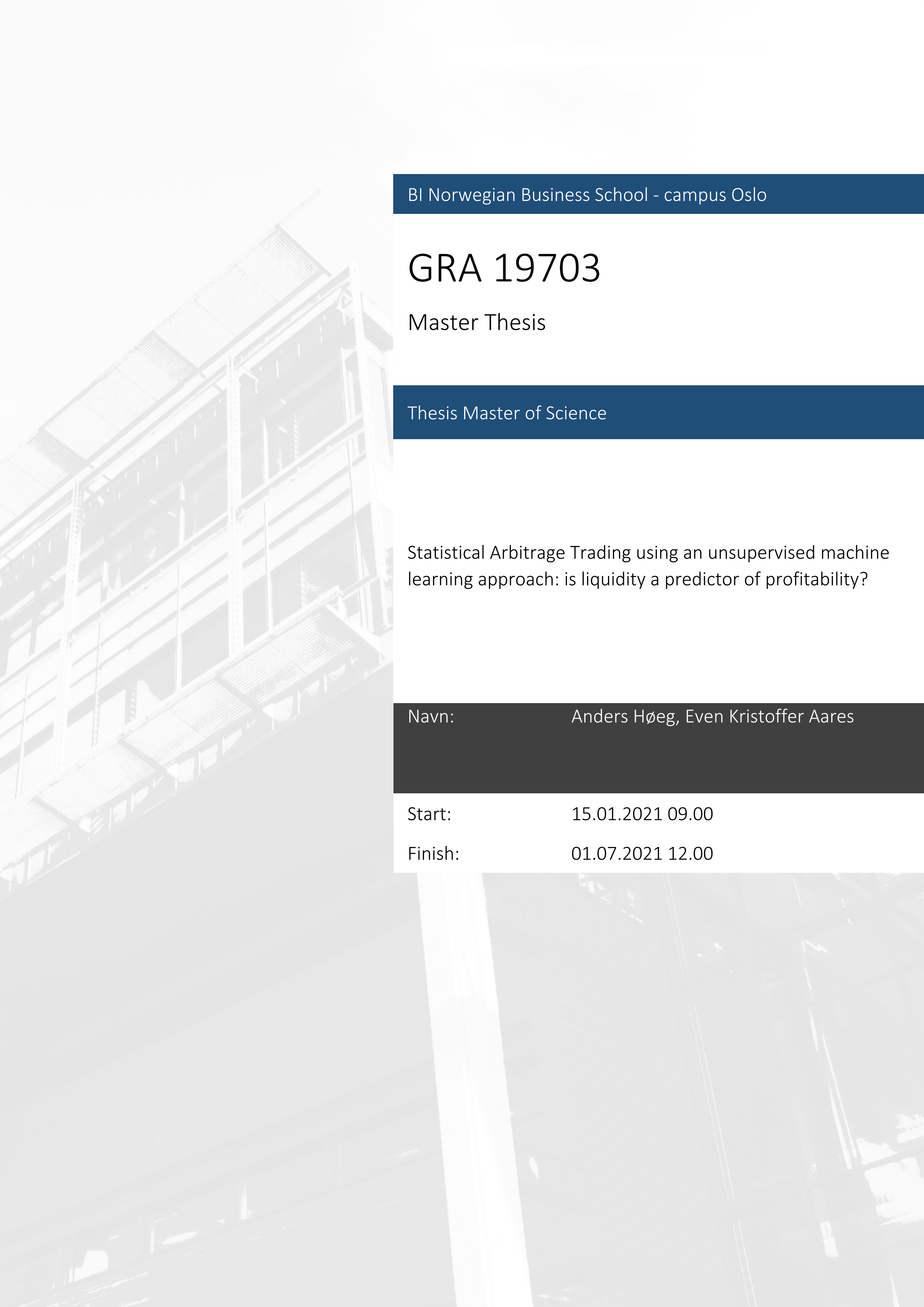


GRA 19703

Master Thesis

Thesis Master of Science

Statistical Arbitrage Trading using an unsupervised machine learning approach: is liquidity a predictor of profitability?

Navn:	Anders Høeg, Even Kristoffer Aares
Start:	15.01.2021 09.00
Finish:	01.07.2021 12.00

Statistical Arbitrage using an unsupervised machine learning approach: is liquidity a predictor of profitability?

Master Thesis

By

Anders Høeg & Even Aares

MSc in Business

ABSTRACT

We test a statistical arbitrage trading strategy, pairs trading, using daily closing prices covering the period 2000 – 2019. Stocks are clustered using an unsupervised machine learning approach and cointegrated stocks from each cluster are then paired. The strategy does not prove to be profitable on S&P500 stocks once adjusted for transaction costs. Conversely, the strategy appears to be profitable on the OSE obtaining annualized excess returns of 22% and a Sharpe Ratio of 0.84 after adjusting for both explicit and implicit transaction costs. We investigate whether a difference in the liquidity can explain why the strategy is more profitable on OSE, and provide evidence suggesting that pairs trading profits are closely related to the liquidity of the stocks traded.

Supervisor:

Costas Xiouros

Acknowledgements

We would like to thank our supervisor, Costas Xiouros, for being a steady sparring partner throughout this project and for offering useful insights. We also thank Eirik Kielland for being a valuable resource for programming related issues.

Table of Contents

List of Figures

List of Tables

1. Introduction and Motivation

1.1 Background

Modern quantitative finance and trading are often said to be dominated by mathematicians, statisticians, physicists, and computer engineers. In the US equity markets, traditional fundamental investors are believed to account for only ten percent of the trading volume (JP Morgan, 2019). With rapid advancements within data analysis and computing power in the last decades, we have seen a similar growth within quantitative trading, leading to new strategies and improvements to existing strategies.

The concept of pairs trading dates back to the 1950s when Alfred Winslow Jones employed the idea of relative value arbitrage in the world's first hedge fund. The strategy was later developed by a team of mathematicians, physicists, and data scientists at Morgan Stanley in the mid-1980s. The initial proprietary nature of the strategy, which falls under the category of statistical arbitrage strategies, made it a popular research topic. Several papers have been published since the early 2000s.

The notion of pairs trading is fairly simple. It entails the buying and selling of two highly correlated securities, exploiting short-term deviations in the relative price between them. As the financial markets are subject to disruptive technological changes, we observe researchers applying more sophisticated versions of these trading strategies. While machine learning itself is not a new concept, literature on its application in statistical arbitrage trading is limited and needs to be further explored. We build on existing literature and employ an unsupervised clustering algorithm to identify stocks with similar risk characteristics suitable for pairs trading.

While the current literature is mainly centered around the question of whether pairs trading is still a profitable trading strategy, we are particularly interested in how the strategy performs in markets with different characteristics regarding liquidity and size. We specifically study and compare our findings in the US and the Norwegian equity market, with focus on how the market liquidity affects the profitability of the strategy.

1.2 Hypothesis

In this paper, we study the profitability of a statistical arbitrage trading strategy with the implementation of machine learning through empirical research. We start by formulating the following hypothesis:

H_0: *A pairs trading strategy using machine learning does not produce positive excess returns*

H_1: *A pairs trading strategy using machine learning produce positive excess returns*

We are particularly interested in how liquidity affects the returns of this strategy and study the performance in both the US and the Norwegian stock market as the two markets are quite different in terms of liquidity. Based on this, we can formulate the following hypothesis:

H_0: *Pairs trading is not more profitable in markets with lower liquidity*

H_1: *Pairs trading is more profitable in markets with lower liquidity*

1.3 Contribution

We explore the use of unsupervised machine learning in pairs trading, a field that is not widely researched. Further, this paper study the performance of the same pairs trading strategy in two markets with different characteristics. This type of analysis does not appear to exist within the most common literature in this field, and is usful to shed some light on how the strategy performs under varuous conditions, such as market liquidity. Finally, the use of machine learning techniques on statistical arbitrage in the Norwegian equity market is not something that is heavily covered in existing literature. This paper paper therefore helps to fill this gap of missing research.

2. Theoretical Framework

This chapter introduces the theoretical framework that the paper builds on. We start by briefly covering the concept of market efficiency before introducing the Arbitrage Pricing Theory. Both serve an important role when discussing the concept of statistical arbitrage and its theoretical implications.

2.1 Market Efficiency

Financial time series are notoriously difficult to model and predict due to its inherent characteristics and nature. Kendall and Hill (1953) discovered, to their surprise, that they were unable to identify any predictable patterns in stock prices, leading to the conclusion that prices were as likely to go up as there were to go down. Researchers soon realized that these findings are evidence of a well-functioning and efficient market (Bodie et al., 2018). If prices are determined rationally by investors, only new information will lead to price changes. By definition, new information must be impossible to predict. Stock prices are thus expected to follow a random walk, meaning that price changes should be random and unpredictable[1].

An efficient market is one in which prices always fully reflect all available information (Fama, 1970). Eugene Fama (1970) specifies three forms of efficiency: weak, semi-strong, and strong form efficiency. In the weak form efficiency, the market prices reflect all historical price information. In such environments, predictive tools based on the analysis of historical data would fail. Semi-strong form efficiency assumes that prices reflect all publicly available information, implying that no abnormal returns can be earned by analyzing public financial data and relevant news. In a strong form efficient market, prices reflect all public *and* private information, such that no entity with monopolistic information of the respective asset profits from that information (Fama, 1970). The inability to predict stock prices assumed in the Efficient Market Hypothesis (EMH) implies that active trading does not result in greater returns than a passive market portfolio. Statistical arbitrage traders who seek to profit from relative mispricings by doing a frequent number of trades in many securities simultaneously would only generate significant

[1] In reality, stock prices may actually follow a submartingale with positive expected price changes as compensation for time value of money and systematic risk.

transaction costs (TC) while failing to outperform the market. Assuming that EMH holds, the Capital Asset Pricing Model (CAPM) states that one should not be compensated for idiosyncratic risk, only systematic risk. Looking at the CAPM formula:

$$\mathbb{E}[R_i] = R_f + \beta_i * (\mathbb{E}[R_m] - R_f) \tag{2.1}$$

β_i represents the systematic risk that cannot be reduced through diversification. The expected returns of a security rely on the risk-free rate and the riskiness of the investment. Most statistical arbitrage trading strategies are *market neutral*, implying no systematic risk related to the investment and $\beta_i = 0$. Thus, the CAPM suggests that:

$$\mathbb{E}[R_i] = R_f \tag{2.2}$$

Statistical arbitrage should not be able to generate abnormal returns, i.e., returns in excess of what the CAPM predicts as it states that there should be no compensation for idiosyncratic risk.

Grossman & Stiglitz (1980) introduce a framework aimed at redefining the notion of efficient markets, arguing that in the case where EMH holds and information is costly, there is no equilibrium and competitive markets break down. They distinguish between informed (arbitrageurs) and uninformed market participants. The uninformed can expend resources to become informed, but since gaining information is costly, they should receive some compensation. As informed individuals trade based on their insights, information is conveyed to the uninformed through the price of the traded security.

Their model consists of two assets: a safe asset yielding the return R and a risky asset yielding a return of u. The return u is defined by:

$$u = \theta + \epsilon \tag{2.3}$$

where both θ and ϵ are random variables, but θ is observable at cost c while ϵ is not observable. The informed will be able to observe θ, which is the true value of the risky asset, while the uninformed will only be able to observe the price of the asset. The informed traders will adjust their demand for the asset based on θ and the risky asset's price P while the demand of the uninformed is only subject to P. An equilibrium can be described by the price function:

$$P_\lambda(\theta, x) \tag{2.4}$$

where x is the supply of the risky asset and λ is a given percentage of informed traders. They further argue that the price system reveals information to the uninformed but in an imperfect fashion. More specifically, the price system reveals the signal:

$$w_\lambda \equiv \theta - \frac{\alpha\sigma_\epsilon^2}{\lambda}(x - \mathbb{E}(x^*)) \tag{2.5}$$

where α is the coefficient of absolute risk aversion. For any given θ it follows that the price system reveals a noisy version of the asset's true value. An important implication of their theory is that if we assume the EMH holds, competitive markets will break down. Once $\sigma_\epsilon^2 = 0$, w_λ and the price of the risky asset reflects all existing information. As all information is observable through the price P, the informed traders no longer have the need to pay c to observe θ as he will do just as well as the uninformed who does not pay for the information. Because all informed traders will share this opinion, this cannot be an equilibrium. Opposite, if the entire fraction is uninformed, there will be an incentive to become informed as this will be profitable, which does not represent an equilibrium. It follows that there will have to be a fraction of informed traders between 0 and 1 depending on the cost of information, how informative the price system is (noise), and how informative the information obtained by the informed is.

The observation that statistical arbitrage is a widely used trading strategy where traders expense a significant amount to gain access to information suggests that this framework bears a close resemblance to the dynamics of the real world. The theory predicts that the market prices might not reflect all relevant information, specifically the type of information that is costly to obtain and privately possessed. This implies that for stocks whose price does not reflect all available information, the risk-adjusted return will be higher than that of other assets. This paper explores the use of machine learning in the process of identifying pairs to trade, a technique that may be able to extract such costly information that is not reflected in the prices. Given that these methods have become more prevalent in recent years, we should expect their ability to obtain this information to decline as time passes.

2.2 Arbitrage Pricing Theory

Arbitrage occurs when an investor can earn a risk-free profit without making a net investment (Bodie et al., 2018). More specifically, the investor is simultaneously buying and selling a security, exploiting a mispricing in the market. According to the Law of One Price (LOP), two assets with identical cash flows should have the same price in the market. Arbitrageurs enforce the LOP by exploiting deviations from the implied market price, buying the "undervalued" security, and selling the "overpriced" security. The selling- and buying pressure will force the price of the two securities to converge until the arbitrage opportunity is eliminated.

In order to exploit mispricing in traded securities, we need a framework that lets us identify deviations from their fair market price, thus leading to arbitrage opportunities. The Arbitrage Pricing Theory (APT) was developed by Ross (1976) as an alternative to the CAPM. The theory introduces the idea that an asset's expected return can be modeled as a linear function of several systematic risk factors. These factors can be economic indicators such as GDP growth and changes in inflation which will impact the risk of the asset, depending on the level of exposure to the factor and thus the expected return. An important implication is that any deviation from the expected price as determined by APT represents a temporary mispricing that will be arbitraged away by market participants such that its price is corrected, and the arbitrage opportunity ceases to exist. More specifically, the excess return of risky assets can be expressed as:

$$r_i = \beta_{o,i} + \sum_{j=1}^{n} \beta_{i,j} F_j + \epsilon_i \tag{2.6}$$

where:

$\beta_{0,i}$ is a constant for asset i

F_j is a systematic risk factor

$\beta_{i,j}$ is the sensitivity of asset i to factor j, called factor loading

ϵ_i is the risky assets idiosyncratic shock with mean zero.

If investors require compensation for taking certain types of risk (F_j), it follows that their expected return should be a compensation for their exposure to these risks. This also means that we can interpret the alpha obtained by regressing returns on these factors as risk-adjusted returns. Common systematic risk factors are for example those proposed in the three-factor model (Fama & French, 1993), such as $r_m - r_f$, SMB, and HML. HML refers to the return of stocks with high book-to-market ratios in excess of the return of stocks with low book-to-market ratios. Similarly, the SMB factor refers to the return of stocks with low market capitalization in excess of the return of stocks with high market capitalization. In the general case, $\mathbb{E}(F_j) = 0$, and we can express excess returns as:

$$\mathbb{E}(r_i) = \beta_{0,i} = \sum_{j=1}^{n} \beta_{i,j} \times \lambda_j \tag{2.7}$$

where λ_j is an expression for prices of risk, i.e., the expected returns of risk factors. Suppose we have a factor X such that $F_1 = R_X - \mathbb{E}(R_X)$ where $\mathbb{E}(R_X) = \lambda_X$ such that:

$$\mathbb{E}(r_i) = \beta_{0,i} = \beta_{i,X} \times \lambda_X \tag{2.8}$$

When pursuing a statistical arbitrage trading strategy, stocks that tend to move together are traded. Therefore, it is not unreasonable to argue that these stocks have similar exposure to the risk factors - that is, they have the same betas. According to the APT, it follows from equation (2.8) that they should also have the same β_0 and hence the same expected excess return. As a pairs trading strategy entails buying one of the stocks and shorting the other, the expected excess returns are actually zero in the case that APT holds true.

$r_m - r_f$ is a factor representing the market risk premium and is similar to what we have in the traditional CAPM, which can be considered a special case of the APT where the only risk factor is the systematic market risk. In simple terms, we can express F_1 as:

$$F_1 = (R_m - R_f) - \mathbb{E}(\mathrm{R_m} - R_f) \tag{2.9}$$

Excess return can followingly be defined as:

$$r_i = R_i - R_f = \beta_{0,i} + \beta_i \times F_1 + \epsilon_i \tag{2.10}$$

Assuming that markets are efficient, $\mathbb{E}(F_1) = 0$:

$$\mathbb{E}(r_i) = \beta_{0,i} = \beta_i \times \mathbb{E}(R_m - R_f) \tag{2.11}$$

Again, in the event that the APT and the CAPM hold true, a statistical arbitrage strategy would produce zero excess returns. Another important implication of the APT is the assumption that idiosyncratic risk is diversifiable. This means that when trading a single pair, while the returns are neutral to the priced risks (F_j), the trader will be exposed to the idiosyncratic risk of the securities. If the trader instead holds a well-diversified portfolio of pairs, the idiosyncratic risk will be negligible considering the portfolio as a whole.

3. Existing Literature on Statistical Arbitrage

This chapter reviews existing literature on statistical arbitrage and pairs trading. We briefly cover the origins of statistical arbitrage trading and its development, before introducing some of the main results found by researchers. Finally, the chapter covers research specifically concerning liquidity and how it relates to pairs trading profits.

3.1 Statistical arbitrage

Statistical arbitrage is believed to have been formalized as a concept at Morgan Stanley in the 1980s and has become a widely used strategy among banks, hedge funds, and proprietary trading desks. Today, Statistical Arbitrage is an umbrella term covering various quantitative trading strategies based on statistical and mathematical models, and where the trades are automatically executed by algorithms.

In contrast to the original concept of arbitrage, statistical arbitrage does not offer entirely risk-free profits. As discussed in chapter 2.2, the returns from statistical arbitrage are directly linked to the idiosyncratic risk that remains. This means that it can only be thought of as an arbitrage if the pair traded is part of a well-diversified portfolio where the idiosyncratic risk is diversified away. A pair trade in isolation can, however, be very risky as one is exposed to events such as M&A activity, defaults, or macroeconomic events, often referred to as fundamental risk (Do & Faff, 2010). The most notable example in this regard is the former hedge fund Long Term Capital Management (LTCM) which utilized statistical arbitrage trading strategies and significantly leveraged a few correlated bets. Following Russia's default on foreign debt in 1998, the fund nearly collapsed and stood to set off a global financial crisis. LTCM was eventually bailed out by some of the largest Wall Street banks and shut down in 2000.

The concept of statistical arbitrage opportunities was first introduced by Bondarenko (2003) and defined as "a zero-cost trading strategy for which (i) the expected payoff is positive, and (ii) the conditional expected payoff in each final state of the economy is nonnegative". In other words, a statistical arbitrage opportunity can result in a negative payoff; however, the average expected payoff in each final state must be nonnegative. According to Becker (2012), the concept often refers to “highly technical short-term mean reversion strategies involving a large number of securities, very short holding periods and substantial computational, trading, and IT infrastructure”. The strategies are usually market neutral with a zero beta to the market and often involve some trading signal based on a mean-reverting relationship between securities. More formally, statistical arbitrage strategies attempt to exploit some mispricing where price relationships are true in expectations in the long run (Becker, 2012):

$$\mathbb{E}(X \times N) > 0 \tag{3.1}$$

where X denotes the payoff matrix, and N denotes the quantities involved. The average payoff will also have to be nonnegative in the final state:

$$\sum_{i=0}^{\infty}(X_i \times N_i) \geq 0 \tag{3.2}$$

The is the time-dimension of statistical arbitrage trading. The idiosyncratic risk can also be diversified away by doing countless trades, meaning that equation (3.2) should always hold as the number of trades (N_i) tends to infinity.

After nearly two decades of high profits, returns have almost diminished for standard statistical arbitrage strategies (Pole, 2007). This does not come as a surprise considering the rapid technological development and increased computing power seen during the same period. This does, however, not imply that no statistical arbitrage strategy is unprofitable today, but rather that the strategies and underlying algorithms have become more complex as the technology needed has become available to a broader audience. This observed development can be interpreted in light of the framework of Grossman and Stiglitz (1980), where they distinguish between informed and uninformed market participants. The informed has spent money to gather information, which is costly, and profits from trading with the uninformed. As technology develops, statistical arbitrage strategies become available to a larger population (i.e., information is cheaper, and more people can access it). Thus, statistical arbitrage opportunities should have been diminishing over time.

3.2 Pairs trading

Pairs Trading is likely the most widely used statistical arbitrage trading strategy and is said to be the predecessor of statistical arbitrage trading strategies. The concept of pairs trading itself is relatively simple: find two securities that historically have been highly correlated and monitor the price difference (spread) between them. If the spread widens, you short the "winner" and buy the "loser" in anticipation that the spread converges and thus make a profit. An attractive feature of the strategy is the *market neutrality*, meaning that profits can be made regardless of how the market moves. While this concept has been around for a long time, it was Nunzio Tartaglia, a quant at Morgan Stanley, that put together a team of highly skilled mathematicians, physicists, and computer scientists in the search for profitable quantitative trading strategies based purely on algorithms with minimal human intervention (Gatev et al., 2006). Among these was a version of the pairs trading strategy, which proved to be highly profitable for the group.

While the strategy itself is relatively simple, extensive research has been done on how these pairs should be selected and how the trading signals should be constructed. While Gatev et al. (2006) introduced a *distance approach* based on their interaction with traders, the *cointegration approach* has also become popular among researchers. A large portion of the research on the subject concerns pairs trading with single stocks, but the concept is also applied to other asset classes such as commodities, fixed income securities, ETFs, cryptocurrencies, and derivatives. Similarly, the strategy can also be used to trade on temporary mispricings between indices and a basket of index constituents.

3.3 Literature on pairs trading

The widely cited study from Gatev et al. (1999;2006) use daily closing prices in the period 1962 – 2002 and find that pairs trading in liquid US stocks has delivered annualized excess returns of 11% for the top 5-20 pairs portfolios when accounting for trading costs and fees. The authors apply a distance approach to identify pairs that tend to be highly correlated. They use a 12-month formation period where pairs are matched followed by 6 months of trading - a setup that appears to be the norm in much of the research on this topic. Their trading rule builds on a relatively simple rule specifying that a position is opened if the spread between a pair of securities diverges by more than two standard deviations from its historical mean. Gatev et al. (2006) also implement a one-day waiting period form receiving theiur trading signal to actually trading to account for the implied bid-ask spread . By opening and closing a position the day after they are signaled to trade, the average monthly excess returns drop by roughly 36%, but are still positive and significant. Although providing robust results, they also find the strategy to be less profitable in recent years. One explanation could be an increased activity from hedge funds and other traders pursuing the strategy, however, the paper argues that the abnormal returns are a compensation for risk given to arbitrageurs for enforcing the "Law of One Price" (LOP). Findings support this statement as the raw returns have fallen but the risk-adjusted returns are consistent throughout the period.

B. Do & Faff (2010) replicate the study of Gatev et al. (1999;2006) and find similar results. They extend the original sample and find that the trend of decline in profitability has continued in recent years. However, the authors find that higher hedge fund activity and increased efficiency are only partly responsible for the declining profits, arguing that worsening arbitrage risk accounts for as much as 70 percent of the decline in profits. Arbitrage risks refer to fundamental risk, noise-trader risk, and synchronization risk. Fundamental risk involves unexpected events affecting the individual securities and thus the spread. In contrast, noise-trader risk refers to traders' behaviour that may seem irrational to other market participants, but in reality can be exaplined by several factors. An example would be a trader that requires liquidity and is forced to liquidate some positions. This might widen the spread between a pair and deter arbitrage activity. Synchronization risks address the issue of the timing of arbitrageurs and how fast a mispricing is corrected. Interestingly, they also report that pairs trading profits are particularly strong in periods of market turmoil, such as the financial crisis in 2007-2009.

In their analysis, Engle and Granger (1987) observe that some variables tend to exhibit a long-term relationship. Followingly, they went on to formalize a test to identify whether variables are cointegrated, i.e., whether there exists a long-term relationship between variables. Based on Engle and Granger's work, Vidyamurthy (2004) constructed the cointegration framework for pairs trading. Vidyamurthy did not present any empirical results in his book, but other researchers widely use his approach. Caldeira & Moura (2013) obtain annual excess returns before TC of

16.38% and an SR of 1.34 when performing a cointegration-based pairs trading strategy in the timespan 2005-2012 on the Sao Paulo stock exchange. Similarly, Rad et al. (2016) apply the cointegration method on the US equity market in the timespan 1962-2014. They obtain annual excess returns of 10.69% and an SR of 0.77 before TC. The alphas were both positive and statistically significant at a 1% levelin the two studies. Even though the strategy shows robustness given the long trading horizon, they both observe a slight decline in trading opportunities in more recent years. However, when comparing the cointegration method to the distance method, they find that cointegration is superior during turbulent market conditions.

Clegg & Krauss (2018) obtain annual excess returns of 12% after TC in the US stock market. Findings suggest that a Partial Cointegration (PCI) method, which is a weakening cointegration where one allows the residuals to contain a mean-reverting and a random-walk component (Clegg & Krauss, 2018), outperforms distance-based pairs trading used by Gatev et al. (2006). Similar to the previous studies, the authors also found that performance has declined over the years, which they argue is due to advancements in pairs trading research.

Following technological developments and increased application of data science in finance, trading models and pairs trading strategies have become more complex. Machine Learning is becoming increasingly popular as the method is well suited for handling large quantities of data and may discover patterns not evident to the naked eye. Avellaneda & Lee (2008) use a Principal Component Analysis (PCA) to extract common risk factors from their universe of securities, allowing for an efficient way to identify potential pairs. The authors find that a PCA strategy on sector ETFs in the US equities market produces an average annual Sharpe ratio (SR) of 1.44 after TC over the period 1997 to 2007, although with a lower level of profitability in later years. Building on the PCA approach used by Avellaneda and Lee (2008), Sarmento and Horta (2020) use a PCA to extract common risk factors from their universe of securities, which they further feed into a clustering algorithm, making it easier to find potentially profitable pairs. This has proven to be advantageous as the authors obtain an annual SR of 3.79 and 86% profitable pairs when clustering, and an annual SR of 3.58 and 79% profitable pairs when performing no clustering (all results before TC), reflecting the robustness of machine learning tools in pairs trading.

3.4 Impact of liquidity on pairs trading profits

Understanding the market dynamics is crucial for any investor or trader. Naturally, there are differences between stock markets in different countries concerning size, liquidity, regulations, laws, etc. A typical feature of a pairs trading strategy is a relatively high frequency of opening and closing positions, relying on many but small returns each time. Thus, it is particularly important to have the ability to trade at observed prices without heavily impacting the market. Næs et al. (2008) expressed that “A market is said to be liquid if traders can quickly buy or sell large numbers of shares at low transaction costs with little price impact”. This means that there exist four dimensions to liquidity. 1) The pace at which one can open/close a position, 2) the volume that can be traded, 3) the size of the spreads and fees, and 4) to which degree the respective stock prices are impacted by a trade. When researching the development of the Oslo Stock Exchange (OSE) liquidity in the timespan 1980-2007, Næs et al. (2008) find significant improvements over the years when testing for all four dimensions. There were considerable differences in firms depending on their market capitalization, where small-cap firms had the biggest improvements in liquidity. In the OSE, the relative bid-ask spreads have declined slightly in the last decades before stabilizing at roughly 4% in the 2000s (Naes et al., 2011). Even though this implies that the Norwegian stock market is becoming *more* liquid, the spreads are still large compared to the US stock market. The average NYSE relative bid-ask spreads in the early 2000s was 1.6% (Naes et al., 2011) indicating a clear difference in market liquidity.

It is reasonable to assume that the market liquidity could influence the actual performance of a pairs trading strategy. A theoretical study might assume that orders can be executed at the observed closing prices. In reality, both the depth of the order book and the bid-ask spread could significantly impact your trading profits. Another aspect is that temporary mispricing in the market is likely to be corrected slower in an illiquid market than in highly liquid markets. As less liquid markets tend to have larger spreads between bid and ask quotes, it could also be the case that what appears to be mispricing, in reality, reflects higher trading costs. Broussard and Vaihekoski (2012) study the profitability in the Finnish stock market, which is assumed to be less liquid than the US market. They found that the strategy delivered annual excess returns of 49.6% before TC using a fully invested portfolio and no lag. Even when adding a one-day lag, due to implications that may arise in low liquidity markets, they found annual excess returns of 11.9%, which is higher than what Gatev found (8.9%). Comparing that to what Gatev found in the US with no lag (15.7%), they see major benefits of the larger bid-ask spreads "that can cause a spread bounce resulting in jumps in the closing price which may be reversed the following day".

The empirical results support our hypothesis that the implementation of pairs trading in low-liquidity markets could be more profitable due to a higher frequency of relative mispricings and slower price convergence. If the Norwegian stock market is much less liquid than the S&P500, as we anticipate, we should expect some of these effects found in previous studies to present also in the Norwegian market.

4. Methodology

This chapter discusses the data used in our analysis and the methods used to implement a pairs trading strategy using an unsupervised machine learning approach. Chapter 4.1 provides an overview of the research design and each stage in the process. All computations are done using Python, and a copy of the code can be found in appendix C1 and C2.

4.1 Research design

Figure 4.1 provides an overview of our research design comprised of five stages. The process begins with stage 1, where we use a principal component analysis to reduce the dimensions of our dataset. The principal components (PCs) are used as input in the clustering algorithm applied in stage 2. The goal of the clustering algorithm is to group the stocks in our dataset in such a way that stocks with similar systematic risk are grouped together and form what we call a "cluster". Once stage 2 is completed and clusters are formed, we move to stage 3 where we try to identify pairs of stocks that exhibit a mean-reverting relationship. This is done by testing all pairs in each cluster for cointegration. Once we have identified the cointegrated pairs in our clusters, we continue to stage 4 where we implement the pairs trading strategy on the cointegrated pairs from each cluster on in-sample data and measure the performance. Finally, we test the strategy using the identified pairs on out-of-sample data to simulate the performance of the strategy. The returns are then analyzed and adjusted for both explicit and implicit transaction costs.

Figure 4.1: *Research design overview*

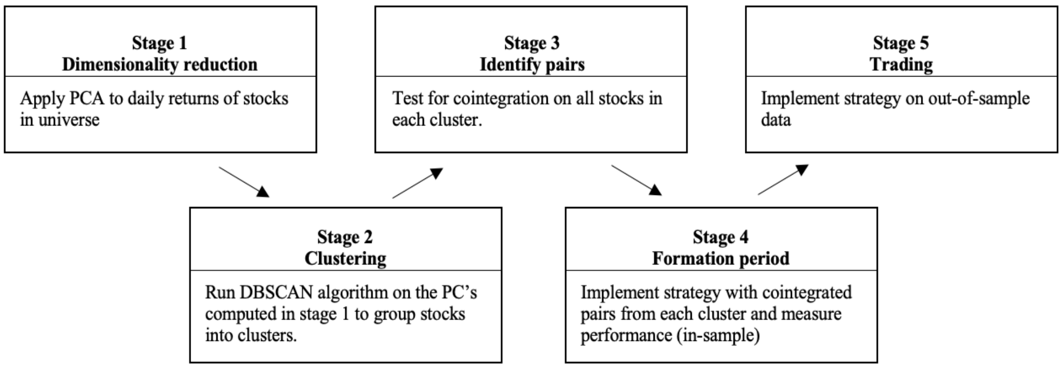

Stage 1
Dimensionality reduction
Apply PCA to daily returns of stocks in universe
Stage 2
Clustering
Run DBSCAN algorithm on the PC's computed in stage 1 to group stocks into clusters.
Stage 3
Identify pairs
Test for cointegration on all stocks in each cluster.
Stage 4
Formation period
Implement strategy with cointegrated pairs from each cluster and measure performance (in-sample)
Stage 5
Trading
Implement strategy on out-of-sample data

Illustration of research design with all stages noted. Stage 1 to 5 is performed for all 37 periods from 2001 to 2019.

4.2 Data

The first step in our study is to import and process the data. We use daily closing prices adjusted for corporate events such as dividends and stock splits as these events may distort price history and produce false trading signals. Our universe is limited to stocks that have been listed on the S&P500 and the Oslo Stock Exchange (OSE) in the sample period. The sample period runs from 2000 to 2019 and covers a period where there have been significant developments within statistical arbitrage trading. The data on US securities is gathered from CRSP, while the OSE data is provided by Oslo Børs Informasjon (OBI). Once imported, we clean the data for missing values to facilitate further computations. The number of securities in the data varies over time, and the OSE goes from having 216 stocks in 2000 to 237 in 2019 while the S&P500 has approximately 500, meaning that potential pairs to trade will vary. We will use Python as our primary tool for processing data and performing computations. Python is particularly well suited for machine learning techniques because of the broad access to various libraries and frameworks suitable for machine learning and other statistical techniques.

4.4 Principal Component Analysis

According to the Arbitrage Pricing Theory, securities containing the same systematic risk should offer the same return. This provides us with a trading framework where deviations from the expected return can be exploited before being corrected by the market. To find the underlying risk factors for each security, we implement a PCA where the PCs will serve as a proxy for systematic risk factors. These principal components will later be used as input in the clustering algorithm (Stage 2), meaning that stocks that appear to share the same systematic risk factors will be put in the same cluster.

PCA is a common statistical technique that reduces the dimensionality of the dataset while preserving as much variability as possible. In practice, we want to obtain the important information from the dataset, create new orthogonal variables referred to as principal components, and then observe similarities between the variables. Our PCA framework builds upon the work of Marco Avellaneda and Jeong-Hyun Lee (2010). To utilize a PCA, we use historical stock price data on a cross section of N

stocks going back M days. In line with Avellaneda and Lee (2010), we assume that the cross section is identical to the investment universe. We define the stocks return data R_{ik} on a given date t_0 going back $M+1$ days, from the daily stock price P_i for a stock i at time t as a matrix:

$$R_{ik} = \frac{P_{i(t_0-(k-1)\Delta t)} - P_{i(t_0-k\Delta t)}}{P_{i(t_0-k\Delta t)}}, \quad k = 1, \dots M, i = 1, \dots N, \Delta t = 1/252 \qquad (4.3)$$

We assure that the variables are measured on the same scale. As some stocks vary more than others, it is helpful to standardize the returns in the following matrix:

$$Y_{ik} = \frac{R_{ik} - \bar{R}_i}{\bar{\sigma}_i} \qquad (4.4)$$

where

$$\bar{R}_i = \frac{1}{M}\sum_{k=1}^{M} R_{ik} \qquad (4.5)$$

and

$$\bar{\sigma}_i^2 = \frac{1}{M-1}\sum_{k=1}^{M}(R_{ik} - \bar{R}_i)^2 \qquad (4.6)$$

In the evaluation of price co-movements, the application of PCA on the *return* series is favorable since the return correlation matrix ρ_{ij} is more informative. Price series might expose spurious correlations due to underlying time trends. The correlation matrix is computed as:

$$\rho_{ij} = \frac{1}{M-1}\sum_{k=1}^{M} Y_{ik}Y_{jk}, \qquad (4.7)$$

To obtain the PCs, we must find the eigenvectors and eigenvalues. The eigenvectors represent the maximum variance directions, while the eigenvalues assess the respective directions variance. There are two main ways to determine these, either by a Singular Value Decomposition (SVD) or by an eigen-decomposition. In our methodology, we will use the SVD due to its mathematical properties, such as giving the best approximation (least square sense) of any rectangular matrix by another rectangular matrix having the same dimensions but smaller rank (Abdi & Williams, 2010). We embed the normalized return series for all the stocks in the respective market in the matrix A, which is through SVD computed as:

$$A = USV^T \qquad (4.8)$$

U is an orthogonal matrix comprised of the *left singular vectors*, V is a transposed orthogonal matrix comprised of the *right singular vectors*, and S is a nonnegative diagonal matrix and consists of *singular values* (eigenvalues) sorted in a descending manner from the highest variance values to the lowest ($\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n \geq 0$). By multiplying the matrix A with its transposed matrix A^T, we attain $A^T A = VS^2V^T$. The former computed correlation matrix ρ_{ij} (4.7) has the property of being symmetrical to $A^T A$, making it possible to discover the eigenvectors and eigenvalues. The next step is to create a new subspace corresponding to the PCs:

$$F_j = \sum_{i=1}^{N} \phi_i^j R_{ik} \tag{4.9}$$

The principal components, F_j, can be seen as the systematic risk factors for the securities in our dataset, outlined in the APT chapter, equation (2.6). These will serve as inputs in our clustering algorithm.

By reducing the dimensionality of the dataset, some information gets lost in the process. There are different ways to select the number of PCs that explain a satisfactory level of variability. Avellaneda & Lee (2010) selected the number of components that explained 55% of the total variance, which naturally varies over time with different datasets. By deciding on a lower explanation level, you sacrifice some information for a simpler data description. Sarmento & Horta (2019) argue that it actually is favorable to give up some information as the likelihood of finding irrelevant features increases with the number of principal component dimensions.

Contrary to Avellaneda and Lee's method of predefining the explanation level and then choosing the corresponding number of components, James et al. (2013) describe an ad hoc method of eyeballing a scree plot (with cumulative variance explained on the y-axis and number of principle components on the x-axis) and choose the number of components where the marginal proportion of variance explained is small. Some researchers argue that selecting the point of the "elbow" of the scree plot is sufficient even though it often leads to few components and thus a somewhat low explanation level. Since an unsupervised learning algorithm will be applied to the PCA output, we should also consider the challenges posed by dimensionality (Sarmento & Horta, 2020). Using a higher number of PCs increases the chances of identifying features in the data that are not relevant. Further, the volume caused by adding more dimensions increases exponentially, causing points in the data to appear very distant from each other, and the clustering algorithm will not prove as efficient (Bellman, 1966). In this paper, we will base our decision on the ad hoc method implied by James et al. (2013), making sure that we choose a number that explains a fair amount of the variance while still avoiding having too many dimensions.

4.5 Unsupervised Machine Learning

Supervised learning is a common form of machine learning where an algorithm is used to learn a mapping function so that it is able to predict the output when being fed with input data. We call this supervised learning as we know the answers from the training data and then teach the algorithm what is correct. Unsupervised Learning is a different machine learning technique where one looks for patterns in the data with no pre-existing labels. As we cannot teach the algorithm what the correct answers should be, it will have to model the underlying structure of the data on its own. Unsupervised learning is often grouped into two methods: clustering and association. Clustering is used for problems where one would like to discover groups within the data, which can be helpful for discovering stocks that have similar systematic risk. Our goal is to classify the stocks into clusters, based on the PCs obtained in Stage 1, before looking for pairs displaying a strong mean-reverting relationship within these clusters. We believe that unsupervised learning will prove valuable in grouping stocks as it removes as much human interference as possible, reducing the risk of human error or bias that might affect our results.

4.5.1 Density Based Spatial Clustering Applications with Noise

Once we have extracted the principal components, F_j, for all our securities, we seek to cluster them such that securities with similar risk factors are grouped together, making it easier to discover highly correlated pairs. The DBSCAN is a clustering algorithm proposed by Ester et al. in 1996, which is designed to discover clusters of arbitrary shapes. The most widely used clustering algorithm is the K-Means; however, the DBSCAN offers a few advantages such as its ability to handle outliers (noise) efficiently, and the number of clusters need not be specified in advance.

DBSCAN requires two input parameters. 1) *Eps*: Specifies the distance required between two points for them to be considered in the same cluster, i.e., the radius around a given point. If the distance between two points is equal to or lower than the Eps, they are considered neighbors. 2) *MinPts*: The minimum number of data points needed to form a cluster (dense region).

The DBCAN starts with an arbitrary point and will classify nearby points as a core point, border point, or outlier. *Core point*: A point is considered a core point if there are at least MinPts within its area with radius Epsilon. *Border point*: A point is considered a border point if it is within Epsilon radius of a Core Point, but there are less than MinPts within its own area. *Outlier*: A point is considered an outlier if it is not classified as a core point nor reachable from any other core points by Epsilon. If there are enough neighboring points wrt. MinPts, a cluster is formed, and the algorithm iterates the process for all other points. Figure 4.2 illustrates the process of the DBSCAN algorithm.

When using DBSCAN, it is essential to correctly specify the input parameters to obtain useful output. This requires knowledge of the dataset as the parameters should be specified according to the user's needs, although some methods can guide the user in the right direction. For 2-dimensional data, the default value for MinPts is set to 4 (Ester et al., 1996). For lower-dimensional data, MinPts is usually set to

Figure 4.2: *DBSCAN process*

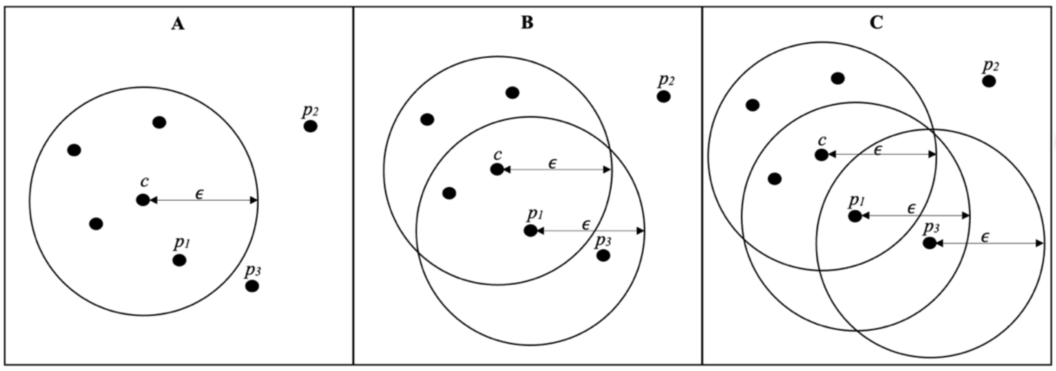


Illustration of the DBSCAN process with MinPts = 4. Panel A: The algorithm identifies a core point c, a border point p_1, an outlier p_2. Panel B: point p_1 is identified as a new core point and p_3 is identified as a new border point. Panel C: As p_3 does not have enough neighboring points within the radius ϵ it is not classified as a core point.

be greater than or equal to the number of dimensions in the data. For higher dimension data, a general rule of thumb is to set MinPts equal to two times the number of dimensions in your dataset and subtract one (Sander et al., 1998). As a small MinPts will produce more clusters from noise, it should not be set too small, but neither too high as it might fail to detect any clusters. The Eps is chosen in an ad hoc fashion that creates meaningful clusters throughout the years in the study.

4.5.2 t-Distributed Stochastic Neighbor Embedding

To better understand the clusters discovered by the DBSCAN algorithm, we can visualize the output. t-Distributed Stochastic Neighbor Embedding (t-SNE) is an unsupervised non-linear technique often used to visualize data of higher dimensions as it lets us plot the data on a two-dimensional map. The method differs from traditional linear methods like PCA as it seeks to keep the low-dimensional representation of close data points instead of preserving the representation of dissimilar data points (van der Maaten & Hinton, 2008).

The t-SNE algorithm measures the similarity between data points using the Euclidian distances of each point to all other points, assigning a higher value to similar pairs. The distances are then converted to conditional probabilities representing the similarity of two points. We use this to determine the probability of two points being neighbors and the conditional probability $p_{(j|i)}$ is given by:

$$p_{(j|i)} = \frac{\exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)}{\sum_{k \neq i} \exp\left(-\|x_i - x_j\|^2 / 2\sigma_i^2\right)} \tag{4.10}$$

where $x_i$and x_j represents datapoints, in our case the principal components serving as risk factors, and σ_i is the variance of the Gaussian distribution. This is then used to calculate the joint probability function:

$$p_{ij} = \frac{p_{(j|i)} + p_{(i|j)}}{2n} \tag{4.11}$$

The next step is to project our dataset onto a lower-dimensional space with k dimensions, setting $k = 2$ in our case. This is done by taking the low-dimensional counterparts of the high dimensional data points, y_i and y_j, and computing a similar conditional probability using a t-distribution instead of a Gaussian distribution given by:

$$q_{(j|i)} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)} \tag{4.12}$$

We want the new data points to yield similar map points in the lower dimensional map; therefore, we seek to minimize the distance between $p_{(j|i)}$ and $q_{(j|i)}$ using the Kullback-Leibler divergence given by:

$$KL(P_i \| Q_i) = \sum_{i,j} p_{(j|i)} log \frac{p_{(j|i)}}{q\_(j|i)} \tag{4.13}$$

Where P and Q represent conditional probability distributions over the given data points x_i and y_i. Finally, the algorithm requires us to choose the standard deviation, σ_i, of the Gaussian distribution by specifying a fixed perplexity. According to van der Maaten and Hinton (2008), perplexity can be thought of as a "smooth measure of the effective number of neighbors". The value will typically vary between 5 and 50.

4.8 Discovering mean-reverting relationships

Once clusters are identified, we will look for pairs exhibiting a mean-reverting relationship within each cluster. This chapter discusses the most common techniques and how they can help us identify potentially profitable pairs to trade.

4.8.1 The Distance Approach

Following the study of Gatev et al. (2006), the distance approach gained recognition among researchers. The approach was developed after discussions with traders about how they implemented this trading strategy, where the sum of squared differences between normalized price series is minimized. More specifically, the authors construct a cumulative total return index for each stock over their formation period. Each stock is then matched with another security that minimizes the sum of Euclidean Squared Distances (SSD) between the two normalized time series. The average SSD can be expressed as:

$$\overline{SSD}_{P_i,P_j} = \frac{1}{T} \sum_{t=1}^{T} \left(P_{i,t} - P_{j,t}\right)^2 \tag{4.14}$$

An optimal pair is found by minimizing equation (4.12), which implies that a pair with a spread of zero would be optimal. However, Sarmento & Horta (2019) argues that this method is counterintuitive as an ideal pair would actually exhibit high spread variance and strong mean-reverting properties.

4.8.2 Cointegration Approach

The cointegration approach may offer a more robust approach to identify mean-reverting relationships as it allows us to discover long-term relationships between securities. We will therefore be using cointegration in our framework to identify pairs within each of the clusters formed.

In order to describe cointegration, it is necessary to introduce the concept of stationarity. Stationarity describes a time series whose probability distributions do not change over time and can be described formally as a stochastic process $\{x_t: t = 1,2,\dots\}$ which is stationary if for every instance of t, the joint distribution of $(x_{t_1}, x_{t_2}, \dots, x_{t_m})$ is equal to the joint distribution of $(x_{t_1+h}, x_{t_2+h}, \dots, x_{t_m+h})$ for integers $h \geq 1$ (Wooldridge, 2015). In contrast, a non-stationary time series violates the above requirement with means and variances varying over time (Hendry & Juselius, 2000). We usually say that a time series is weak stationary if the two first moments are constant over time, i.e., the series has a constant mean and variance for all periods.

Cointegration was formally introduced by Engle & Granger (1987) and is widely used by practitioners in finance and economics to identify long-term relationships between a set of variables. Generally, time series are said to be cointegrated when they are integrated of order 1, $I(1)$, while a linear combination of the series is $I(0)$ (Wooldridge, 2015). Saramento & Horta (2020) offer the following formal definition: Two time series, y_t, and x_t, which are $I(1)$, are cointegrated if there exist coefficients μ and β such that:

$$y_t - \beta x_t = u_t + \mu, \tag{4.15}$$

where u_t is a stationary time series. This approach lets us create stationary time series from our stock prices by calculating the spread between two stocks that are $I(1)$. As stationary time series tend to fluctuate around its mean with a constant variance, we can exploit short-term deviations in the spread and bet that the spread will converge back to its mean.

There are several ways to test for cointegration. In this paper, we follow the widely used Engle-Granger two-step test to identify pairs that are cointegrated. First, by applying an Augmented Dickey-Fuller (ADF) test, we observe whether y_t and x_t are non-stationary (has a unit root). If this holds true, we run an Ordinary Least Squares (OLS) regression on equation (4.15). Lastly, with an ADF test, we observe a potential unit root in the residuals. To conclude on the existence of a cointegrating relationship, we formulate the following hypothesis test:

$$H_0\text{: } There\ is\ a\ unit\ root\ in\ the\ residuals$$
$$H_1\text{: } There\ is\ no\ unit\ root\ in\ the\ residuals$$

If the P-value is < 0.05, we reject the null hypothesis. Thus, the residual series are stationary, and the respective securities are cointegrated.

4.9 Trading execution

After filtering the stocks in our universe and identifying pairs exhibiting a mean-reverting relationship, i.e., cointegrated pairs, we must define when and how trades should be placed. This usually involves setting a threshold level that will trigger a trade.

4.9.1 Signal generation

Gatev et al. (2006) propose a simple threshold-based trading model based on the divergence of the observed spread between two securities. More specifically, if the spread diverges by more than two standard deviations from the mean measured in the formation period, a trade is placed. The trade is closed once the spread converges and the prices cross. We apply a similar trading rule where we define the spread between the two stocks forming a pair:

$$S_t = \frac{Y_t - X_t}{Y_t} \tag{4.16}$$

Y_t and X_t are the two different securities. We further compute a z-score, measuring the distance to the mean in units of standard deviation Caldeira (2013):

$$z_t = \frac{S_t - \mu_t}{\sigma_t} \tag{4.17}$$

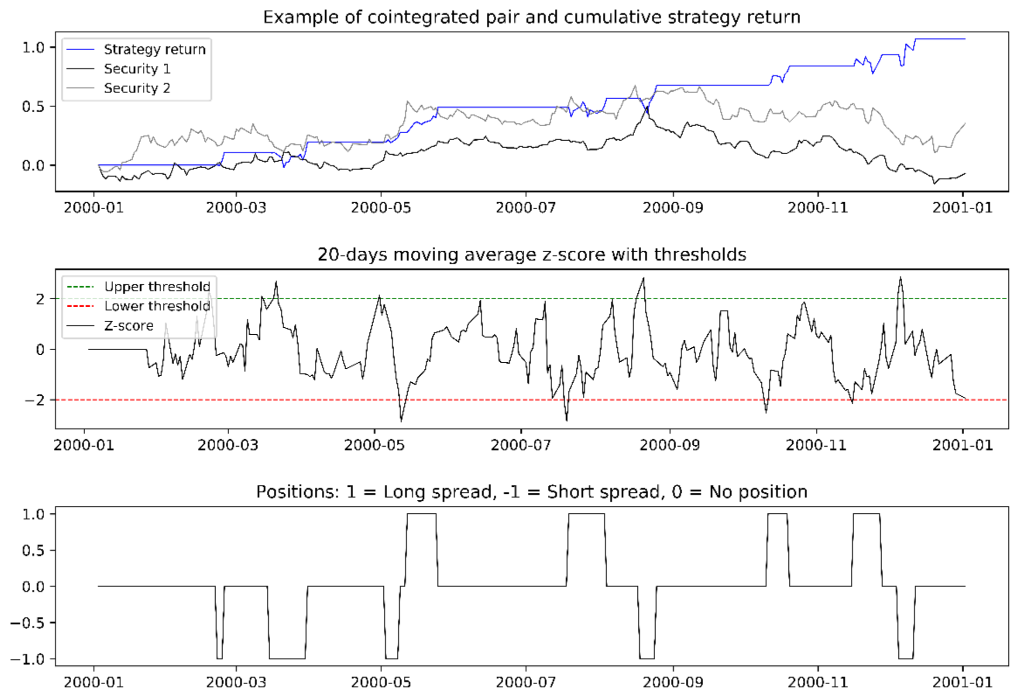
Instead of using the observed mean from the formation period, we apply a 20-day moving average. We will use the *z*-score to determine when positions should be opened, using a threshold of two standard deviations as an upper and lower bound. If the observed spread exceeds the threshold, one of the securities in the pair is said to be "significantly" overpriced relative to the other security, and a bet is taken that the relative value will converge towards the long-term equilibrium.

z-score > 2: Enter a short spread trade

z-score < -2: Enter a long spread trade

Once a trade is opened, we will keep the position open until the sign of the *z*-score is reversed, similar to the approach employed by Gatev et al. (2006). Figure 4.3 illustrates the trading setup and execution using a randomly chosen pair from our universe in the first formation period (January 2000 to January 2001).

Figure 4.3: *Pairs trading example*



Example of a pair that is found to be cointegrated in the formation period running from January 2000 to January 2001. The top chart shows the stock prices of the two securities and the blue line plots the cumulative return that the strategy would obtain. The middle chart plots the z-score based on the 20 day moving average spread and the thresholds of +/- two standard deviations. The bottom chart shows the positions that would be held by this strategy. +1 indicates a long spread position, -1 indicates a short spread position, while 0 indicates that no position is open.

4.9.2 Formation and trading period

Testing a trading strategy on historical data requires a formation period where the algorithm is trained and a testing period where we observe the performance of the strategy out-of-sample. There is no single answer as to how long the formation and the testing periods should be. Still, since we are using the cointegration approach, we should ensure that the formation period contains enough data for a cointegrating relationship to be identified. Gatev et al. (1999;2006) use a 12-month formation period followed by 6 months of trading in their original study, and we will be using a similar setup in this study (figure 4.4). Our data runs from 2000 to 2019, resulting in 37 formation- and trading periods for both markets.

Figure 4.4: *Formation and trading setup*

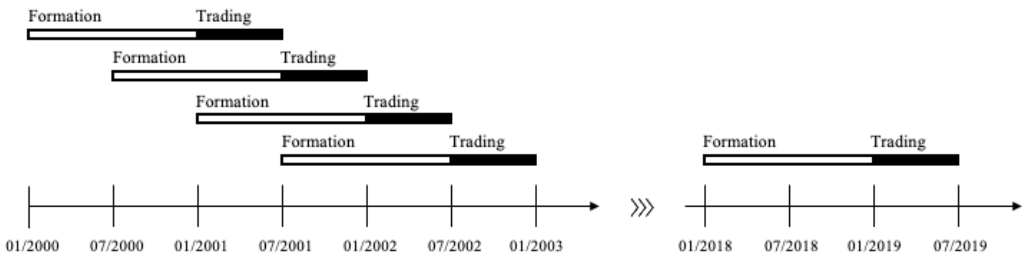


Illustration of the rolling formation and trading setup. In the formation period, the stocks are clustered and tested for cointegration. In the trading period, the identified pairs from the formation period are traded according to our prespecified thresholds.

4.9.3 Computing returns

As the strategy involves taking both a long and a short position, where the long leg of the trade is financed by the short leg, calculating the returns may not be entirely intuitive. The payoffs can be thought of as a string of randomly distributed cash flows incurred at different points in time. A positive cash flow will occur once a successful trade is closed, which may happen multiple times for each pair. Open positions that are not closed by the end of the trading period will only incur a cash flow at the last trading day based on the closing prices. Because the return is computed on long and short positions on one dollar invested, Gatev et al. (2006) argue that the payoffs have the interpretation of excess returns. They further suggest two measures of excess return: return on committed capital and return on actual employed capital. Return on committed capital takes the sum of all payoffs and divides it by the total number of identified pairs in the portfolio. As this approach is fairly conservative and may not represent the capital sourcing of a hedge fund, they argue that the return on actual capital employed seems like a more appropriate measure as this divides the sum of the payoffs by the number of pairs opened during the trading period:

$$r_p = \frac{\sum_{i=1}^{n} CF_i}{L + \tau S} \tag{4.16}$$

L and S are the amounts placed in the long and short leg, respectively. τ refers to the fraction of capital required (margin) required by the broker for the short position. This requirement will vary based on the volatility and liquidity of the individual security, but we will, similar to Hoel (2013), set $\tau = 1$, giving us a more conservative return estimate.

4.9.4 Transaction costs

Bearing in mind the liquidity sensitiveness of our study, we pay particular attention to transaction costs and attempt to obtain a realistic estimate of the costs that a trader would incur when applying this strategy. In their research on trading costs, Do & Faff (2011) find that most earlier studies fail to adequately adjust strategy performance for costs, thus leading to a "material upward bias". We follow Do & Faff's approach with three main components of costs: commissions, short selling fees, and market impact.

The nature of pairs trading implies that commissions accrue two times when opening a position and two times when closing the position, hence two roundtrips of costs. Fees charged for short selling accrue only for the security that is, in relative terms, overpriced. Commissions and short selling fees are explicit trading costs and are generally easy to observe. Based on an analysis of historical data, we set the commission per trade to 5 bps and the annual short-selling fee to 450 bps. To compute the short-selling expenses per trade, we convert the annual fee into a daily fee of 1.79 bps. To get the total cost, the average number of days we hold a position open is multiplied by the daily short-selling fee and added to the fixed commission. The implicit trading costs are slightly harder to estimate, and we apply two methods to increase the reliability of the estimates:

1) Apply a one-day lag both when opening and closing the positions to estimate the implied bid-ask spread in the market.

2) Adjust for relative bid-ask spreads to simulate a "worst-case" scenario of trading costs.

The first approach is related to the bid-ask bounce as described by Jegadeesh & Titman (1993). Any movements in the stock prices observed are potentially due to movements in the bid-ask quotes. Once the spread between a pair converges, we are more likely to trade on an ask quote for the "winner" and the bid quote for the "loser". Since we are implicitly buying at bid-quotes and selling at ask-quotes, with the opposite case for the unwinding of the position, there is a chance that our returns are biased upwards. The second approach serves as a conservative measure of profitability that stresses the robustness to transaction costs. While this estimate might be a bit too conservative, we argue that it is interesting to observe whether the strategy would survive this worst-case estimate of costs.

4.10 Assessing performance of the strategy

To assess the performance, we study the excess returns generated by the strategy and the risk profile of these returns. We can measure the Sharpe Ratio (SR) as defined by William F. Sharpe (1966):

$$SR = \frac{r_p - r_f}{\sigma_p} \tag{4.17}$$

where r_p is the return of the portfolio, r_f is the risk-free rate and σ_p is the observed standard deviation of the portfolio. Because the pairs trading strategy being tested is dollar neutral, we do not subtract the risk-free rate when computing the SR. The SR measures the return obtained per unit of risk, as defined by the standard deviation. A higher SR generally means higher risk-adjusted returns. Similar to Gatev et al. (2006), we control for traditional risk factors to explore the strategy's systematic risk exposure. This will give us an estimate of the strategy's ability to generate returns not captured by the most common factors, in other words, the risk-adjusted returns. More specifically, we will be using the three-factor model of Fama & French (1993) and Carhart's (1997) momentum factor:

$$\alpha = \beta_1(mkt - rf) + \beta_2(SMB) + \beta_3(HML) + \beta_4(UMD) \tag{4.18}$$

where α represents the risk-adjusted returns obtained by the strategy.

4.11 Liquidity

To comment on the liquidity of the two markets, we apply measuring tools building on the works of Næs et al. (2008). As a proxy for liquidity, we apply one order-based measure using current available (ex-ante) liquidity and one trade-based measure using realized (ex-post) liquidity. For the order-based measure, we observe the relative spreads (in percent) at closing as:

$$S = \frac{P_{Ask} - P_{Bid}}{\frac{1}{2}(P_{Ask} + P_{Bid})} \tag{4.1}$$

where $P_{Ask}\ and\ P_{Bid}$ represents the "best" quoted prices in the order book.

To assess liquidity via trade-measures, we will compute the turnover as:

$$T = \frac{Number\ of\ shares\ traded}{Number\ of\ shares\ outstanding} \tag{4.2}$$

As we are interested in the effect of liquidity on the strategy's profits, we will use these measures to determine the overall liquidity in the two markets studied. Additionally, we use the observed relative bid-ask spread to construct separate portfolios containing stocks with the highest and lowest liquidity. This analysis is covered in chapter 5.4.

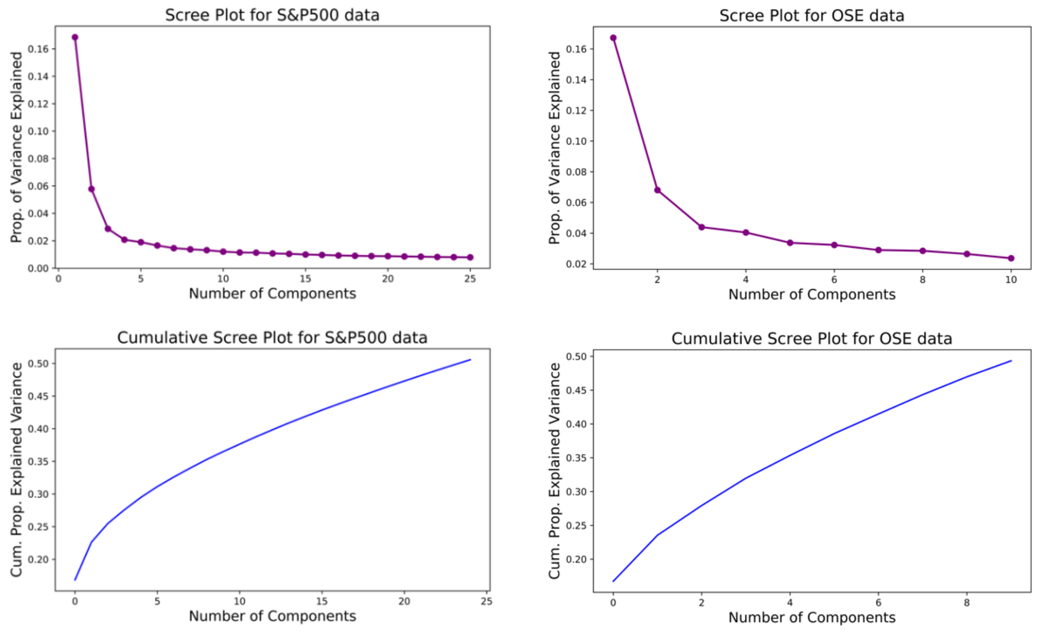
5. Results and Analysis

This section discusses the results obtained when studying the strategy in both the US and Norwegian stock market. We start by discussing the output of the dimensionality reduction technique, PCA, and the unsupervised clustering algorithm before describing the results obtained by the model. The section proceeds to assess the liquidity of the two markets, followed by an in-depth analysis of how market liquidity affects the profitability of the strategy. Finally, we discuss the robostness of the strategy to both explicit and implicit transaction costs.

5. 1 Number of Principal Components

After running the PCA algorithm for every formation period on our data, we begin by studying the proportion of variance explained by each of the principal components as well as the cumulative explained variance. Panel A in figure 5.1 takes the *number of principal components* on the x-axis and the corresponding *proportion of variance explained* on the y-axis. We see that the first component explains the greatest proportion of the variance in both the S&P500 and the OSE data and that the additional proportion of explained variance for the following components quickly becomes marginal.

Figure 5.1: *PCA output*



The figure contains a graphical representation of the proportion of variance explained per number of principal component as well as the cumulative explained variance for the S&P500 and OSE respectively. The figure is chosen to best portray the average results of each formation period.The figure is constructed for illustration purposes only, and is chosen to best portray the average results of each formation period in our results.

Panel B in figure 5.1 presents the *number of principal components* on the x-axis, and the corresponding *cumulative variance explained* on the y-axis. Using the ad hoc approach described in chapter 4.4, we decide to use 20 PCs for the S&P500 data and 7 for the OSE data, ensuring that we have enough components to capture a sufficient amount of variance in the data while limiting the number of dimensions. The chosen number of components capture on average roughly 45% - 55% of the variance, similar to what Avellaneda (2008) used as predefined variance when selecting the number of PCs. To be consistent in our research, we hold the number of principal components constant throughout all formation periods.

To optimize the output of the clustering tool, we want to choose the number of components that produce the most meaningful clusters. We are interested in knowing how the number of PCs used affects the output of the DBSCAN and run a sensitivity analysis stressing the number of PCs on cluster characteristics. Table 5.1 summarizes the cluster statistics as we vary the number of principal components. Throughout the sensitivity analysis, the ϵ is held constant at 0.6 for the OSE and at 1.0 for the S&P500. We observe clear variation in clusters formed, and thus the number of stocks in each cluster, as PC change. The trend is similar for both markets: too many or too few components produce fewer clusters. As we aim to let the algorithm form meaningful clusters, we seek a balance between clusters formed and the number of stocks in each cluster.

Table 5.1: *Cluster characteristics*

A: S&P500

Principal Components	**10**	**15**	**18**	**20**	**22**	**25**	**30**	**45**
Number of clusters	*3*	*3.75*	*3.80*	*3.81*	*3.54*	*2.30*	*1.14*	*0*
Unique stocks in each cluster	*172*	*73*	*34*	*20*	*15*	*10*	*7*	*0*
Cointegrated pairs in each cluster	*3681*	*614*	*188*	*66*	*32*	*17*	*10*	*0*
Of which are unique stocks	*125*	*49*	*23*	*14*	*9*	*7*	*5*	*0*

B: OSE

Principal Components	**4**	**6**	**7**	**8**	**10**	**12**	**15**	**25**
Number of clusters	*1.24*	*1.32*	*1.35*	*1.31*	*1.19*	*0.97*	*0.59*	*0*
Unique stocks in each cluster	*105*	*86*	*75*	*68*	*79*	*23*	*9*	*0*
Cointegrated pairs in each cluster	*971*	*720*	*588*	*464*	*150*	*86*	*34*	*0*
Of which are unique stocks	*86*	*69*	*57*	*47*	*37*	*27*	*16*	*0*

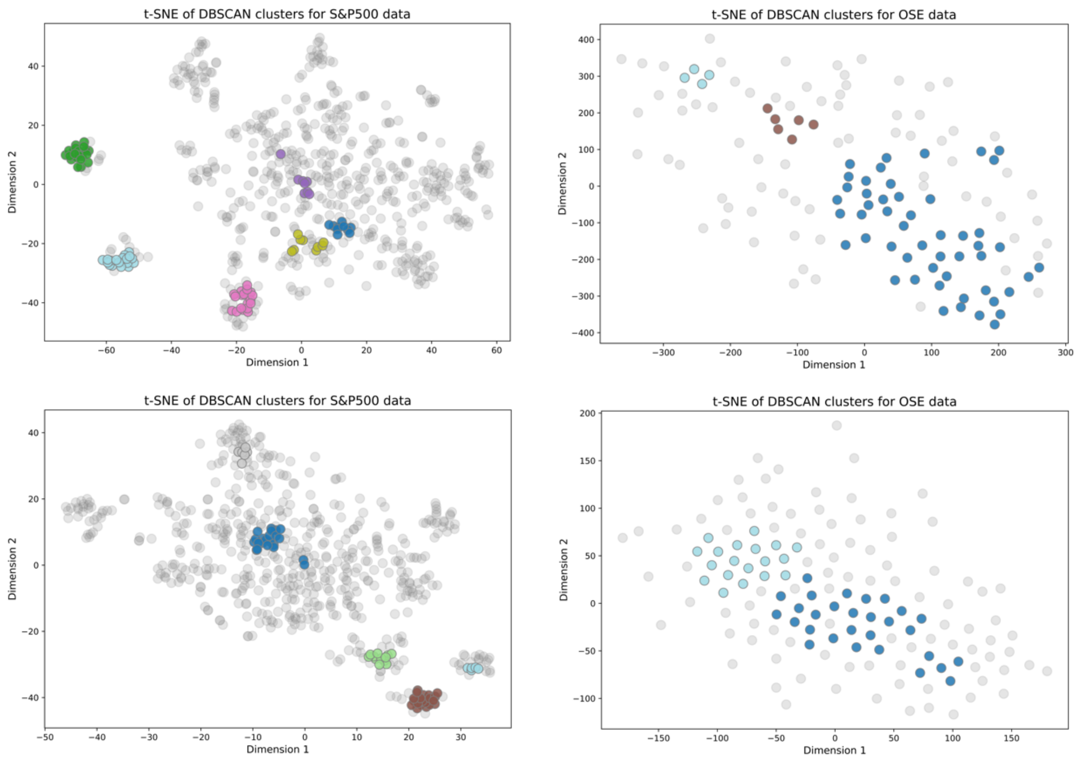
The table contains cluster characteristics stressed by number of principal components. The values are gathered from the formation periods in the timespan 2000-2019 and then averaged. The colored columns represent the characteristics related to our chosen number of components.

We find that 7 components in the OSE and 20 components in the S&P500 on average produce the most clusters while maintaining a similar number of potential stocks to trade per period.

5.2 Cluster discovery

Looking at the US data, we observe that we on average identify between three and four clusters each period, with one period having as many as ten unique clusters. This clearly differs from the OSE data where we on average identify between one and two clusters in every formation period, with the highest number of clusters in a period being three. The size of each cluster varies, but on average, there are 2895 pairs in each cluster for the US data and 8479 for the OSE data (before checking for cointegration). In figure 5.2, we plot a few of the clusters using t-SNE on the DBSCAN results to visualize the output of the clustering algorithm, and we clearly see that clusters form in areas of higher density. This also works as a sanity check for the DBSCAN uoutput, as we want to see that both the t-SNE and the DBSCAN are able to find our clusters.

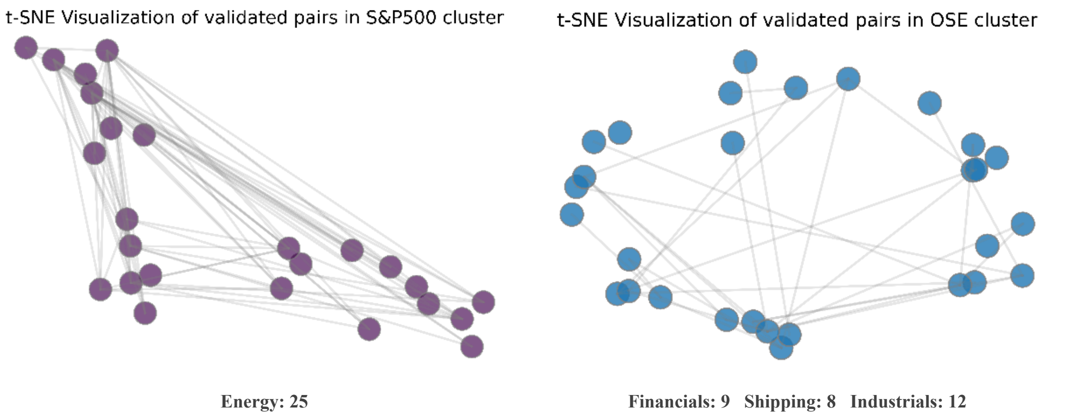
Figure 5.2: *t-SNE plots of clusters*



t-SNE plots of some of the clusters formed by the stocks in the US market and the Norwegian market. Each color represents different clusters while the gray dots are unclustered stocks.

To further investigate the output of the DBSCAN algorithm, we examine one arbitrary cluster from each of the two datasets in detail. The two clusters shown in figure 5.3 are both formed in the formation period running from January 2001 through December 2001. The cluster formed by the S&P500 data is entirely made up of stocks related to the energy industry and contains 22 unique stocks. When paired, several of these stocks are cointegrated at the 5% level, implying that a long-term relationship between them may exist (indicated by grey connecting lines between points in figure 5.3). The cluster formed by the OSE data is roughly the same size but are less concentrated, meaning that it is formed by stocks from several different industries. It is worth noting that a cointegrating relationship is found between several pairs, also pairs made up of stocks belonging to various industries. This is in line with our hypothesis that an unsupervised machine learning technique might be able to uncover patterns not necessarily obvious at first glance. We observe that the OSE cluster contains quite a few stocks from both the financial industry and the shipping industry. This might reflect the fact that many Norwegian banks historically have had significant exposure to the shipping industry, although this is not something we can conclude.

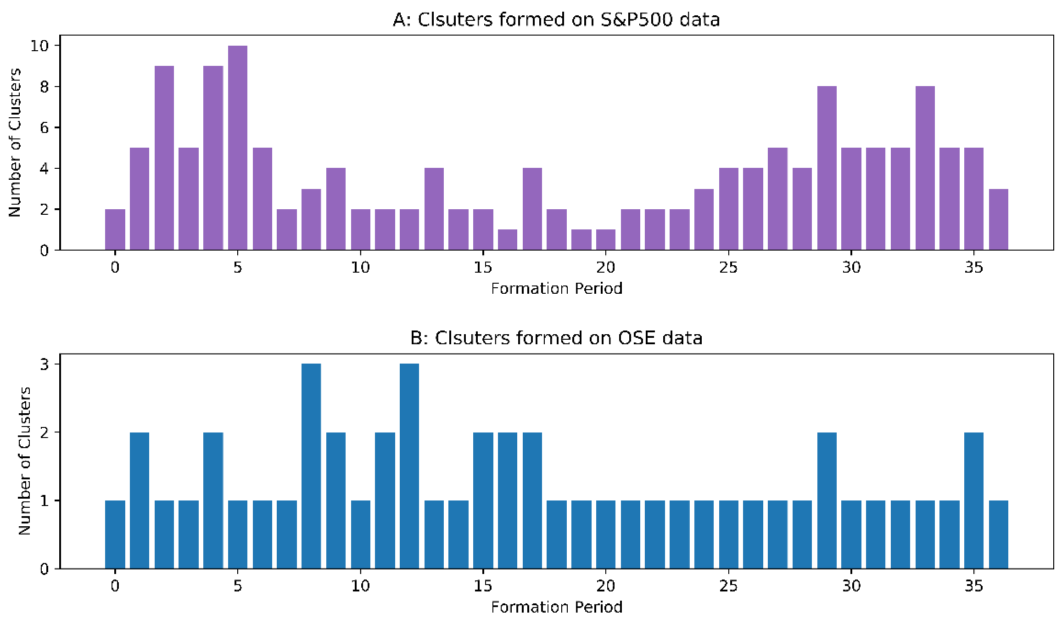
Figure 5.3: *Example of clusters with validated pairs*



t-SNE plots of some cluster from each of the two markets for illustrational purposes. Each point represents a stock while a grey line between two points mean that they are cointegrated. The cluster from the S&P500 stocks contains only stocks related to the energy industry. The cluster from the OSE data is less concentrated and contains stocks related to

It is worth noting that after formation period 18 (2009), the algorithm struggles to form more than one cluster in the OSE dataset (figure 5.4). Instead of forming several clusters with fewer stocks in each cluster, it forms one big cluster comprising several stocks and only removes outliers. Generally, it is not a problem;

Figure 5.4: *Number of clusters identified*



Number of clusters formed in each formation period in the two markets. The algorithm seems to be working well in the beginning and in the end if the sample period for the S&P500 data. The trend is different on the OSE data and we see that the algorithm is struggling to make more than one cluster after formation period 17 (2008).

however, the effect of the DBSCAN declines when the inputs are no longer the best fit for the respective period. We observe that this is a minor pitfall of the "one size fits all" idea of applying the same inputs over the entireness of the sample period. For the purpose of this study, comparing the same strategy on two different markets, we are not necessarily exploring new ways to optimize the strategy. Nevertheless, if we were to optimize the model, we observe that applying a time-varying EPS that tests each period separately could improve the potential of the DBSCAN.

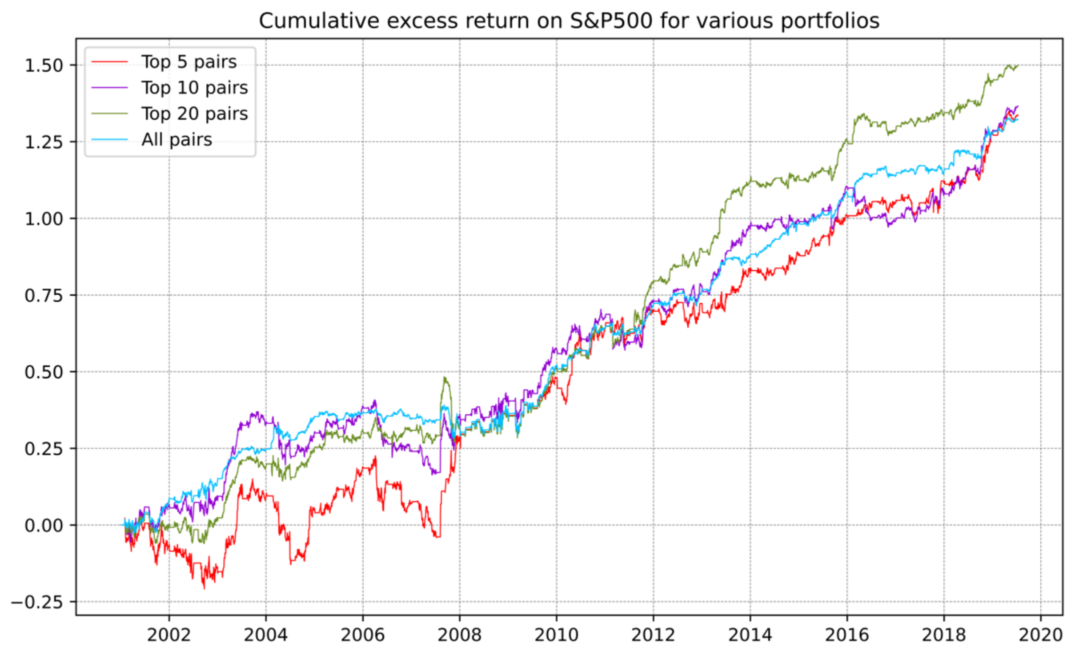
5.3 Strategy performance

We begin by looking at how the strategy performs on the S&P500 and OSE separately, and comment on the distribution of excess returns and its risk characteristics.

5.3.1 Pairs trading on the S&P500

Panel A in Table 5.2 summarizes the descriptive statistics of the excess returns before TC and the systematic risk of the strategy in the US. Statistics are computed for different portfolios containing the n number of pairs with the highest Sharpe Ratio from the formation period, as well as a portfolio containing all cointegrated pairs in each cluster. The first rows show that the top 5 pairs on average deliver excess returns of 8.32% per year (t-statistic of 2.84) while the top 20 pairs on average generate annualized excess returns of 10.58% (t-statistic of 4.61), suggesting that pairs trading in the US stock market is profitable (Figure 5.5). This is almost identical to what Gatev et al. (2006) found on average in 1962-2002. Looking at the distribution of the excess returns, we see diversification benefits from trading on several pairs as the standard deviation decreases when adding more pairs to the portfolio. This is consistent with the idea that pairs trading is only considered a form of arbitrage when trading on many pairs simultaneously. Comparing the greatest daily loss and gain for the top 5 portfolio and the portfolio including all pairs, we observe that the maximum daily *loss* is lower, and the maximum daily *gain* is larger for the portfolio containing more pairs. This trend is also evidenced by the increased positive skewness for the portfolios containing more pairs.

Figure 5.5: *Strategy performance on the S&P500*



Cumulative excess return for portfolios containing the top 5, 10, 20 and all pairs from 2001 to 2019.

Table 5.2: *Summary of descriptive statistics and systematic risk of pairs trading in US equities*

Pairs portfolio	*Top 5*	*Top 10*	*Top 20*	*All*
A: Daily excess return distribution				
Average excess return	*0.00033*	*0.00033*	*0.00042*	*0.00036*
Annualized excess return	*0.08316*	*0.08316*	*0.10584*	*0.09072*
t-Statistic	*2.84731*	*3.29851*	*4.61972*	*4.90302*
P-value	*0.00443*	*0.00097*	*0.00000*	*0.00000*
Excess return distribution				
Median	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Standard deviation	*0.00791*	*0.00687*	*0.00617*	*0.00501*
Skewness	*0.64727*	*0.47832*	*1.37969*	*1.38413*
Kurtosis	*11.41694*	*21.28843*	*24.95730*	*27.50951*
Minimum	*-0.05467*	*-0.09028*	*-0.06072*	*-0.04615*
Maximum	*0.07452*	*0.07452*	*0.07938*	*0.06258*
B: Systematic risk of pairs trading				
Annualized Sharpe Ratio	*0.66*	*0.77*	*1.07*	*1.14*
Intercept	*0.00030*	*0.00030*	*0.00040*	*0.00040*
	*(2.89800)****	*(3.32600)****	*(4.64800)****	*(4.89900)****
Market	*-0.02020*	*-0.01040*	*-0.00650*	*0.00180*
	*(-1.89500)**	*(-1.12100)*	*(-0.78300)*	*(0.27300)*
SMB	*0.04380*	*0.04840*	*0.05790*	*0.04870*
	*(2.12600)***	*(2.70600)****	*(3.60700)****	*(3.74200)****
HML	*-0.04370*	*-0.04360*	*-0.04640*	*-0.04120*
	*(-2.25300)***	*(-2.58600)****	*(-3.07000)****	*(-3.35900)****
Momentum	*-0.02330*	*-0.02400*	*-0.03540*	*-0.02440*
	*(-1.71600)**	*(-2.03800)***	*(-3.34700)****	*(-2.84100)****
R^2	*0.00300*	*0.00400*	*0.00700*	*0.00700*

Summary statistics of the daily excess returns on the portfolios of pairs between January 2001 and July 2019 containing 4662 observations. Trades are made according to the prespecified rule where trades are opened once the spread between two cointegrated stocks in the same cluster deviate by more than two standard deviations, measured over a 20-day rolling window (Panel A). Top "n" portfolios consists of the "n" number of pairs with the highest reported Sharpe Ratio in the formation period. Panel B presents a summary of the strategy's risk characteristics with daily returns regressed on the Fama & French (1993) three factor model and Carharts (1997) Momentum factor.

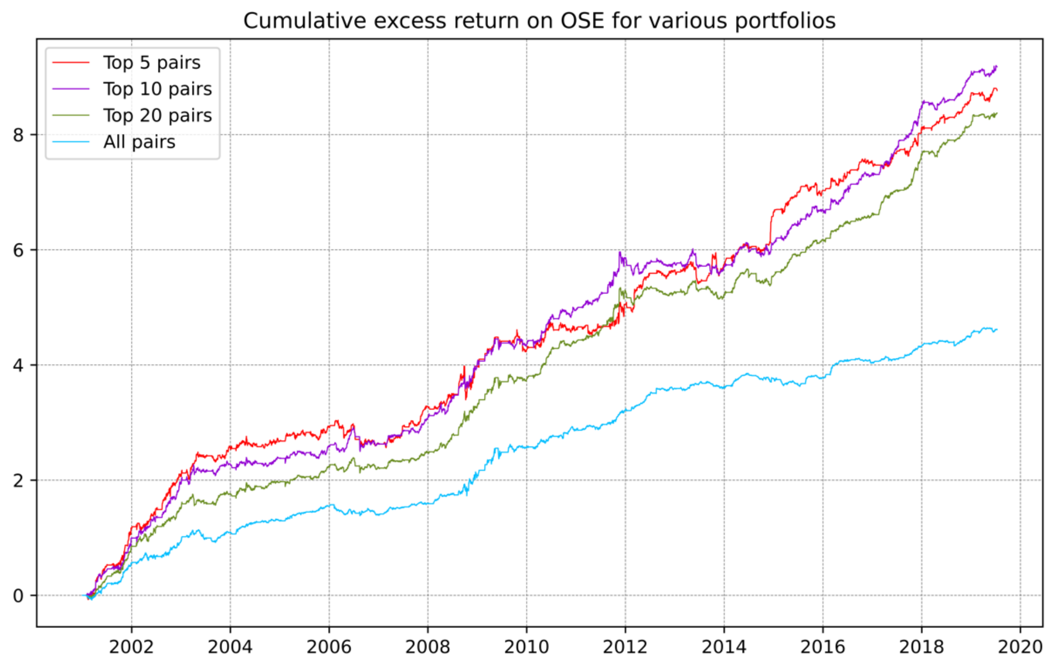
Panel B summarizes the risk characteristics associated with the pairs trading strategy. With a declining standard deviation and relatively stable excess returns as we add more pairs to the portfolio, the Sharpe Ratio is positively correlated to the number of pairs in the portfolio. The portfolio containing all pairs delivered an annualized SR of 1.14. In the same period, the S&P500 index obtained an annualized SR of 0.26, suggesting that the strategy has a more attractive risk profile than the overall stock market (Figure A1 in appendix). To assess the systematic risk of the strategy, we regress the daily excess returns on the three factors of Fama & French (1993) and the momentum factor as constructed by Carhart (1997). Overall, the four risk factors appear to explain a relatively small portion of the excess returns with a very low R squared. The risk-adjusted returns are all significant and lower than the excess returns, except for the portfolio containing all pairs which has a slightly higher risk-adjusted return than raw excess returns.

As expected, being a market-neutral strategy, the exposure to the market premium is small, and the estimates' sign is shifting. Although small, the exposure to both the SMB and HML factors appears to be positive and significant for all portfolios. The exposure to the momentum factor is negative and statistically significant. This does not come as a surprise as pairs trading is a contrarian trading strategy where one in many cases will short stocks that have performed well recently and buy stocks that have underperformed, opposite of what a momentum strategy would do.

5.3.2 Pairs trading on the OSE

Panel A in Table 5.3 summarizes the descriptive statistics of the excess returns and the systematic risk of the strategy on securities traded on OSE. Statistics are computed in the same manner as for the US equities above, presenting trading results for the n pairs with the highest Sharpe Ratio in the formation period. The top 5, 10, and 20 portfolios generate on average 46.87%, 48.34%, and 46.12% excess returns per year before TC, and the observed returns are all significantly different from zero at a 1% level. In addition, the portfolio where all cointegrated stocks in each cluster are traded delivers annualized excess returns of 24.19% (t-stat of 6.15), which is clearly lower than the concentrated portfolios (Figure 5.6). Still, the results suggest that Pairs Trading on the OSE is profitable and even more profitable than on the S&P500 exchange.

Figure 5.6: *Strategy performance on the OSE*



Cumulative excess return for portfolios containing the top 5, 10, 20 and all pairs from 2001 to 2019.

Panel B summarizes the risk characteristics associated with the pairs trading strategy on OSE. Similar to the US results, we observe diversification benefits as the standard deviation decreases, also reflected by the decline in maximum gain or loss in a day by adding more potential trading pairs in the portfolio. Not surprisingly, as the average excess returns remain stable, the Sharpe Ratios experience an almost linear upward trend from an annualized SR before TC of 1.28 in the top 5 to an SR of 1.90 in the top 20. Even though the standard deviation is lowest for the portfolio containing all possible pairs, the average excess returns are lower, resulting in an annualized SR of 1.43. Again, we regress the daily excess returns on the three factors of Fama & French (1993) and Carharts (1997) momentum factor to assess the systematic risk of the strategy. The risk factors' coefficients are in general similar to those in the US, both in terms of size and signs; however, the regression produces much fewer significant results implying that these factors cannot properly explain the return we obtain at OSE. The risk-adjusted returns are all positive and significant at a 1% level, and the portfolio containing all cointegrated pairs obtain higher risk-adjusted returns than raw excess returns also in this case. The R^2 is very low, ranging from 0.002-0.005, implying that the risk factors generally fail to explain the returns obtained by the strategy.

Table 5.3: *Summary of descriptive statistics and systematic risk of pairs trading in Norwegian equities*

Pairs portfolio	*Top 5*	*Top 10*	*Top 20*	*All*
A: Excess return distribution				
Average daily excess return	*0.00186*	*0.00192*	*0.00183*	*0.00096*
Annualized excess return	*0.46872*	*0.48384*	*0.46116*	*0.24192*
t-Statistic	*5.48480*	*6.78365*	*8.18620*	*6.15468*
P-value	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Excess return distribution				
Median	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Standard deviation	*0.02312*	*0.01927*	*0.01526*	*0.01066*
Skewness	*0.62709*	*0.37796*	*0.74100*	*0.12691*
Kurtosis	*15.39224*	*9.05495*	*11.31566*	*16.28963*
Minimum	*-0.18488*	*-0.12301*	*-0.11293*	*-0.11293*
Maximum	*0.27059*	*0.21724*	*0.18203*	*0.09332*
B: Risk Characteristics				
Annualized Sharpe Ratio	*1.28*	*1.58*	*1.90*	*1.43*
Intercept	*0.00180*	*0.00180*	*0.00170*	*0.00100*
	*(5.24400)****	*(6.33800)****	*(7.73100)****	*(6.18900)****
Market	*-0.04250*	*-0.08970*	*-0.07480*	*-0.00170*
	(-1.03600)	*(-2.62000)****	*(-2.75800)****	*(-0.08800)*
SMB	*0.09720*	*0.04840*	*0.05790*	*0.04870*
	*(2.11800)***	*(-0.18500)*	*(-0.50200)*	*(0.29500)*
HML	*-0.01910*	*-0.05730*	*-0.02120*	*0.00660*
	(-0.54500)	*(-1.96300)***	*(-0.91600)*	*(0.40900)*
Momentum	*-0.05010*	*-0.03070*	*-0.03660*	*-0.04120*
	(-1.63000)	*(-1.19800)*	*(-1.80700)**	*(-2.90700)****
R^2	*0.00500*	*0.00400*	*0.00300*	*0.00200*

Summary statistics of the daily excess returns on OSE for various portfolios of pairs between January 2001 and July 2019 containing 4662 observations. Trades are made according to the prespecified rule where trades are opened once the spread between two cointegrated stocks in the same cluster deviate by more than two standard deviations, measured over a 20-day rolling window (Panel A). Top "n" portfolios consists of the "n" number of pairs with the highest reported Sharpe Ratio in the formation period. Panel B presents a summary of the strategy's risk characteristics with daily returns regressed on the Fama & French (1993) three factor model and Carharts (1997) Momentum factor.

5.4 Market liquidity

This chapters looks at the liquidity in the two markets covered in this study. We begin by studying the overall historical liquidity in the markets using the two measures covered in chapter 4.11. We proceed to sort stocks based on their measured liquidity, and measure the performance of the strategy when trading at the most liquid and the least liquid stocks separately. We do this to analyse how the liquidity of the stocks traded impact the strategy performance.

5.4.1 Liquidity on the S&P500 and OSE

To help us understand the difference in profitability in the two markets using the strategy, we examine the overall market liquidity looking at both an order-based measure (relative spread) and a trade-based measure (turnover). Given that the S&P500 index consists of large US companies while the OSE data includes all companies listed on OSE, we should expect to see a difference in liquidity in the two datasets.

The average relative bid-ask spread for the S&P500 stocks in the period 2000 – 2019 is 0.30%, while for the OSE stocks, the average relative spread in the same period is 2.78%. Looking at figure 5.7, we observe a downwards sloping trend for OSE, while the average spread at S&P500 appears to have been relatively stable since 2004. In both markets, there were large spikes around periods of market turmoil, such as the global financial crisis in 2008.

Figure 5.7: *Relative bid-ask spread on the S&P500 and OSE*

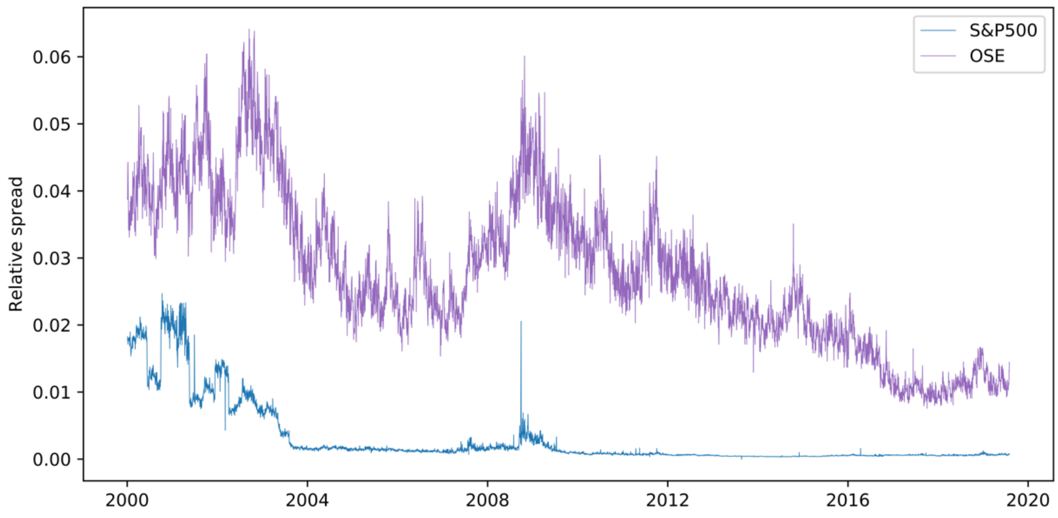
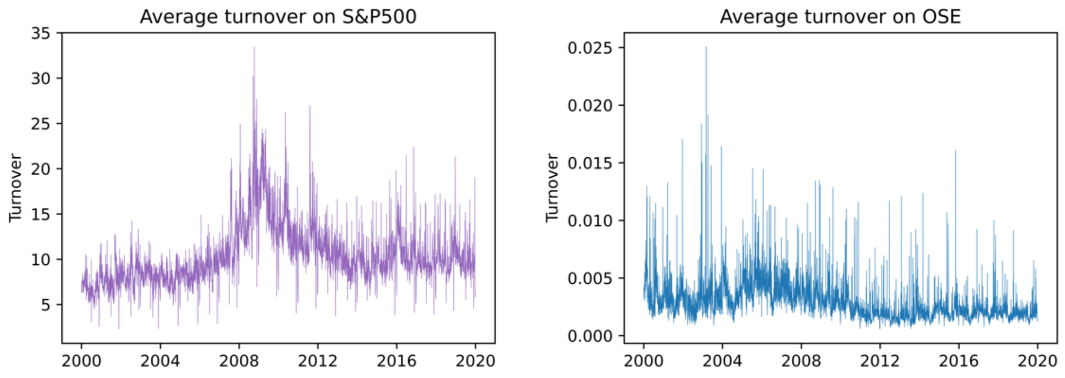


Chart displaying the relative bid-ask spread measured at closing for both the S&P500 and OSE from 2000 through 2020.

Additionally, we examine the average turnover in the two markets. Panel A and B in figure 5.8 display the average turnover in the two markets calculated as the daily number of shares traded divided by the number of shares outstanding. Again, we observe considerable differences between the two markets with a much higher turnover on the S&P500 than on the OSE, averaging at 1035% and 0.3%, respectively. In the period considered, the turnover appears to have declined on the OSE, while on the S&P500 it has increased.

Figure 5.8: *Turnover on the S&P500 and OSE*



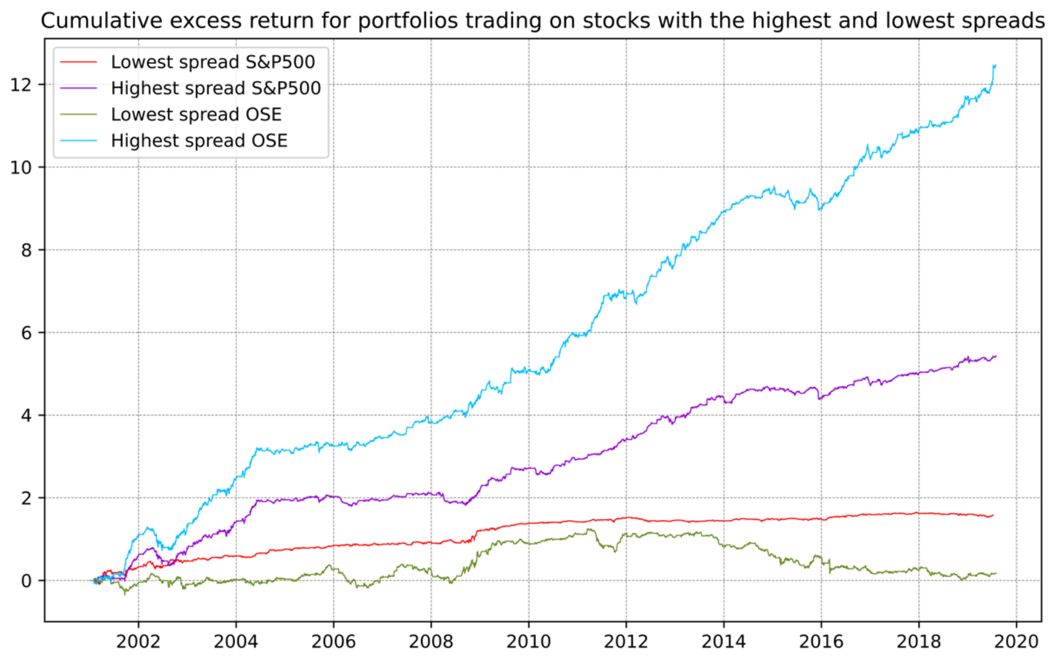
Panel A and B shows the average turnover (equal weighted) on the S&P500 and OSE respectively in the period 2000 – 2020, calculated as the number of shares traded divided by the number of shares outstanding.

Based on these measures, stocks listed at OSE appear to be less liquid in terms of both orders-based and trade-based measures. This increases our suspicion that the difference in market liquidity could help us explain why pairs trading appears to be much more profitable at OSE than on the S&P500.

5.4.2 Trading on the most liquid and the least liquid stocks

To further investigate the effect of liquidity on pairs trading profits, we want to observe the performance of the strategy when trading only on the least liquid stocks and the most liquid stocks, separately. As a measure of liquidity we use the observed relative bid-ask spread. At the end of very formation period, the average daily relative spread is computed and we use this information to construct separate portfolios containing stocks with the highest and lowest spread which will be traded in the following six months. Specifically, we select the 30th percentile of stocks that had the highest relative spread and the 30th percentile of stocks that had lowest relative spread in the formation period. We measure the spread and construct new portfolios after each formation period and for each of the two markets. We do not run the clustering algorithm but test for cointegration and trade according to the same rule as before. Table A3 in the appendix summarizes the excess return distribution for the constructed portfolios. The excess returns obtained by trading on stocks with the highest spread evidently outperform those obtained by trading on the stocks with lowest spread (figure 5.9). The difference is more prominent on the OSE, where the difference in average daily excess return between the two portfolios is 26 bps compared to 8 bps for the S&P500 portfolios. The portfolio trading on the *least* liquid stocks on OSE obtains statistically significant annualized excess returns of 67.36% over the 20-year long period (before TC) with an annualized SR of 1.82. Contrary, when trading on the *most* liquid stocks on OSE, the annualized excess returns are close to zero and not statistically significant. The excess returns on the portfolios trading stocks on the S&P500 are all statistically significant at the 1% level. The "top spread" portfolio returns 29.36% annually compared to 8.54% for the "bottom spread" portfolio. These results indicate that the profitability of pairs trading might be linked to the liquidity of the traded stocks, which potentially explains why pairs trading seems to be more profitable on OSE than on the S&P500. It may also be the case that the larger bid-ask spread increases the upward bias of returns because of a bid-ask bounce, and that the extra profits are, in reality, eaten up by the costs of trading illiquid stocks.

Figure 5.9: *Performance of the least liquid and most liquid stocks*



Cumulative excess returns from the portfolios trading on the most liquid stocks and the portfolios trading on the least liquid stocks on the S&P500 and OSE. Stocks are sorted by the relative spread. The top and bottom 30th percentile are used to construct portfolios.

5.5 Trading costs

As mentioned previously, we make the distinction between explicit and implicit transaction costs. In this chapter, we adjust for both explicit and implicit TC that a trader would incur when pursuing the strategy. The implicit costs are difficult to measure precisely, and we therefore apply one standard estimate as well as one “worst-case scenario” estimate to stress the results. First, we add a 1-day lag from we receive our trading signal until we trade to obtain an estimate of the implied bid-ask spread in the respective markets. Second, to simulate a worst-case scenario, we directly impose a TC equal to the average relative bid-ask spread in each respective market. This approach assumes that we must cross the order book and pay the bid-ask spread on every transaction. Other implementation costs, such as slippage, are not considered in this study.

5.5.1 Robustness to explicit transaction costs

We begin by testing whether the results obtained in chapters 5.3.1 and 5.3.2 are robust to explicit TC, including commission and short selling fees. For both the S&P500 and OSE portfolios, the average number of days a portfolio is held open is roughly nine days. Combined with the fixed commission of 5 bps gives us a total TC of 36.11 bps per trade (two roundtrips with short-selling fees). Table A1 in the appendix section shows that the average daily excess returns are close to zero and not statistically significant for any of the S&P500 portfolios once adjusted for explicit TC. It is worth noting that pairs trading is a strategy with a relatively high frequency of trades, meaning that the returns will be highly susceptible to the estimate of TC. We observe that while the results before TC are similar to what previous studies find, we obtain much poorer results after TC. A potential explanation could be the frequency of trades made, which is considerably higher than for example Gatev et al. (2006). On OSE, we observe slightly reduced excess returns for all portfolios. The average annualized excess returns decrease by 7.71 percentage points (pp) for the top 5 portfolio, and 8.44 pp for the portfolio containing all pairs (Table A2). We argue that the reason why loss from TC increases with the size of the portfolio is that more trades are executed, leading to additional roundtrips of costs. Nevertheless, the excess returns are still positive with the average annualized excess return for the “top” portfolios being 41.08%. The returns are statistically significant at the 1% level for all OSE portfolios.

5.5.2 Adjusting for bid-ask spreads

As discussed in chapter 4.11, the returns obtained may be biased upwards because of the bid-ask bounce. To adjust for this effect, we test the strategy's performance with one day delay from the time we receive the trading signal to a position is taken, both when opening and closing a position. The average daily excess returns on the top 20 portfolio drop by 0.72 bps and 6.89 bps for the S&P500 and OSE, respectively, compared to the results obtained when only including explicit TC (Table 5.4). The daily excess returns from the S&P500 are still close to zero and not statistically significant. These results corresponds to a drop in the annual excess return from 2.34% to 0.53% on the S&P500, and from 39.39% to 22.02% on the OSE, which provides us with an estimate of the average bid-ask spread and hence the implicit TC. Thus, the results suggest that a substantial part of the excess returns on OSE may be driven by the bid-ask bounce, although it is difficult to measure how much of the decline is due to actual price convergence. We observe that the decline is greater on the OSE, which may be due to the higher bid-ask spread, causing bid-ask bounces of greater magnitude. The annualized sharpe ratio is still relatively high at 0.84 for the top 20 portfolio. Although the excess returns on OSE are lower than before applying the 1 day lag, they are all still positive and statistically significant at the 1% level, indicating that the trading strategy survives this estimate of explicit *and* implicit TC.

We also apply the 1-day lag and the explicit TC estimate to the "top spread" portfolios constructed in chapter 5.4.2 and observe that even the most illiquid stocks on OSE survives the transaction costs (Table A4). The average annualized excess returns decline by 14.87 percentage points from 67.36% to 52.49%. While the drop in excess returns are quite large, the excess returns are still very high and statistically significant at the 1% level. For the S&P500 top spread portfolio, the annualized excess returns drop from 29.48% to 2.09%. It is interesting to see that the returns obtained on the OSE survives the estimated explicit and implicit costs while the returns obtained on the S&P500 disappear using these estimates. A possible explanation could be that even the least liquid stocks in the S&P500 are still more liquid than most of the OSE stocks, although we cannot conclude that this is causing the difference.

We recall from the definition of liquid markets that an important factor is being able to trade large numbers of shares with little price impact. After observing the evidenced inferior liquidity at the OSE, measured in turnover and spreads relative to the S&P500, we expect that the possibility to trade at observed prices heavily depends on the size of the trades. Given the difficult task of adequately accounting for this, we argue that applying the 1-day delay also provides a fitting estimate of the execution price because the trader might not be able to fill an order at the exact market close. It should also be noted that while the portfolio trading on the least liquid stocks on OSE perfoms very well, the volume that can be traded is likely to be very limited.

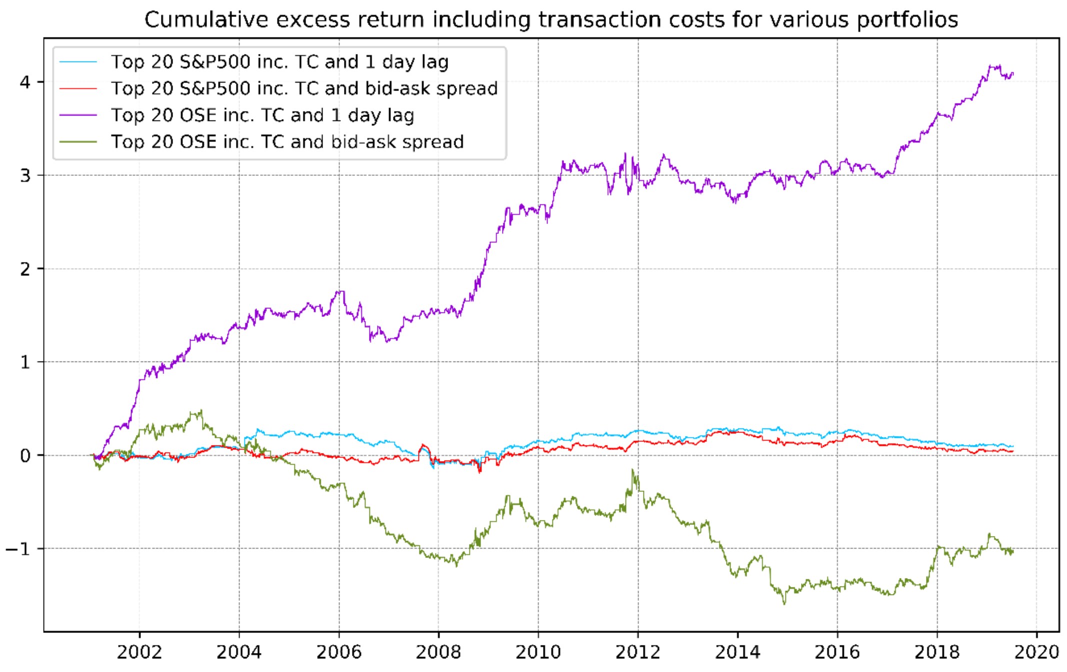
Table 5.4: *Summary of descriptive statistics and risk characteristics of pairs trading with 1-day lag and explicit TC*

Pairs portfolio	*Top 5*	*Top 10*	*Top 20*	*All*
A: S&P500 Daily excess return distribution with 1 day lag and explicit TC				
Average excess return	*0.00023*	*0.00007*	*0.00002*	*0.00001*
Annualized excess return	*0.05796*	*0.01764*	*0.00504*	*0.00252*
t-Statistic	*2.13060*	*0.77774*	*0.25933*	*0.22251*
pvalue	*0.03317*	*0.43676*	*0.79539*	*0.82393*
Excess return distribution				
Median	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Standard deviation	*0.00722*	*0.00618*	*0.00541*	*0.00419*
Skewness	*1.70128*	*2.19739*	*3.01913*	*1.19975*
Kurtosis	*26.03100*	*40.47192*	*63.01301*	*23.25694*
Minimum	*-0.04826*	*-0.04351*	*-0.04351*	*-0.04351*
Maximum	*0.11149*	*0.11944*	*0.11944*	*0.05289*
Annualized Sharpe ratio	*0.51*	*0.18*	*0.06*	*0.04*
B: OSE Daily excess return distribution with 1 day lag and explicit TC				
Average excess return	*0.00123*	*0.00109*	*0.00087*	*0.00075*
Annualized excess return	*0.30895*	*0.27367*	*0.22025*	*0.18900*
t-Statistic	*3.33290*	*3.75884*	*3.62494*	*4.81246*
p-value	*0.00087*	*0.00017*	*0.00029*	*0.00000*
Excess return distribution				
Median	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Standard deviation	*0.02512*	*0.01972*	*0.01647*	*0.01064*
Skewness	*2.21012*	*0.01662*	*-0.21437*	*0.08400*
Kurtosis	*53.03421*	*22.03774*	*14.43742*	*16.75985*
Minimum	*-0.36398*	*-0.25299*	*-0.17157*	*-0.11654*
Maximum	*0.42244*	*0.27067*	*0.14072*	*0.09300*
Annualized Sharpe ratio	*0.77*	*0.87*	*0.84*	*1.12*

Summary statistics of the daily excess returns applying the same strategy as before but with 1 day lag on the opening and closing of positions.

While applying a 1-day lag might give us a decent estimate of trading costs in the sample, we also run the strategy without the 1-day lag but instead adding the observed average relative bid-ask spread as a transaction cost on each trade executed. This gives us an idea of how robust the returns are to a "worst-case" scenario where the bid-ask spread is paid on every transaction. We see from table A4 in the appendix section that this adjustment leads to negative excess returns for most of the S&P500 portfolios, which were barely positive when we adjusted for explicit TC earlier. None of the excess returns are statistically significant, and we argue that when considering all transaction costs, the strategy is not profitable on the S&P500. Similarly, adding the relative spread on the OSE stocks eliminates most of the profits that were left after adjusting for the explicit TC only. The top 5 and top 10 portfolios obtain positive excess returns, but neither are statistically significant. The portfolio trading on all pairs actually generates significant negative excess returns. Interestingly, the method applied does not seem to make much of a difference on the S&P500 portfolios, while the returns on the OSE are highly sensitive to the method used to estimate implicit TC (Figure 5.10). This is likely a result of higher trading activity than the other portfolio combined with a much higher observed relative spread on the OSE than for the S&P500 portfolios.

Figure 5.10: *Strategy performance after TC*



Plot showing the cumulative excess returns when applying the estimated transaction costs and two different measures of bid-ask spreads.

The estimates obtained by using the 1-day lag approach may be more realistic, but it is interesting to observe how robust the returns are to the "worst-case" estimate. While the excess returns are close to zero for the S&P500 portfolios irrespective of which method is applied, the OSE portfolios obtain positive and statistically significant returns when adjusting for explicit TC and applying a 1-day lag to simulate the implied bid-ask spread. On the other hand, applying the observed bid-ask spread as an additional transaction cost suggest that the cost of trading at OSE eats up the excess returns generated by the strategy. Most previous studies appear to use the 1-day lag as an estimate of the bid-ask spreads, and we emphasize these results as we believe they gives us the most realistic representation of trading costs as well as it facilitates comparison of results with previous studies.

5.6 Summary of results and theoretical implications

In chapter 1.2, we outlined the following hypotheses which are tested separately for the two markets in our study:

H_0: *A pairs trading strategy using machine learning does not produce positive excess returns*

H_1: *A pairs trading strategy using machine learning produce positive excess returns*

We find that a pairs trading strategy using machine learning is not able to generate positive excess returns on the S&P500 once adjusted for TCs, and we therefore fail to reject the null hypothesis for the US market. When trading at OSE however, we are able to generate positive and statistically significant excess returns after adjusting for TCs; thus, we reject the null hypothesis and claim that pairs trading using a machine learning approach produce positive excess returns on the OSE. Further, we outlined two additional hypotheses:

H_0: *Pairs trading is not more profitable in markets with lower liquidity*

H_1: *Pairs trading is more profitable in markets with lower liquidity*

In our analysis, we prove that the OSE is far less liquid than the S&P500. As we find the strategy to be more profitable on the OSE than the S&P500, we reject the null hypothesis and claim that pairs trading is more profitable in markets with lower liquidity. To back up this hypothesis, we go on to show that applying the pairs trading strategy on the least liquid stocks yields significantly higher excess returns than when applied to the most liquid stocks before adjusting for TC. This is the case for both the S&P500 and OSE, which again suggest that market liquidity is an essential driver of profitability. To address the concern that what looks like larger profits may represent higher trading costs, we adjust for the implied bid-ask spread and still obtain positive and significant results when trading on the OSE. Using a more aggressive worst-case estimate of the impact of the bid-ask spread, our returns are however eliminated. Still, we argue that the probability of this worst-case estimate to occur is low, and base our conclusion on the less aggressive estimate using a 1-day lag.

We also adjust for the implied bid-ask spread using a 1 da lag on the portfolios trading on the least liquid stocks in each of the two markets. The returns from the S&P500 stocks with the highest spread are eliminated, while the OSE portfolio still obtain significantly positive excess returns.

Analyzing the results in light of our theoretical point of departure, we are able to comment on a few interesting observations. While we are able to produce positive excess returns on the S&P500 before TC, most of these returns are eliminated once adjusted for TC using a conservative estimate. This indicates that the US markets exhibit a weak form efficiency and that we are not able to exploit any inefficiencies or mispricings. Even when trading on the least liquid stocks, the market seems to be relatively efficient and the strategy does not produce positive results. The strategy proves to be more profitable on the OSE, suggesting that there might be some inefficiencies that we are able to exploit in this market that is much less traded than the S&P500. Going back to the framework by Grossman & Stiglitz (1980), it appears that the unsupervised learning model is able to extract information of value on OSE but not on the S&P500. A possible explanation could be that there are a much larger fraction of informed traders on the S&P500 than on OSE, resulting in lower returns in the former market. Another interesting observation is that the profitability does not appear to be time dependant in the sample period, contrary to most literature that reports declining profits in recent years. This is not in line with our expectations that a machine learning approach would be able to produce higher returns in the past where such techniques were not widely available.

5.7 Caveats

This analysis is exposed to a few pitfalls that are worth mentioning. We assume that we can trade on closing prices when implementing the trading strategy, presenting us with a “look-ahead bias”. While it is not entirely unreasonable to assume that we can execute our order on the exact closing price, we should note that this may not always be the case in the “real world”, and the slippage costs could potentially be significant. Further, some of the stocks included on OSE are relatively illiquid and rarely traded, meaning that we have a few missing values in the dataset on days that no trades were made. As we chose to backfill these empty data points with the previous day’s close (unless there are more than ten days without data, in which case we remove the stock from the dataset), we assume that we could trade on the previous price, which might not always be the case. Gatev et al. (2006) addressed this concern and did not find it to be a major issue in the obtained results. Another problem with illiquid stocks is that it can be difficult to find shares available to borrow so that the stock can be shorted. While this is not a concern for stocks included in the S&P500, it is likely an issue for many of the illiquid stocks at OSE.

6. Conclusion

We prove that an unsupervised machine learning algorithm is able to identify stocks that are similar in terms of risk and often linked to the same industry, making them good candidates for pairs trading. Additionally, the algorithm form clusters with stocks from different industries which still prove to be cointegrated. This suggests that unsupervised machine learning does help us discover patterns that are not entirely intuitive. Our results have shown that a simple pairs trading strategy building on an unsupervised machine learning approach does not generate sufficient excess returns to cover a conservative estimate of explicit transaction costs on the S&P500. Conversely, the same trading strategy appears to be profitable on OSE even when adjusting for both explicit and implicit transaction costs. We have shown that the profitability of pairs trading appears to be closely related to the market liquidity of the stocks that are traded, which might explain why the trading strategy appears to be more profitable at OSE.

7. Further Research

The output of the clustering algorithm indicates that keeping the parameters fixed throughout all periods might not be optimal. Further research should investigate whether non-constant parameters impact the clustering and thus the profitability of the strategy. Additionally, the application of other clustering techniques such as OPTICS in the context of pairs trading requires additional research. While this study makes an effort to determine the profitability of a pairs trading strategy in two markets with different liquidity, further analysis on the actual costs of applying the strategy on illiquid stocks is required. Further, we only compare two different markets, and expanding the data to cover additional markets could potentially increase the validity of the results.

Bibliography

Abdi, H., & Williams, L. J. (2010). Principal component analysis. *WIREs Computational Statistics*, *2*(4), 433–459. https://doi.org/10.1002/wics.101

Avellaneda, M., & Lee, J.-H. (2008). Statistical Arbitrage in the U.S. Equities Market. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1153505

Avellaneda, M., & Lee, J.-H. (2010). Statistical arbitrage in the US equities market. *Quantitative Finance*, *10*(7), 761–782. https://doi.org/10.1080/14697680903124632

Becker, J. (2012). *Making Money with statistical Arbitrage: Generating Alpha in sideway Markets with this Option Strategy: Generating Alpha in sideway Markets with this Option Strategy*. Diplomica Verlag. http://ebookcentral.proquest.com/lib/bilibrary/detail.action?docID=3033280

Bellman, R. (1966). Dynamic Programming. *Science*, *153*(3731), 34. https://doi.org/10.1126/science.153.3731.34

Bodie, Z., Kane, A., & Marcus, A. J. (2018). *Investments* (Eleventh edition). McGraw-Hill Education.

Bondarenko, O. (2003). Statistical Arbitrage and Securities Prices. *The Review of Financial Studies*, *16*(3), 875–919.

Broussard, J. P., & Vaihekoski, M. (2012). Profitability of pairs trading strategy in an illiquid market with multiple share classes. *Journal of International Financial Markets, Institutions and Money*, *22*(5), 1188–1201. https://doi.org/10.1016/j.intfin.2012.06.002

Caldeira, J., & Moura, G. V. (2013). Selection of a Portfolio of Pairs Based on Cointegration: A Statistical Arbitrage Strategy. *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.2196391

Clegg, M., & Krauss, C. (2018). Pairs trading with partial cointegration. *Quantitative Finance*, *18*(1), 121–138. https://doi.org/10.1080/14697688.2017.1370122

Do, B., & Faff, R. (2010). Does Simple Pairs Trading Still Work? *Financial Analysts Journal*, *66*(4), 83–95. https://doi.org/10.2469/faj.v66.n4.1

Do, B. H., & Faff, R. W. (2011). Are Pairs Trading Profits Robust to Trading Costs? *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.1707125

Elliott, R. J., Van Der Hoek *, J., & Malcolm, W. P. (2005). Pairs trading. *Quantitative Finance*, *5*(3), 271–276. https://doi.org/10.1080/14697680500149370

Engle, R. F., & Granger, C. W. J. (1987). Co-Integration and Error Correction: Representation, Estimation, and Testing. *Econometrica*, *55*(2), 251. https://doi.org/10.2307/1913236

Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance*, *25*(2), 383. https://doi.org/10.2307/2325486

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, *33*(1), 3–56. https://doi.org/10.1016/0304-405X(93)90023-5

Gatev, E., Goetzmann, W. N., & Rouwenhorst, K. G. (2006). Pairs Trading: Performance of a Relative-Value Arbitrage Rule. *The Review of Financial Studies*, *19*(3), 797–827. https://doi.org/10.1093/rfs/hhj020

Gatev, E., Goetzmann, W., & Rouwenhorst, K. G. (1999). *Pairs Trading: Performance of a Relative Value Arbitrage Rule* (No. w7032; p. w7032). National Bureau of Economic Research. https://doi.org/10.3386/w7032

Grossman, S. J., & Stiglitz, J. (1980). On the Impossibility of Informationally Efficient Markets. *The American Economic Review*, *70*(3), 393–408.

Hendry, D. F., & Juselius, K. (2000). Explaining Cointegration Analysis: Part 1. *The Energy Journal*, *21*(1), 1–42. JSTOR.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (Eds.). (2013). *An introduction to statistical learning: With applications in R*. Springer.

Jegadeesh, N., & Titman, S. (1993). Returns to Buying Winners and Selling Losers: Implications for Stock Market Efficiency. *The Journal of Finance*, *48*(1), 65–91. https://doi.org/10.1111/j.1540-6261.1993.tb04702.x

Kendall, M. G., & Hill, A. B. (1953). The Analysis of Economic Time-Series-Part I: Prices. *Journal of the Royal Statistical Society. Series A (General)*, *116*(1), 11. https://doi.org/10.2307/2980947

Næs, R., Skjeltorp, J. A., & Ødegaard, B. A. (2008). *Liquidity at the Oslo Stock Exchange*.

Naes, R., Skjeltorp, J. A., & Ødegaard, B. A. (2011). Stock Market Liquidity and the Business Cycle. *The Journal of Finance*, *66*(1), 139–176. https://doi.org/10.1111/j.1540-6261.2010.01628.x

Pole, A. (2007). *Statistical arbitrage: Algorithmic trading insights and techniques*. J. Wiley & Sons.

Rad, H., Low, R. K. Y., & Faff, R. (2016). The profitability of pairs trading strategies: Distance, cointegration and copula methods. *Quantitative Finance*, *16*(10), 1541–1558. https://doi.org/10.1080/14697688.2016.1164337

Ross, S. A. (1976). The arbitrage theory of capital asset pricing. *Journal of Economic Theory*, *13*(3), 341–360. https://doi.org/10.1016/0022-0531(76)90046-6

Sander, J., Ester, M., Kriegel, H.-P., & Xu, X. (1998). Density-Based Clustering in Spatial Databases: The Algorithm GDBSCAN and Its Applications. *Data Mining and Knowledge Discovery*, *2*(2), 169–194. https://doi.org/10.1023/A:1009745219419

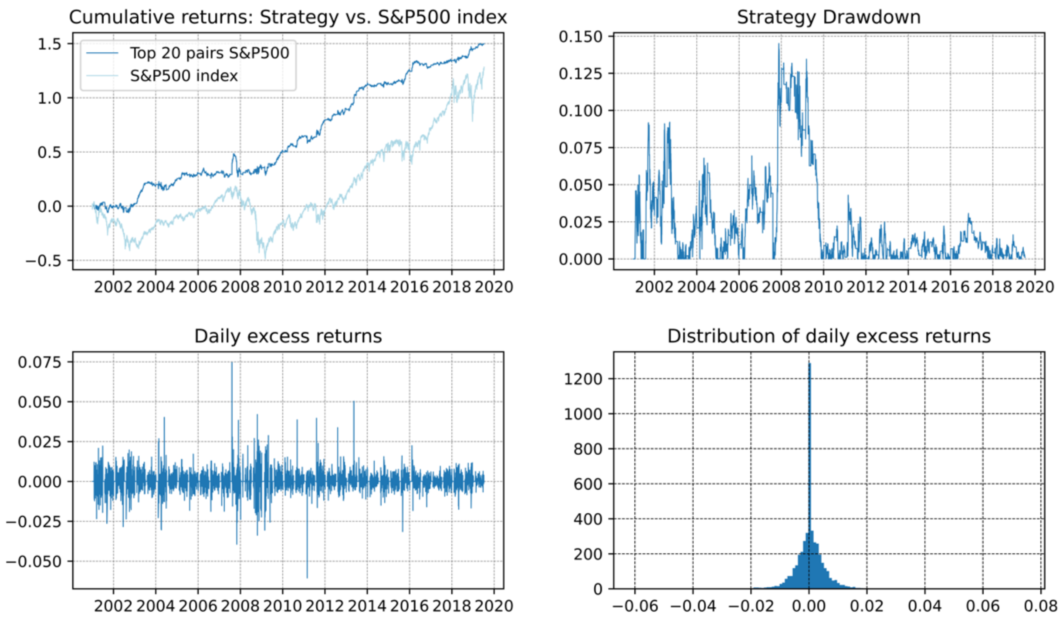
Sarmento, S. M., & Horta, N. (2020). Enhancing a Pairs Trading strategy with the application of Machine Learning. *Expert Systems with Applications*, *158*, 113490. https://doi.org/10.1016/j.eswa.2020.113490

van der Maaten, L., & Hinton, G. (2008). Visualizing Data using t-SNE. *Journal of Machine Learning Research*, *9*, 2579–2605.

Wooldridge. (2015). *Introductory Econometrics*.

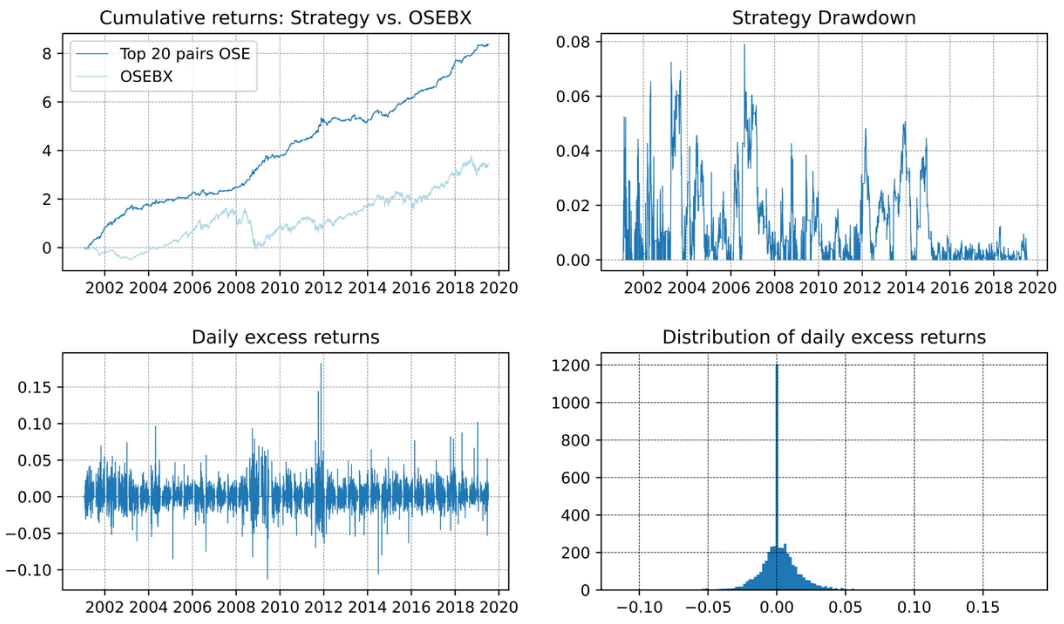
Appendix

Figure A1: *Top 20 pairs S&P500*



Summary of strategy performance for the top 20 portfolio on S&P500. The cumulative strategy return is compared with the return of the S&P500 index in the top left chart.

Figure A2: *Top 20 pairs OSE*



Summary of strategy performance for the top 20 portfolio on the OSE. The cumulative strategy return is compared with the return of the OSEBX index in the top left chart.

Table A1: *S&P500 results with explicit TC*

Pairs portfolio	*Top 5*	*Top 10*	*Top 20*	*All*
A: Excess return distribution				
Average excess return	*0.00002*	*0.00003*	*0.00009*	*0.00003*
Annualized excess return	*0.00605*	*0.00630*	*0.02344*	*0.00630*
t-Statistic	*0.21580*	*0.26925*	*1.14699*	*0.38215*
pvalue	*0.82915*	*0.78775*	*0.25144*	*0.70237*
Excess return distribution				
Median	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Standard deviation	*0.00745*	*0.00633*	*0.00551*	*0.00450*
Skewness	*0.15589*	*0.44111*	*0.78204*	*0.79083*
Kurtosis	*9.73473*	*12.42605*	*16.84887*	*19.09057*
Minimum	*-0.05467*	*-0.04748*	*-0.04351*	*-0.04351*
Maximum	*0.07091*	*0.07091*	*0.07091*	*0.05289*
B: Systematic risk of pairs trading				
Sharpe Ratio	*0.00322*	*0.00395*	*0.01688*	*0.00556*
Intercept	*0.00007*	*0.00003*	*0.00009*	*0.00002*
	(0.64500)	*(0.29200)*	*(1.13600)*	*(0.34300)*
Market	*-0.01890*	*-0.00340*	*0.00580*	*0.01310*
	*(-1.87000)**	*(-0.39600)*	*(0.78400)*	*(2.17000)***
SMB	*0.01340*	*0.03790*	*0.04400*	*0.03140*
	(0.68300)	*(2.30100)***	*(3.07000)****	*(2.69100)****
HML	*-0.02710*	*-0.02830*	*-0.02060*	*-0.02580*
	(-1.47200)	*(-1.82400)**	*(-1.52500)*	*(-2.34600)***
Momentum	*-0.03170*	*-0.03700*	*-0.04020*	*-0.03200*
	*(-2.46600)***	*(-3.40900)****	*(-4.25900)****	*(-4.15800)****
R^2	*0.00200*	*0.00400*	*0.00700*	*0.00900*

Panel A: Summary statistics of the daily excess returns applying the strategy on the S&P500 but adjusted for standard transaction cost such as commissions and short selling fees. Panel B: Summary of risk profile of the obtained returns. Daily returns regressed against Fama-French three factor model and Carhart's momentum factor.

Table A2: *OSE results with explicit TC*

Pairs portfolio	*Top 5*	*Top 10*	*Top 20*	*All*
A: Excess return distribution				
Average excess return	*0.00163*	*0.00169*	*0.00156*	*0.00072*
Annualized excess return	*0.41177*	*0.42664*	*0.39388*	*0.18043*
t-Statistic	*4.98742*	*6.07211*	*7.01941*	*4.60163*
p-value	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Excess return distribution				
Median	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Standard deviation	*0.02237*	*0.01904*	*0.01520*	*0.01062*
Skewness	*0.67228*	*0.38684*	*0.64957*	*0.08351*
Kurtosis	*16.40217*	*9.20571*	*11.51896*	*16.56312*
Minimum	*-0.18488*	*-0.12482*	*-0.11293*	*-0.11293*
Maximum	*0.27059*	*0.21724*	*0.18203*	*0.09326*
B: Systematic risk of pairs trading				
Sharpe ratio	*0.07305*	*0.08894*	*0.10281*	*0.06741*
Intercept	*0.00160*	*0.00160*	*0.00140*	*0.00070*
	*(4.85600)****	*(5.70600)****	*(6.44900)****	*(4.69200)****
Market	*-0.01360*	*-0.07420*	*-0.06840*	*0.00170*
	(-0.34100)	*(-2.19300)***	*(-2.56800)****	*(0.09100)*
SMB	*0.10080*	*-0.01030*	*-0.01260*	*0.00490*
	*(2.26800)***	*(-0.27300)*	*(-0.42300)*	*(0.23400)*
HML	*0.00370*	*-0.03900*	*-0.01030*	*0.01590*
	(0.10800)	*(-1.35300)*	*(-0.45200)*	*(0.98600)*
Momentum	*-0.04790*	*-0.03280*	*-0.03440*	*-0.04460*
	(-1.61100)	*(-1.29600)*	*(-1.72600)***	*(-3.15500)****
R^2	*0.00400*	*0.00200*	*0.00300*	*0.00200*

Panel A: Summary statistics of the daily excess returns applying the strategy on the OSE but adjusted for standard transaction cost such as commissions and short selling fees. Panel B: Summary of risk profile of the obtained returns. Daily returns regressed against Fama-French three factor model and Carhart's momentum factor.

Table A3: *Performance on Bottom and Top spread portfolios*

	S&P500	*OSE*
A: Bottom spread portfolio		
Average excess return	*0.00034*	*0.00004*
Annualized excess return	*0.08568*	*0.01008*
t-Statistic	*3.62283*	*0.16730*
p-value	*0.00029*	*0.86714*
Excess return distribution		
Median	*0.00000*	*0.00000*
Standard deviation	*0.00638*	*0.01532*
Skewness	*1.69139*	*-0.23631*
Kurtosis	*21.02821*	*14.98725*
Minimum	*-0.04615*	*-0.18605*
Maximum	*0.08895*	*0.12610*
Annualized Sharpe Ratio	*0.85*	*0.04*
B: Top spread portfolio		
Average excess return	*0.00117*	*0.00267*
Annualized excess return	*0.29484*	*0.67284*
t-Statistic	*5.82546*	*7.83020*
p-value	*0.00000*	*0.00000*
Excess return distribution		
Median	*0.00000*	*0.00000*
Standard deviation	*0.01366*	*0.02331*
Skewness	*0.51067*	*0.85283*
Kurtosis	*9.35137*	*10.35755*
Minimum	*-0.11031*	*-0.16418*
Maximum	*0.11193*	*0.27856*
Annualized Sharpe Ratio	*1.36*	*1.82*

The table summarize the excess returns and its distribution for the portfolios containing the 30% most liquid stocks and the portfolios containing the 30% least liquid stocks for both the S&P500 and OSE, measured by the relative bid-ask spread.

Table A4: *Performance on Bottom and Top spread portfolios with 1 day lag and including explicit TC*

	S&P500	*OSE*
A: Bottom spread portfolio inc. 1 day lag and explicit TC		
Average excess return	*0.00005*	*0.00003*
Annualized excess return	*0.01310*	*0.00781*
t-Statistic	*0.59829*	*0.09267*
pvalue	*0.54967*	*0.92617*
Excess return distribution:		
Median	*0.00000*	*0.00000*
Standard deviation	*0.00594*	*0.02272*
Skewness	*1.71382*	*3.06829*
Kurtosis	*23.29081*	*158.90822*
Minimum	*-0.04544*	*-0.39856*
Maximum	*0.08193*	*0.58065*
Annualized Sharpe Ratio	*0.13362*	*0.02166*
B: Top spread portfolio inc. 1 day lag and explicit TC		
Average excess return	*0.00008*	*0.00208*
Annualized excess return	*0.02092*	*0.52492*
t-Statistic	*0.70225*	*4.49086*
pvalue	*0.48256*	*0.00001*
Excess return distribution:		
Median	*0.00000*	*0.00000*
Standard deviation	*0.00807*	*0.03168*
Skewness	*1.05147*	*0.82808*
Kurtosis	*10.93792*	*12.94110*
Minimum	*-0.04891*	*-0.24811*
Maximum	*0.07218*	*0.38182*
Annualized Sharpe Ratio	*0.16319*	*1.04393*

The table summarize the excess returns and its distribution for the portfolios containing the top and bottom 30th percentile of stocks sorted by their liquidity. The relative bid-ask spread is used as a measure of liquidity.

Table A5: *S&P500 Performance with explicit TC and relative BA-spread*

Pairs portfolio	*Top 5*	*Top 10*	*Top 20*	*All*
A: Excess return distribution				
Average excess return	*-0.00005*	*-0.00005*	*0.00001*	*-0.00006*
Annualized excess return	*-0.01210*	*-0.01361*	*0.00227*	*-0.01512*
t-Statistic	*-0.43665*	*-0.58213*	*0.11314*	*-0.90302*
p-value	*0.66238*	*0.56051*	*0.90993*	*0.36656*
Excess return distribution:				
Median	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Standard deviation	*0.00751*	*0.00634*	*0.00552*	*0.00451*
Skewness	*0.30682*	*0.40876*	*0.73320*	*0.75858*
Kurtosis	*9.26784*	*12.16502*	*16.47296*	*18.99981*
Minimum	*-0.05467*	*-0.04748*	*-0.04351*	*-0.04351*
Maximum	*0.06978*	*0.06978*	*0.06978*	*0.05289*
B: Risk characteristics of pairs trading				
Sharpe Ratio	*-0.00639*	*-0.00851*	*0.00163*	*-0.01329*
Intercept	*-0.00004*	*-0.00005*	*0.00001*	*-0.00006*
	(-0.39100)	*(-0.55800)*	*(0.64500)*	*(-0.94500)*
Market	*-0.01070*	*-0.00380*	*-0.01890*	*0.01300*
	(-1.06300)	*(-0.44100)*	*(-1.87000)**	*(2.14500)***
SMB	*0.03260*	*0.03750*	*0.01340*	*0.03110*
	*(1.66400)**	*(2.27000)***	*(0.68300)*	*(2.65500)****
HML	*-0.03440*	*-0.02830*	*-0.02710*	*-0.02610*
	*(-1.86800)**	*(-1.81700)**	*(-1.47200)*	*(-2.36300)***
Momentum	*-0.03760*	*-0.03700*	*-0.03170*	*-0.03200*
	*(-2.91800)****	*(-3.40300)****	*(-2.46600)***	*(-4.14400)****
R^2	*0.00300*	*0.00400*	*0.00200*	*0.00900*

Panel A: Summary statistics of the daily excess returns applying the same pairs trading strategy as before but adjusted for standard transaction and average relative bid ask spreads. Panel B: Summary of risk profile of the obtained returns. Daily returns regressed against Fama-French three factor model and Carhart's momentum factor.

Table A6: *OSE Performance with explicit TC and relative BA-spread*

Pairs portfolio	*Top 5*	*Top 10*	*Top 20*	*All*
A: Excess return distribution				
Average excess return	*0.00015*	*0.00003*	*-0.00022*	*-0.00094*
Annualized excess return	*0.03755*	*0.00731*	*-0.05645*	*-0.23587*
t-Statistic	*0.44500*	*0.10192*	*0.99915*	*5.86567*
pvalue	*0.65634*	*0.91883*	*0.31777*	*0.00000*
Excess return distribution				
Median	*0.00000*	*0.00000*	*0.00000*	*0.00000*
Standard deviation	*0.02279*	*0.01946*	*0.01533*	*0.01090*
Skewness	*0.53614*	*0.21687*	*0.41488*	*-0.31438*
Kurtosis	*15.76179*	*9.07968*	*11.74605*	*16.18632*
Minimum	*-0.18488*	*-0.13571*	*-0.11293*	*-0.11293*
Maximum	*0.27059*	*0.21724*	*0.18203*	*0.09288*
B: Systematic risk of pairs trading	*0.10377*	*0.02365*	*-0.23197*	*-1.36317*
Sharpe ratio	*0.00654*	*0.00149*	*-0.01461*	*-0.08587*
Intercept	*0.00010*	*-0.00005*	*-0.00030*	*-0.00090*
	(0.40000)	*(-0.16000)*	*(-1.28100)*	*(-5.58600)*
Market	*0.00120*	*-0.06160*	*-0.05420*	*0.01540*
	(0.03000)	*(-1.77900)*	*(-1.98900)**	*(0.79200)*
SMB	*0.11840*	*0.00440*	*0.00240*	*0.02130*
	*(2.61400)****	*(0.11300)*	*(0.08000)*	*(0.98200)*
HML	*0.00830*	*-0.03380*	*0.00140*	*0.02190*
	(0.24000)	*(-1.14600)*	*(0.06200)*	*(1.32400)*
Momentum	*-0.04310*	*-0.02550*	*-0.02840*	*-0.04260*
	(-1.42400)	*(-0.98600)*	*(-1.39300)*	*(-2.93900)*
R^2	*0.00400*	*0.00200*	*0.00300*	*0.00200*

Panel A: Summary statistics of the daily excess returns applying the same pairs trading strategy as before but adjusted for standard transaction and average relative bid ask spreads. Panel B: Summary of risk profile of the obtained returns. Daily returns regressed against Fama-French three factor model and Carhart's momentum factor.

```
import pandas_datareader as web
import matplotlib.pyplot as plt
import datetime as datetime
import seaborn as sns
import matplotlib.cm as cm
import statsmodels.api as sm
from sklearn import linear_model
from sklearn.cluster import KMeans, DBSCAN
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
from sklearn import preprocessing
from statsmodels.tsa.stattools import coint
from statsmodels.tsa.stattools import adfuller
import statsmodels.regression.linear_model as rg
from scipy import stats
import statsmodels.api as sm
import statsmodels.tsa.stattools as ts
from statsmodels.api import add_constant
import os


#Import data
crsp_data = pd.read_csv('CRSP_data_FIXED.csv', index_col=0, sep=',')


# Import benchmarks
oslo_bors_benchmark_index = pd.read_csv(\
                            'Oslo_bors_benchmark_index.csv', index_col=0,\
                            sep=',', encoding='latin-1')
oslo_bors_benchmark_index.index = \
    pd.to_datetime(oslo_bors_benchmark_index.index, format='%Y%m%d')

SP500_index = pd.read_csv('SP500_benchmark_index.csv', index_col=0, sep=','\
                          , encoding='latin-1')
SP500_index.index = pd.to_datetime(SP500_index.index, format='%Y%m%d')

# Calculate cumulative return on benchmarks
oslo_bors_benchmark_index['return'] = \
    oslo_bors_benchmark_index['Oslo BĂ¸rs Benchmark Index_GI'].pct_change()
oslo_bors_benchmark_index['cumulative return'] = \
    np.cumprod(1+oslo_bors_benchmark_index['return'])-1
oslo_bors_benchmark_index.fillna(0)

SP500_index.index = pd.to_datetime(SP500_index.index, format='%Y%m%d')
SP500_index['cum_ret'] =  np.cumprod(1+SP500_index['sprtrn'])-1


#%%
#-----------------------------------------------------------------------------------------------
# Create FORMATION period datasets with daily stock prices
#-----------------------------------------------------------------------------------------------

list_of_formation_datasets_prices = []
y = 0
while y < (5040-126):
    temp = crsp_data.iloc[y:y+252]
    list_of_formation_datasets_prices.append(temp)
    y += 126

list_of_formation_datasets_prices.pop()
list_of_formation_datasets_prices.pop()
```

```
#-----------------------------------------------------------------------------------------------

list_of_trading_datasets_prices = []
y = 0
while y < (5040-126):
    temp = crsp_data.iloc[y:y+126]
    list_of_trading_datasets_prices.append(temp)
    y += 126

list_of_trading_datasets_prices.pop(0)
list_of_trading_datasets_prices.pop(0)


#-----------------------------------------------------------------------------------------------
# Clean data for missing values
#-----------------------------------------------------------------------------------------------

# for formation datasets
for dataset in list_of_formation_datasets_prices:
    dataset.dropna(axis=1, how='all', thresh=None, subset=None, inplace=True)

for dataset in list_of_formation_datasets_prices:
    dataset.fillna(method = 'bfill', inplace=True, limit=10)

for dataset in list_of_formation_datasets_prices:
    dataset.dropna(axis=1, how='any', thresh=None, subset=None, inplace=True)


# for trading datasets:
for dataset in list_of_trading_datasets_prices:
    dataset.dropna(axis=1, how='all', thresh=None, subset=None, inplace=True)

for dataset in list_of_trading_datasets_prices:
    dataset.fillna(method = 'bfill', inplace=True, limit=10)

for dataset in list_of_trading_datasets_prices:
    dataset.dropna(axis=1, how='any', thresh=None, subset=None, inplace=True)


# Make sure we have the same securities in the formation and trading period.
# Remove securities that are not present in both periods

common_tickers = []
for i in range(len(list_of_formation_datasets_prices)):
    common_cols = list_of_formation_datasets_prices[i].drop([col for col in\
                list_of_formation_datasets_prices[i].columns if col in\
                list_of_formation_datasets_prices[i].columns and col not in\
                list_of_trading_datasets_prices[i].columns], axis = 1)

    common_tickers.append(common_cols)

for i in range(len(list_of_formation_datasets_prices)):
    list_of_formation_datasets_prices[i] = list_of_formation_datasets_prices[i]\
    [common_tickers[i].columns]
    list_of_trading_datasets_prices[i] = list_of_trading_datasets_prices[i]\
    [common_tickers[i].columns]

# Calculating returns from closing prices
list_of_training_returns = []
for dataset in list_of_formation_datasets_prices:
    stock_returns = dataset.pct_change()
    list_of_training_returns.append(stock_returns)
```

```
list_of_trading_returns = []
for dataset in list_of_trading_datasets_prices:
    stock_returns = dataset.pct_change()
    list_of_trading_returns.append(stock_returns)

for dataset in list_of_trading_returns:
    dataset.iloc[0:1] = 0

#%%

#-----------------------------------------------------------------------------------------------
# PCA
#-----------------------------------------------------------------------------------------------
extracted_pca_data = []
for dataset in list_of_training_returns:
    pca = PCA(n_components = 20) # nr. of components is set to 12
    pca.fit(dataset)
    pca.explained_variance_ratio_.cumsum() # determine nr . of components
    print('The shape of the array after PCA is : ', pca.components_.T.shape)
    extracted_data = preprocessing.StandardScaler().\
    fit_transform(pca.components_.T)
    print ('The shape of the array is now:', extracted_data.shape)
    extracted_pca_data.append(extracted_data)

    PC_values = np.arange(pca.n_components_) + 1
    plt.plot(PC_values, pca.explained_variance_ratio_, 'ro-', linewidth=2)
    plt.title('Scree Plot')
    plt.xlabel('Principal Component')
    plt.ylabel('Proportion of Variance Explained')
    plt.title('Scree Plot for US data')
    plt.show()
    plt.close()

    plt.plot(np.cumsum(pca.explained_variance_ratio_), color = 'blue')
    plt.xlabel('number of components')
    plt.ylabel('cumulative explained variance');
    plt.title('Cumulative Scree Plot for US data')
    plt.show()

#-----------------------------------------------------------------------------------------------
# DBSCAN
#-----------------------------------------------------------------------------------------------
extracted_DBSCAN_data = []
extracted_labels = []
extracted_clustered_series = []
extracted_clustered_series_all = []
extracted_labels = []
extracted_ticker_count_reduced = []
extracted_n_clusters = []

for i in range(len(extracted_pca_data)):
    clustering = DBSCAN(eps=1, min_samples=4)
    # eps = 1 for SP500, eps = 0.6 for OSE
    print(clustering)
    clustering.fit(extracted_pca_data[i])
    labels =clustering.labels_
    extracted_labels.append(labels)
    n_clusters_ = len(set(labels)) - (1 if -1 in labels else 0)
    print ('\nClusters discovered : %d' % n_clusters_ )
    extracted_n_clusters.append(n_clusters_)
    clustered = clustering.labels_
    extracted_DBSCAN_data.append(clustered)
```

```
                   data=extracted_DBSCAN_data[i])
    extracted_clustered_series_all.append(clustered_series)
    clustered_series = clustered_series[clustered_series != -1]
    extracted_clustered_series.append(clustered_series)

    CLUSTER_SIZE_LIMIT = 200
    counts = clustered_series.value_counts()
    ticker_count_reduced = counts[(counts>1) & counts<=CLUSTER_SIZE_LIMIT]
    extracted_ticker_count_reduced.append(ticker_count_reduced)
    print('Clusters formed: %d' % len(ticker_count_reduced))
    print('Pairs to evaluate: %d' % (ticker_count_reduced*\
                                (ticker_count_reduced-1)).sum())


#%%

#Plot multidimension dataset of returns into 2D
#This creates a t-SNE plot of all stocks with clusters noted
for i in range(36):
    extracted_data_tsne = TSNE(learning_rate=500, perplexity=18, \
                   random_state=1337).fit_transform(extracted_pca_data[i])

    #PLOT
    plt.figure(1, facecolor='white', figsize=(10,6))
    plt.clf()
    #plt.axis('off')

    #unclustered in the background
    plt.scatter(
        extracted_data_tsne[(extracted_clustered_series_all[i]==-1).values, 0],
        extracted_data_tsne[(extracted_clustered_series_all[i]==-1).values, 1],
        s=120,
        alpha=0.2,
        c='grey'
        )

    #clustered
    plt.scatter(
        extracted_data_tsne[(extracted_labels[i]!=-1), 0],
        extracted_data_tsne[(extracted_labels[i]!=-1), 1],
        s=120,
        alpha=0.85,
        c=extracted_labels[i][extracted_labels[i]!=-1],
        cmap=cm.cool,
        edgecolors = 'grey'
        )

    plt.title('T-SNE of DBSCAN clusters for US data', fontsize = 20)
    plt.xlabel('Dimension 1', fontsize = 12)
    plt.ylabel('Dimension 2', fontsize = 12)
    plt.show()


#%%
#-----------------------------------------------------------------------------------------------
# This part prepares the data for performing a cointegration test on all pairs
# in each cluster
#-----------------------------------------------------------------------------------------------
# Get the number of stocks in each cluster
extracted_counts = []
for i in range(len(extracted_clustered_series)):
    counts = extracted_clustered_series[i].value_counts()
    extracted_counts.append(counts)
    clusters_vis_list = list(extracted_counts[i][(extracted_counts[i]<500) & \
                                        (extracted_counts[i]>1)].index[::-1])
    extracted_clusters_vis_list.append(clusters_vis_list)

extracted_training_new = []
for dataset in list_of_training_returns:
    training_new_draft = ((dataset + 1).cumprod()-1)
    training_new = training_new_draft[0:252]
    extracted_training_new.append(training_new)

# Create a list to use as x-axis in plot:
x=list(range(1,253))

# Plot the stock time series for all clusters
extracted_tickers_list = []
for i in range(len(extracted_clusters_vis_list)):
    temp = extracted_clustered_series[i]
    tickers2 = temp[temp==0]
    tickers1 = temp[temp==1]
    tickers3 = temp[temp==2]
    tickers4 = temp[temp==3]
    tickers5 = temp[temp==4]
    tickers6 = temp[temp==5]
    tickers7 = temp[temp==6]
    tickers8 = temp[temp==7]
    tickers9 = temp[temp==8]
    tickers10 = temp[temp==9]
    tickers11 = temp[temp==10]
    tickers12 = temp[temp==11]
    tickers13 = temp[temp==12]
    tickers14 = temp[temp==13]
    tickers15 = temp[temp==14]

   # tickers4 = temp[temp ==[]]
    if list(tickers1)!=[]:
        extracted_tickers_list.append(tickers1)
    if list(tickers2)!=[]:
        extracted_tickers_list.append(tickers2)
    if list(tickers3)!=[]:
        extracted_tickers_list.append(tickers3)
    if list(tickers4)!=[]:
        extracted_tickers_list.append(tickers4)
    if list(tickers5)!=[]:
        extracted_tickers_list.append(tickers5)
    if list(tickers6)!=[]:
        extracted_tickers_list.append(tickers6)
    if list(tickers7)!=[]:
        extracted_tickers_list.append(tickers7)
    if list(tickers8)!=[]:
        extracted_tickers_list.append(tickers8)
    if list(tickers9)!=[]:
        extracted_tickers_list.append(tickers9)
    if list(tickers10)!=[]:
        extracted_tickers_list.append(tickers10)
    if list(tickers11)!=[]:
        extracted_tickers_list.append(tickers11)
    if list(tickers12)!=[]:
        extracted_tickers_list.append(tickers12)
    if list(tickers13)!=[]:
        extracted_tickers_list.append(tickers13)
    if list(tickers14)!=[]:
        extracted_tickers_list.append(tickers14)
    if list(tickers1) == [] and list(tickers2) == [] and list(tickers3) == []\
        and list(tickers4) == [] and list(tickers5) == [] and \
        list(tickers6) == [] and list(tickers7) == [] and list(tickers8) == []\
        and list(tickers9) == [] and list(tickers10) == [] and \
        list(tickers11) == [] and list(tickers12) == [] and \
        list(tickers13) == [] and list(tickers14) == [] and \
        list(tickers15) == []:
        extracted_tickers_list.append([])

#%%
#------------------------------------------------------------------------------------------------
# Setting up cointegration test
#------------------------------------------------------------------------------------------------
# COINTEGRATION TEST (From Larkin (2017))
def cointegrated_stocks(data, significance=0.05):
    n = data.shape[1] # gives us the number of stocks in cluster
    score_matrix = np.zeros((n, n)) # creates an n*n array of zeros
    pvalue_matrix = np.ones((n, n))
    # ^ this array will be updated with cointegration p-values
    keys = data.keys() # store the ticker symbol of stocks
    pairs = [] # create an empty list

    for i in range(n):
        for j in range(i+1, n):
            S1 = data[keys[i]]
            S2 = data[keys[j]]

            result = coint(S1, S2) # no intercept needed
            score = result[0] # store result index[0]
            pvalue = result[1]
            score_matrix[i, j] = score
            pvalue_matrix[i, j] = pvalue

            if pvalue < significance:
                pairs.append((keys[i], keys[j]))
    return score_matrix, pvalue_matrix, pairs

# Create a new index to allow for several clusters in each formaiton period
new_index = []
for i in range(len(extracted_counts)):
    x = i
    if len(extracted_counts[i]) == 2:
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 3:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 4:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 5:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 6:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 7:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 8:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 9:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 10:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 11:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 12:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 14:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    elif len(extracted_counts[i]) == 15:
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
        new_index.append(x)
    else:
        new_index.append(x)

#-------------------------------------------------------------------------------------------------
# Loop through formation periods to find cointegrated pairs in each cluster
#-------------------------------------------------------------------------------------------------
cluster_dictionary = {}
score_matrix_list = []
pvalue_matrix_list = []
pairs_list = []

count = 0

for i in range(0, len(new_index)):

    if len(extracted_tickers_list[i]) == 0:
        count += 1
        print(i)
        continue
    stock_ticks = extracted_tickers_list[count]
    print(stock_ticks)# An index list of all stocks in cluster
    print(list_of_formation_datasets_prices[period].columns)
    score_matrix, pvalue_matrix, pairs = cointegrated_stocks\
        (list_of_formation_datasets_prices[period][stock_ticks.index]
    )
    score_matrix_list.append(score_matrix)
    pvalue_matrix_list.append(pvalue_matrix)
    pairs_list.append(pairs)

    cluster_dictionary[i] = {}
    cluster_dictionary[i]['period'] = period
    cluster_dictionary[i]['score_matrix'] = score_matrix
    cluster_dictionary[i]['pvalue_matrix'] = pvalue_matrix
    cluster_dictionary[i]['pairs'] = pairs

    count += 1

potential_pairs = []
for clust in cluster_dictionary.keys():
    potential_pairs = cluster_dictionary[clust]['pairs']

    print('The following pairs will be traded in this period:')
    print(set(cluster_dictionary[clust]['pairs']))

    print('We found %d pairs.' % len(potential_pairs))
    print('In those pairs, there are %d unique tickers.' % \
          len(np.unique(potential_pairs)))

    potential_pairs.extend(cluster_dictionary[clust]['pairs'])


#---------------------------------------------------------------------------------------------
# Plot a cluster with cointegrated pairs noted
#---------------------------------------------------------------------------------------------
Cluster = extracted_clustered_series[1][extracted_clustered_series[1]==0]
our_pairs = cluster_dictionary[3]['pairs']

stocks = list(np.unique(our_pairs))
X_df = pd.DataFrame(index=list_of_training_returns[1].T.index, \
                    data=extracted_pca_data[1])

stocks = list(np.unique(our_pairs))
X_pairs = X_df.loc[Cluster.index]

X_tsne = TSNE(learning_rate=50, perplexity=3, random_state=1337).\
    fit_transform(X_pairs)

plt.figure(1, facecolor='white')
plt.clf()
plt.axis('off')
for pair in our_pairs:
    ticker1 = pair[0]
    loc1 = X_pairs.index.get_loc(pair[0])
    x1, y1 = X_tsne[loc1, :]

    ticker2 = pair[0]
    loc2 = X_pairs.index.get_loc(pair[1])
    x2, y2 = X_tsne[loc2, :]

    plt.plot([x1, x2], [y1, y2], 'k-', alpha=0.2, c='gray');

plt.scatter(X_tsne[:, 0], X_tsne[:, 1], s=220, alpha=1, c=[Cluster.values], \
#%%
#-----------------------------------------------------------------------------------------------
# Trading setup and execution in FORMATION periods
#-----------------------------------------------------------------------------------------------
# creating datasets for all potential pairs in the formation period containing
#returns, spread, trading signals, positions and spread returns

pairs_datasets = []
for clust in cluster_dictionary:
    for i in range(len(set(cluster_dictionary[clust]['pairs']))):
        pair_tickers = list(cluster_dictionary[clust]['pairs'][i])
        period = cluster_dictionary[clust]['period']
        trading_pair = list_of_formation_datasets_prices[period][pair_tickers]
        #trading_pair.columns = ['S1', 'S2']

        trading_pair['S1_ret'] = trading_pair[pair_tickers[0]].pct_change(1)
        trading_pair.iloc[0,2] = 0
        trading_pair['S2_ret'] = trading_pair[pair_tickers[1]].pct_change(1)
        trading_pair.iloc[0,3] = 0

        # CALCULATE ROLLING Z-SCORE
        rolling_window = 20

        # OLS Regression (can be used to decide long/short position size)
        lm_pair = rg.OLS(trading_pair[pair_tickers[0]],\
                         trading_pair[pair_tickers[1]]).fit()
        trading_pair_b1 = lm_pair.params[0]

        # Create new column called pairs spread
        trading_pair['pairs_spread'] = \
            (trading_pair[pair_tickers[0]] - trading_pair[pair_tickers[1]])\
                / trading_pair[pair_tickers[1]]

        # Rolling 10-day covariance
        rolling_pair_cov = trading_pair.loc[:, [pair_tickers[0],\
                          pair_tickers[1]]].rolling(window=rolling_window)\
              .cov(trading_pair.loc[:, [pair_tickers[0], pair_tickers[1]]],\
                                                        pairwise=True)

        # Slice multi index df to single index df if pairs covariance
        idx = pd.IndexSlice
        rolling_pair_cov = rolling_pair_cov.loc[idx[:, pair_tickers[0]], \
                                                    pair_tickers[1]]

        # Convert Date and Stock index into date index by making stock at
        # index level 1 intp a new column
        rolling_pair_cov = rolling_pair_cov.reset_index(level=1)

        # Calculate the 10-day rolling variance
        rolling_pair_var = trading_pair[pair_tickers[0]].\
                                      rolling(window=rolling_window).var()

        # Rolling Beta
        trading_pair['rolling_pair_b1'] = rolling_pair_cov[pair_tickers[1]] \
                                                      / rolling_pair_var

        # Calculation of 10-day rolling spread
        trading_pair['rolling_pair_spread'] = trading_pair['pairs_spread'].\
                                  rolling(window=rolling_window).mean()

        trading_pair['spread_std'] = trading_pair['pairs_spread'].\
                                  rolling(window = rolling_window).std()
        # TRADING SIGNAL ALGORITHM
        # z-score the day before
        trading_pair['rolling_Z_score(-1)'] = \
                                trading_pair['rolling_Z_score'].shift(1)
         # z-score two days before

        trading_pair['pair_signal'] = 0
        pair_signal = 0

        # Signal generation
        for i, r in enumerate(trading_pair.iterrows()):
            if r[1]['rolling_Z_score(-1)'] > -2 and \
                                        r[1]['rolling_Z_score'] < -2:
                pair_signal = -2
            elif r[1]['rolling_Z_score(-1)'] < -0 and \
                                        r[1]['rolling_Z_score'] > -0:
                pair_signal = -1
            elif r[1]['rolling_Z_score(-1)'] < 2 and \
                                         r[1]['rolling_Z_score'] > 2:
                pair_signal = 2
            elif r[1]['rolling_Z_score(-1)'] > 0 and \
                                         r[1]['rolling_Z_score'] < 0:
                pair_signal = 1
            else:
                pair_signal = 0
            trading_pair.iloc[i, 10] = pair_signal

        # Positions: 1 = Long Spread Trade, -1 = Short Spread Trade
        trading_pair['position'] = 0
        for i, r in enumerate(trading_pair.iterrows()):
            if r[1]['pair_signal'] == -2:
                position = 1
            elif r[1]['pair_signal'] == -1:
                position = 0
            elif r[1]['pair_signal'] == 2:
                position = -1
            elif r[1]['pair_signal'] == 1:
                position = 0
            else:
                position = trading_pair['position'].iloc[i-1]
            trading_pair.iloc[i,11] = position

        # Computing returns without beta
        trading_pair['spread_returns'] = trading_pair['S1_ret'] - \
                                            trading_pair['S2_ret']
        trading_pair['return'] = trading_pair['spread_returns'] * \
                                    trading_pair['position'].shift(1)
                                    #set this to .shift(2) to impose
                                    #a 1 day lag

        # checking which period the pair is from and adding it to the datasets
        trading_pair['period'] = cluster_dictionary[clust]['period']

        # append trading_pair to the list containing all datasets
        pairs_datasets.append(trading_pair)

#%%
#---------------------------------------------------------------------------------------------------
# Calculate the sharpe ratios for all pairs in the training periods
#---------------------------------------------------------------------------------------------------

sharpe_ratios = []
```

```
    pairs_datasets[i]['SR'] = pairs_datasets[i]['avg_ret'] / \
                                          pairs_datasets[i]['std_dev_ret']

# Put all sharpe ratios in a list together with their period number
list_of_all_sharpe_ratios = []
list_of_all_periods = []
for i in range(len(pairs_datasets)):
    sharpe_ratio = pairs_datasets[i]['SR'].mean()
    period = pairs_datasets[i]['period'].mean()
    list_of_all_sharpe_ratios.append(sharpe_ratio)
    list_of_all_periods.append(period)

SR_p_merged = pd.DataFrame()
SR_p_merged['SR'] = list_of_all_sharpe_ratios
SR_p_merged['period'] = list_of_all_periods

# Group by highest sharpe and period
# We pick the 10 pairs with highest sharpe from each training period
groups = SR_p_merged.sort_values(['period', 'SR']).groupby('period').tail(5)
# adjust the .tail() to the number of stocks you want to trade on
groups.reset_index(inplace=True)

# Match the highest sharpe ratios with the tickers that belongs to these
tickers_with_highest_sharpe = []

for i in range(len(groups)):
    for j in range(len(pairs_datasets)):
        if groups['SR'][i] == pairs_datasets[j]['SR'].mean() and \
                    groups['period'][i] == pairs_datasets[j]['period'].mean():
            tickers_with_highest_sharpe.append(pairs_datasets[j].iloc[:,0:21])

optimal_trading_pairs = []
for i in range(len(tickers_with_highest_sharpe)):
    print(i)
    new_trading_pair = list_of_trading_datasets_prices\
    [int(tickers_with_highest_sharpe[i]['period'].mean())]\
        [tickers_with_highest_sharpe[i].iloc[:,:2].columns]
    optimal_trading_pairs.append(new_trading_pair)

#%%
#-----------------------------------------------------------------------------------------------
# Calculate daily return when trading at every cointegrated pair in each
# cluster every training period
#-----------------------------------------------------------------------------------------------

dataframe = pd.DataFrame()
dataframe_index = pd.DataFrame()
for i in range(len(pairs_datasets)):
    temp = pairs_datasets[i]
    temp2 = temp['return']
    temp = temp.shift(-1)[temp['position']!=0]['return']
    dataframe = pd.concat([dataframe, temp])
    dataframe_index = pd.concat([dataframe_index, temp2])
    #trading_dataframe.drop_duplicates(inplace=True)
dataframe.columns = ['return']

dataframe.reset_index(inplace=True)
grouped_dataframe = dataframe.groupby('index').agg('mean')

dataframe_index.reset_index(inplace=True)
grouped_dataframe_index = dataframe_index.groupby('index').agg('mean')
```

```
training_results['cumulative return'] = np.cumsum(training_results['return'])
training_results.reset_index(inplace=True)

plt.figure(figsize=(10,7))
plt.plot(training_results['cumulative return'], linewidth=1, color='blue')
plt.grid(color = 'black', linestyle = '--', linewidth = 0.5)
plt.title('Cumulative return for all pairs in the training periods ')
plt.show()

training_std_of_returns = training_results['return'].std()
training_average_return = training_results['return'].mean()
print('The annualized return in the training period is:', \
                                        (training_average_return * 252))
print('The annualized SR in the training period is:', \
          ((training_average_return/training_std_of_returns) * np.sqrt(252)))

#-----------------------------------------------------------------------------------------------
# Calculate the daily return on the x number of pairs with the highest sharpe
# ratio. The x is decided by the .tail()
#-----------------------------------------------------------------------------------------------

highest_SR_dataframe = pd.DataFrame()
highest_SR_dataframe_index = pd.DataFrame()
for i in range(len(tickers_with_highest_sharpe)):
    temp = tickers_with_highest_sharpe[i]
    temp2 = temp['return']
    temp = temp.shift(-1)[temp['position']!=0]['return']
    highest_SR_dataframe = pd.concat([highest_SR_dataframe, temp])
    highest_SR_dataframe_index = pd.concat([highest_SR_dataframe_index,temp2])
    #trading_dataframe.drop_duplicates(inplace=True)
highest_SR_dataframe.columns = ['return']

highest_SR_dataframe.reset_index(inplace=True)
grouped_highest_SR_dataframe=highest_SR_dataframe.groupby('index').agg('mean')

highest_SR_dataframe_index.reset_index(inplace=True)
grouped_highest_SR_dataframe_index = \
                    highest_SR_dataframe_index.groupby('index').agg('mean')

training_results_high_SR = pd.DataFrame(index = \
                        grouped_highest_SR_dataframe_index.index, data = \
                                grouped_highest_SR_dataframe['return'])
training_results_high_SR = training_results_high_SR.fillna(0)

training_results_high_SR['cumulative return'] = \
                             np.cumsum(training_results_high_SR['return'])
training_results_high_SR.reset_index(inplace=True)

plt.figure(figsize=(10,7))
plt.plot(training_results_high_SR['cumulative return'], linewidth=1, \
                                                   color='blue')
plt.grid(color = 'black', linestyle = '--', linewidth = 0.5)
plt.title('Cumulative return for the pairs with highest sharpe ratio in the \
                                              formation periods ')
plt.show()

training_std_of_returns_high_SR = training_results_high_SR['return'].std()
training_average_return_high_SR = training_results_high_SR['return'].mean()
print('The annualized return in the training period is:', \
                                  (training_average_return_high_SR * 252))
print('The annualized SR in the training period is:', \
#%%
#------------------------------------------------------------------------------------------------
# Trading setup and execution in FORMATION periods
#------------------------------------------------------------------------------------------------

semi_annual_spread = \
            pd.read_csv('semi_annual_liquidity.csv', index_col=0, sep=',')

commission = 0.0005
short_fee = 0.000179

optimal_pairs_datasets = []
# create new datasets for pairs that will be traded
# these are the pairs chosen in the .tail() above
for i in range(len(optimal_trading_pairs)):
      trading_pair = optimal_trading_pairs[i]
      #trading_pair.columns = ['S1', 'S2']
      trading_pair['S1_ret'] = trading_pair.iloc[:,0].pct_change(1)
      trading_pair.iloc[0,2] = 0
      trading_pair['S2_ret'] = trading_pair.iloc[:,1].pct_change(1)
      trading_pair.iloc[0,3] = 0
      pair_tickers = [trading_pair.iloc[:,0].name, \
                              trading_pair.iloc[:,1].name]

      # CALCULATE ROLLING Z-SCORE
      rolling_window = 20

      # OLS Regression
      lm_pair = rg.OLS(trading_pair[pair_tickers[0]], \
                          trading_pair[pair_tickers[1]]).fit()
      trading_pair_b1 = lm_pair.params[0]

      # Create new column called pairs spread
      trading_pair['pairs_spread'] = \
      (trading_pair[pair_tickers[0]] - trading_pair[pair_tickers[1]]) / \
                              trading_pair[pair_tickers[1]]

      # Rolling 10-day covariance
      rolling_pair_cov = trading_pair.loc[:, [pair_tickers[0], \
                  pair_tickers[1]]].rolling(window=rolling_window)\
      .cov(trading_pair.loc[:, [pair_tickers[0], pair_tickers[1]]], \
                                    pairwise=True)

      # Slice multi index df to single index df if pairs covariance
      idx = pd.IndexSlice
      rolling_pair_cov = rolling_pair_cov.loc[idx[:, pair_tickers[0]],\
                                    pair_tickers[1]]

      # Convert Date and Stock index into date index by making stock at index
      #level 1 intp a new column
      rolling_pair_cov = rolling_pair_cov.reset_index(level=1)

      # Calculate the 10-day rolling variance
      rolling_pair_var = trading_pair[pair_tickers[0]].\
                          rolling(window=rolling_window).var()

      # Rolling Beta
      trading_pair['rolling_pair_b1'] = rolling_pair_cov[pair_tickers[1]] \
                                    / rolling_pair_var

      # Calculation of 10-day rolling spread
      trading_pair['rolling_pair_spread'] = trading_pair['pairs_spread'].\
                          rolling(window=rolling_window).mean()
```

```
# 10-day rolling z-score
trading_pair['rolling_Z_score'] = (trading_pair['pairs_spread'] - \
    trading_pair['rolling_pair_spread']) / trading_pair['spread_std']


# TRADING SIGNAL ALGORITHM
# z-score the day before
trading_pair['rolling_Z_score(-1)'] = trading_pair['rolling_Z_score']\
                                                    .shift(1)
 # z-score two days before
trading_pair['pair_signal'] = 0
pair_signal = 0

# Signal generation
for i, r in enumerate(trading_pair.iterrows()):
   if r[1]['rolling_Z_score(-1)'] > -2 and \
                                 r[1]['rolling_Z_score'] < -2:
       pair_signal = -2
   elif r[1]['rolling_Z_score(-1)'] < -0 and \
                                 r[1]['rolling_Z_score'] > -0:
       pair_signal = -1
   elif r[1]['rolling_Z_score(-1)'] < 2 and \
                                 r[1]['rolling_Z_score'] > 2:
       pair_signal = 2
   elif r[1]['rolling_Z_score(-1)'] > 0 and \
                                 r[1]['rolling_Z_score'] < 0:
       pair_signal = 1
   else:
       pair_signal = 0
   trading_pair.iloc[i, 10] = pair_signal

# Positions: 1 = Long Spread Trade, -1 = Short Spread Trade
trading_pair['position'] = 0
for i, r in enumerate(trading_pair.iterrows()):
   if r[1]['pair_signal'] == -2:
       position = 1
   elif r[1]['pair_signal'] == -1:
       position = 0
   elif r[1]['pair_signal'] == 2:
       position = -1
   elif r[1]['pair_signal'] == 1:
       position = 0
   else:
       position = trading_pair['position'].iloc[i-1]
   trading_pair.iloc[i,11] = position

# Computing returns without beta
trading_pair['spread_returns'] = trading_pair['S1_ret'] - \
                                       trading_pair['S2_ret']
trading_pair['return'] = trading_pair['spread_returns'] * \
                            trading_pair['position'].shift(1)
                            #set this to .shift(2) to impose
                            #a 1 day lag

# checking period
count = 0
for j in list_of_trading_datasets_prices:
   if sum(trading_pair.index == j.index) == len(j.index):
       trading_pair['period'] = count
       j['period'] = count
       break
```

```
        count = 1
        prev = 0
        TC_Op = 0
        trading_pair['TC_Op'] = 0
        for i, row in enumerate(trading_pair.iterrows()):
            if (row[1]['pair_signal'] == -2 or row[1]['pair_signal'] == 2) \
                                                    and prev == 0:
                TC_Op =  4*commission + 9 * short_fee
            else:
                TC_Op = 0
            trading_pair.iloc[i,15] = TC_Op
            prev = row[1]['position']
            count += 1

        TC_Cl = 0
        trading_pair['TC_Cl'] = 0
        for i, row in enumerate(trading_pair.iterrows()):
            if ((row[1]['pair_signal'] == 1 or row[1]['pair_signal'] == -1) \
                                          and (row[1]['return'] != 0)):
                TC_Cl =  0
            else:
                TC_Cl = 0
            trading_pair.iloc[i,16] = TC_Cl

        optimal_pairs_datasets.append(trading_pair)

for i in range(len(optimal_pairs_datasets)):
    optimal_pairs_datasets[i]['new_TC'] = \
                                optimal_pairs_datasets[i]['TC_Op'].shift(1)
                                        #set this to .shift(2) to impose
                                        #a 1 day lag
    optimal_pairs_datasets[i]['return_inc_TC'] = optimal_pairs_datasets[i]\
                            ['return'] - optimal_pairs_datasets[i]['new_TC']

#%%
#-----------------------------------------------------------------------------------------------
# Visualizing return of some of the optimal pairs
#-----------------------------------------------------------------------------------------------

for i in range(len(optimal_pairs_datasets)):
    # Create x-axis to use in plot
    x_axis=list(range(len(optimal_pairs_datasets[i])))
    optimal_pairs_datasets[i][np.isnan(optimal_pairs_datasets[i])] = 0

for i in range(10):
    plt.figure(figsize=(10,7))
    optimal_pairs_datasets[i]['Cumulative return'] = \
                    np.cumprod(optimal_pairs_datasets[i]['return']+1) - 1
    optimal_pairs_datasets[i]['Cumulative return with TC'] = \
               np.cumprod(optimal_pairs_datasets[i]['return_inc_TC']+1) - 1
    optimal_pairs_datasets[i]['Security 1 return'] = \
                    np.cumprod(optimal_pairs_datasets[i]['S1_ret']+1) - 1
    optimal_pairs_datasets[i]['Security 2 return'] = \
                    np.cumprod(optimal_pairs_datasets[i]['S2_ret']+1) - 1

    plt.plot(x_axis, optimal_pairs_datasets[i]['Cumulative return'], \
                                      c='blue', label = 'Strategy return')
    plt.plot(x_axis, optimal_pairs_datasets[i]['Cumulative return with TC'],\
                                    c='orange', label = 'Strategy return')
    plt.plot(x_axis, optimal_pairs_datasets[i]['Security 1 return'], \
                                          c='grey', label = 'Security 1')
    plt.plot(x_axis, optimal_pairs_datasets[i]['Security 2 return'], \
                                         c='black', label = 'Security 2')
```

```
#-----------------------------------------------------------------------------------------------
# Visualizing trading signal and positions of a few pairs
#-----------------------------------------------------------------------------------------------

for i in range(10):
    x_axis=list(range(len(optimal_pairs_datasets[i])))
    plt.figure(figsize=(10,7))
    plt.axhline(y =2, color='green', linestyle='--', linewidth=.7, \
                                                    label='Upper threshold')
    plt.axhline(y =-2, color='red', linestyle='--', linewidth=.7, \
                                                    label='Lower threshold')
    plt.plot(x_axis, optimal_pairs_datasets[i]['rolling_Z_score'], \
                                     color='blue', alpha=.5, label='Z-score')
    plt.legend(loc='upper left')
    plt.show()

    plt.figure(figsize=(10,2))
    plt.plot(x_axis, optimal_pairs_datasets[i]['position'], color='black', \
                                                          label='Position')
    plt.show()

#%%
#-----------------------------------------------------------------------------------------------
# Compute sharpe ratio for every pair traded
#-----------------------------------------------------------------------------------------------
for i in range(len(optimal_pairs_datasets)):
    optimal_pairs_datasets[i]['avg_ret'] = \
                                  optimal_pairs_datasets[i]['return'].mean()
    optimal_pairs_datasets[i]['std_dev_ret'] = \
                                   optimal_pairs_datasets[i]['return'].std()
    optimal_pairs_datasets[i]['SR'] = optimal_pairs_datasets[i]['avg_ret'] \
                                  / optimal_pairs_datasets[i]['std_dev_ret']

for i in range(len(optimal_pairs_datasets)):
    optimal_pairs_datasets[i][np.isnan(optimal_pairs_datasets[i])] = 0

#%%
#-----------------------------------------------------------------------------------------------
# Compute the daily excess return in the trading periods
#-----------------------------------------------------------------------------------------------

trading_dataframe = pd.DataFrame()
trading_dataframe_index = pd.DataFrame()
for i in range(len(optimal_pairs_datasets)):
    temp = optimal_pairs_datasets[i]
    temp2 = temp['return']
    temp = temp.shift(-1)[temp['position']!=0]['return']
    trading_dataframe = pd.concat([trading_dataframe, temp])
    trading_dataframe_index = pd.concat([trading_dataframe_index, temp2])
    #trading_dataframe.drop_duplicates(inplace=True)
trading_dataframe.columns = ['return']

temp_index = pd.read_csv('temp_index.csv', index_col=0, sep=',')

trading_dataframe.reset_index(inplace=True)
grouped_trading_dataframe = trading_dataframe.groupby('index').agg('mean')
trading_dataframe_index.reset_index(inplace=True)
grouped_trading_dataframe_index = trading_dataframe_index.groupby('index').\
                                                          agg('mean')

trading_results = pd.DataFrame(index = grouped_trading_dataframe_index.index,\
                                data = grouped_trading_dataframe['return'])
```

```
trading_results['cumulative return'] = np.cumsum(trading_results['return'])
trading_results.reset_index(inplace=True)

# Calculate the strategy drawdown over the trading period
trading_results['HWM'] = trading_results['cumulative return'].cummax()
trading_results['Drawdown'] = ((1+trading_results['HWM'])-\
        (1+trading_results['cumulative return']))/(1+trading_results['HWM'])

#%%
# Fill in days or were no trades are made with zero return
test_trading_dataframe = pd.DataFrame()
for i in range(len(list_of_trading_datasets_prices)):
    temp = list_of_trading_datasets_prices[i]
    test_trading_dataframe = pd.concat([test_trading_dataframe, temp])

trading_results.set_index('index', inplace=True)

test_trading_results = pd.DataFrame(index = temp_index.index, \
                                    data = grouped_trading_dataframe['return'])
test_trading_results = test_trading_results.fillna(0)

test_trading_results['cumulative return'] = \
                                      np.cumsum(test_trading_results['return'])
test_trading_results.reset_index(inplace=True)

# Calculate the strategy drawdown over the trading period
test_trading_results['HWM'] = \
                            test_trading_results['cumulative return'].cummax()
test_trading_results['Drawdown'] = ((1+test_trading_results['HWM'])-\
(1+test_trading_results['cumulative return']))/(1+test_trading_results['HWM'])

#%%
#---------------------------------------------------------------------------------------------------
# Plot cumulatice return and some performance measures
#---------------------------------------------------------------------------------------------------
# Plot cumulative return of strategy and benchmark
plt.figure(figsize=(10,7))
plt.plot(x_axis, SP500_index['cumulative return'], linewidth=1, color='red', label='S&P500 Index')
plt.plot(test_trading_results['cumulative return'], linewidth=1, color='blue', label='Strategy')
plt.grid(color = 'black', linestyle = '--', linewidth = 0.5)
plt.legend(loc='upper left')
plt.title('Cumulative strategy return 2000 - 2019 vs. benchmark')
plt.show()

# Plot drawdown
plt.figure(figsize=(10,7))
plt.plot(test_trading_results['Drawdown'], linewidth=1, color = 'red')
plt.title('Strategy drawdown 2000 - 2019')
plt.show()

# Plot daily return
plt.figure(figsize=(10,7))
plt.plot(test_trading_results['return'], linewidth=1, color = 'blue')
plt.title('Strategy daily return 2000 - 2019')
plt.show()

# Plot distribution of daily returns
plt.figure(figsize=(10,7))
plt.hist(grouped_trading_dataframe['return'], color = 'blue', bins = 150)
plt.grid(color = 'black', linestyle = '--', linewidth = 0.5)
plt.title('Distribution of daily returns')
plt.show()
```

```
trading_std_of_returns = test_trading_results['return'].std()
trading_average_return = test_trading_results['return'].mean()
print('The annualized return in the trading period is:', \
                                            (trading_average_return * 252))
print('The annualized SR in the trading period is:', \
            ((trading_average_return/trading_std_of_returns) * np.sqrt(252)))

# Additional summary statistics
test_trading_results.agg(
    {
     'return': ['mean', 'median', 'std', 'skew', 'kurtosis', 'min', 'max'],
     }
    )
# T-test to check significance of daily excess returns
stats.ttest_1samp(test_trading_results['return'], popmean=0)

#%%
#--------------------------------------------------------------------------------------------------
# Calculate the daily return and cumulative return after transaction costs
#--------------------------------------------------------------------------------------------------

grouped_trading_dataframe_index = pd.DataFrame()
grouped_trading_dataframe_index = pd.DataFrame()
trading_results_inc_TC = pd.DataFrame()
new_trading_dataframe = pd.DataFrame()
new_trading_dataframe_index = pd.DataFrame()
for i in range(len(optimal_pairs_datasets)):
    new_temp = optimal_pairs_datasets[i]
    new_temp2 = new_temp['return']
    new_temp = new_temp.shift(-1)[new_temp['position']!=0]['return_inc_TC']
    new_trading_dataframe = pd.concat([new_trading_dataframe, new_temp])
    new_trading_dataframe_index = pd.concat([new_trading_dataframe_index, \
                                                            new_temp2])
    #trading_dataframe.drop_duplicates(inplace=True)
new_trading_dataframe.columns = ['return_inc_TC']
new_trading_dataframe.reset_index(inplace=True)
new_grouped_trading_dataframe = new_trading_dataframe.groupby('index')\
                                                           .agg('mean')
new_trading_dataframe_index.reset_index(inplace=True)
new_grouped_trading_dataframe_index = \
                    new_trading_dataframe_index.groupby('index').agg('mean')

trading_results_inc_TC = pd.DataFrame(index = \
                              new_grouped_trading_dataframe_index.index,\
                     data = new_grouped_trading_dataframe['return_inc_TC'])
trading_results_inc_TC = trading_results_inc_TC.fillna(0)

# Calculate cumulative return of the strategy
trading_results_inc_TC['cumulative return'] = \
                           np.cumsum(trading_results_inc_TC['return_inc_TC'])
trading_results_inc_TC.reset_index(inplace=True)

# Calculate the strategy drawdown over the trading period
trading_results_inc_TC['HWM'] = \
                         trading_results_inc_TC['cumulative return'].cummax()
trading_results_inc_TC['Drawdown'] = ((1+trading_results_inc_TC['HWM'])-\
                           (1+trading_results_inc_TC['cumulative return']))/\
                                         (1+trading_results_inc_TC['HWM'])

#--------------------------------------------------------------------------------------------------
# Descriptive statistics
#--------------------------------------------------------------------------------------------------
trading_std_of_returns = trading_results_inc_TC['return_inc_TC'].std()
```

```
                                        (trading_average_return * 252))
print('The annualized SR in the trading period is:',\
           ((trading_average_return/trading_std_of_returns) * np.sqrt(252)))

# Additional Summary statistics
trading_results_inc_TC.agg(
    {
 'return_inc_TC': ['mean', 'median', 'std', 'skew', 'kurtosis', 'min', 'max'],
     }
    )

#T-test to check significance of daily excess returns
stats.ttest_1samp(trading_results_inc_TC['return_inc_TC'], popmean=0)

#%%
#-----------------------------------------------------------------------------------------------
# Analyzing systemtic risk of strategy by regressing returns on known pricing
# factors
#-----------------------------------------------------------------------------------------------


FF_factors_daily = pd.read_csv('FF_factors.csv', index_col=0, sep=',')
# Reset index of trading restuls dattaset
trading_results_inc_TC.set_index('index', inplace=True)
# Make sure that only the same dates are included in the pricing factor dataset
FF_factors_daily = pd.DataFrame(index = temp_index.index ,data = \
FF_factors_daily[FF_factors_daily.index.isin(trading_results_inc_TC.index)])
FF_factors_daily = FF_factors_daily.fillna(0)

factors = FF_factors_daily[['mktrf', 'smb', 'hml', 'umd']]
returns = trading_results_inc_TC['return_inc_TC']
factors = add_constant(factors)
model = sm.OLS(returns, factors)
results = model.fit()
results.summary()
```

```
#importing modules
import pandas as pd
import numpy as np
import pandas_datareader as web
import matplotlib.pyplot as plt
import datetime as datetime
import seaborn as sns
import matplotlib .cm as cm
from sklearn import linear_model
from sklearn . cluster import KMeans, DBSCAN
from sklearn . decomposition import PCA
from sklearn . manifold import TSNE
from sklearn import preprocessing
from statsmodels . tsa . stattools import coint
from statsmodels . tsa . stattools import adfuller
import statsmodels . regression . linear_model as rg
from scipy import stats
import statsmodels.api as sm
import statsmodels.tsa.stattools as ts



#Import data

ose_dataset_close_2000_2019 = pd.read_csv('CRSP_data_FIXED.csv', index_col=0, sep=',')

# Import benchmark
oslo_bors_benchmark_index = pd.read_csv('Oslo_bors_benchmark_index.csv', index_col=0, sep=',', encoding='latin-1')

# Datasets containing the daily relative spread for all stocks at OSE
ose_rel_spread_close_2000_2019 = pd.read_csv('CRSP_rel_spread.csv', index_col=0, sep=',')

# Calculate cumulative return on benchmark
oslo_bors_benchmark_index['return'] = oslo_bors_benchmark_index['Oslo BĂ¸rs Benchmark Index_GI'].pct_change()
oslo_bors_benchmark_index['cumulative return'] = np.cumprod(1+oslo_bors_benchmark_index['return'])-1
oslo_bors_benchmark_index.fillna(0)

#%%
#-------------------------------------------------------------------------------------------------------
#################### Creating TRAINING period datasets : ####################################################
#-------------------------------------------------------------------------------------------------------

list_of_training_datasets_prices = []

y = 0
while y < (5040-126):

    temp = ose_dataset_close_2000_2019.iloc[y:y+252]
    list_of_training_datasets_prices.append(temp)

    y += 126

list_of_training_datasets_prices.pop()
list_of_training_datasets_prices.pop()

#-------------------------------------------------------------------------------------------------------

##################### Creating TRADING period datasets : ####################################################

list_of_trading_datasets_prices = []

y = 0
```

```
    list_of_trading_datasets_prices.append(temp)

    y += 126

list_of_trading_datasets_prices.pop(0)
list_of_trading_datasets_prices.pop(0)



#-----------------------------------------------------------------------------------------------------------
##################### Creating spread portfolios TRAINING period  :
####################################################
#-----------------------------------------------------------------------------------------------------------
list_of_training_spread_datasets = []
y = 0
while y < (5040-126):

    temp = ose_rel_spread_close_2000_2019.iloc[y:y+252]
    list_of_training_spread_datasets.append(temp)

    y += 126

list_of_training_spread_datasets.pop()
list_of_training_spread_datasets.pop()

#-----------------------------------------------------------------------------------------------------------
##################### Creating spread portfolios TRAINING period  :
####################################################
#-----------------------------------------------------------------------------------------------------------
list_of_trading_spread_datasets = []
y = 0
while y < (5040-126):

    temp = ose_rel_spread_close_2000_2019.iloc[y:y+126]
    list_of_trading_spread_datasets.append(temp)

    y += 126


list_of_trading_spread_datasets.pop(0)
list_of_trading_spread_datasets.pop(0)


#Removing missing values:
# for training datasets
for dataset in list_of_training_datasets_prices:
    dataset.dropna(axis=1, how='all', thresh=None, subset=None, inplace=True)

for dataset in list_of_training_datasets_prices:
    dataset.fillna(method = 'bfill', inplace=True, limit=10)

for dataset in list_of_training_datasets_prices:
    dataset.dropna(axis=1, how='any', thresh=None, subset=None, inplace=True)


# for trading datasets:
for dataset in list_of_trading_datasets_prices:
    dataset.dropna(axis=1, how='all', thresh=None, subset=None, inplace=True)

for dataset in list_of_trading_datasets_prices:
    dataset.fillna(method = 'bfill', inplace=True, limit=10)

for dataset in list_of_trading_datasets_prices:
```

```
#### for spread portfolios ####
#Training:
for dataset in list_of_training_spread_datasets:
    dataset.dropna(axis=1, how='all', thresh=None, subset=None, inplace=True)

for dataset in list_of_training_spread_datasets:
    dataset.fillna(method = 'bfill', inplace=True, limit=10)

for dataset in list_of_training_spread_datasets:
    dataset.dropna(axis=1, how='any', thresh=None, subset=None, inplace=True)
# Trading
for dataset in list_of_trading_spread_datasets:
    dataset.dropna(axis=1, how='all', thresh=None, subset=None, inplace=True)

for dataset in list_of_trading_spread_datasets:
    dataset.fillna(method = 'bfill', inplace=True, limit=10)

for dataset in list_of_trading_spread_datasets:
    dataset.dropna(axis=1, how='any', thresh=None, subset=None, inplace=True)




# Make sure that we have the same securities in both the training and trading period. Remove securities that are not
# present in both periods

# --------For daily close data -------
common_tickers = []
common_tickers_2 = []
for i in range(len(list_of_training_datasets_prices)):
    common_cols = list_of_training_datasets_prices[i].drop([col for col in list_of_training_datasets_prices[i].columns if col in
list_of_training_datasets_prices[i].columns and col not in list_of_trading_datasets_prices[i].columns], axis = 1)

    common_tickers.append(common_cols)

for i in range(len(list_of_training_datasets_prices)):
    list_of_training_datasets_prices[i] = list_of_training_datasets_prices[i][common_tickers[i].columns]
    list_of_trading_datasets_prices[i] = list_of_trading_datasets_prices[i][common_tickers[i].columns]


# -------- For spread portfolios ---------
common_tickers = []
common_tickers_2 = []
for i in range(len(list_of_training_spread_datasets)):
    common_cols = list_of_training_spread_datasets[i].drop([col for col in list_of_training_spread_datasets[i].columns if col in
list_of_training_spread_datasets[i].columns and col not in list_of_trading_spread_datasets[i].columns], axis = 1)

    common_tickers.append(common_cols)

for i in range(len(list_of_training_spread_datasets)):
    list_of_training_spread_datasets[i] = list_of_training_spread_datasets[i][common_tickers[i].columns]
    list_of_trading_spread_datasets[i] = list_of_trading_spread_datasets[i][common_tickers[i].columns]

common_tickers = []
common_tickers_2 = []
for i in range(len(list_of_trading_spread_datasets)):
    common_cols = list_of_trading_spread_datasets[i].drop([col for col in list_of_trading_spread_datasets[i].columns if col in
list_of_trading_spread_datasets[i].columns and col not in list_of_trading_datasets_prices[i].columns], axis = 1)

    common_tickers.append(common_cols)
```

```
common_tickers = []
common_tickers_2 = []
for i in range(len(list_of_trading_spread_datasets)):
    common_cols = list_of_trading_spread_datasets[i].drop([col for col in list_of_trading_spread_datasets[i].columns if col in 
list_of_trading_spread_datasets[i].columns and col not in list_of_training_spread_datasets[i].columns], axis = 1)

    common_tickers.append(common_cols)

for i in range(len(list_of_trading_spread_datasets)):
    list_of_trading_spread_datasets[i] = list_of_trading_spread_datasets[i][common_tickers[i].columns]
    list_of_training_spread_datasets[i] = list_of_training_spread_datasets[i][common_tickers[i].columns]




# Calculating returns from closing prices

list_of_training_returns = []
for dataset in list_of_training_datasets_prices:
    stock_returns = dataset.pct_change()
    list_of_training_returns.append(stock_returns)


for dataset in list_of_training_returns:
    dataset.iloc[0:2] = 0


list_of_trading_returns = []
for dataset in list_of_trading_datasets_prices:
    stock_returns = dataset.pct_change()
    list_of_trading_returns.append(stock_returns)


for dataset in list_of_trading_returns:
    dataset.iloc[0:2] = 0

#%%

# Create portfolios based on the size of the relative bid-ask spread in the training period

list_of_top_spreads = []
list_of_bottom_spreads = []
for i in range(len(list_of_training_spread_datasets)):
    avg_rel_spread = pd.DataFrame(list_of_training_spread_datasets[i].mean())
    avg_rel_spread.columns= ['rel_spread']

    top_rel_spread = avg_rel_spread.nlargest(178, 'rel_spread', keep='first')
    list_of_top_spreads.append(top_rel_spread)

    bottom_rel_spread = avg_rel_spread. nsmallest(178, 'rel_spread', keep='first')
    list_of_bottom_spreads.append(bottom_rel_spread)


list_of_training_bottom_spreads = []
list_of_training_top_spreads = []
for i in range(len(list_of_training_spread_datasets)):
    tickers = list(list_of_bottom_spreads[i].index.values)
    training_pairs = list_of_training_spread_datasets[i][tickers]
    list_of_training_bottom_spreads.append(training_pairs)

    tickers2 = list(list_of_top_spreads[i].index.values)
    training_pairs2 = list_of_training_spread_datasets[i][tickers2]
```

```
top_spreads_prices = []
bottom_spreads_prices = []
for i in range(len(list_of_training_datasets_prices)):
    pairs = list_of_training_datasets_prices[i][list_of_training_bottom_spreads[i].columns]
    bottom_spreads_prices.append(pairs)

    pairs2 = list_of_training_datasets_prices[i][list_of_training_top_spreads[i].columns]
    top_spreads_prices.append(pairs2)

top_spreads_trading_prices = []
bottom_spreads_trading_prices = []
for i in range(len(list_of_trading_datasets_prices)):
    pairs = list_of_trading_datasets_prices[i][list_of_training_bottom_spreads[i].columns]
    bottom_spreads_trading_prices.append(pairs)

    pairs2 = list_of_trading_datasets_prices[i][list_of_training_top_spreads[i].columns]
    top_spreads_trading_prices.append(pairs2)


top_spreads_returns = []
bottom_spreads_returns = []
for dataset in top_spreads_prices:
    returns = dataset.pct_change()
    top_spreads_returns.append(returns)

for dataset in top_spreads_returns:
    dataset.iloc[0:1] = 0

for dataset in bottom_spreads_returns:
    returns = dataset.pct_change()
    bottom_spreads_returns.append(returns)

for dataset in bottom_spreads_returns:
    dataset.iloc[0:1] = 0
```