**Pairs Trading:**

**Cluster-based PCA, k-means, and t-SNE Pair Identification using Financial and Closing Data for Russell 3000 Stocks**

Jerry Li

Saarthak Maheshwari

Aryan Shah

1. **Abstract:**

This project is a pairs trading pipeline that creatively explores finding better ways to determine pairs using a mixture of financial metrics, financial health, and closing price data for Russell 3000 stocks in order to create clusters of stocks using k-means clustering and t-SNE that are strong candidates for cointegration using initial dimensionality reduction through PCA. The pairs of stocks that are determined by clusters that then meet the threshold for a low p-value using a cointegration test are then fed through the remaining pairs trading pipeline, which normalizes the ratio of the closing prices of the pairs by evaluating the ratio’s z-score using a 5-day moving average as the current price ratio and a 30-day moving average for the mean and standard deviation. Finally, the closing price data for the pair of stocks uses a 75:25 split for “training” and “testing”, for which the training data is used to determine the rolling z-scores at which to buy or sell one or the other stocks within the pair using a simple trading strategy based on a z-score of 1 or -1 (peaks or troughs). This simple trading strategy is what is used to determine the efficacy of our clustering paradigm for our uniquely structured and feature-engineered dataset.

1. **Introduction:**

Pairs trading is a strategy that is considered market neutral because it identifies pairs of securities who are intrinsically similar to each other and also looks for times when the two securities are trending in opposing directions or aren’t in their usual trading range. Here, pairs traders can take advantage of this discrepancy by shorting and longing securities within the pair based on their paths. Pairs trading is a quantitative technique that has been heavily used in history. Through our research, however, we wanted to get a better understanding of pairs trading as well as come up with a new dataset and explore new techniques to find pairs. Currently, some implementations and techniques used for finding pairs in pairs trading include various cointegration tests, such as Engle Granger’s OLS regression, the distance approach which includes simply finding squared differences between prices within a pair that are normalized, the stochastic approach which assumes continuous time and added measurement errors. Through our project, we wanted to explore a newer technique for finding pairs which includes clustering and dimensionality reduction in order to use the principal components of stocks to determine the similarity of a pair through k-SNE coefficients that can be plotted. This technique allows us to try constructing a more robust dataset to use for our pipeline rather than just the closing prices of stocks. Therefore, we were able to incorporate a 743-feature dataset which included financial health and metrics data as well as time series data for closing prices over a 2 year period. By conducting PCA on this dataset and reducing its dimensionality, we can determine the optimal number of clusters to cluster our stock dataset (Russell 3000 stocks) based on inertia (or variance) of cluster sizes. Then, we can use k-means clustering algorithms to determine a set number of centroids that all the stocks should be centered around. Now that we used a mixture of financial data and closing price data to determine a set number of clusters to put our stocks in, each cluster already tells us something about the intrinsic similarity between stocks within a cluster due to the PCA analysis. Now, we can integrate another testing mechanism to further evaluate the clusters. We chose to run a simple cointegration test on each cluster and find pairs with p-values at .001 or less.

Our hypothesis for this project is that using PCA, k-means clustering, and t-SNE to find clusters of similar stocks would perform better than simply randomly selecting two stocks from the Russell 3000 and feeding it into a trading strategy. In order to test this hypothesis, we then began building out a simple trading strategy to test the efficacy of our pairs selecting paradigm. We used the ratio of the optimal pair(s) we got from our clustering scheme and then normalized the closing data of these stocks to compute the ratio as a z-score. Then, we calculated the rolling z-score by using the 5-day and 30-day moving/rolling averages. From this, we were able to see the spikes and dips in the rolling z-score for our pair(s) and approximately at which z-scores the rolling z-score would revert back to the mean or in the other direction. Now, we decided to use a simple trading strategy where we sell short for a z-score that was larger than 1, and buy long for a z-score that was less than 1. We split our rolling z-score data into “training” and “testing”, where we used the “training” data to determine that 1 and -1 z-scores were optimal as signals, and then applied that directly to our “testing” data in order to compare net returns on both the training and testing data over a window of around 300 days.

1. **Literature Review:**

<http://home.cse.ust.hk/~rossiter/independent_studies_projects/stocks_arbitrage/stocks_arbitrage.pdf>:

This paper, like the stanford paper below, goes in depth into an entire pipeline for pairs trading using a clustering scheme. They implemented a density-based spacial clustering algorithm to identify pairs, and talked about how PCA works. From there, they also ran t-SNE algorithms on their validated and PCA’ed pairs in order to plot their clusters. They used some more advanced algorithms for their actual trading scheme, like the Huerst Exponent and half-life test to determine lag between the two stocks within a pair. They also used the spread from Kalman Filters to determine their trading strategy, which consisted of signals at an open z-score of 1 and a close z-score of 0. They also benchmarked using the S&P 500. They also included some great visualizations on their cumulative returns for the two stocks within a pair as well.

<https://core.ac.uk/download/pdf/52072275.pdf> -

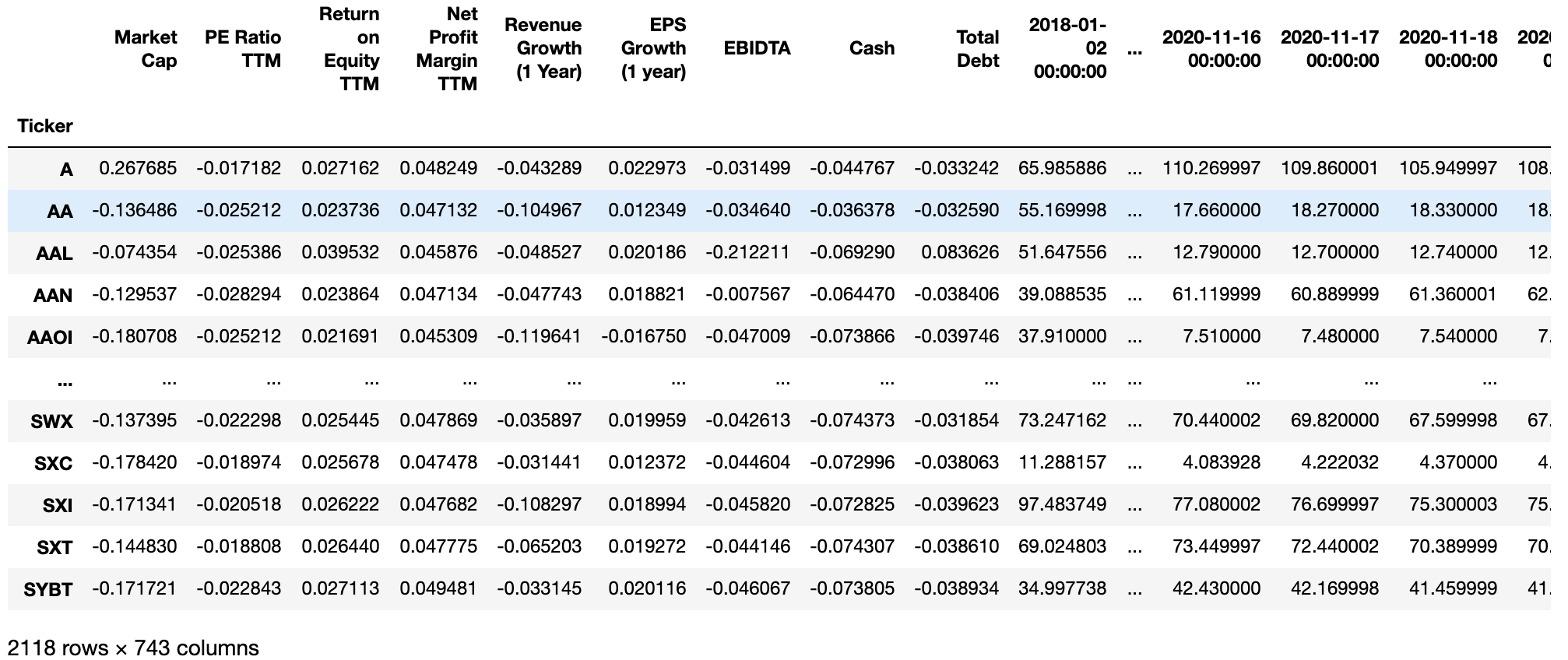
This paper from the Norwegian School of Economics details exactly how statistical arbitrage pairs work, various techniques to find pairs, and trading schemes as well. They go into depth on the mathematical intuition behind the distance, scholastic, and cointegration approaches of finding pairs. They also explained how to derive trading thresholds, which is how we decided to use a normalized z-score based threshold scheme, although it’s very simple. They also discuss hedge ratios and how they come into play when you sell short or buy long. They tested their pairs trading pipeline on the Olso Stock Exchange over a 10 year period, using long formation periods and short trading periods that increase in time linearly. They tested for cointegration using 2 years of data, but used a 5% threshold (we on the other hand used a 1% threshold for our p value). They also filtered out pairs based on transaction costs and and used standard deviation as their threshold for shorting and longing.

<http://stanford.edu/class/msande448/2017/Midterm/gr6.pdf>:

This is actually a slide deck from a group of students in a class at Stanford University called MS&E 448 - Big Financial Data for Algorithmic Trading, where the group outlines what their pipeline looked like for their multi-factor statistical arbitrage model. It gave us a lot of good insight on what our pipeline should look like, all the way from collecting data, to exploring various methods to find pairs, and getting results. Their model design revolved around using returns and fundamental finance data, using PCA and clustering on them, and then finding cointegrated pairs from those clusters. They didn’t end up actually using their S&P 500 fundamental factors dataset! They used an elbow graph to determine variances in components after using PCA. Their trading algorithm included passing the Augmented Dickey Fuller Test as well as the Engel-Granger test. They also included some factors of their trading algorithm, where they specified the spread of the ratio being in a certain range, which gave us the idea for our simple algorithm as well.

1. **Datasets**

We decided to use two main datasets for our project. The first dataset was closing prices from 2018 to 2020 for all stocks in Russell 3000. To gather this data, we used the Yahoo Finance API. Usually in pairs trading, most research and experimentation uses closing prices of stocks as the main way to determine pairs. We, however, wanted to expand our data on each stock by also gathering data on the market cap, PE ratio, return on equity, 1 year revenue growth and EPS growth, EBITA, cash, and total debt. Our assumption was that these financial metrics and ratios would allow for better cluster identification when performing PCA on our dataset, and hopefully better cointegrated stocks that would perform better on our simple trading strategy revolving around z-scores. So, we collected our data on these financial metrics using a financial metrics service called Financial Modeling Prep. Through this service, we used several API calls to pull data on all the Russell 3000 stocks in batches of around 200. These API calls were directly for financial growth, income statement, and ratios-ttm datasets, each API call resulting in a dictionary for each stock. We parsed the data within each dictionary in each API call and aggregated them together for each batch. Then, we merged the data in these batches together and finally merged the financial metric dataset with the closing price dataset to get a robust 743-feature dataframe that we believed would be a creative dataset to use for PCA and cluster analysis in order to determine pairs to test for cointegration. After we merged our dataset, we had to continue preprocessing and standardizing the data. We used scikit-learn’s StandardScaler class in order to scale all 743 features of our merged dataset in order to have the distribution centered at 0 and have a standard deviation of 1. This was important to do to our dataset because our financial data and our stock price data are of different units and types, so by scaling our data using StandardScaler, we can work with all our features without having any negative externalities in the dataset that cause our clustering functions and z-score manipulation to not be intuitive. We decided to use the assumption that our data was normal, which obviously may not always be the case with stocks because in real life we often see stocks with tail distributions and other types of distributions, but regardless by normalizing the data it was easier to work with and compute z scores for the ratio of pair prices in the rest of the pipeline. The final preprocessing we did to our dataset was essential to the modeling portion of our project. Essentially, once we had found tradable pairs using our clustering and PCA pipeline (see Modeling section), we wanted to make sure that the ratio between the two stocks could be converted to a z-score with a mean as close to 0 as possible. If the ratio between two stocks closing prices have a mean as close, then it becomes easier to determine at what z-scores the ratio increases or decreases and arrives back at the mean, which is important if you want to implement any sort of trading strategy based off of z-scores.



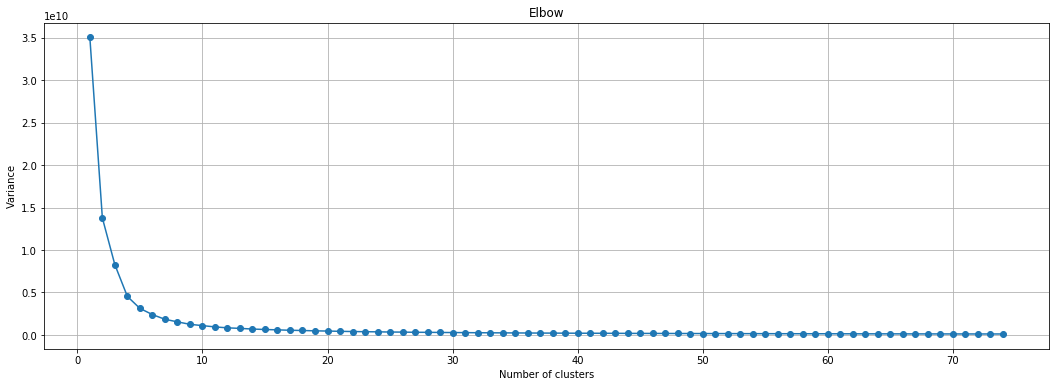
1. **Methods and Modeling**

Our literature and other sources for our ideas for our methods and modeling come from:

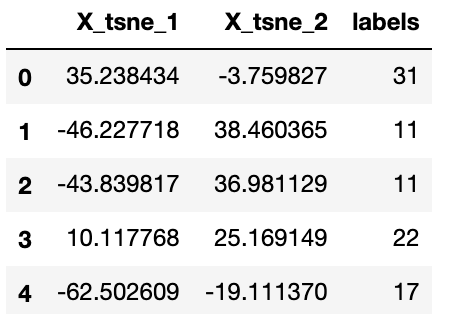
1. <https://towardsdatascience.com/an-introduction-to-t-sne-with-python-example-5a3a293108d1>
2. <http://home.cse.ust.hk/~rossiter/independent_studies_projects/stocks_arbitrage/stocks_arbitrage.pdf>
3. <https://towardsdatascience.com/quant-trading-an-introduction-to-pairs-trading-5ce50c03177e>
4. <http://stanford.edu/class/msande448/2017/Midterm/gr6.pdf>
5. <https://core.ac.uk/download/pdf/52072275.pdf>

The first step to our modeling started with having our “control” be the p-values of the best cointegrated stocks and eventual net profit made using a simple z-score-based tradable strategy by simply randomly selecting n stocks from Russell 3000 and running Scipy.stats’s cointegration test on them. Our general hypothesis was that our “control” would perform worse in terms of cointegration p-values as well as net profit.

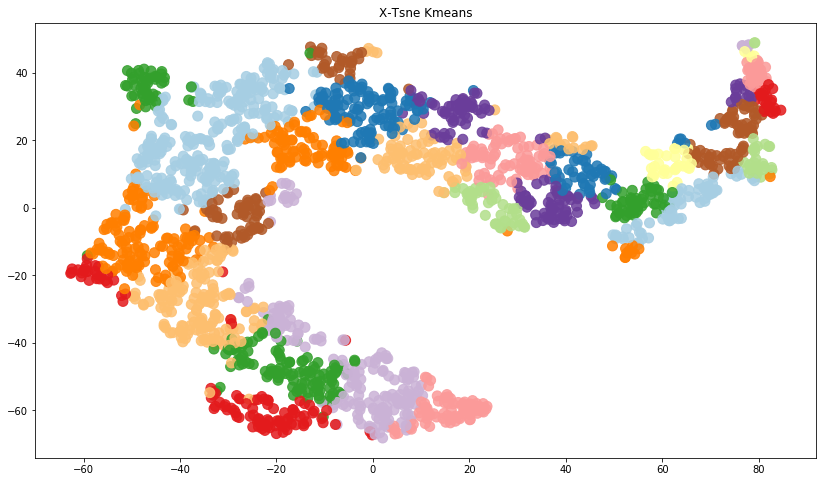
The first part of our modeling involved using a PCA clustering strategy to cluster all the Russell 3000 stocks into n clusters. To do this, we used the scikit-learn decomposing subpackage by fit transforming our dataset. We experimented with several component parameters ranging from 2 to 20, and ended up using 7 as our value. So essentially, PCA will allow us to reduce the dimensionality of our merged dataset from 743 components/features to 7 principal components that can numerically and mathematically represent the original 743 components. Now, our dataset’s shape is (2118, 7). The next step of the model is to determine the optimal number of clusters to assign all 2118 stocks to. Each cluster will essentially represent stocks that are similar in nature based on the variances in their principal components. In order to do this, we used a k-means clustering algorithm using scikit-learn. We first need to determine which number of clusters has the optimal and lowest variance, but at the same time limit the total number of clusters. The Russell 3000 has 75 different sectors, so we essentially calculate the inertia for each cluster size, which is the sum of squared distances to their respective cluster centroid. This acts as the “variance”, which we can then plot for each cluster size. We can then visualize the variance through an elbow graph. It’s called an elbow graph because the variances decrease as the size of the number of clusters increases, aka logarithmically.



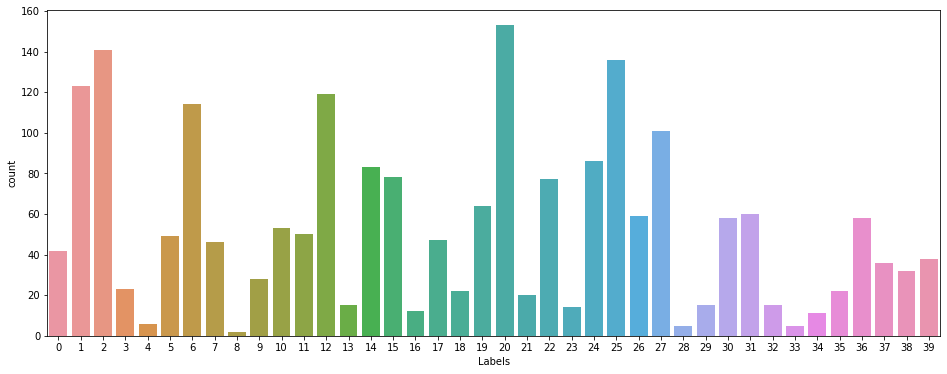
Through this Elbow graph, we can see that the inertia (variance) of each number of clusters is inversely proportional to the size of the number of clusters. We want to choose a number of clusters that has a low variance. For our data, we can see through this graph that the variance/inertia for the data points that fit the various cluster centers decreases to nearly 0 after reaching a size of 10 and going till 70. We decided to use a variety of cluster sizes (between 20 and 60) for the rest of our model, but ended up using 40 as our value for the number of clusters because it’s a centric value in terms of all the cluster sizes that have close to 0 variance/inertia, and at the same time isn’t a super large size for number of clusters or a super small size. After deciding on using 40 as our number of clusters, we then ran the K-means algorithm once again but specifying 40 as our number of clusters. We used the elkan algorithm as one of our hyperparameters because it is said to be more efficient on data that has well-defined clusters already, and we assumed that stock data that is already based on industry where the financials of industries are already pretty similar would mean that our data would have well-defined clusters. After fitting this model to our data, we had a final table that consisted of all the stocks in our dataset, and the respective cluster it would belong to. The clusters were arbitrarily labeled 0-39 for our purposes. Finally, we want to be able to visualize the clusters, so we used a t-Distributed Stochastic Neighbor Embedding (t-SNE) model. We actually ran this model on our dimensionality-reduced dataset (PCA) that only had 7 features for each stock instead of 743. The reason we wanted to use the t-SNE model to see the clusters and where each stock would fall is because t-SNE cares about small and localized similarities between objects whereas PCA cares about the opposite and does this through linear dimensionality reduction, trying to maximize the variance. Therefore, t-SNE visualizations are easier to see and easier to understand. So, after transforming the PCA’ed data using this t-SNE model, we were able to find the 2 t-SNE components for each stock for each label:



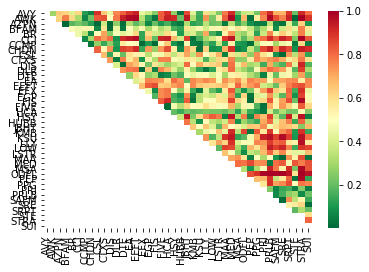
After this, we decided to plot each stock using its t-SNE components, and group them by their labels to see if our clusters actually make sense and are actually clustered together properly :



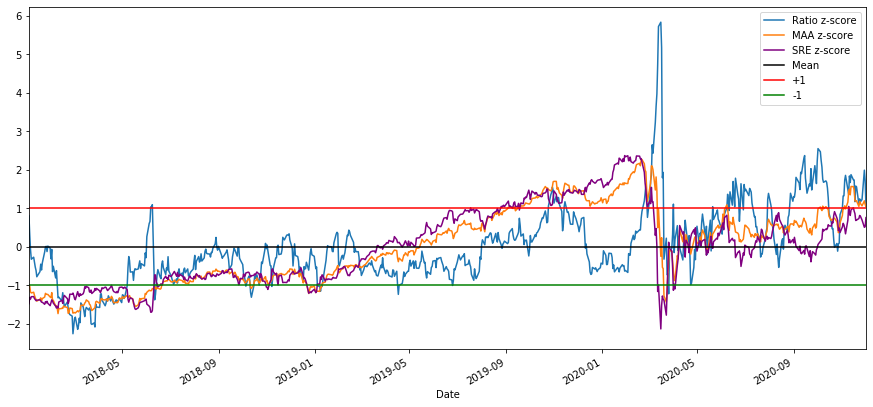
This cluster plot has every stock in our dataset plotted based on its k-SNE components, which themselves were based on the 7 features that the original data was dimensionally reduced from 743 using PCA. We can clearly see that stocks with similar points on the graph are grouped relatively close to others, meaning that our clustering scheme makes sense. We can see some discrepancies of course, as certain color clusters exist in multiple places, but we don’t see a hodgepodge of random colors within a given cluster, which is good. The next step is to determine which cluster label we want to run our cointegration test on. So, to get a bit more insight on how many stocks exist in each of the 40 clusters, we made a simple bar graph that specifies exactly how many stocks ended up being in each of the 40 clusters. Based on this visualization below, we can see a wide variance in the sizes of each cluster. For our project, since we can assume that we don’t actually have any details on what each cluster represents, only that each cluster contains stocks that should be intrinsically similar, we decided to choose cluster label 0 as the cluster to run the cointegration test on because 0 only contains around 40 stocks, which seemed like a number that wasn’t too high or too low. We felt that if a cluster label had 100 or more stocks, then that could possibly mean that the principal components of that cluster had a larger variance. The opposite goes for the smaller clusters. Therefore, we decided to choose something in the middle.



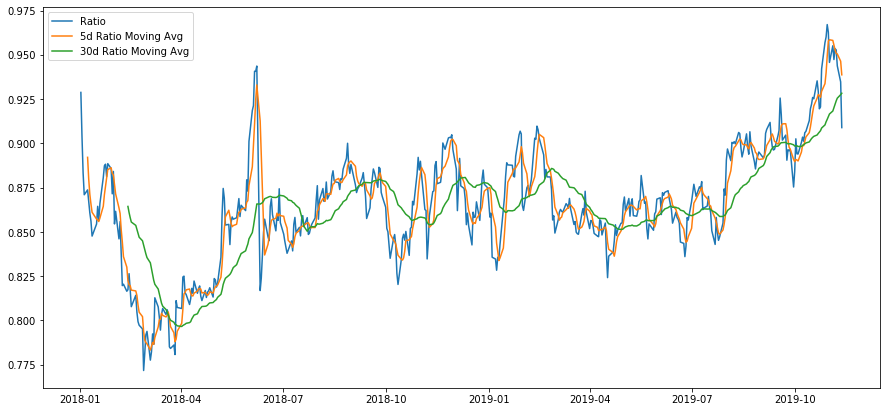
After deciding on choosing the 0th label cluster to run our cointegration test on, we ran the statsmodels coint test on all the pairs of 40 stocks within the 0th cluster and outputted a heat map based on the p-values that each pair received as an output from the cointegration test. Through this heatmap, we can spot many pairs that have low p-values, which is ideal, but also several pairs with higher p-values that we wouldn’t want to use for our project. Therefore, after getting the p-values for all possible pairs in the 0th cluster, we decided to condition the cointegrated pairs on if the p-value of the pair was less than .001. After doing this, we found that one pair: MAA - Mid America Apartment Communities and SRE - Sempra Energy passed all of our conditions and test, and this is the pair we decided to use for testing using a simple trading strategy:



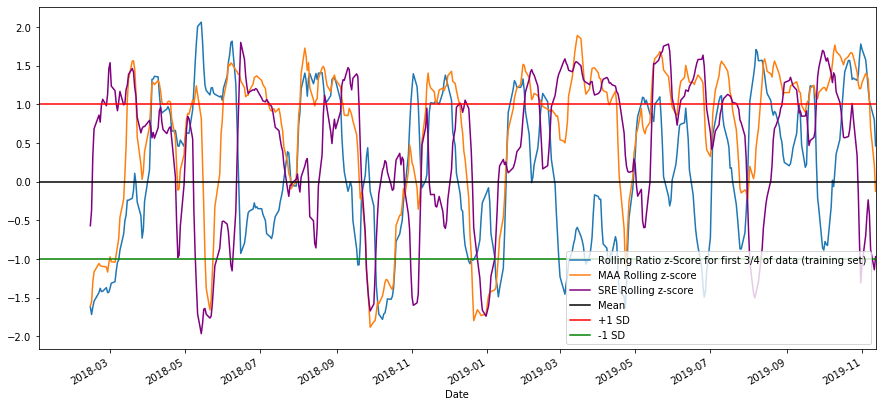
Now, the next part of our project pipeline is building a simple tradable strategy to test our pair on. We first found the ratio of the two stocks in our pair, and then normalized the closing prices of our data by having the time series’ in terms of z-score. We understand that by doing this, we are assuming a normal distribution for these stocks and we understand that normally, stocks don’t always follow a normal distribution because there’s always all sorts of market fluctuations that occur. However, we decided to normalize our data so that we could get some sense of how our pair looks in terms of its mean-reverting behavior and the behavior of a ratio and to see if the z-score ratio would fit in between any 2 z-scores. In the below graph, we plotted the ratio of the two stocks closing prices that were then converted to a z-score, and we did the same for both the MAA and SRE stocks. Then, we also plotted the mean of the ratio as well as where the -1 and +1 SD’s would be on our plot. We can already see that generally, these two stocks mostly fit in between -1 and 1 z-score except for the spike in the ratio because of the pandemic. Therefore, our assumption of normalizing our data actually makes sense. In order to determine signals, however, and get rid of daily discrepancies and market fluctuations, we need to calculate moving averages for our z-score ratio in order to smooth out fluctuations and get a better understanding of how the ratio fits in terms of z-scores.



Instead of just calculating the regular z-score graphs, we also wanted to calculate two moving averages to use in our z-score calculations as well. We settled on using the 5 day rolling average as the current price ratio, and the 30 day rolling average for the mean and SD. The reason we chose the 5 day for the current price ratio is because within a trading week (5 days), there can always be various discrepancies and severe fluctuations in prices of an asset, and this would severely affect signals for pairs trading. Therefore, by using the 5 day rolling average, we can smooth out any discrepancies. This can be seen explicitly during the huge spike in ratio z-score in the above graph due to the covid-19 pandemic. We want to smooth things like that out for the “current price”. Then, we chose a 30-day moving average as our mean and SD because our data only spans about 2 years. Therefore, a 30-day moving average seems reasonable to capture average price ratios and the SD of price ratios better than the average of the entire dataset over two years. Below, you can see the ratio itself (non-normalized) as well as the 5 and 30 day moving averages to understand how the smoothing process helps eliminate discrepancies and huge fluctuations in the ratio of the prices of our two stocks in our pair:

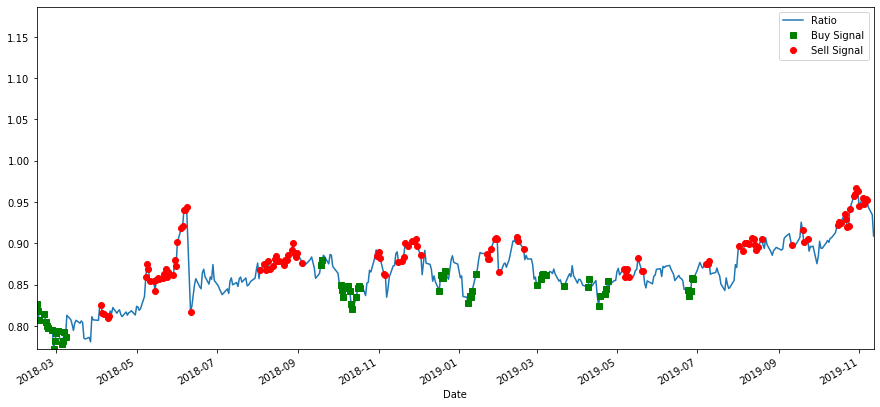


After computing the new moving-average-based z-scores for our ratio, for MAA, and for SRe, we get an even better understanding of the mean-reverting nature of the rolling average ratio as well as the fact that it normally stays within 1 and -1, which is a good sign. In this graph, as well as the above graphs we only used the first ¾ of our data on closing prices and use this as our “training data”. By training data, we mean data that will allow us to determine what day rolling averages to use to calculate our rolling z-score, as well as what z-score values to use for our simple trading strategy. So, after looking at the visualizations from our “training data”, we see that using a 5-day and 30-day rolling average to compute the rolling z-score seems like a good fit, and we also see that using -1 and 1 z-score as our baseline for signals seems like a good fit. In the results section, we can then implement a simple trading strategy and test out these values on our “test” dataset, which is just the last ¼ of our closing price datasets for the two stocks:

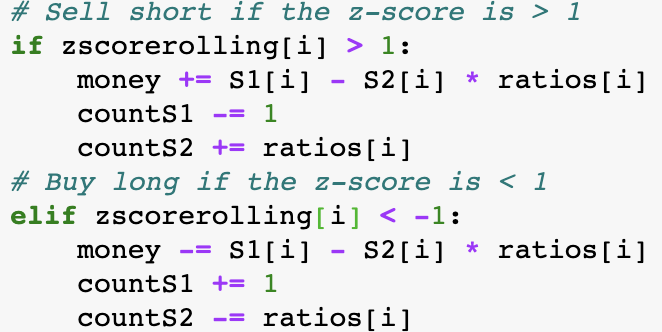


1. **Results**

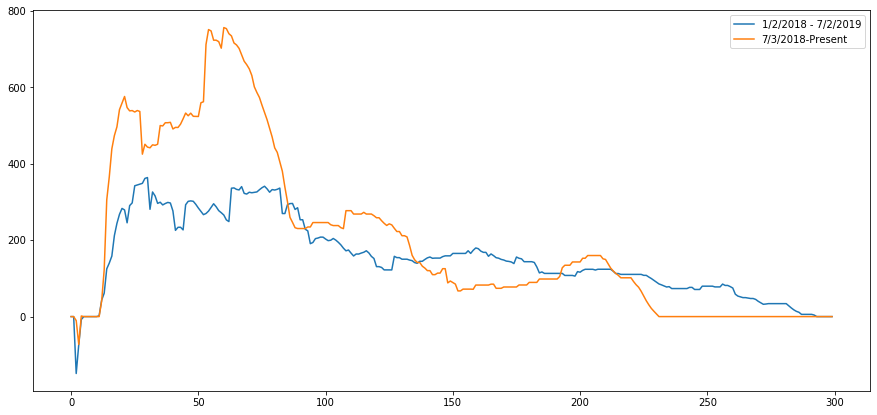
After calculating the rolling ratio z-score for MAA and SRE, landing on using 5-day and 30-day moving averages to calculate the rolling ratio z-score, and landing on using -1 and 1 z-scores as our thresholds for signals, we can visualize what the signals would look like. This plot uses the “training” data and plots the buy and sell signals based on if the z-score exceeds 1 or -1:



Looking at the graph, it seems like the buy and sell signals make sense. When the ratio is increasing, we can see a plethora of sell signals popping up, but when the ratio is low or starting to increase, we can see a plethora of buy signals. Now, we can test our strategy and quantify how much money we could potentially make. We decided to implement a very simple trading strategy. The way it works is after computing the rolling z-score of the ratio of the two stocks, if that z-score is larger than 1, we sell short by subtracting the price of the second stock \* the hedge ratio from the price of the first stock, and adding that value to our total money. If the z-score is lower than -1, we buy long by subtracting the price of the second stock \* the hedge ratio from the price of the first stock, subtracting that from the value of our total money:



Using this simple strategy, we can compute how much money we would make with this pair over any period of time we want to choose (from our dataset), and we can also determine the optimal time period in days using the argmax function. So, we found that for our stocks MAA and SRE, the optimal trading window on the “training” data, or the data for the first ¾ of our data, was 30. For the “test” data, it was 60.



We can see that in the first 30 days (x-axis depicting the window in days), we make around 400 dollars on the “training” data (blue line) if we start with 0. On the “test” data however, we can make around 800 dollars in the first 60 days. It was surprising that the test data actually performed better than the train data because we optimized the 5-day and 30-day rolling averages as well as the +1, -1 z-score thresholds using the training data. With this graph, we can also see that after a significant period of time, our returns revert back to 0, so this means that our pair is better suited for pairs trading in shorter periods of time!

1. **Conclusion**

In conclusion, it seems that the pair of stocks MAA and SRE that we got through using a dataset with financial health and metrics as well as closing price data and then ran through a clustering paradigm actually resulted in making net profit, which performs better than simply choosing two random stocks to run a cointegration test on. Overall, using a PCA, k-means clustering, and t-SNE paradigm with a robust and diverse dataset allowed us to identify very prolific stock pairs in the Russell 3000 which can result in net profit within a short period of time.

1. **Future Work**

One main addition that we would like to make to this project in the future is to make the entire pipeline more automated. Because we used Jupyter notebooks, it’s harder to completely automate the entire pipeline. We would also like to implement some sort of gridsearch on this pipeline so we can experiment with hundreds to thousands of different combinations of cluster sizes, pairs, and stocks from other indexes. This would obviously take a lot more computation power and time than what we need for our project right now. Finally, the current tradable strategy we have is very simple. We could potentially use better unsupervised learning algorithms to determine the z-score cutoff for shorting and longing stocks rather than the simple implementation we have right now.