Mean Reversion Analysis

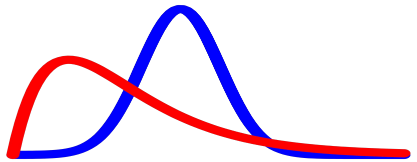
by Vincent Lee, Ryan Pierce, and Raul Martinez

Motivation & Background

Our team has an interest in financial data, specifically stock data, so we wanted to create a stock trading algorithm. The pairs trading algorithm is a common model in systematic trading. Our version of the pairs trading algorithm seeks to utilize mean reversion theory to create a profitable trading strategy by buying and selling two cointegrated stocks. Mean reversion theory states that over time, assets will converge to their average price, so when they stray from that average, there is arbitrage, or potential for profit. Mean reversion is a very commonly used concept in trading, not only being used for pairs trading, but also moving-average crossover trading and calendar spread trading.

Objective

Our algorithm will conduct an analysis on the top 100 companies from three different sectors: Healthcare, Technology, and Industrials. It will find a pair of cointegrated stocks, then use our pairs trading algorithm to trade this stock, hopefully leading to positive returns.



Financial Definitions

Stock

Stocks represent a tiny ownership stake in a company. They can be bought and sold on stock exchanges such as the New York Stock Exchange, and their prices can go up or down based on factors such as the company's performance, economic conditions, and investor sentiment.

Long Position

A long position in when you buy a stock with the expectation that its value will increase over time, allowing you to sell it later at a higher price for a profit. One can enter a long position by buying a stock or selling a call option.

Short Position

A short position is a way to make money when the price of an investment goes down. One can enter a short position by selling a call option.

Buy Low, Sell High

Buy low, sell high is a motto among traders. Buy means to take a long position, and sell means to take a short position. You make money if you take a long position when a stock is low, then the stock price increases, or if you take a short position when a stock is high, then the stock price decreases. If you want to make money, you should buy low and sell high!

Spread

Stock spread refers to the price difference between two stocks at a certain period of time. If the current spread between two stocks moves away from the historical spread between those stocks, the stocks are said to be diverging.

Data Sources

Website - stockanalysis.com

Description - Stock Analysis is a website which contains information about stocks as well as stock market news articles. In addition, they also have pages dedicated to showcasing stocks which belong to particular sector of the stock market. We used this site to scrape a list of the top 100 stocks for the healthcare, industrials, and technology sector of the stock market.

Size - <1mb
Location - https://stockanalysis.com/
Format - HTML
Access Method - Web Scraping

Python Library - yfinance

Description - yfinance is a python library which allows users to obtain time series information about assets using the Yahoo Finance API, particularly stocks traded in the NYSE. We used this library to obtain all the closing prices for the stocks we analyzed.

Size - <1mb
Location - https://pypi.org/project/yfinance/
Format - DataFrame
Access Method - python library (yfinance)

Data Manipulation

Web Scraping

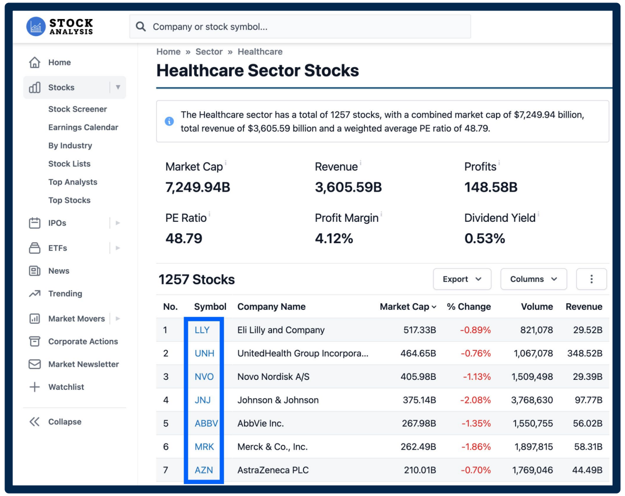
The team utilized web scraping to obtain a list of the top 100 stocks from the healthcare, industrials, and technology sectors of the market. To accomplish this the team utilized the python libraries requests, and beautifulsoup to scrape the stock tickers from the site Stock Analysis.

The process of scraping the stocks was encapsulated into a function called *scrape_stock_symbols()*. This function takes two arguments. The first argument is *arr* which is an array of dictionaries, each of which contains two keys the url to scrape and the sector that the stocks belong to. The second argument is *top_n* which specifies how many of the top n stock tickers the function should scrape.

The function begins by establishing an empty list called *stocks*, which is used to aggregate the responses from each page. Then the function loops through the dictionaries in *arr*. For each url, it sends a GET request to the url, then parses the HTML response into a BeautifulSoup object and stores it in a variable soup. The function then utilizes soup to target the text inside of HTML tags which contain the stock tickers which are required, and generate a list of all the stock ticker symbols for the sector. The list of stock ticker symbols and the sector are appended to stocks in a dictionary containing the list and the string for the sector.

The function returns a list of dictionaries. Each with two keys, *sector* which is a string for the particular sector which was scraped and *stock_symbols* which is a list of top 100 tickers corresponding to the sector.

Below is an example of one of the pages scraped:



Home » Sector » Healthcare

Healthcare Sector Stocks

The Healthcare sector has a total of 1257 stocks, with a combined market cap of $7,249.94 billion, total revenue of $3,605.59 billion and a weighted average PE ratio of 48.79.

Market Cap	Revenue	Profits
7,249.94B	3,605.59B	148.58B

PE Ratio	Profit Margin	Dividend Yield
48.79	4.12%	0.53%

1257 Stocks

No.	Symbol	Company Name	Market Cap	% Change	Volume	Revenue
1	LLY	Eli Lilly and Company	517.33B	-0.89%	821,078	29.52B
2	UNH	UnitedHealth Group Incorpora...	464.65B	-0.76%	1,067,078	348.52B
3	NVO	Novo Nordisk A/S	405.98B	-1.13%	1,509,498	29.39B
4	JNJ	Johnson & Johnson	375.14B	-2.08%	3,768,630	97.77B
5	ABBV	AbbVie Inc.	267.98B	-1.35%	1,550,755	56.02B
6	MRK	Merck & Co., Inc.	262.49B	-1.86%	1,897,815	58.31B
7	AZN	AstraZeneca PLC	210.01B	-0.70%	1,769,046	44.49B

Data Manipulation

Database Stock Price Seeding

For a single source of truth the team decided to use a PostgreSQL database hosted on Heroku. To execute any queries to the database, a function *sql_execution_wrapper(sql_statement)* was created to manage opening, closing, executing, and committing any sql query. To create the necessary table for the database a function was created *create_stock_price_table()* which created the table *stock_prices*. To prevent against duplicate entries, we restricted a UNIQUE combination of the *ticker* and *date* columns.

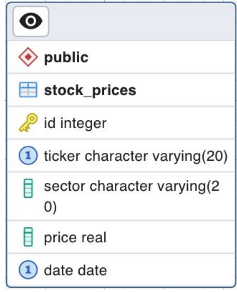
To seed the database, a function called *seed_stock_info()* was created to orchestrate the scraping and seeding of stock symbols. First, the function would scrape the three stock sectors of interest using the *scrape_stock_symbols()* function. The resulting list of dictionaries, would be iterated and the *stock_symbols* list within each dictionary would be iterated creating a for-loop within a for-loop. In the innermost for-loop, we would loop through each ticker for a particular industry. For each ticker, a function *insert_stock_records_to_db(ticker, sector, start_date, end_date)* was executed. This function handled retrieving all available stock price history from yfinance and transforming the DataFrame response from yfinance into a string which combined the necessary columns into a SQL INSERT query.

Here is an example insert query that the function would generate and execute for one day of stock info:

INSERT INTO public.stock_prices(price, ticker, date, sector)
VALUES (85.82,'AMZN','2023-01-03','technology');

The actual query, inserted values for every available price from 08/31/2019 - 08/31/2023, in a single query. This inner loop ran 300 times, one for every 100 stocks in the 3 sectors the team chose. Out of the 300 stocks, two were not available in yfinance, and 18 did not have complete information. To prevent errors from breaking the loops, the insert statements were wrapped in a *try/except* block, logging whenever an error was encountered. In downstream analyses, only stocks with complete information were considered.

Once the data was in the database, team members were able to access the stock information through a function called *get_stock_prices_from_db()* which would return the entire table as a DataFrame of shape (294277, 5).


public
stock_prices
id integer
ticker character varying(20)
sector character varying(20)
price real
date date

UNIVERSITY OF MICHIGAN

Analysis

Cointegration

Cointegration can be interpreted as a long-term relationship between two or more time series variables over time. Two series that are cointegrated tend to move together over the long run, even if they independently fluctuate in the short term. Mathematically, if two series are cointegrated, they can be linearly combined to form a stationary series. It is important to note that cointegration is not correlation and two assets can be cointegrated while not being correlated, as well as the inverse.

To determine stocks which are cointegrated, we divided the stocks by sector to only analyze same sector stocks. From there, we further filtered the stocks to ensure that only stocks with complete data for the time period of 2019-08-31 to 2023-08-31 were included in the analysis. For these stocks we chose the time period 2019-08-31 to 2022-08-31 to analyze cointegration between the stocks. This was done intentionally to hold out the last year of data to test the stock prediction algorithm. In addition, this prevented the potential for data leakage, where the algorithm would have access to cointegrated values over a period of time which the algorithm should not have access to.

The test used to determine cointegration between a pair of stocks was the augmented Engle-Granger cointegration test. This test is implemented in a function named *coint* which is available in the statsmodels python library. For the augmented Engle-Granger test, the null hypothesis is that there is no cointegration, meaning if the p-value from the test is small we can reject the hypothesis that there is no cointegrating relationship (6).

The team decided the critical size of the p-value to be below 0.05. Any cointegration pair with a p-value below 0.05 would be considered for further stationarity analysis.

Note that our cointegration test is susceptible to multiple comparison bias. We tried to minimize this issue by only comparing stocks to other stocks in the same sector. Large hedge funds usually take this a step further, and have large teams of economic and financial analysts narrowing down the list to just a few stocks before running cointegration tests, but given our lack of time and resources, we were unable to do this. Thus, we must acknowledge multiple comparison bias as a limitation to our current model.

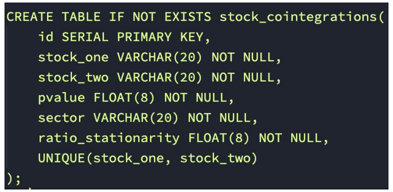
Analysis

Stationary

A stationary series is a time series in which the statistical properties, such as the mean and variance are constant over time.

For each asset pair which was determined to be cointegrated, an additional test was performed to determine stationarity in the spread between the two stocks. The spread can be interpreted in several ways in quantitative stock analysis, for our analysis we used the ratio spread. The price ratio spread is obtained by dividing one stock series by the other, the result is a singular time series of data with the price ratio between two individual assets.

To analyze the stationarity in the price ratio spread, the team used the Augmented Dickey-Fuller test. This test is implemented in a function named *adfuller* which is available in the statsmodels python library. For the Augmented Dickey-Fuller test, the null hypothesis is that there is a unit root (6). For every pair which was identified to be cointegrated, an additional Augmented Dickey-Fuller test was performed on the price ratio spread to determine the p-value of the test.



```
CREATE TABLE IF NOT EXISTS stock_cointegrations(
    id SERIAL PRIMARY KEY,
    stock_one VARCHAR(20) NOT NULL,
    stock_two VARCHAR(20) NOT NULL,
    pvalue FLOAT(8) NOT NULL,
    sector VARCHAR(20) NOT NULL,
    ratio_stationarity FLOAT(8) NOT NULL,
    UNIQUE(stock_one, stock_two)
);
```

Cointegration and Stationarity Results

The main drawback encountered with the *coint* and *adfuller* functions are the absence of a vectorized implementation, requiring the team to use an inner and outer for loop to analyze every pair resulting in a long O(n^2) runtime to analyze all the pairs. Comparing all the stock pairs for the 3 sectors, took around 15 minutes to complete. To combat this long running execution time, the team made use of a PostgreSQL Database table to persist the results of the analysis. Note that this limitation would not be a problem if we were able to narrow our list of stocks to compare, as mentioned in the Cointegration slide.

To store the results, the team used a table called *stock_cointegrations* with four columns. One column for each stock in the pair, as well as a column for the p-value for the result from the coint test and a column for the p-value result from the adfuller test.

The analysis returned 1303 pairs of same sector stocks which contained a p-value below the 0.05 significance level cutoff.

In addition, an access function was provided named *get_stock_coint_pairs_from_db()*. This function allowed team members to retrieve all the pairs into a pandas DataFrame, from the database without needing to rerun the cointegration analysis.

Analysis

Mean Reversion

Mean reversion refers to the idea that over time, things tend to return to their average or "mean" level. As it relates to stock prices, if you have a stock that is performing much better than it has in the past, the mean reversion theory states that it is likely to go back to its previous performance in the future.

The algorithm that we have built, called pairs trading, takes advantage of mean reversion in a clever way. If we find two cointegrated stocks, we know that they move together. We can take advantage of the fact that these cointegrated stocks can still experience independent, short term fluctuations. If one starts to overperform and the other starts to underperform, there is opportunity for us to make money by taking a long position in the under performer and a short position in the over performer. The hope is that both stocks return to normal performance. The under performer that we have a long position in goes up, and the over performer we have a short position in goes down. We have bought low and sold high, so both our positions yield profit.

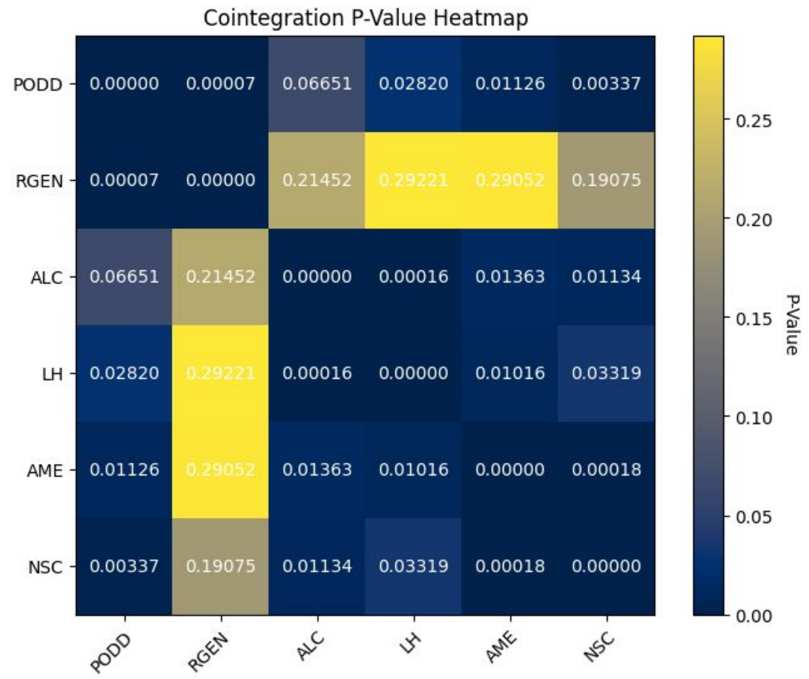
We identify over performing and underperforming by finding two cointegrated stocks. With those two stocks, we want to find the spread between these stocks, or how different they are from each other. We have opted to use the ratio between the two stocks rather than the difference because cointegrated stock prices are usually on a multiplicative scale to each other (Ex. Stock A's price is always ~2x Stock B's price, so if Stock B increases by $1, Stock A increases ~$2). It is highly likely that the spread between a pair of cointegrated stocks is stationary.

The main way this strategy can fail is if our assumption of cointegration does not hold. As one example, our tests could incorrectly show us that two stocks are cointegrated when they are not in reality. As a second example, our tests could correctly show us that two stocks are cointegrated, but in the future, for whatever reason, they move apart and are no longer cointegrated. If cointegration no longer holds, it is possible that the two stocks diverge and do not mean revert, resulting in both our long and short positions losing money.

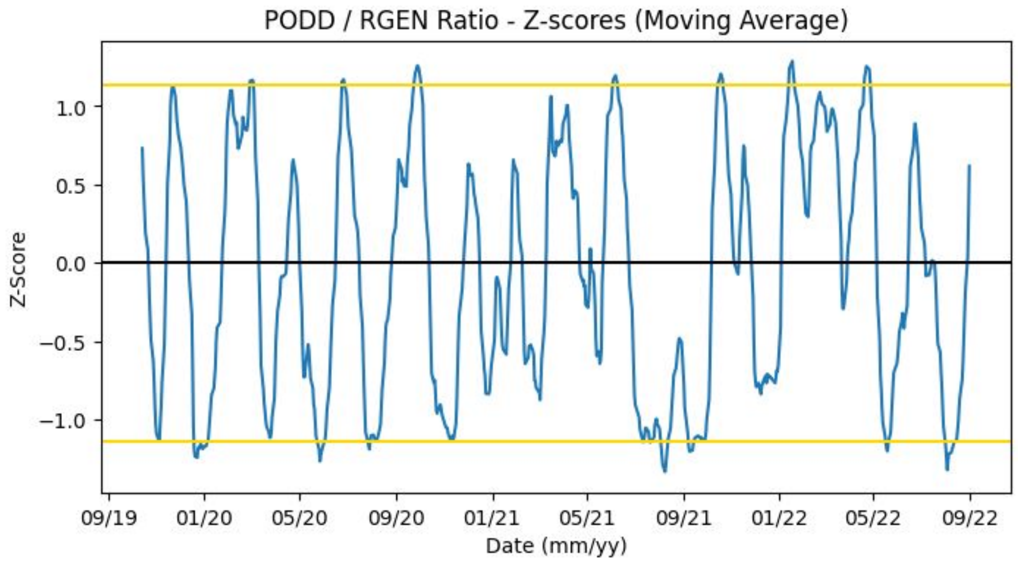
Of course, there will always be potential points of failure for an algorithm that could cause us to lose money, but as they say on Wall Street, there is no such thing as a "free lunch".

Visualizations

The heatmap below shows the results of our cointegration test. We can see that PODD and RGEN, both in the healthcare industry, had the lowest p-value for our cointegration test, so we used this pair of stocks to demonstrate our pairs trading algorithm.



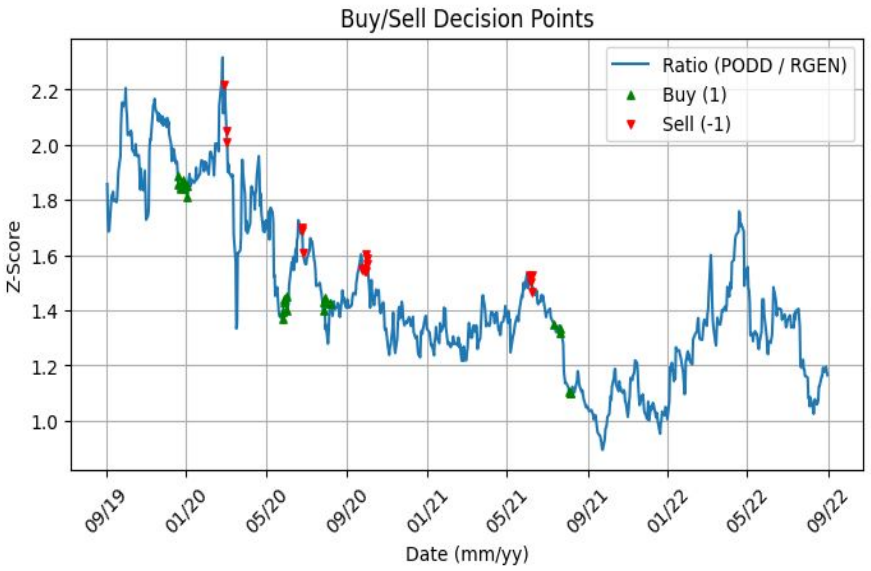
	PODD	RGEN	ALC	LH	AME	NSC
PODD	0.00000	0.00007	0.06651	0.02820	0.01126	0.00337
RGEN	0.00007	0.00000	0.21452	0.29221	0.29052	0.19075
ALC	0.06651	0.21452	0.00000	0.00016	0.01363	0.01134
LH	0.02820	0.29221	0.00016	0.00000	0.01016	0.03319
AME	0.01126	0.29052	0.01363	0.01016	0.00000	0.00018
NSC	0.00337	0.19075	0.01134	0.03319	0.00018	0.00000

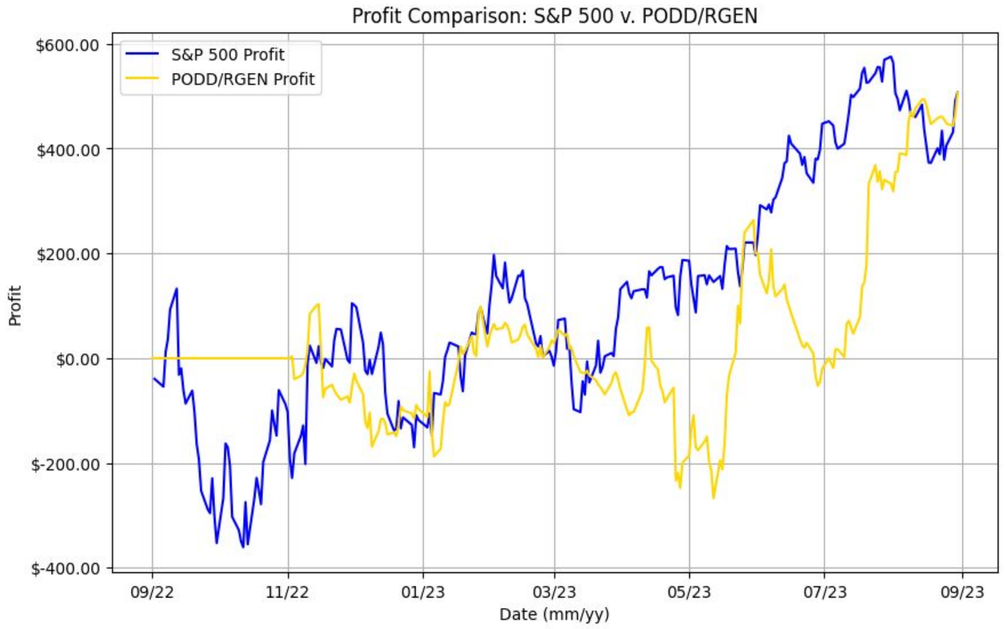


This visualization above shows the spread between PODD and RGEN (represented as price ratio) with bands representing an upper and lower z-score. When the z-score is below the band, that means that stock 1 (PODD) is undervalued compared to stock 2 (RGEN), so we will buy the spread, which means long stock 1 and short stock 2. When the z-score is above the band, that means that stock 1 (PODD) is overvalued compared to stock 2 (RGEN), so we will sell the spread, which means short stock 1 and long stock 2.

Note: Our spread was calculated with moving averages to avoid look-ahead bias. Details can be found in our code.

Conclusions





The green arrows represent signals that our algorithm bought the spread, and the red arrows represent signals that our algorithm sold the spread. Luckily, it looks like our algorithm did very well with buying low and selling high.

The profit comparison between our pair trading algorithm and investing into the S&P 500 is shown above. Our pair trading algorithm requires $0 initial cash outlay, while we assumed a $3,670 initial investment in the S&P 500 to have the same ~$507 profit. Note, these dollar values are under the assumption that there are no transaction fees for either trading strategy.

Statement of Work

Statement of Work

All - Responsible for documentation of processes and final report editing and review

Raul Martinez - Lead for data cleaning and manipulation

Vincent Lee - Lead for data analysis

Ryan Pierce - Lead for data visualizations

Collaborative Tools

Slack - Communication and collaboration with team to share ideas, links, and files

GitHub - Version control, documentation, and code storage & collaboration

Google Slides - Collaborate on, review, and edit final report

References

1. https://pypi.org/project/yfinance/
2. https://stockanalysis.com/
3. https://requests.readthedocs.io/en/latest/
4. https://www.crummy.com/software/BeautifulSoup/bs4/doc/
5. https://brand.umich.edu/design-resources/templates/
6. https://www.statsmodels.org/
7. https://people.duke.edu/~rnau/411diff.htm
8. http://uu.diva-portal.org/smash/get/diva2:1477748/FULLTEXT01.pdf