bid-ask spreads and increased slippage, impacting the performance of our strategy.

3. **Remove Stocks for Which Correlation with ETF Returns is Less Than 0.6**: To ensure that the stocks in our portfolio are representative of their respective sectors, we required a minimum correlation coefficient of 0.6 between the stock's returns and the returns of its corresponding sector ETF. This threshold helps in aligning the stock's movements with broader sector trends, enhancing the effectiveness of the hedging strategy.

4. **Remove Stocks for Which the Stationarity Check Using ADF Test Failed**: Stationarity in time series data is a crucial assumption for many financial models,

including mean-reversion strategies. We used the Augmented Dickey-Fuller (ADF) test to check for stationarity in the price series of each stock. Stocks that did not pass the stationarity test were excluded, as their inclusion could undermine the assumptions underpinning our statistical models and potentially lead to erroneous signals and predictions.

5.3.2 Bucketing of Z-scores and Gridsearch for optimal Z-scores

In our trading strategy, z-scores are used as a standardized measure to identify when a stock is significantly overbought or oversold relative to its historical norms. These z-scores help us decide when to take long or short positions in stocks, and the effectiveness of these positions depends heavily on choosing the right z-score thresholds.

To find the optimal set of z-score thresholds, we employed a technique similar to a grid search, commonly used in machine learning for hyperparameter tuning. A grid search systematically explores a predefined space of hyperparameters and evaluates their performance to identify the best combination.

1. **Initial Thresholds**: We began with the following initial z-score thresholds:
   - **sbc (Sell to Close Long Position)**: $z\_score > -0.5$
   - **sbo (Sell to Open Short Position)**: $z\_score < -1.25$
   - **sso (Sell to Open Long Position)**: $z\_score > 2$
   - **ssc (Sell to Close Short Position)**: $z\_score < 1.25$
2. **Grid Search Process**:
   - We tweaked each threshold incrementally by 0.25. For instance, the `sbc` threshold was adjusted from -0.5 to -0.75, -1.0, and so on. Similar adjustments were made for `sbo`, `sso`, and `ssc`.
   - This systematic tweaking created a grid of possible z-score threshold combinations. For example, if we considered three possible values for each threshold, we would have $3^4 = 81$ combinations to evaluate.

3. **Evaluation Criteria**:

    - For each combination of thresholds, we backtested the strategy over the same period and calculated the resulting Sharpe ratio. The Sharpe ratio is a measure of risk-adjusted return, which helps us understand the effectiveness of the strategy in balancing returns and risk.

4. **Selecting Optimal Thresholds**:

    - The goal was to find the set of z-score thresholds that yielded the highest Sharpe ratio, indicating the most favorable balance between returns and risk.
    - Through this grid search process, the optimal set of thresholds was identified as:
        - **sbc (Sell to Close Long Position)**: $z\_score > -1.0$
        - **sbo (Sell to Open Short Position)**: $z\_score < -1.75$
        - **sso (Sell to Open Long Position)**: $z\_score > 2$
        - **ssc (Sell to Close Short Position)**: $z\_score < 1.25$

These adjusted thresholds allowed for a more refined approach to the trading strategy, balancing the trade-off between capturing profitable opportunities and managing risk. The final thresholds provided the best Sharpe ratio, indicating an improved return per unit of risk taken.

Results:

We get the best Sharpe for:

- sbc = -1
- sbo = -1.75
- sso = 2
- ssc = 1.25

Table 1: PnL metrics for different Z-Scores

Buy Close (SBC)	**Buy Open (SBO)**	**Short Open (SSO)**	**Short Close (SSC)**	**Annual % return**	**Sharpe Ratio**
-0.5	-1.25	2	1.25	7.80%	1.10035
-0.75	-1.25	2	1.25	8.98%	1.2905
-1	**-1.25**	2	**1.25**	**13.15%**	**1.83635**
-0.5	-1.5	2	1.25	10.46%	1.52886
-0.5	-1.75	2	1.25	9.12%	1.434347
-0.5	-1.25	2.25	1.25	2.37%	0.454077
-0.5	-1.25	2.5	1.25	2.30%	0.449384
-0.5	-1.25	2	1.5	3.17%	0.529814
-0.5	-1.25	2	1.75	2.79%	0.479423
-1	-1.5	2	1.25	12.51%	1.79327
-1	-1.75	2	1.25	13.44%	1.97548

Figure 2: PnL over time for optimal z-scores

6 Acknowledgements

We would like to thank Prof. Terrence Hendershott for giving us the opportunity to work on this project. We are very grateful for his guidance.

Appendix

In [1]:
```
import requests as req
import pandas as pd
import os
import json
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
sns.set_theme(style="darkgrid")
import datetime
from statsmodels.tools import add_constant
from sklearn.linear_model import LinearRegression
%matplotlib inline
pd.options.mode.chained_assignment = None

KEY = ''
TIMESCALE = '1y'
DATA_FILE = 'data'#'drive/MyDrive/data'
ETF_DATA_FILE = 'etf_data'
```

```
C:\Users\srava\anaconda3\Lib\site-packages\pandas\core\arrays\masked.py:60: UserWarning: Pandas requires versio
n '1.3.6' or newer of 'bottleneck' (version '1.3.5' currently installed).
  from pandas.core import (
```

In [2]:
```
def get_SP():
  df = pd.read_csv('S&P500-Info.csv')
  return df

def get_sector_etf(sector):
    categories = ['Communication Services', 'Consumer Discretionary', 'Consumer Staples', 'Energy', 'Financials
    return ETF_TICKERS[categories.index(sector)]

# symb = get_SP()[::]['Symbol']
#TICKERS = ['AAPL','XOM']#[symb[i] for i in range(len(symb))]
ETF_TICKERS = ['XLC', 'XLY', 'XLP', 'XLE', 'XLF', 'XLV', 'XLI', 'XLB', 'XLRE', 'XLK', 'XLU','VPU','SMH','RTH','
# print()
# print(ETF_TICKERS)
```

In [3]:
```
def get_returns_data_from_csv(df):
    # df = df.dropna(subset='PX_LAST')
    prices = df['PX_LAST']
    ret = np.log(prices/prices.shift(1))
    df['Return'] = ret.fillna(0.0)
    return df
```

In [4]:
```
# def get_price_volume_data(dataloc=DATA_FILE):
#     folder = os.listdir(dataloc)
#     N = len(folder)
#     dfs = {}
#     for n, filename in enumerate(folder):
#         if filename == '.DS_Store' or os.path.isdir(filename):
#             continue
#         data = pd.read_csv(os.path.join(dataloc,filename))
#         if dataloc == 'etf_data':
#             isETF = True
#         data = preprocess_data(data, isETF = True)
#         if isinstance(data,str) and data == 'empty':
#             print(filename)
#             continue

#         dfs[filename] = data
#     return dfs
```

In [5]:
```
# adtvs = get_price_volume_data('data')
```

In [6]:
```
def preprocess_data(df,isETF=False):
    if df.empty:
        return 'empty'
    if isETF:
        df = df[2:]
        df['PX_LAST'] = df['PX_LAST'].astype(float)
    df['Dates'] = pd.to_datetime(df['Dates'])
    df = df[(df['Dates'] >= '2020-01-01') & (df['Dates'] <= '2023-12-31')]
    if df['PX_LAST'].isna().any():
        return 'empty'
    return df
```

In [7]:
```
def avg_daily_volume_filter(adtv):
    if adtv['ADTV'].mean() >= 1e8:
        return True
    return False
```

In [8]:
```
def get_returns_data(dataloc=DATA_FILE):
    folder = os.listdir(dataloc)
```

```
    N = len(folder)
    # SP500 Data Frame datascrape
    SP = get_SP()
    # map the ticker name to index in the dataframe
    ticker_to_index = {v:k for k,v in SP['Symbol'].to_dict().items()}
    # map returns index to ticker name and sector
    ticker_map = {}
    returns = []
    dates = []
    offset = 0
    isETF = False
    for n, filename in enumerate(folder):

        if filename == '.DS_Store' or os.path.isdir(filename):
            offset += 1
            continue
        data = pd.read_csv(os.path.join(dataloc,filename))
        if dataloc == 'etf_data':
            isETF = True
        data = preprocess_data(data, isETF = True)
        #missing prices data
        if isinstance(data,str) and data == 'empty':
            offset+=1
            print(filename)
            continue

        if not isETF:
            data['DTV'] = data['PX_VOLUME']*data['PX_LAST']
            data['ADTV'] = data['DTV'].rolling(window=60).mean()
            #average daily volume threshold
            if not avg_daily_volume_filter(data):
                offset+=1
                print(f'{filename} not meeting volume threshold condition')
                continue
        #get returns
        data = get_returns_data_from_csv(data)
        # # only use adjusted close data
        ticker = filename.split('.csv')[0]
        # exclude the first day in returns data
        ret = data.iloc[1:,:]['Return']*100
        ticker_return = ret.to_list()
        # map index -> ticker string, ticker etf
        # map ticker string -> index
        if ticker not in ETF_TICKERS:
            etf_ticker_name = get_sector_etf(SP['GICS Sector'][ticker_to_index[ticker]])
            if etf_ticker_name not in ['XLE', 'XLF', 'XLI', 'XLK', 'XLP','XLV', 'XLY']:
                offset+=1
                continue
            ticker_map[n-offset] = (ticker, etf_ticker_name)
            ticker_map[ticker] = n-offset
        else:
            ticker_map[n-offset] = ticker
            ticker_map[ticker] = n-offset


        dates.append(data.dropna(subset='PX_LAST')['Dates'].to_list())

        returns.append(ticker_return)

    return_mat = np.zeros((len(returns), len(returns[0])))

    for i in range(len(returns)):
        for j in range(len(returns[i])):
            return_mat[i,j] = returns[i][j]

    return return_mat, ticker_map, dates
```

In [9]:
```
returns, ticker_map, dates = get_returns_data(DATA_FILE)
```

```
ABNB.csv
AIZ.csv not meeting volume threshold condition
ALLE.csv not meeting volume threshold condition
AMCR.csv not meeting volume threshold condition
AOS.csv not meeting volume threshold condition
ATO.csv not meeting volume threshold condition
AVY.csv not meeting volume threshold condition
AXON.csv not meeting volume threshold condition
BEN.csv not meeting volume threshold condition
BIO.csv not meeting volume threshold condition
BR.csv not meeting volume threshold condition
BRO.csv not meeting volume threshold condition
BWA.csv not meeting volume threshold condition
CARR.csv
CBOE.csv not meeting volume threshold condition
CEG.csv
CINF.csv not meeting volume threshold condition
CPT.csv not meeting volume threshold condition
DAY.csv not meeting volume threshold condition
DVA.csv not meeting volume threshold condition
EG.csv not meeting volume threshold condition
EMN.csv not meeting volume threshold condition
EVRG.csv not meeting volume threshold condition
FDS.csv not meeting volume threshold condition
FFIV.csv not meeting volume threshold condition
FMC.csv not meeting volume threshold condition
FOX.csv not meeting volume threshold condition
FRT.csv not meeting volume threshold condition
GEHC.csv
GEV.csv
GL.csv not meeting volume threshold condition
GRMN.csv not meeting volume threshold condition
HAS.csv not meeting volume threshold condition
HII.csv not meeting volume threshold condition
HRL.csv not meeting volume threshold condition
HSIC.csv not meeting volume threshold condition
HUBB.csv not meeting volume threshold condition
HWM.csv not meeting volume threshold condition
IEX.csv not meeting volume threshold condition
IRM.csv not meeting volume threshold condition
IVZ.csv not meeting volume threshold condition
J.csv not meeting volume threshold condition
JBL.csv not meeting volume threshold condition
JKHY.csv not meeting volume threshold condition
KIM.csv not meeting volume threshold condition
KVUE.csv
L.csv not meeting volume threshold condition
LDOS.csv not meeting volume threshold condition
LKQ.csv not meeting volume threshold condition
LNT.csv not meeting volume threshold condition
MHK.csv not meeting volume threshold condition
MTCH.csv
NDSN.csv not meeting volume threshold condition
NTRS.csv not meeting volume threshold condition
NWS.csv not meeting volume threshold condition
NWSA.csv not meeting volume threshold condition
OTIS.csv
PFG.csv not meeting volume threshold condition
PKG.csv not meeting volume threshold condition
PNR.csv not meeting volume threshold condition
PNW.csv not meeting volume threshold condition
PTC.csv not meeting volume threshold condition
REG.csv not meeting volume threshold condition
ROL.csv not meeting volume threshold condition
SNA.csv not meeting volume threshold condition
SOLV.csv
TAP.csv not meeting volume threshold condition
TECH.csv not meeting volume threshold condition
TFX.csv not meeting volume threshold condition
TRGP.csv not meeting volume threshold condition
TRMB.csv not meeting volume threshold condition
TXT.csv not meeting volume threshold condition
TYL.csv not meeting volume threshold condition
UDR.csv not meeting volume threshold condition
UHS.csv not meeting volume threshold condition
VLTO.csv
VST.csv not meeting volume threshold condition
WAB.csv not meeting volume threshold condition
WBD.csv
WRB.csv not meeting volume threshold condition
```

In [10]:
```
returns.shape
```

Out[10]:
```
(331, 1042)
```

In [11]:
```
etf_returns, etf_map, _ = get_returns_data(ETF_DATA_FILE)
```

In [12]:
```
stock_num = 0
```

```
print(returns[stock_num].shape)
index = etf_map[ticker_map[stock_num][1]]
sector_returns = etf_returns[index]
print(sector_returns.shape)
```

```
(1042,)
(1042,)
```

In [13]:
```
list(ticker_map.values())
set([i[1] for i in list(ticker_map.values()) if type(i) is tuple])
```

Out[13]:
```
{'XLE', 'XLF', 'XLI', 'XLK', 'XLP', 'XLV', 'XLY'}
```

In [14]:
```
from scipy.stats import pearsonr

def ETF_factors(returns, etf_returns, period):
    N = len(returns)
    L = np.zeros((N, 1))
    U = np.zeros((N, period))
    cs = []
    for stock in range(N):
        etf = ticker_map[stock][1]
        index = etf_map[etf]
        sector_returns = etf_returns[index]
        # use etf returns as market factors
        if pearsonr(sector_returns[:period], returns[stock][:period])[0] > 0.7:
            cs.append(stock)
        model = LinearRegression().fit(sector_returns.reshape(period, 1), returns[stock].reshape(period, 1))#tr
        # factor loadings are the slope of regression
        L[stock] = model.coef_
        # calculate residuals
        U[stock] = returns[stock] - np.matmul(L[stock], sector_returns.reshape(1, period))

    return L, U, cs
```

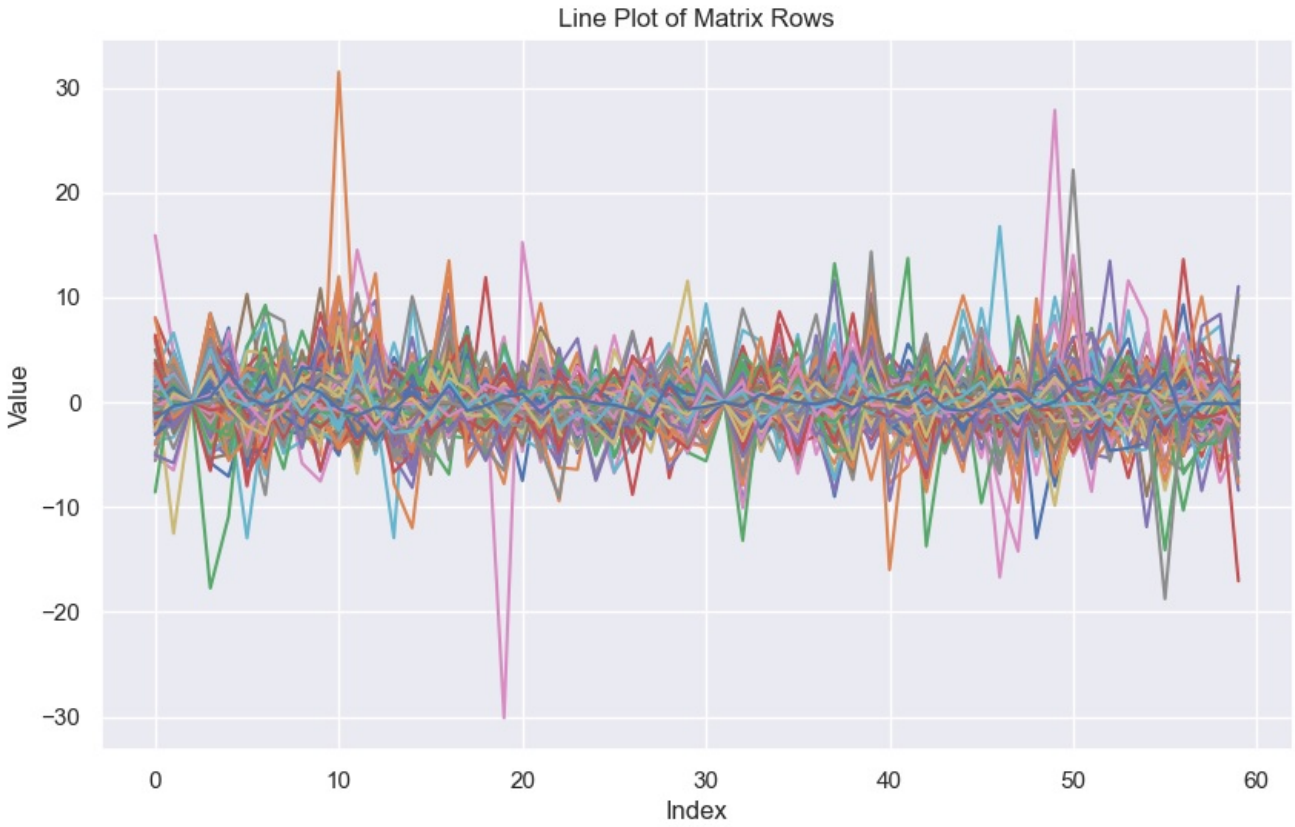
In [15]:
```
start = 100
period = 60
end = start + period
snap_returns = np.transpose(returns.T[start:end])
snap_etf_returns = np.transpose(etf_returns.T[start:end])
L, U, cs = ETF_factors(snap_returns, snap_etf_returns, period)
```

In [16]:
```
L.shape
U.shape
returns.shape
```

Out[16]:
```
(331, 1042)
```

In [17]:
```
# Plotting each row as a different line
plt.figure(figsize=(10, 6))
for i, row in enumerate(U):
    plt.plot(row, label=f'Row {i}')

plt.xlabel('Index')
plt.ylabel('Value')
plt.title('Line Plot of Matrix Rows')
plt.show()
```



In [18]:
```python
from statsmodels.tsa.stattools import adfuller
# Perform the ADF test

def remove_non_mean_reverting_residuals(U):
    '''function to remove the non mean reverting residuals'''
    output = []
    for i,residual in enumerate(U):
        adf_result = adfuller(residual)
        # # Print the results
        # print("ADF Statistic:", adf_result[0])
        #print("p-value:", adf_result[1])
        # print("Critical Values:")
        # for key, value in adf_result[4].items():
        #     print(f'   {key}: {value}')

        # Interpretation
        if adf_result[1] < 0.01:
            1
            #print("The residuals are mean-reverting (reject null hypothesis of unit root).")
        else:
            output.append(i)
            #print("The residuals are not mean-reverting (fail to reject null hypothesis of unit root).")
        return output
```

In [19]:
```python
def OU_estimation(L, U):
    stock_score = []
    stock_kappa = []

    for i in range(len(U)):
        X = np.cumsum(U[i])
        t = len(X) - 1
        Xt = X[0:-1].reshape(t,1)
        Xt1 = X[1:].reshape(t,1)
        ls = LinearRegression(fit_intercept=True).fit(Xt, Xt1)
        beta = ls.coef_[0][0]
        if beta > 1 :
            beta = .9672 #why this?
        alpha = ls.intercept_[0]
        # epsilon -> random process of t+1 residual
        epsilon = Xt1 - alpha - beta * Xt
        k = -np.log10(beta) * 252
        m = alpha / (1 - beta)
        sigma = np.sqrt((np.var(epsilon))/(1-np.power(beta, 2)))
        score = (X[t]-m) / sigma
        # print(f'score {score}, {i}')
        # adjust score for drift
        modscore = (score - (alpha/(k*sigma)))
        stock_kappa.append(k)
        stock_score.append(modscore)

    return stock_kappa, stock_score
```

In [23]:
```python
def backtest_main(sbc, sbo, ssc, sso):
```

```
    # Backtesting program
    first = 60
    last = 250
    cash = 300000
    estimation_window = 60
    total_return = []
    portfolio_breakdown = []
    portfolio = {}
    positions = {n:0 for n in range(len(returns))}
    ticker_stats = {ticker_map[n][0]:0 for n in range(len(returns))}
    etf_stats = {etf_map[n]:0 for n in range(len(etf_returns))}
    kval = []
    signal_evo = []
    for day in range(first, last):
        end = day
        start = day - estimation_window
        period = end - start
        # get returns matrix of just start:end
        snap_returns = np.transpose(returns.T[start:end])
        snap_etf_returns = np.transpose(etf_returns.T[start:end])
        # calculate market factors and residuals
        L, U, correlated_stocks = ETF_factors(snap_returns, snap_etf_returns, period)
        non_mean_reverting_stocks = remove_non_mean_reverting_residuals(U)

    #     # Create a mask that is True for rows to keep
    #     mask = np.ones(matrix_data.shape[0], dtype=bool)
    #     mask[rows_to_remove] = False

    #     # Filtered matrix
    #     filtered_matrix = matrix_data[mask]

        kappa, signal = OU_estimation(L, U)
        # filter out stocks based on mean reversion factor
        # estimated mean reversion must occur within ~8 days
        filtered = [i for i,x in enumerate(kappa) if (x > (252/(estimation_window * 0.5)))] #why not x > 8 days


        # generate returns from portfolio at the start of the trading day
        returns_mat = returns.T[day]
        etf_returns_mat = etf_returns.T[day]
        day_returns = 0

        for ticker in portfolio.keys():
            if ticker in ETF_TICKERS:
                index = etf_map[ticker]
                etf_return = portfolio[ticker] * (etf_returns_mat[index]/100)
                # print(f'{ticker} return: {etf_return}, {etf_returns_mat[index]/100}')
                day_returns += etf_return
                etf_stats[ticker] += etf_return
            else:
                index = ticker_map[ticker]
                stock_return = portfolio[ticker] * (returns_mat[index]/100)
                # print(f'{ticker} return: {stock_return}, {returns_mat[index]/100}')
                day_returns += stock_return
                ticker_stats[ticker] += stock_return

        #print(f'Day {day} returns: {day_returns}')
        #print('\n')

        # calculate long/short percentages
        total_long = sum([val for val in portfolio.values() if val > 0])
        total_short = sum([val for val in portfolio.values() if val < 0])
        portfolio_breakdown.append((total_long/cash, total_short/cash))

        # track returns, kappa values, and signals for graphing
        total_return.append(float(day_returns))
        kval.append(kappa)
        signal_evo.append(signal)

        # assign new positions using closing prices
        for n, sig in enumerate(signal):
            take_pos = n in filtered and n not in non_mean_reverting_stocks and n in correlated_stocks
            # if we already have a position, check for exit condition
            if take_pos: #only take position if model is valid K > 8.4
                if positions[n] != 0:
                    if (positions[n] == 1 and sig > sbc) or (positions[n] == -1 and sig < ssc): #-.0.5, 1.25
                        positions[n] = 0
                    if not take_pos:
                        positions[n] = 0

                # if we don't have a position, check for entry condition
                else:
                    if sig < sbo and take_pos: #-1.25
                        positions[n] = 1

                    elif sig > sso and take_pos: #2
                        positions[n] = -1
```

```
                    else:
                        positions[n] = 0


        # equal weight every position in the portfolio
        # could modify to an optimization problem
        total_pos = len([pos for pos in positions.values() if pos != 0])

        # create new portfolio
        # pair trade stock and associated etf
        portfolio = {}
        for pos in positions.items():
            index = pos[0]
            stock, etf = ticker_map[index]
            # long stock, short etf
            if pos[1] == 1:
                long = (1 / total_pos) * cash
                short = L[index][0] * long * -1
                portfolio[stock] = long
                if etf in portfolio.keys():
                    portfolio[etf] += short
                else:
                    portfolio[etf] = short

            # short stock, long etf
            elif pos[1] == -1:
                short = (1 / total_pos) * cash * -1
                long = short * L[index][0] * -1
                portfolio[stock] = short
                if etf in portfolio.keys():
                    portfolio[etf] += long
                else:
                    portfolio[etf] = long




    print('\n')
    plt.figure(1)
    # plot the cumulative pnl for the backtest
    plt.xlabel('Business Days In Backtest')
    plt.ylabel('Cumulative PnL ($)')
    sns.lineplot(data=np.cumsum(total_return, dtype=object))
    plt.savefig('cumulativepnl')


    print('\n')
    print(f'Total Return: {np.sum(total_return)}')
    percent = (np.sum(total_return)/cash)*100
    print('Percentage Return: {:.2f}%'.format(percent))
    sharpe = (np.mean(total_return)/np.std(total_return))* (252 ** .5)
    print(f'Sharpe Ratio: {sharpe}')
    return [percent, sharpe]
```

In [24]:
```
sbc = -0.5
sbo = -1.25
sso = 2
ssc = 1.25
ans = backtest_main(sbc, sbo, ssc, sso)
for i in [0.25,0.5]:
    sbc = sbc - i
    out = backtest_main(sbc, sbo, ssc, sso)
    ans.append(out)

for i in [0.25,0.5]:
    sbc = sbo - i
    out = backtest_main(sbc, sbo, ssc, sso)
    ans.append(out)

for i in [0.25,0.5]:
    sbc = sso + i
    out = backtest_main(sbc, sbo, ssc, sso)
    ans.append(out)

for i in [0.25,0.5]:
    sbc = ssc + i
    out = backtest_main(sbc, sbo, ssc, sso)
    ans.append(out)
```

```
Total Return: 23408.735175781047
Percentage Return: 7.80%
Sharpe Ratio: 1.1003509505157425
```

```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
Total Return: 26925.097626313316
Percentage Return: 8.98%
Sharpe Ratio: 1.2904998080746448
```

```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
Total Return: 39441.42096339095
Percentage Return: 13.15%
Sharpe Ratio: 1.8363509660506114
```

```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
Total Return: 31365.012021716786
Percentage Return: 10.46%
Sharpe Ratio: 1.528862688877771
```

```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
Total Return: 27357.08316650596
Percentage Return: 9.12%
Sharpe Ratio: 1.4343472880029544
```

```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
Total Return: 7097.07593731037
Percentage Return: 2.37%
Sharpe Ratio: 0.4540773264431008
```

```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
Total Return: 6900.521272848089
Percentage Return: 2.30%
Sharpe Ratio: 0.4493841914091877
```

```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

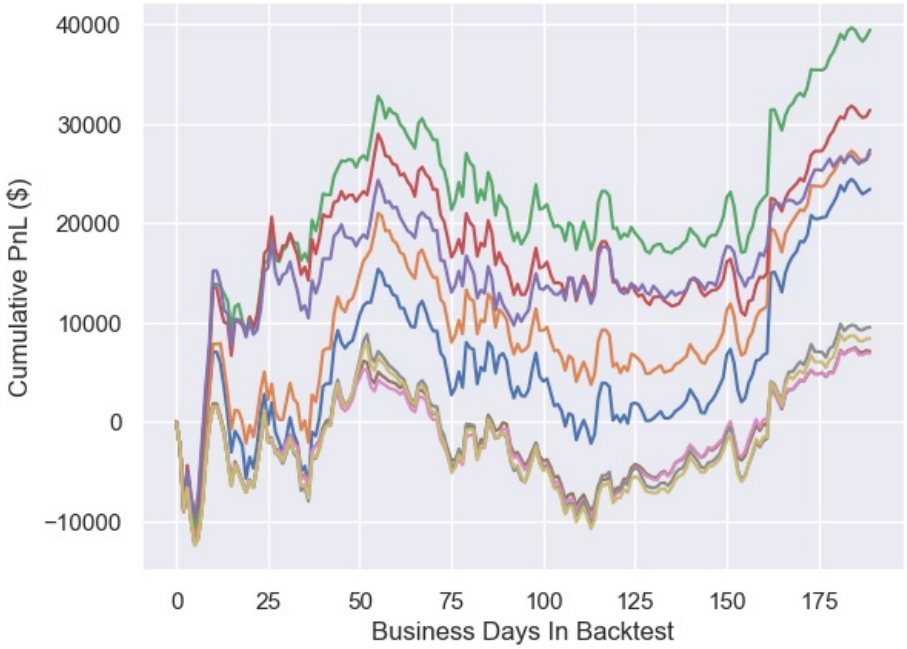
```
Total Return: 9513.599217916126
Percentage Return: 3.17%
Sharpe Ratio: 0.5298814539538512
```

```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep

recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```

```
Total Return: 8355.290436968848
Percentage Return: 2.79%
Sharpe Ratio: 0.4794231592116388
```

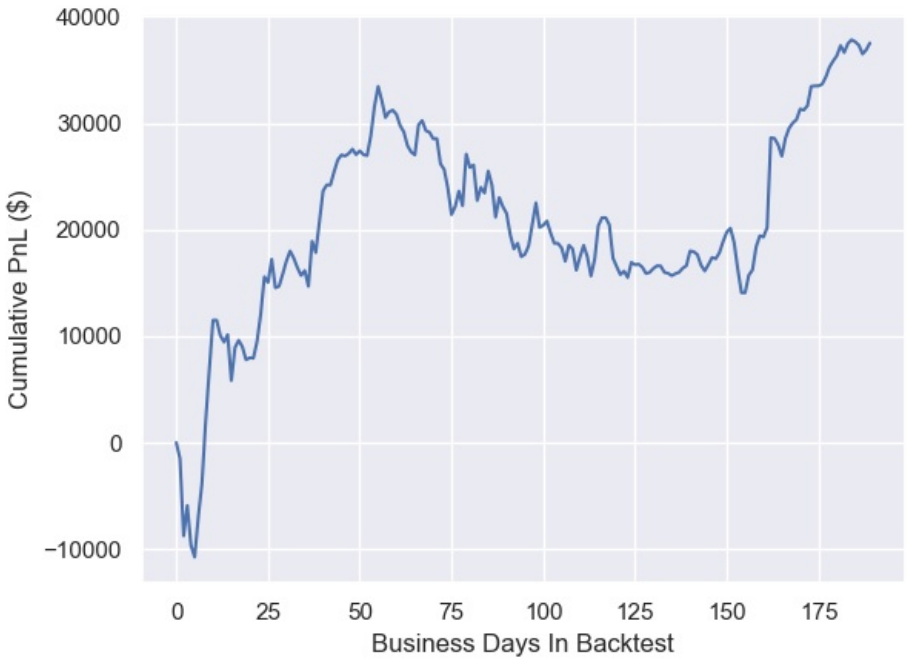
```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [25]:
```
sbc = -1
sbo = -1.75
sso = 2
ssc = 1.25
out1 = backtest_main(sbc, sbo, ssc, sso)
```

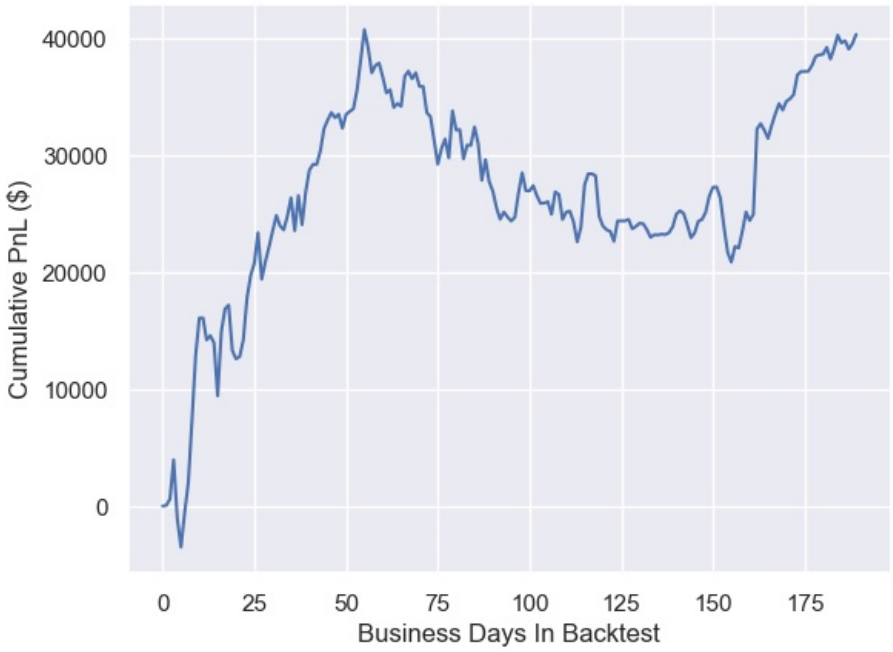
```
Total Return: 37532.3707625829
Percentage Return: 12.51%
Sharpe Ratio: 1.7932755145475954
```

```
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [26]:
```
out2 = backtest_main(sbc, sbo-0.25, ssc, sso)
```

```
Total Return: 40322.01546844425
Percentage Return: 13.44%
Sharpe Ratio: 1.9754785548941465
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
C:\Users\srava\anaconda3\Lib\site-packages\seaborn\_oldcore.py:1119: FutureWarning: use_inf_as_na option is dep
recated and will be removed in a future version. Convert inf values to NaN before operating instead.
  with pd.option_context('mode.use_inf_as_na', True):
```



In [ ]: