Haaga-Helia
University of Applied Sciences

Implementing and comparing various algorithmic trading strategies

Eric Mc Laren

Bachelor's Thesis
Degree Programme in
Business Administration
2018

Haaga-Helia
University of Applied Sciences

Abstract

April 24, 2018

Author(s)
Eric Mc Laren

Degree programme
Bachelor of Science, Business Information Technology

Report/thesis title
Implementing and comparing various algorithmic trading strategies

Number of pages and appendix pages
91 + 11

Most financial firms use algorithms to buy and sell financial assets. It is possible for amateur investors with programming knowledge or vice-versa, to implement algorithms and improve their strategies. In the theoretical part, the history of stock exchanges, analytical methodologies, strategies and testing process are discussed. For the empirical part, a few algorithms based on different strategies are written. Though the algorithms' results are analysed and compared, the main goal is to implement strategies, not find the one with the best results.

A website called Quantopian is used to recuperate the data, write the algorithms, run the program and analyse the results. The financial assets that are traded are US stocks. The period used to test the algorithms goes from the 1st of January 2004 to the 1st of January 2018. This enables us to test long term strategies: at least 10 years and see the impact of an important crisis: the subprime crisis of 2007. Though the algorithms trade US stocks, they are stock-generic, which means that they will work with stocks from any stock market. The programming language used is Python, since it is the only language supported by Quantopian.

This paper discusses basic strategies for different investment styles and how to improve them. The two algorithms and their different versions are compared to each other and the best algorithm is chosen based on the results and risks. Tools are used to try and find the best elements possible so that the second algorithm can beat the first. In this paper, we see that the first version of the first algorithm has the best returns. The focus is on the use of tools to implement and improve these algorithms.

The two main algorithms beat the chosen benchmark. However, it was expected that the second algorithm would beat the first one, since a separate tool was used to try and improve it. The results do not show that the tool is useless. Demonstrating the use and results of the tool in question requires more tests and data.

Keywords
Algorithm, Stock, Strategy, Stock exchange, Stock market, Analysis.

Table of contents

1 Introduction

This thesis evolves around writing algorithms that buy and sell financial assets in a fully automated fashion. Stocks, commodities, currencies and many more used to be bought and sold in specific physical locations called stock exchanges. Today, most transactions are done on computers and do not require traders to be on-site anymore. Computers are also trading by themselves, without having humans interfere. This is called algorithmic trading or program trading.

In the theoretical part, the history of stock markets, main analytical methods, common strategies and testing methods are discussed. In the empirical part, the tools used, the python notebook, the implementation of the two algorithms and their comparison are discussed. The python notebook is a tool used to write live documents. This means that the data used for the results is updated each time the paper is read.

I am going to write two algorithms. The first one will have four versions and use various methods to try different strategies. The second is written with the use of a library to try and find better factors in order to beat the first algorithm's best strategy. Starting with some financial knowledge, this is an interesting exercise to find out if it is possible to learn how to write these algorithms and have good returns with moderate risk. The two algorithms are long-term focused and represent an investment strategy a person can use while only having a limited time to devote to tracking the stock market. This enables us to see how the algorithms can improve the investment strategies used by humans. The focus is on learning and writing the algorithms. Setting a specific goal to reach, i.e. minimum returns is not part of my thesis because this would have a bigger finance part, and I want to focus on the IT side.

Throughout the thesis, numerical examples are given to explain strategies. These examples and the algorithms themselves do not include any taxes or fees. The decision to not include these elements is based on the fact that each country has its own taxes and each bank its own fees. In real life situation, these fees and taxes must be considered when doing predictions.

A glossary has been written to provide the readers with more detailed explanations of technical terms. It can be found in the appendix 1 at the end of the document.

2 Theoretical framework

The first Stock Exchange to be established was the Amsterdam Stock Exchange in 1602. From then on, other countries opened their market places[1] and some of them brought innovations of their own. Although being a very interesting subject, I will not be discussing the history, improvements of stock exchanges and the financial industry between 1602 and 1970. The reason for this is that the period does not include any IT implementation at a level that is significant for computerized trading workflows. The methodologies used before the 1970s will be briefly discussed. From the 1970s, there is an important evolution due to IT being incorporated, and this is directly linked to this Thesis' subject. For this reason, focus will be brought on the 1970 to today period. (Beattie 2018.)

2.1 History

Before the 1970s, all trading was done on "trading floors". Trading floors are found in the buildings of stock exchanges and are zones where traders complete the buying and selling of assets to try to make a profit (Banks 2010, 219; Bannock & Manser 2003, 110; Bannock & Manser 2003, 267.). A (financial) asset represents any element that can be sold and bought in stock exchanges: stocks, commodities, currencies, etc (Bannock & Manser 2003, 11; Banks 2010, 28.). The process in which these traders buy and sell stocks is called an "open outcry". An open outcry is the process in which a trader wishing to sell a share at a certain price, cries out the asset and its price. A buyer responds to that offer. If the price is the same, the contract is made. The use of hand signals is also used to help with communication. The open outcry method, requires that the bids: the price a trader is willing to pay, and offers are made in the open market, giving everyone a chance to compete. If several people are interested in the asset being sold, the bid is done as an auction, and the buyer with the highest bid wins (Banks 2010, 34.). (Banks 2010, 364; Bannock & Manser 2003, 196; Tewels & Bradley 1998, 110.)

The decade from 1970 to 1980 is a transition period, where we see the arrival of the first "trading turrets" and computerized orders. A trading turret or dealer board is a specific telephone key system that was mostly used by traders. These phones were specifically built to enable traders to exchange or give information between them and their organizations, customers or counter-parties (Bannock & Manser 2003, 67.). In 1971, Nasdaq implemented the first computers used for trading. At that time, they did not pass orders, but

[1] Other term commonly used for stock exchange.

routed them electronically to the correct trading post in order to save time. The orders where then executed manually. (Tewels & Bradley 1998, 128-129.)

During the period of 1980 to 1990, trading turrets became dominant and were the principal method used for exchanging information and concluding deals. Computerization continued to develop, but was still only used to route orders electronically, not fulfil them. In 1987, CME Group came up with one of the earliest widespread electronic trading system. Although being a proof of concept, this opened huge possibilities for the future of stock exchanges. (Tewels & Bradley 1998, 130-131; Blythe 2017.)

The 1990s to 2000s period represents the automation of stock markets. In 1992, the CME Group fully launched the project that had been conceived five years prior. Other companies like Oak Trading Systems launched their own electronic trading systems (Oak Trading Systems 2010.). These systems were capable of executing orders. The traders were being moved from the trading floors to external locations and were passing orders from their computers. This resulted in a decrease of personnel working at the physical stock exchanges' locations and the replacement of trading turrets by computers. (Tewels & Bradley 1998, 130-131; Blythe 2017.)

From 2000 we have acknowledgement of algorithmic trading. In 2001, IBM researchers published a paper at the International Joint Conference on Artificial intelligence. They used the two following algorithms in experimental laboratory versions of financial markets:

- MGD: modified version of the "GD" algorithm invented by Steven Gjerstad & John Dickhaut in 1996 -1997.
- ZIP: invented at HP by professor David Cliff in 1996

The research team showed that they could consistently out-perform non-professional human traders with simplistic strategies. According to their paper, the financial impact could be measured in billions of US dollars annually. (Das, Hanson, Kephart & Tesaura 2001.)

The difference between 1980 and 2010 is notable. In 1980, there were 5'500 people working on the trading floor, while in 2013, only 700 people were left. The atmosphere has also changed. Where there was a lot of shouting in the trading pit, now there is a "near-soundstage" that is perfect for Television Networks like Bloomberg and CNBC. These networks have set up permanent broadcasting outposts on the trading floor for all the financial press coverage. (Levine, D. 2013; Hiltzik 2014; Schaffler 2015.)

The two figures below represent the personnel working at the New York Stock Exchange's physical location. The first picture was taken in 1987 and shows a crowded area which is bustling with traders. The second was taken in 2017 and represents the emptiness that is now a part of the NYSE.

Figure 1: The New York Stock Exchange trading floor on October 19th, 1987 (Yahoo Finance 2017.)

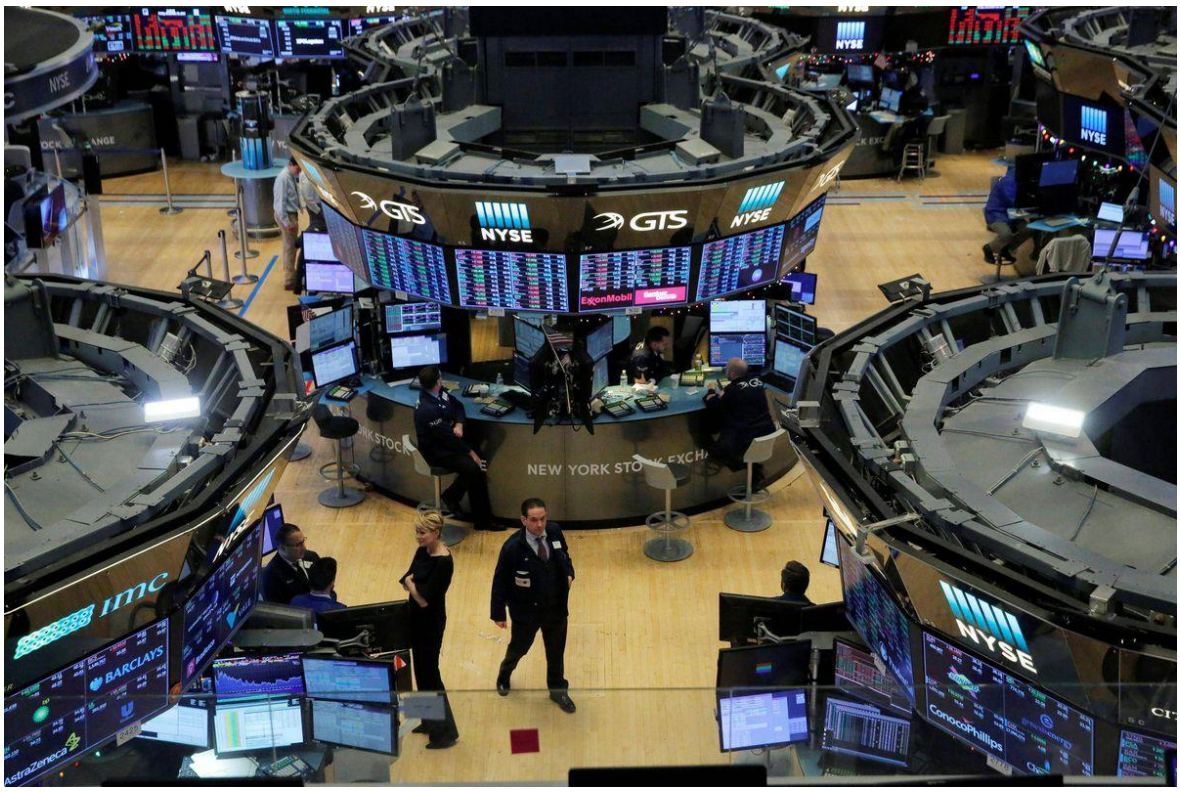

Figure 2: The New York Stock Exchange trading floor on December 29th, 2017 (Shaffler 2015.).

2.2 Market analysing

There are many different techniques used to analyse the stock market. To the contrary of stock exchanges, stock markets are non-physical entities that represent all the owners of a certain asset. Some analysts rely on charts or number crunching, while others investigate the company to understand its business model and functioning. Even if some of these methodologies are in direct contradiction with each other, it is not uncommon to see them being used side by side. It is even approved and recommended to mix these different procedures, since it provides traders with more tools and perspective to analyse the situation. The principal techniques are:

- Fundamental analysis for long term strategies.
- Technical analysis for short term strategies.
- Qualitative analysis: used in fundamental analysis.
- Quantitative analysis: used in fundamental and technical analysis.
- Efficient market hypothesis.

Since I am replicating an investment style that an amateur trader with limited time to devote to investing will use, my thesis will be focusing on Fundamental analysis with quanti-

tative techniques. One of the versions of my first algorithm will implement technical analysis, and depending on the results, it is possible that the second algorithm uses technical analysis. Qualitative analysis requires assessments made on unquantifiable factors. This makes this task extremely difficult for computers, which are not able to handle it yet. Efficient market hypothesis is not used either in my thesis, since it does not provide much elements to discuss. I will discuss each analytical process separately, in order to give a good notion of what each one brings to the table.

2.2.1 Fundamental Analysis

Fundamental analysis is generally used for long-term investment and is based on two major assumptions:

- The stock is not trading at its “true” price.
- In the long term, the stock market always reflects the fundamentals.

The “true” price of a company is called the intrinsic value. According to these assumptions, a security can be:

- Overvalued: stock price > intrinsic value.
- Undervalued: stock price < intrinsic value.

Fundamental analysts focus on underlying factors that affect the company’s business such as financial ratios, the company’s brand, market share, etc. They believe that it is possible to evaluate the intrinsic values of firms and select those that are trading at a lower price. They then wait for the stock price to catch up with the intrinsic value and sell the share at a profit. This scheme is called: stock picking or stock selection. In order to do this analysis, investors focus on fundamental factors that can be grouped into two categories: Qualitative or Quantitative.

These two categories are discussed separately below, since technical analysis also uses quantitative factors. Fundamental analysis has two big caveats. The first one is that an investor cannot be sure if his intrinsic value estimation is correct. The second is that the investor does not know how long it will take for the stock market to reflect the intrinsic value. (Folger 2018a.)

A fundamental analyst looks at the macroeconomic situation: how the general economy is performing and how the different actors are interacting with each other. This is done by looking at various levels: international, national, industry and sector. Various metrics are

used like growth domestic product (GDP), price indices, unemployment rates, inflation, growth rates and more.

A fundamental analyst will also look at the microeconomic situation: how companies and individuals make decisions regarding price levels, resource allocation, supply and demand, etc. An investor can take a top-down approach, where the economy is analysed as a whole, and then stocks are selected by going into detail: Global economy > Country > Industry > Sector > Company. It is also possible to do a bottom-up approach. This happens when an investor hears of or sees a worthwhile investment and decides to look at the general picture by going from a detailed perspective to a general overview. (Rodrigo 2017)

To add to that, some fundamental analysts also rely on intuition and emotions when investing. This is called “gut feeling”. Warren Buffet, CEO of Berkshire Hathaway, famously advises: “to be fearful when others are greedy and to be greedy when others are fearful.” He relies on his emotions and on the emotions of the general public to help him in his decision making. (Mehta 2012; Maidique 2011.)

2.2.2 Technical Analysis

Technical analysis is based on three assumptions:

- The market discounts everything.
- Price moves in trends.
- History tends to repeat itself.

The first is in complete contradiction with the fundamental analysts’ believes. While fundamental analysts think that a stock is not trading at its intrinsic value, technical analysts believe that the stock’s price has already taken all factors: companies fundamentals, micro and macro-economic situations and even market psychology such as excitement and fear into account and is trading at its true value.

The second assumption states that a stock’s price is more likely to move accordingly to a past trend, rather than in an erratic way.

The third assumption asserts that due to market psychology, emotions like fear and excitement make the market movements repetitive and predictable. These emotions are analysed with the means of chart patterns.

In contrast with fundamental analysis, technical analysis focuses on short to mid-term timeframes. The analyst does not look for the stocks' values but tries to find the best time to enter or exit a trade. This is called: market timing. The principal tools a technical analyst uses is charts. He works with different chart types: line chart, bar chart, candlestick charts, point and figures chart. Each chart contains different information, and the analyst looks for different patterns in each of them. The trades are made depending on the patterns that are seen. Quantitative analysis is used to determine price limits and other features that are common in graphs and enable traders to take decisions on when to enter and exit a trade. (Thomsett 2006, 34-39.)

2.2.3 Quantitative Analysis

Quantitative analysis is the use of mathematical measurements and calculations, statistical modelling and research to understand or predict behaviour or events. The objective is to represent a situation with a numerical value. Using mathematical values or variables enables the analysts, also known as "quants" or "quant/algo jockeys" to compare them with each other (Yager 2012.).

This type of analysis is widely used for long and short-term strategies. The first being used by fundamental analysts to calculate ratios and assess them based on the values found in the balance-sheet and results of a company. Price to earnings (pe) ratio, price to book (pb) ratio and earnings per share (eps) are examples of calculations used. The latter is used by technical analysts to calculate values based on historical and current stock prices. Simple moving averages, Bollinger bands, relative strength index and accelerator decelerator oscillator are examples of metrics used to plot values on graphs that indicate when to enter and exit trades

Governments, central banks and non-governmental organisations (NGOs), among others also use quantitative analysis for many reasons like monetary or economic policy decisions. They track data such as GDP, employment figures, and more.

The fundamental ratios and the simple moving averages listed above are explained in the first algorithm's section. The other technical metrics are not explained since they are not used for the algorithms in this thesis.

Due to its number crunching methodology, this type of analysis is perfect for computerized workflows. The capacity to solve problems and calculate ratios for analysis is vast, therefore it is used in various domains. The algorithms in this thesis will be based on quantitative analysis. (Scherbaum & Shockley 2015, 1-12.)

2.2.4 Qualitative Analysis

Qualitative analysis is a non-mathematical review investors and business managers use to take investment and business decisions. Examples of unquantifiable factors are: management expertise, industry cycles, labour relations and strength of research and development, among many others. This approach is purely subjective and necessitates its users to fully comprehend how businesses function. Therefore, advocates of this practice believe it takes many years to master. This procedure is only performed by humans for the moment, since computers cannot yet understand factors that are not possible to capture with numerical inputs. To add to the factors mentioned above, there is also: brand recognition, positive associations with a brand, management trustworthiness, competitive advantage, cultural shifts, customer satisfaction, patents worthiness and many more. Qualitative investors famously state to “Invest in people, not in companies”. These factors help us understand what defines this methodology and why computers are not yet capable of attempting it.

One big difficulty, is getting the information related to all these factors. CEOs do not open the doors of their companies to permit small investors to assess the managers, employees and company’s ethics. Big investors like Warren Buffet have the privilege to go and see the companies functioning because CEOs are willing to give them their time. This reason makes it harder for small investors to perform qualitative analysis. (Investopedia 2018.)

Due to the type of non-mathematical factors and complexity for computers, qualitative analysis will not be used in any of the algorithms discussed in this paper.

2.2.5 Efficient Market Hypothesis (EMH)

This investment theory reports that it is impossible “to beat the market”. This statement is based on “Market Efficiency”, which asserts that all available and relevant information is always incorporated into the stock prices. This means that neither stock selection: buying undervalued securities and selling at a higher price, or market timing is effective in outperforming the stock market. According to this theory, outperforming the market requires to

buy riskier assets. This means that there is a higher chance to lose part or all of the initial investment. EMH proponents conclude that:

- In order to outperform the stock market, an investor must buy riskier assets.
- The best solution is to invest in low-cost passive index funds or exchange traded funds (ETF), which replicate stock indices.

A fund regroups various assets into one portfolio, and investors can invest directly into that fund, instead of having to buy each element. While fund managers try to beat stock indices like the S&P500, passive funds only follow its performance, drastically reducing transaction costs and fees. (Syndicateroom 2018)

Standard & Poor's released a study at the end of 2016 stating that 1 in 20 actively managed funds actually beat the stock market during 2016. To add to that, the one-year period ended with more than 94% of managers trailing behind the index by 17.13%. The results for the last 5 and 15 years are on par with the one-year period's numbers. (Soe & Poirier 2016; Hulbert 2017.)

I will not be discussing any further this hypothesis since buying funds that replicate financial indices does not provide enough material for a thesis in my opinion. The topic evolving around risk related assets is interesting and I will be using some of these factors when analysing my algorithms' results. I will not be evaluating these metrics since they belong to a whole other topic that could suffice itself. (Thomsett 2006, 46-47.)

2.3 Algorithmic trading strategies

Algorithms are used in trading for many different purposes. They can replicate a human style strategy, where the number of stocks is relatively low (around 20) and the quantity of data to process can be done in a feasible time-frame. This can be a time-frame for amateur traders that trade on the side of their job, or for professionals that do that all day.

The algorithms can also improve human strategies. This means that the underlying logic is the same, but the quantity of data and number of stocks is higher. The execution speed can also be faster, for instance: strategies like arbitrage (Arbitrage: 2.3.1) and pairs trading (Paris trading: 2.3.2) can be done by human investors, but the greatly improved execution speed of computers makes these trades a lot more profitable.

The last algorithms, mainly high frequency trading (High Frequency Trading: 2.3.4), trade in a completely new way and the methodology is not feasible by humans. The time-frame

they are based on is at a sub-second level, making these trades only possible for computers. These strategies require fast connexion rates to the stock markets, huge amount of data and fast execution speeds. Different methods exist. Some evolve around finding a buyer and seller and profiting from the price difference by being a “middle-man”: buying from the seller and selling to the buyer as an intermediary. Others pertain in more obscure strategies like emitting fake orders to slow down the concurrence so that they don’t have access to the real trades. These strategies become a battle ground where algorithms evolve and try to beat other algorithms by understanding their logic and using it to they’re advantage. (Meerman 2013.)

2.3.1 Arbitrage

There are several financial markets and stock exchanges. Companies can list themselves in several stock exchanges to increase liquidity and gain popularity. The price of a company’s stock must be updated for each market, and they do not all update the information at the exact same time. This is called market inefficiencies. Currencies or commodities are also listed in several places, and the price can differ from one place to the other. As an example: before going on holiday, you want to change your EUR to USD. The two banks at the shopping centre propose these rates:

- Bank A: 1.- EUR = 1.18 USD.
- Bank B: 1.- EUR = 1.20 USD.

This means that if you want to change 1’000.- EUR and you go to bank 1, you will get 1’180.- USD. But if you go to bank 2, you will get 1’200.- USD. It is a better choice to go to the second one.

Algorithms doing arbitrage will profit from these market inefficiencies to make benefits. A simple arbitrage example: a company’s stock is trading at 20.- USD on the NYSE (New York Stock Exchange) and is also trading at 20.10 USD on the SIX Swiss Exchange. The algorithm buys the stock at 20.- USD and sells it on the other stock market at 20.10 USD. The algorithm proceeds until the NYSE runs out of inventory for the company’s stock, or until the price readjustment is made by the specialists of one of the stock exchanges.

A complex arbitrage example: The algorithm starts off with 1’000’000.- USD. The algorithm has the following information:

- Institution A: 1 EUR = 0.934 USD.
- Institution B: 1 GBP = 1.275 EUR.
- Institution C: 1 GBP = 1.400 USD.

As the table below demonstrates, the algorithm proceeds in three steps to change its USD to other currencies. It finishes by buying back some USD at a higher rate to make profits.

Table 1: Complex arbitrage example

Steps	*Before*	*After*
Step 1	1’000’000.- USD	934’000.- EUR
Step 2	934’000.- EUR	732’549.02 GBP
Step 3	732’549.02 GBP	1’025’568.63 USD

The use of algorithms and computerized workflows here is important since the tasks must be executed swiftly, before the readjustments take place or the inventories empty out. When talking about inventory, it is important to note that many algorithms might be competing. The algorithm starts by finding the different rates, and then proceeds to trade the asset. If other algorithms see the difference, they will start trading the same stock, currency, commodity, etc., diminishing the inventory and by consequence, the first algorithm’s benefits. Therefore, the algorithm must be efficient and fast enough to see the differences first and do the trading as fast as possible, before the other algorithms chip in. (Moffatt 2017.)

2.3.2 Pairs trading

Pairs trading functions a lot like Arbitrage, in the sense that it profits from market inefficiencies, also called market weaknesses. Instead of trading the same stock in different markets, the trade is made on the same market with two securities that are very strongly correlated: their prices move in the same direction and with the similar intensity. The weakness come from taking two opposite positions of same value. The investor takes a long position and a short position. Buying a stock is called going long or taking a long position while going short or taking a short position is selling a borrowed stock. In the first case profit comes from the price increasing, while the second’s profits come from the price decreasing. Normally, taking opposite positions of same value cancel out all potential loses or gains. The following table demonstrates that.

Table 2: Contradicting positions

Stock price	*Goes up*	*Goes down*
Long position	Goes up	Goes down
Short position	Goes down	Goes up
Together	Doesn't move	Doesn't move

Table 3: Pairs trading example without market weakness

	Initial price	*Outcome A*	*Gain*	*Outcome B*	*Gain*
Stock A price	100.- USD	200.- USD	+100%	50.- USD	- 50%
Stock B price	100.- USD	200.- USD	+100%	50.- USD	-50%
Long A	100.- USD	200.- USD	+100%	50.- USD	-50%
Short B	100.- USD	0.- USD	-100%	150.- USD	+50%
Long + Short	200.- USD	200.- USD	0%	200.- USD	0%

The second table shows that taking contradicting positions cancels out any profits or losses. In this example, the correlation is a one to one, meaning that if Stock A moves up of X points, stock B will also move up of X points. The second example shows a "real life" situation, where two highly correlated securities are affected by market weaknesses. This means that for short periods, the one to one correlation will change and not be equal. The algorithms will profit from these differences to make a profit.
Here is an example:

Table 4: Pairs trading example with market weakness

	Initial price	*After period 1*	*After period 2*	*Gain*
Stock A price	19.90 USD	20.- USD	20.25 USD	+1.25%
Stock B price	19.90 USD	20.50 USD	20.25 USD	- 1.22%
Long A	0.- USD	200.- USD	202.50 USD	+1.25%
Short B	0.- USD	205.- USD	207.50 USD	+1.22%
Long + Short	0.- USD	405.- USD	410.- USD	+1.23%

In this example, the initial price for stock A and B is 19.90 USD. There is no long or short positions taken yet, since the trader is waiting for a difference.

After period 1, stock A is trading at 20.- USD and stock B at 20.50 USD. The correlation is not one to one anymore. It has moved to 1 to 1.025. That means that for every unit, A goes up, B will go up by 1.025. This is verified with the values in the table: 20 * 1.025 = 20.50 USD.

The analyst will now go long on the under-performing asset: Stock A, and short the over-performing asset. The idea is that the over-performing asset's price will depreciate, while the under-performing asset's value appreciates, and their prices will meet somewhere in the middle.

At the end of period 2, the gap has closed. Stock A's price has gone up, providing benefits on the long position. Stock B's price has gone down, providing benefits on the short position.

It is important to note that these periods are often very short. This strategy has two specific risks:

- Model risk: the model used to trade does not perform as expected, i.e. instead of closing, the gap opens. This creates a loss on both positions.
- Execution risk: This happens when only part of the trade is executed, or if the trader experiences slippage: the trader receives a less favourable price than the one expected. Slippage is very important here, since the differences in values are small in trading pairs. (Folger 2018b; Mitchell 2014.)

2.3.3 Trading Before Index Fund Rebalancing, Mean Reversion & Scalping

Index funds track indices or commodities and replicates their pricing. This requires them to readjust their portfolio periodically. They release their schedules to the market well before the trading takes place. Due to the size of index funds and the massive amounts of assets that are bought and sold, the price of these items is strongly impacted. The algorithms use that information to buy and sell assets before their price changes. For example: an index fund announces that it will buy a certain stock in one month, meaning that its price will rise. The trader will proceed to buy the asset before the fund does, and sell it after it after the price has risen. If the index fund announces it will sell a stock, the trader will sell it, if it was purchased it beforehand or short it (Maverick 2015.).

Mean reversion, also called trading range, states that the stock price's highs and lows are temporary. The algorithm determines the trading price range of the stock for a certain period, and then calculates the average for that period. If the price is below that average, the

asset is considered undervalued and is bought. If the price is above the average, the asset is considered overvalued and is sold. Standard deviation is often used on recent prices as a buy and sell indicator. Some stock reporting services like Yahoo Finance, Morningstar, Google Finance, etc. report the moving averages for 50 and 100 days. However, evaluating the highs and lows must still be done by the analyst. (Butler 2016.)

Scalping is a fast-paced strategy. The algorithm buys and holds on to a security for a very short period: minutes or seconds. It trades within a short time-frame in order to make profits from small differences in assets' prices. For this strategy to be profitable, the algorithm must dispose of a lot of money. Investing huge sums is a necessity when the benefits are small. With fibre optic cable, and the increased communication speed, trades can even be done in milliseconds, or lower. This will be discussed in more details in the point below, since it is an important part of the high frequency trading strategies. (Milton 2016.)

2.3.4 High Frequency Trading

High frequency trading is the most publicized form of algorithmic trading. It is suspected that in 2009, 73% of all equity trading volume is done by HFTs, even though only 2% of financial companies pertain in these algorithms (Lati 2009.). This new methodology has ushered in a new era of algorithms that use newly created strategies or improve already existing strategies. It is difficult to define these algorithms since no specific definition has yet been given. To characterize these, we will look at their main attributes.

In order to do algorithmic trading, a financial company must produce algorithms that will do automated trading. But HFT needs much more. These algorithms work on a sub-millisecond level, which requires efficient and optimized code coupled with high connection rates to the stock exchanges. A high turnover and high order-to-trade ratio: percentage of passed trades that are executed, also characterise these algorithms.

HFT poses new challenges to regulators and governments which makes it difficult for them to track and enforce laws and regulations. To add to that, the sub-second positions and high volumes increase volatility and are the cause of flash crashes.

The infrastructures needed for HFT make it impossible for small investors to pertain in these strategies. This results in big companies having more power and more weight in the financial markets. Some people argue that these firms only compete among themselves and not against small investors.

All requirements, pros, cons and main strategies are discussed below in the appropriate sections. (Meerman 2013.)

2.3.5 How do companies do High frequency trading?

Data centres, fast connections to stocks exchanges and algorithms are three main elements high frequency traders need.

Financial institutions build huge data centres with redundant systems, preventing any form of technical failure as much as possible. They usually have at least two copies of every part of the building and it must be as close as possible to the stock exchange, or at a strategic location between stock exchanges.

These companies then put fibre optic cables that go straight from the data centre to the stock exchange. The ultimate bottleneck here is the speed of light. It takes signals 67 milliseconds to travel half-way round the earth. As of summer 2015, London and New York's stock exchanges could communicate at a speed of 2.6 milliseconds.

The figure below shows the fibre optic cables that have been installed between North America and Europe. We can also see lines that are going to Africa, Central and South America.

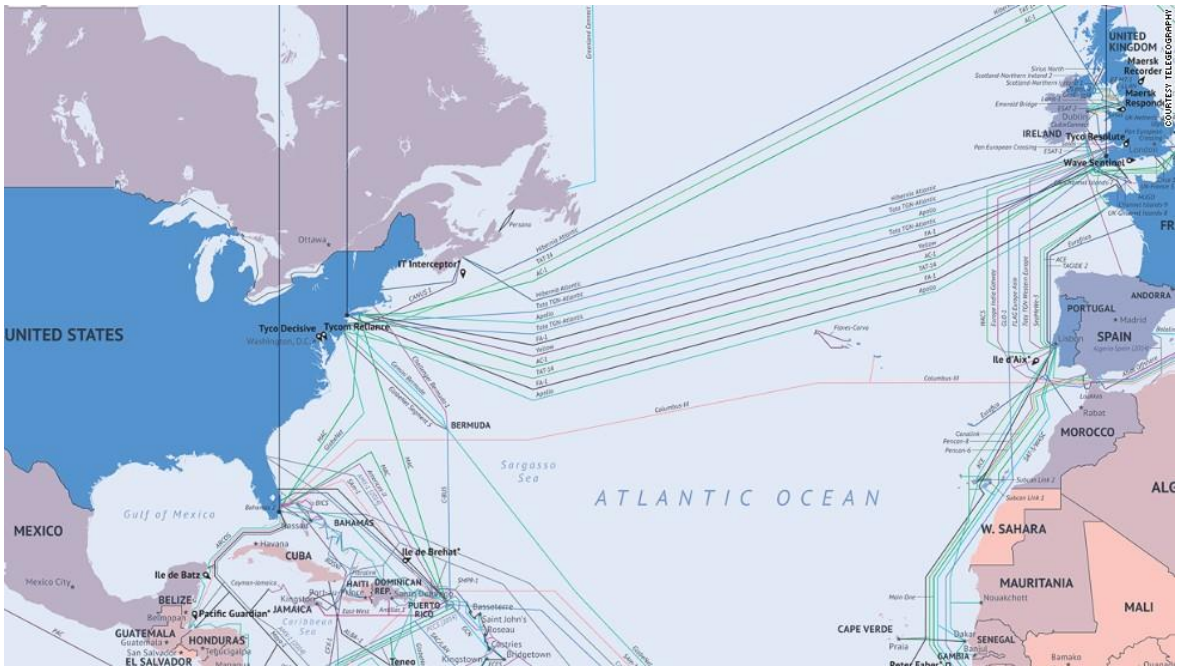

Figure 3: The submarine lines between New York and London (Competent Hustling 2014.)

The algorithms used by high frequency traders are very criticized as they belong to a set of algorithms called: “Black box algorithms”. The picture below shows how a black box process works.

Figure 4: Black box model (Lewis & Monett 2017.)

A black box is an object, system or device that is analysed only by its inputs and outputs. Its internal structure, like a complex mathematical model, is unknown. This type of methodology is heavily criticized because its true risk is unknown to investors and regulators. To add to that, the rising popularity of Artificial Intelligence and its sophisticated quantitative methods is adding complexity to the undisclosed black boxes’ processes. (Lewis & Monett 2017.)

2.3.6 The pros and cons of HFT

Being highly publicized and criticized, HFT has several arguments going for and against it.

Pros:

- Sharpe ratio: compares risk taken to reward to find out if returns are due to higher risk or good strategy: The higher the number, the better. HFT has a ratio 10 times higher than buy and hold strategies (long term).
- Provides liquidity to the market (see cons).
- Facilitates order execution (see cons).
- Lowers volatility: risk associated with an asset. This results in a surplus liquidity (see cons).
- Narrowing spreads: difference between bid and ask price, for other market participants, which makes trading cheaper for everyone (see cons).

Cons:

- Only big structures can profit from HFT.
- Causes flash crashes (HFT Flash crashes: 2.3.7).
- Erodes liquidity, since the High Frequency Traders trade stocks until long term investors get their hands on the stock.
- Does not facilitate order execution when the market has irregularities, as most of these algorithms sell massive amounts in that circumstance. Thus, taking all the networks' resources for their trades and eroding the other participants' access.
- Increases volatility: as soon as something seems strange to the algorithm, it sells, and this creates flash crashes (HFT Flash crashes: 2.3.7).
- Regulation is scarce, since the algorithms are not fully understood, and the timeframe is too small for human comprehension.

As you can see, most pros and cons contradict themselves. Some studies show that HFT is beneficial to the market, while others state the opposite. Although having a lot of publicity, these lists show us that HFT is still very opaque for the public, regulators and even high frequency traders themselves. Governments and regulators are investigating the subject due to the size, impact and risks linked to these types of algorithms. (Shmuel 2014; Sethi 2012.)

2.3.7 HFT Flash crashes

Financial crises, like the Great Depression of the 1930s, or the Subprime Mortgage crisis from 2007 – 2008 took years for the economy to recover from. The first started in 1929 and lasted until 1941 (Galbraith 2009, 124-125.). The second started in 2007 and lasted until 2009, with many people arguing that the economy was still recovering years later (Indran 2017.). The Flash crash on the 6th of May 2010 started at 2:32 P.M. and lasted approximatively 36 minutes. It is this short duration that explains the name of these types of crashes. The Dow Jones dropped 9% in a matter of minutes, before rebounding back to its previous rate. Other indices like the S&P500 and Nasdaq also plunged in the same period. Trillions of dollars were lost during these 36 minutes. Share prices of big companies and ETFs dropped to a few cents a share before going back to their usual trading price. There have been other flash crashes since 2010. While not as big as the 2010 crash, it shows that the regulations in place are still not effective enough. (Treanor 2015.)

The three figures below show in order: the Dow Jones index during the Great Depression, the Subprime crisis and the Flash crash on the 6th of May 2010. The caption associated with each image details how long it took for the stock market to reach its former height after each crash.

Figure 5: The Dow Jones plummeted in October 1929: it took 30 years for the stock market to reach the level before the crash, in May 1959 (Macrotrends 2018.)

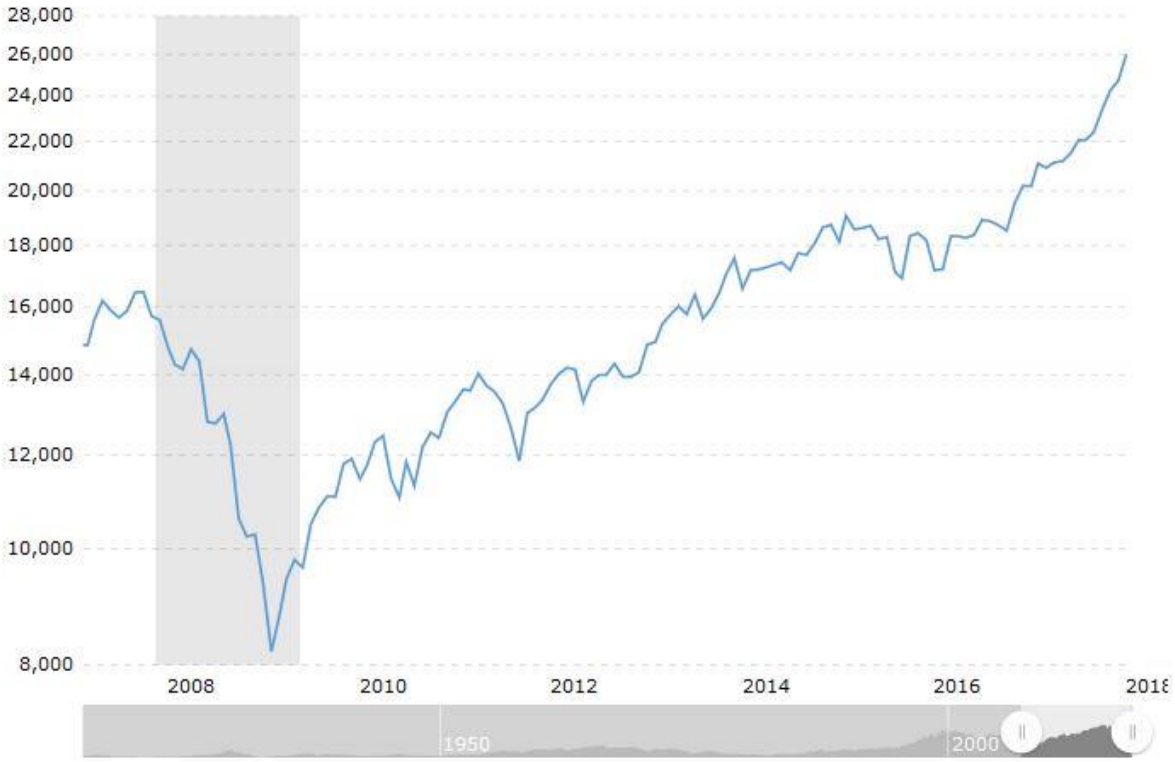

Figure 6: The Dow Jones fell in October 2007, and only recuperated in October 2013, 6 years later (Macrotrends 2018.)

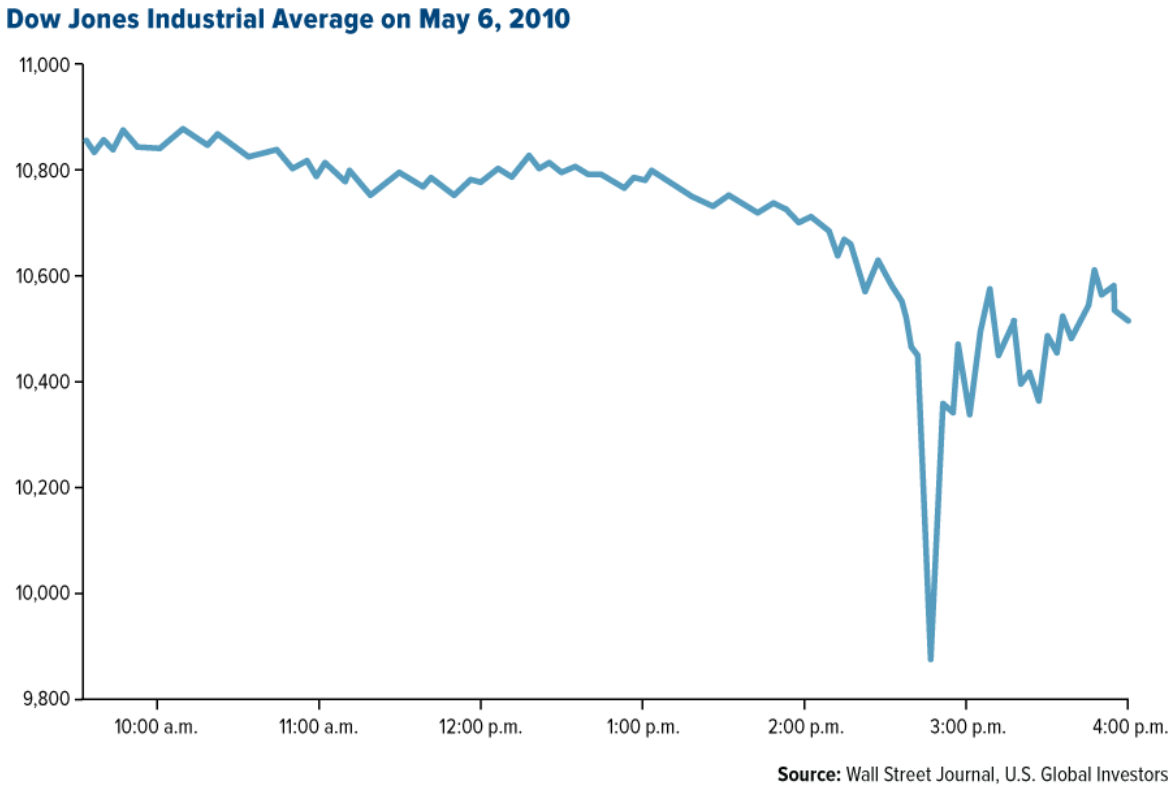

Figure 7: The Dow Jones plunged at 2.32 P.M. on May 6th, 2010 and recovered 36 minutes later (Twin 2010.)

2.3.8 Solutions to counter HFT

An ideal solution would be having regulators follow every bid and ask on every stock exchange. This means they could enforce laws linked to market manipulation by condemning firms and traders using these strategies. This requires them to have computers and systems that have enough computational power to assess all the data on a sub-second level. Unfortunately, they are very far from having that computational power and other solutions must be found. One solution would be to enforce a minimum speed: trades cannot be done under a certain amount of time. In theory this might prevent flash crashes and would enable regulators to follow the stock markets closely.

For the moment, this is only an idea and is not enforced. It will be some time before more regulations are put in place, and it is possible that regulations are removed. For instance, the Dodd Frank act created in 2010, during the Obama administration, in the United States of America enforced regulations for the whole financial industry, including HFT. This was a government response to the subprime crisis. In 2017, this regulation is being voted on in order to dismantle it (Yu 2017.). This will result in less (strict) regulations, more liberty for HFT institutions and thus, more risk on the financial markets. (McFarlane 2018.) Although being difficult to monitor, regulators have fined companies for not complying with rules. A few examples are:

- Octeg LCC in March 2012 for not maintaining sufficient supervision over its financial activities (Demos 2012.).
- Knight Capital in October 2013 for violating market access rules by ignoring error messages before passing millions of orders (Mamudi 2013).
- Latour Trading LCC in September 2014 for underestimating risk by using faulty calculations to buy and sell assets without having enough capital (McCrank 2015.).

2.3.9 HFT Market Making

Market making is one of the most known HFT strategy. The financial firm will find a buyer and seller for a specific stock. Once they are found, the company will buy from the seller, and sell to the buyer, providing the selling price is lower than the buying price. The example below illustrates the concept:

- Trader A wants to sell 100 shares at 50.- EUR each for a total of 5000.- EUR.
- Trader B wants to buy 100 shares at 50.10 EUR each for a total of 5010.- EUR.
- The financial corporation will buy the shares for 5000.- EUR and sell them for 5010.- EUR, thus making a 10.- EUR profit.

This schema is very profitable for companies since they are doing this with millions of shares each day. The financial institution must have a fast connection to the stock market in order to see a seller's order and fulfil it before the buyer sees it. The firm will then sell to the buyer, making a profit on the spread. It is also very common to have more than one intermediary between the original seller and final buyer.

This type of strategy is subject to many criticisms. Since these companies act as middlemen, the buyer will not profit from buying the stock at a lower price than his stop limit: maximum price that investor will purchase a good for. This means that financial institutions not using HFT and investors that trade often are losing a few cents for each transaction. This may not seem a lot at first, but if the firm is passing hundreds, thousands or even more transactions each day, the losses become substantial. (Meerman 2013; Levine, M. 2013.)

2.3.10 HFT Market Weaknesses and Manipulation

Due to its fast-paced trading, HFT is the best way to profit from market weaknesses and use market manipulation strategies. The following algorithms profit from either weaknesses or use manipulation:

HFT Arbitrage and pairs trading. Theses algorithms profit from market weaknesses and function in the exact same way as they have been described above (Arbitrage: 2.3.1; Paris trading: 2.3.2). High frequency trading algorithms take full advantage of the logic and implementation of these strategies. The time it takes a machine to look for discrepancies in the different markets is far lower that the time it takes a human to do. Therefore, these strategies are commonly used in HFT. (Hanson & Hall 2013, 3-10.)

HFT Spoofing. This strategy is an implementation of market manipulation. Before selling the owned shares, the High Frequency Trader will make offers to buy shares from the same company at a higher price. However, there is no will to fulfil the order and will cancel it once the price has gone up. Enforced since 2010, the Dodd-Frank Act forbids spoofing, but the law is vague, and it is up to the market regulators to enforce specific rules. (McLeod 2013.)

Quote stuffing is the practice of flooding the market with the creations and withdrawals of enormous amounts of orders. These must be processed by each market participant, and this delays their access to information concerning the following trades. This practice is very controversial since none of the orders in this strategy are real. The only point is to have an edge over other market participants, which need to assess the “fake” orders. This enables the HFT algorithms to be ahead in the trades that follow. Like spoofing, quote stuffing is considered to be market manipulation. Even though it is illegal, the law is vague, and regulators have the responsibility of writing and enforcing specific laws. It is also important to note that it is difficult for regulators to track these practices. The reason for this is that they do not have the required computational power to check all the different inputs from each trader. (Durden 2011.)

The two last strategies have become a battle ground for algorithms, with each other competing against each other. The algorithms try to understand the logic of other algorithms and use that information to evolve and trade in a manner that makes the latter’s strategies useless. (Meerman 2013.)

2.3.11 Thematic investment & Alpha factor seeking

The two strategies described below will be used to write the two algorithms for this thesis. The first will use thematic investment, while the second uses alpha factor seeking to try and beat the first algorithm.

Thematic investment is interesting because it can be done entirely by humans. The idea is to choose a theme or trend and invest accordingly. For instance, a thematic investment can be done for a specific sector: only industrial companies, green energy, only small or big companies. Investments can also be done by trends: technology companies, cannabis, emerging markets. Last but not least, it can also follow portfolio rules. The O'Shares ETF created by Kevin O'Leary is a good example:

- At least 100 stocks
- Each stock must pay dividends
- Each sector invested in must represent less than 20% of the portfolio
- Each stock invested in must represent less than 5% of the portfolio.

These types of strategies are well complimented with the use of algorithms, since they help process more data and look into more stocks. The first algorithm will use thematic investment with the following rules:

- 22 shares in the portfolio.
- Each share has the same weight (100%/22).
- Each sector is invested in with two shares. (Motif, 2015.)

Alpha factor seeking methodology takes a factor that is a ratio, calculation or number which has a financial meaning and analyses the evolution of that factor through time. This process is quite special, since it does not aim to use factors to determine when to buy or sell a stock but tries to determine the best factors to use. Through this examination, we try to determine how predictive a factor can be, compared to the returns of our portfolio. In order to do so, we must separate this part from the trading strategy completely. This prevents the results being biased due to a good or bad strategy. (Christopher 2017.)

The Python Notebook (explained in the Quantopian Research notebook chapter: 2.7) is a useful tool in this situation. It provides the developer with an environment that facilitates testing factors independently from strategies. The second algorithm will be using an alpha factor seeking strategy with the use of the Python Notebook.

2.4 Testing the strategies

Testing the strategies is an important part in the development of a new algorithm. It enables the developer to fine tune the algorithm and check that the performance, risks, etc. are acceptable. The type of trading done during testing is called paper trading. This means that no money is involved. Testing a strategy is done in two steps:

- Backtesting.
  The first step after an algorithm is written is to test it on historical data. This enables the developer to analyse its performance, volatility, drawdown: difference between a peek and a trough, annual rate of return: yearly profit expressed as a percentage, risk metrics, etc. As the name suggests, the testing is not done on real-time data.

- Forward testing.
  The second step is to test the algorithm on real-time data. The developer verifies that the analytical metrics used in the previous step are conform and have roughly the same values. Forward testing is also used to check that the algorithm can handle the live data feed.

- (Live trading).
  Live trading is often mentioned as the third step, even though it is not part of the testing process. This is the final part of the algorithm's development. It is now used with real time data and real money. (Automated Trading 2018.)

2.5 Tools used

In order to write the algorithms, an integrated development environment (IDE), programming language, dataset and analytical structure are required. An IDE is a program that contains various elements that help developers write applications. The core components needed are a source code editor and a code compiler and execution (or interpreter). Additional features like debugger or code completion are not compulsory but facilitate the workflow. When choosing an IDE, there are many options. The first step is to know what language is used. The most popular IDEs handle many languages, but some are more specific than others. Here is a short and non-exhaustive list of some common IDEs:

- Notepad++.
- Visual Studio.
- Sublime Text.
- Eclipse.
- IntelliJ.

It is important to note that some of these IDEs are text editors. The use of plugins enables these text editors to gain the features of an IDE and perform the same tasks. (O'Dell 2010.)

There are many different options possible when choosing a programming language. The names that come to mind at first are R and Python (Gregory 2014). These are data science languages used for data mining and statistical computing. To add to these, there are the more common programming languages that are used for applications outside data science like C, C++, Java. These languages all have libraries specifically built for data analysis, so are not uncommon choices. On forums, I often see people getting caught up in debates about what language is the most efficient or has the most features for data analysis.

Simply put, people are looking for the “best” language. While this debate is interesting and does provide arguments that help us choose a language over another, the most crucial factor is the logic. The syntax changes from language to language, but the logic stays the same. Understanding it enables the developer to write the code in a different language if needed. A few values that are not inherent to a programming language are: the amount of documentation, posts on forums and people ready to answer questions. In my opinion, these factors are the most important and outweigh the specifics of each programming language and/or libraries. A novice needs advice and help for trouble shooting, and the three factors quoted above are essential. Once one becomes an expert, the features offered by specific libraries or languages have a bigger impact and can be a defining factor. (Akiwatkar 2017; Piatetsky 2014.)

The two polls below provide a view of the languages used in data-driven science. While the first one is from a study matching programming languages to job postings, the second is a Survey.

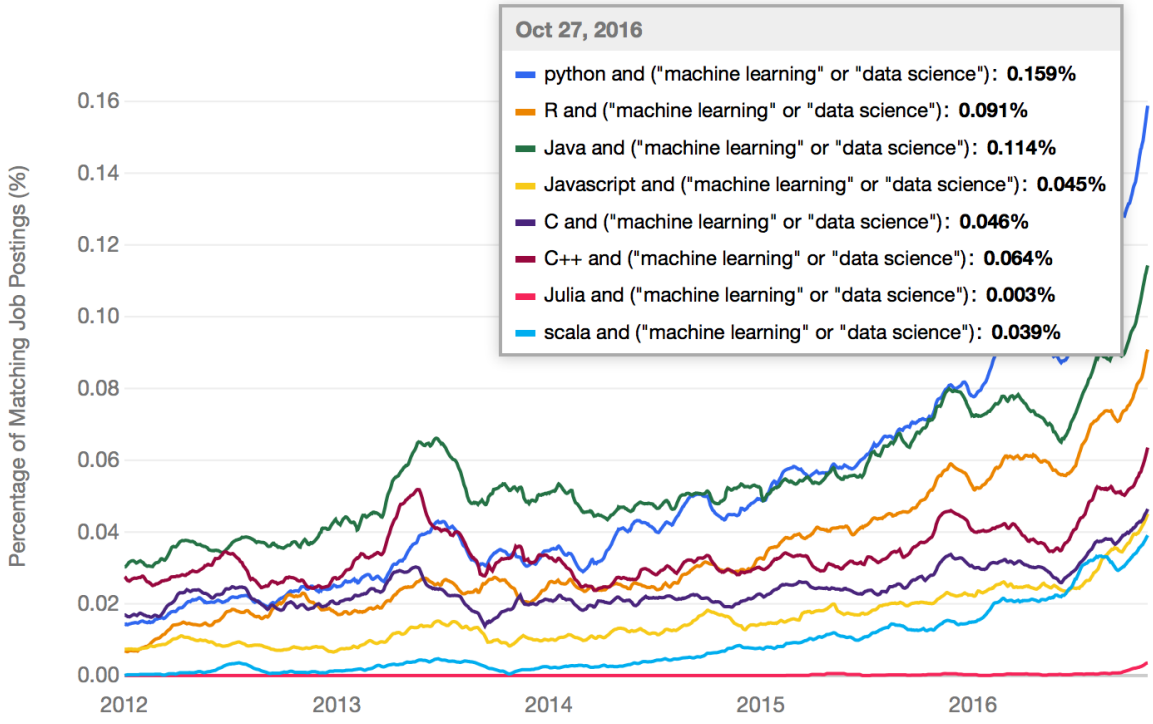

Figure 8: Percentage of Matching Job Positions for programming languages used for machine learning or data science (Akiwatkar 2017.)

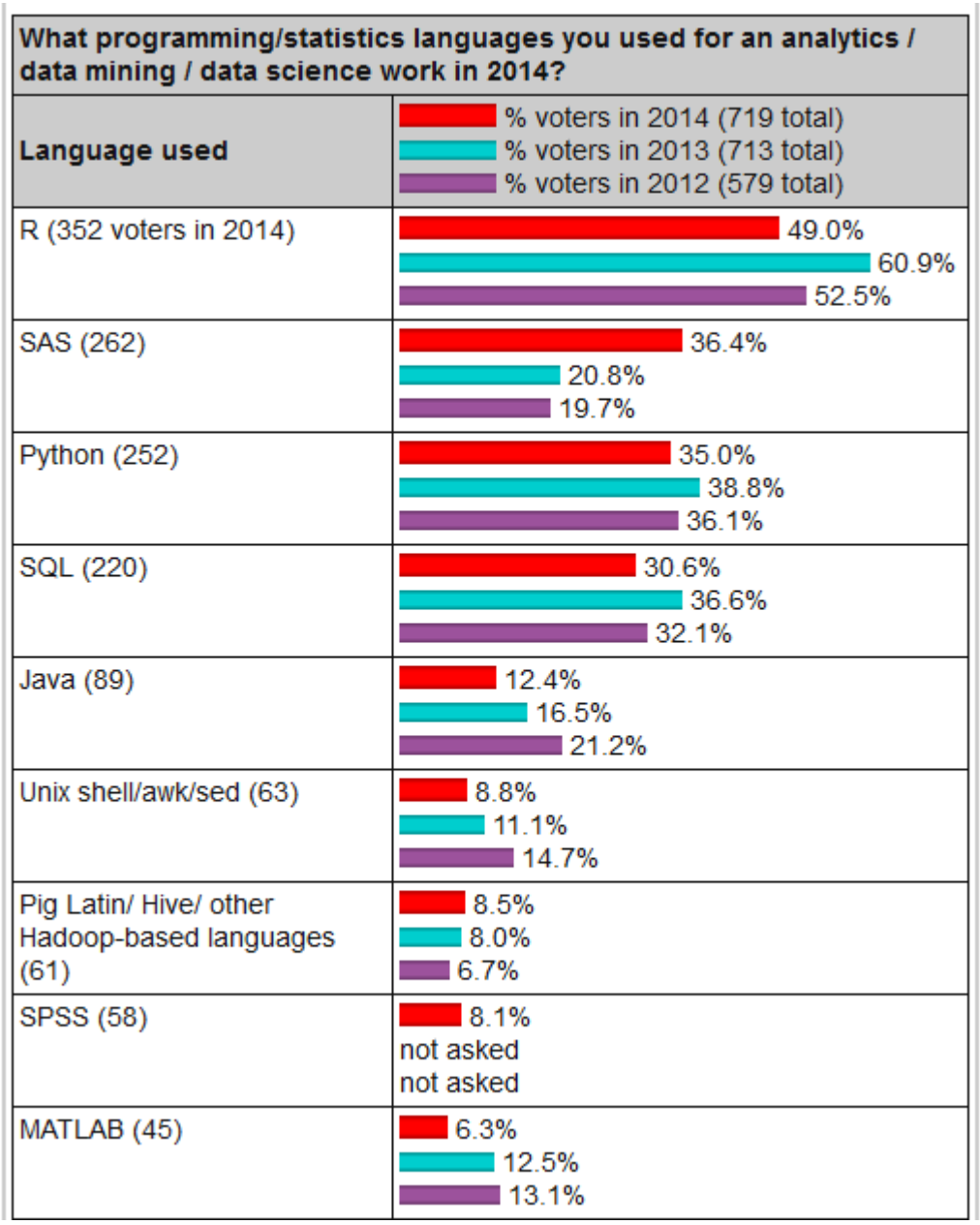

Figure 9: Survey representing the programming languages used for data science and data mining (Piatetsky 2014.)

The dataset is the asset which is the hardest to get. The quality and quantity of the information enables companies to have an advantage over other companies. Yahoo finance and Google finance's APIs are the most notable ones. Morningstar, Reuters, Bloomberg and Alpha Vantage also provide APIs. It is then a matter of finding what each service proposes and what information is included. The other point to note, is that some companies require a subscription to get real time prices and information. It is not uncommon that free versions have a "15 minute delay". (Csiszar 2017)

In my opinion, it is not compulsory for long term investors and amateurs to have real time access. It is possible to have profitable investments with the "15 minute delay".

The analysis structure is the last element we need. Once the IDE, programming language and dataset have been chosen, we need a way to analyse the data, i.e. transform the data from numbers to human readable forms, like graphs, (sorted) tables, and more. This generally comes from the programming languages' features or from libraries.

2.6 Ready-made structures & my selection

Some companies have made custom solutions that regroup the four components above into one structure. These systems have many advantages and disadvantages.
Advantages:

- No need to look for four components and make them work together.
- Troubleshooting is easier since everyone is using the same four constituents.
- Helps build a community.
- Provides documentation for this specific domain.
- Automated methods: the basic methods are already created and are called internally (back-end).

Disadvantages:

- No choice concerning the components, developers must use their IDE, supported language(s), dataset and data analysis structure.
- Less liberty: developers must follow their logical structure.
- When passing to another platform, the code must be partially rewritten since some parts are automated.

These solutions are preferred for novices. It is a complex domain that requires financial, programming and data analysis knowledge. Getting started and having sufficient insight in each of these domains to be able to write algorithms and analyse the results is a feat in itself. Having to deal with the different components and build everything up from scratch does not facilitate the process. That is why these solutions are a good entry point, simplifying some of the process and leaving the analysts work on the algorithms and grasp the logic.

Quantopian, Quantiacs and QuantConnect are the three main readymade structures (Quantopian 2018a; Quantiacs 2018; QuantConnect 2018).

Table 5: All-readymade structures comparison

	Quantopian	*Quantiacs*	*QuantConnect*
Supported language(s)	Python	Python, Mathlab	C#, Python
Competition	Yes	Yes	No
Total prize money	5 – 50 USD cash prize/ Day awarded to the top 10 algorithms (5 USD for 10th place, 50 USD for 1st place, by increments of 5 USD per place).	2.25 million USD investment/ Quarter awarded to the top three algorithms	/
Live trading	Yes	Yes	No for the free subscription. Yes for the paying subscriptions.
Markets	US Equities, US Futures	US Equites, US Futures	Free subscription: Equity, Forex, Contract for differencies (Futures contract with cash instead of physical goods), Cryptocurrencies. Paid subscriptions: the same as the free version plus Options and Futures.
Intellectual Property	Retained by the creator. But has to give Quantopian an exclusivity over the algorithm.	Retained completely by the creator	Retained completely by the creator.
Price	Free	Free	Free subscription 20.- USD / Month 250.- USD / Month
IDE	Browser based	Installations required	Browser based
Dataset	Morningstar	In house solution	QuantQuote

The table above shows the different options that each of these three solutions propose. Each of them has its strengths and weaknesses. It is up to the developer to decide which features are important and which are not.

Because of its subscription-based service, I did not want to start learning with QuantConnect. For more experienced users, this can be a good idea since it provides access to more markets than the other two services. Quantiacs has better prizes, but I am not entering any competitions for the Thesis. The other downside is that the setup is more complex since the other two services are browser based. However, this provides more flexibility since the developer is not locked to Quantiacs dataset, data analysis or code structure.
I decided to go with Quantopian since there is a big community and Python as a programming language, which has a substantial community to itself. To add to that, I am only interested in equities, since the other markets need more financial knowledge and I want to keep the financial part as simple as possible for neophytes.

2.7 Quantopian Research Notebook

The Quantopian notebook is a derived version of IPython. Project Jupyter with its Jupyter Notebook is also another derived application. These notebooks are documents that contain live code and are used to write/publish live documents and tutorials. The code is written in cells and can be executed within the document. The output: graphs, results, etc. are previewed in the document directly. To the contrary of a “basic” document, the data and results are not limited to its timeframe. A paper published in 2010 with access to an online dataset will only show results linked to the data leading up to 2010. A notebook’s results will be up to date as long as the dataset exists. Someone reading the notebook in 2018 will see the results linked to the data for 2018. Quantopian’s community writes and publishes a lot of tutorials for many different strategies. The cells help separate the code and make it more comprehensible: the developer can learn step by step. The code is then transcribed into a full-fledged algorithm, once the tutorial is understood.
To add to that, Quantopian’s notebook is often used to test strategies, parts of strategies or factors before implementing them in complete algorithms (IPython 2018; Quantopian 2018b.).

I used the notebook when learning how to write algorithms for trading and have tested parts of strategies with it. For my second Algorithm I will be using the notebook to test for various factors and find out if I can improve my first algorithm.

2.8 Alphalens

Alphalens is a Python package that is used to evaluate the relationship between factors and future returns. A factor is given, for instance: price to equity ratio, and a time-frame. The package will find a relationship between the evolution of this ratio and the returns for that time-frame. Based on that analysis, we can determine if that factor has good predictive information or not. This is determined by the value returned from the package: the higher the better. We use this analysis with various factors to compare them. Once good factors are found, it is possible to merge them together by adding their values and testing the predictive factors for that group. The objective is to find the best combination of factors. It is important to note that this package is used solely to find factors and not to test strategies. It is important to not use any strategy, because a good strategy can offset poor factors with good results. Developers use this package to gain time: it enables them to focus on creating strategies with good factors, instead of trying to create both at the same time. Once the factors are found, the strategy is then created.

Alphalens provides us with extensive charts and metrics to evaluate the risks and returns for our factor(s). I will be using four metrics to test numerous factors for my second algorithm. The number of metrics is the same as the number of factors used for the first algorithm, so as to keep comparisons fair. These metrics are:

- Annualized alpha.
- Beta.
- Information Coefficient mean.
- Information Coefficient standard deviation. (Alphalens 2018.)

3 Empirical part

This is a personal project that I undertook to create algorithms that are fully automated and trade without any human interaction. This paper aims to improve the readers theoretical and technological skills in finance and programming. The created algorithms and used strategies are basic and provide ground work for more complicated algorithms to be developed in the future. This product was not created for a commissioning party.

3.1 Product development process

I started the development process by following tutorials, written by the Quantopian community. The next step evolved around writing the first version of the first Algorithm. The idea was to create a simple stock selection strategy to ensure I could write algorithms without basing myself on those provided in the tutorials. Once competed, I wrote the second version, which uses a stock selection strategy that is more sophisticated than the previous version. The third and fourth versions were written after that and implement hedging strategies: sacrificing returns to have less risk. The development process then continued on to the second algorithm, and the Alphalens package was used with the python notebook to find factors that had a chance of beating the first algorithm's factors. Each version of both algorithms was immediately tested after being written, and any errors were corrected straight away. Once both algorithms were implemented, another set of tests were run and any required modifications were made. They were then benchmarked and compared to one another.

3.2 Product overview and choices

Each algorithm has a specific number of rules that define the strategy, for instance:

- How will the stocks be selected?
- What strategy will be implemented?
- How frequently is the portfolio rebalanced?
- Etc.

Once these rules are defined and the program is run, trading is done in a fully automated fashion without any human interaction. These algorithms demonstrate different strategies and how they are implemented with the use of an independent computerized workflow.

The first algorithm has four versions that trade using a “human strategy”. This means that the trades are not carried out on a sub minute level and the number of stocks in the portfolio is small enough for a person to manage.
For the second algorithm, the objective is to try to find a set of factors that select stocks which provide better returns than the first algorithms. Originally, Machine Learning was going to be used to determine the best factors. After looking at a few tutorials, the idea was abandoned for two reasons:

- The accuracy resulting from the main Machine Learning tutorial for Quantopian was 49% (Wiecki 2017.).
- A python package called Alphalens was created for that purpose.

The algorithms will be compared to the S&P500 stock index and to each other. The returns and risks will be accounted for in determining which algorithm is better. The tools used are Python for the programming language and Quantopian for the IDE, data recuperation, analysis, etc.

The algorithms will only be bactkested for the thesis. The reason is that there is not enough time to do forward testing. This testing scenario might seem incomplete, since it is only using known data. But no knowledge of past trends or crises are used in the strategies. Of course, any serious testing should include forward testing before trading with real money.

3.3 Quantopian code structure & screen representation

Quantopian has a well-defined code structure and regroups all important information into one screen. This gives developers the basic methods that are needed to write algorithms and displays all additional important information at the same place. For neophytes, this is a good starting point since the logical structure is provided and gives a clear indication of where to start. When debugging or testing new strategies, time is gained since there is no need to change screen to look at the results, logs or errors. The trade-off is that the structure is a bit rigid, so more experienced developers might want to look into building their own structures in their IDE.

Quantopian’s algorithm development screen is divided into three parts: the code, backtests and logs and runtime errors. Quantopian provides a structure for the algorithm that includes six methods, an object structure and imports. Some methods are not required to run the algorithm, while others are essential. The developers are free to then add their own methods if needed. In the following chapters, the structure is divided into

two parts: compulsory (Quantopian compulsory elements: 3.3.1) and facultative (Quantopian optional elements: 3.3.2). The first resumes the items needed for the program to run, while the second contains those that are not necessarily required. The latter can be commented or deleted without affecting the functioning of the algorithm. Deleting the unused methods helps clean the code and only keep the used elements.

The picture below shows the development screen. The centre part contains the code with a save button and build algorithm button. To the top right, the screen contains the backtesting zone, and to the bottom left the logs, compiling and runtime error messages. These elements are developed in the section after the compulsory and optional elements (Backtesting, logs and outputs: 3.3.3).

Figure 10: Quantopian full screen

3.3.1 Quantopian compulsory elements

This section contains the imports, object structure, initialize and make_pipeline methods.

Importing is the process in which a module gains access to the code contained in another module. Quantopian's algorithms start off with four basic imports that are represented in the picture below. The first import gives access to a module that contains general purpose code. This code contains methods linked to the Pipeline, calendar, portfolio calculations, etc. The second one provides access to the Pipeline. This is a very important part of the algorithm building process and will be explained in detail below in the make_pipeline method. To keep it summarized, it enables the developer to query and compute large datasets. The third one links to a dataset. A common misunderstanding is that a dataset contains actual data. Datasets contain collections of objects that tell where and how the Pipeline API can find the inputs to compute. These datasets often conform to database tables. Due to this particularity, the dataset's attributes are often referred to as columns. The last one provides access to filters. These filters enable us to choose assets based on certain conditions. The most known ones are the Q500US and the Q1500US. The first one selects the 500 most liquid US stocks and the second one selects the 1500 most liquid US stocks traded at that moment. The code snippet below shows the four imports.

```
import quantopian.algorithm as algo
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.filters import Q1500US
```

Quantopian has two objects that most of its six basic methods use. If more methods are created by the developer, it is highly probable that these methods will use one or both of these objects. Context is an augmented Python dictionary that is used to maintain the state of the program. The state contains variables and objects that can be accessed from any method that has the context passed as a parameter. Data is used to get discrete prices or windows of prices (several days for example) for stocks and futures. It is important to note that using global methods is not recommended and is a bad practice. Accessing these variables should be done by using the context object.

The figure below shows the initialize method which is, according to the comment in the method, "Called once at the start of the algorithm". This is where we do all the setup for the rest of the program. We will define all global variables through the context object and initialize the program's state. Scheduling is used in initialize to call methods based on a recurring date and time. The make_pipeline method is called, initializing the pipeline that

will be used throughout the program. The following code snippet shows the initialize method with the scheduling functions and the call to the make_pipeline method. The context object is passed in parameter and can be passed to any method from here.

```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    # Rebalance every day, 1 hour after market open.
    algo.schedule_function(
        rebalance,
        algo.date_rules.every_day(),
        algo.time_rules.market_open(hours=1),
    )

    # Record tracking variables at the end of each day.
    algo.schedule_function(
        record_vars,
        algo.date_rules.every_day(),
        algo.time_rules.market_close(),
    )

    # Create our dynamic stock selector.
    algo.attach_pipeline(make_pipeline(), 'pipeline')
```

Figure 11: Initialize method

The make_pipline method featured in the figure below creates the pipeline, which is at the heart of every algorithm. Many algorithms use a method of calculation called: Cross-sectional trailing-window computation, and the pipeline makes it easy to define and execute them. These calculations follow a similar pattern: fetch the last N days from a data source for a large amount of assets and apply a reduction function to produce a single value per asset. A simple example of that type of calculation is: retrieve the assets' closing prices for the two last days and calculate the average/median/percentage change between the two prices. In this function we declare our base universe which is the stocks or futures we are going to be analysing. We define columns for our pipeline: this is the data that must be extracted for each asset. The function returns the constructed pipeline. In my algorithms, the pipeline contains the previous day's closing price for the 1500 US stocks with highest liquidity. The other methods' logic will use the pipeline, hence making it a central part of the process.

```
def make_pipeline():
    """
    A function to create our dynamic stock selector (pipeline).
Documentation
    on pipeline can be found here:
    https://www.quantopian.com/help#pipeline-title
    """

    # Base universe set to the Q1500US
    base_universe = Q1500US()

    # Factor of yesterday's close price.
    yesterday_close = USEquityPricing.close.latest

    pipe = Pipeline(
        screen=base_universe,
        columns={
            'close': yesterday_close,
        }
    )
    return pipe
```

Figure 12: Make_pipeline method

3.3.2 Quantopian optional elements

This section contains: before_trading_start, rebalance, record_vars and handle_data methods.

Before_trading_start is an optional method. There is no need to implement it and it can just be deleted if not used. It is called once a day before the market opens, and orders (buying or selling stocks) cannot be placed. The point of this method is to create a set of securities, which can then be used by other methods. The method recuperates the data from the pipeline, and then puts it in a list. This list is declared in the context, giving access to it from any other method using the context object. The following code snippet shows the before_trading_start method, which uses the two objects: context and data.

```
def before_trading_start(context, data):
    """
    Called every day before market open.
    """
    context.output = algo.pipeline_output('pipeline')

    # These are the securities that we are interested in trading each day.
    context.security_list = context.output.index
```

Figure 13: Before_trading_start method

Rebalance is a recurring method based on a specific timeframe. It is scheduled in the initialize method and is optional, since the recurring call can be deleted. This method is used when the trader wants to define a weight for certain assets. For example: The trader has two stocks and wants them to each have 50% of the portfolio's value. After one year, one stock has increased by 100%, doubling its value while the other has decreased by 50%, halving its value. The rebalance method will sell enough of the first stock and buy a sufficient amount of the second stock so that they are back to 50% each. Rebalancing can be done weekly, monthly, yearly or any other timeframe. The more frequent the better as it flattens out volatility. However, each transaction has fees, and doing it too often can highly reduce the portfolio's profit.

Record-vars is a recurring method, like the rebalance method. The timeframe of the recurring call is defined by the scheduling function declared in the initialize method. This method is also optional since it is possible to delete the scheduling declaration. The method records variables and then prints them on a graph. This is an important feature that helps in the analysing process after the algorithm is run. Example of variables: stock or futures prices, index value, leverage: defines the amount of money borrowed for an investment, and any other variable that can be plotted.

Handle_data is an optional method that is called every minute. The call for this method is done "behind the scenes" by Quantopian. It is a good example of methods and calls defined in an all in one solution, removing the hassle for the developer to have to take care of these aspects. The handle_data method is used for any sort of logic that requires minute by minute actions. This can be trading or recording variables that need a shorter timeframe than the record_vars method. Calling a method every minute has a drastic impact on performance. This method should only be used if it is necessary to do actions every minute. It is crucial that any operations not requiring actions every minute are de-

clared in another function. If no actions require such a short timeframe, it is better to delete this method and implement another method using the scheduler. Quantopian's documentation specifically states that handle_data should rarely be used, and scheduled functions are preferred. (Quantopian, 2018b.)

The code snippet below shows the three methods above: rebalance, record_vars and handle_data. The reason these methods are grouped together is that they do not contain any logic, only the structure for the developer to take care of the implementation.

```
def rebalance(context, data):
    """
    Execute orders according to our schedule_function() timing.
    """
    pass


def record_vars(context, data):
    """
    Plot variables at the end of each day.
    """
    pass


def handle_data(context, data):
    """
    Called every minute.
    """
    pass
```

Figure 14: Rebalance, record_vars and handle_data methods

3.3.3 Backtesting, logs and outputs

When testing, it is possible to either build the algorithm or run a full backtest. The first stays on the same screen as the code. The view of the results and metrics is summarized, and the developer can see the logs and outputs. Running a full backtest redirects to another screen which has more detailed information concerning the returns and metrics but does not provide any logs. It will display errors if any are encountered. These options are detailed below.

The figure below shows that this part of the screen is empty before doing a backtest,. The developer can choose the starting and finishing dates, the amount of money and the type of asset: US Equities or US Futures. The basic metrics are displayed with no values.

Figure 15: Before the backtest

The “partial” backtest is run when the “build algorithm” button (see Figure 10: Quantopian full screen) is clicked. Once finished, the same part of the screen than before is now filled with information. This is usefull when starting the testing process, since it gives basic information and does not load another window. The metrics are filled with information, the performance of the algorithm is graphed and the variables that are recorded appear in another graph below the performance one.

To the right of the screen a zone contains the output logs that are written in the code to help debug or show variables’ attributes while the program is running. The logs also show any compiling and runtime errors. It also shows the variables that are declared but never used. The figure below shows the logs and errors part of the screen. No logs are present since it is only the default algorithm which contains no trading or records of variables.

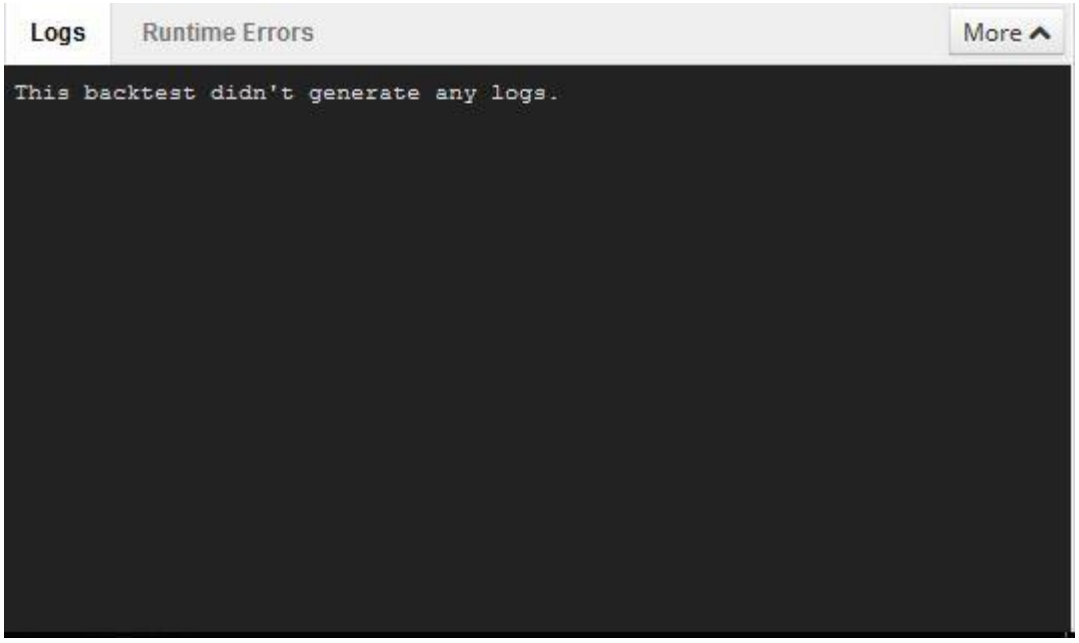

Figure 16: Logs and runtime errors

The picture below shows a backtest of the default algorithm. It does not contain any trading: represented by the blue line, since its purpose is to provide the structure. The red line represents the default benchmark which is a financial index.

Figure 17: After the backtest

Once the algorithm has been written and a few partial backtests have been completed, a full one is run. This is done by clicking the “Run Full Backtest” button. This loads a new window with the backtest taking the whole screen. There are a lot more metrics in this page and it enables developers to fine-tune the algorithm.

Figure 18: Full backtest window

3.4 Algorithms & comparison

Two algorithms were created. The first algorithm replicates a strategy that is feasible by a human being. It is implemented in four versions: version 0 is the basic implementation, and each next version tries to improve the previous one with different strategies. The second algorithm uses a python package called Alphalens in an attempt to find better factors and use these to improve the first algorithm's version that has the best returns. Each algorithm trades with 80'000.- USD. In 2013, the median for Finish households' net wealth was 110'000.- Euros (Statistics Finland 2015.). Implementing strategies with 30'000.- EUR below median represents middle class citizens salaries. This shows how they can improve their savings and earnings through investing in the stock market. The algorithms are stock-generic, so they will work with stocks from any country. For this paper, the algorithms only trade US stocks since that is what Quantopian provides. To add to that, the strategies are specifically written for amateur traders to follow. This means that it is not required to follow the stock market every day or spend a lot of time to use these strategies. The trading period ranges from the 1st of January 2004 to the 1st of January 2018. This period shows a long-term investment strategy: more than ten years and includes a major crisis: the subprime crisis in 2007 – 2008. The comparison will be done using the main metrics from the full backtest:

- Returns.
- Benchmark returns.
- (Alpha).
- Beta.
- Sharpe.
- Sortino.
- Volatility.
- Max Drawdown.

The metrics are divided into two groups: returns and risks. The values we are looking for are detailed in the first algorithm's comparison part (Algorithm 1 comparison: 3.6). Even though alpha is part of the metrics, its numbers do not match the difference in returns between the algorithm and the benchmark. For this reason, it is not included it in the tests.

3.5 Algorithm 1

This algorithm buys two shares for each of the eleven sectors. Each month, it proceeds to buy the shares that have lost value and sell the shares that have gained value. This process is called rebalancing. The aim is to have each share representing the same weight in

the portfolio. If a company's share price increases by a factor of 10, it will have a considerable impact on the portfolio. Maintaining the same weight for each asset prevents the portfolio's value from going up or down due to one share only.

The shares are selected based on four factors that have arbitrarily been chosen since these are the metrics I use personally. For each stock, the four factors' values are added together, and then sorted from highest to lowest. The top two shares for each sector are bought. The fours factors are detailed below (Algorithm 1 – Version 0: 3.5.1).

This algorithm has four versions. Each version implements the same strategy with different variants. When discussing each version, I will detail the specificities of each one. The results of each algorithm will be used to compare to each other.

Each version uses the same structure and only a few methods change from one version to another. The complete code will be included for the first version of the algorithm. All the other versions will only include the modified methods. Each method will be discussed from a theoretical point of view. The technical part is provided by the inclusion of the code, and it is commented to help the reader understand it.

3.5.1 Algorithm 1 – Version 0

I decided to start the version numbering by 0. For this strategy, I only merge the factors' values and pick the two stocks with the highest values for each sector. These factors are:

- Basic earnings per share (eps): a rough calculation of the company's profit for one share. (Banks 2010, 173; Bannock & Manser 2003, 80.)
- Price to equity (pe) ratio: A measurement used to determine the value of a company based on the basic eps and the stock price. A ratio of 25 indicates that an investor is paying 25.- USD for 1.- USD of returns. (Banks 2010, 400; Bannock & Manser 2003, 213.)
- Price to book (pb) ratio: Compares the company's book value to its share price. (Banks 2010, 399.)
- Total debt to equity ratio: measures how much debt a company, relative to the shrare price, is using to finance its assets. (Peavler 2018.)

```
from quantopian.algorithm import attach_pipeline, pipeline_output
from quantopian.pipeline import Pipeline
from quantopian.pipeline.data.builtin import USEquityPricing
from quantopian.pipeline.filters import Q1500US
from quantopian.pipeline.data import Fundamentals
import math
import numpy as np
```

Figure 19: Imports

The figure above contains the four basic imports. To add to these four, the Fundamentals provides methods that recuperate the data for the four factors listed above. The math import provides us with complex mathematical functions and numpy facilitates querying matrices.

```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    # boolean that tells us if the portfolio has been created or  not
    context.portfolio_created = False
    context.stocks_in_portfolio = []
    context.sectors = []

    # Rebalance every month, 1 hour after market open.
    schedule_function(rebalance, date_rules.month_end(),
time_rules.market_open(hours=1))

    # Record tracking variables at the end of each day.
    schedule_function(my_record_vars, date_rules.every_day(),
time_rules.market_close())

    # Creates the portfolio
    schedule_function(my_define_portfolio, date_rules.month_start(),
time_rules.market_open(hours=1))

    # Create our dynamic stock selector.
    attach_pipeline(make_pipeline(context), 'my_pipeline')
```

Figure 20: Initialize method

The code snippet above contains all the global variables created with the use of context object. The scheduled functions and the call to make_pipeline are also declared in this method. All these elements are crucial for the rest of the algorithm.

```
def make_pipeline(context):
    """
    A function to create our dynamic stock selector (pipeline).
Documentation on
    pipeline can be found here: https://www.quantopian.com/help#pipeline-
title
    """
    # columns
    basic_eps = Fundamentals.basic_eps_earnings_reports.latest
    pe_ratio = Fundamentals.pe_ratio.latest
    pb_ratio = Fundamentals.pb_ratio.latest
    total_debt_equity_ratio = Fundamentals.total_debt_equity_ratio.latest
    sector = Fundamentals.morningstar_sector_code.latest

    # universe definition
    universe = Q1500US()

    # rank factors
    basic_eps = basic_eps.rank(mask=universe, method="average")
    pe_ratio = pe_ratio.rank(mask=universe, method="average")
    pb_ratio = pb_ratio.rank(mask=universe, method="average")
    total_debt_equity_ratio = total_debt_equity_ratio.rank(mask=universe,
method="average")

    # adding factors
    factor = basic_eps + pe_ratio + total_debt_equity_ratio + pb_ratio

    pipe = Pipeline(
        columns={
            'factor':factor,
            'sector': sector
        },
        screen = universe)

    return pipe
```

Figure 21: Make_pipeline method

In this method, the sum of the factors is used to define the pipeline. This method returns a pipeline containing a list of stocks with two columns: the sum of the factors and the sectors.

```
def my_define_portfolio(context, data):
    # if true, the portfolio has been created, so we don't re-create it.
    if context.portfolio_created == False:
        sectors = define_sectors_securies(context, data)
        define_portfolio(context, data, sectors)
        buy_stocks(context, data)
        context.portfolio_created = True
```

Figure 22: My_define_portfolio method

This method is declared by the scheduler, but the first condition is only met once: at the beginning of the execution of the program. This means that all other calls to this method do nothing. The reason for a useless recurring call is that no concrete procedure exists to call a method only once. To add to that, this method's logic cannot be implemented in the initialize method, since the pipeline must be built first, and Quantopian's logic does not permit to use the pipeline results in the initialize method.

```
def define_sectors_securities(context, data):
    #recuperate the pipeline
    context.output = pipeline_output('my_pipeline')

    # These are the securities that we are interested in trading each day.
    context.security_list = context.output.fillna(0)

    # These are the sectors. We will be buying 2 shares for each sector.
    sectors = context.security_list['sector'].unique()
    sectors.sort()
    return sectors
```

Figure 23: Define_sectors_securities method

In the figure above, the method recuperates the sectors from the pipeline with the use of the Fundamentals import. The sectors are then sorted by numerical value (101, 102, …) and returned without duplicates.

```
def define_portfolio(context, data, sectors):
    for sector in sectors:
        context.sectors.append(sector)

        #select the current sector
        current_sector =
context.security_list[context.security_list['sector'] == sector]

        #sort the stocks in the sector by factor in descending order
        factor = current_sector.sort_values('factor', ascending=False)

        # add stocks to be bought
        stocks = factor.head(n=2)
        context.stocks_in_portfolio.extend(stocks.index)
```

Figure 24: Define_portfolio method

The method above defines which stocks will be bought by the algorithm. For each sector, the stocks are sorted by their factor's value, from highest to lowest. The top two listed stocks are added to the global object: context.stocks_in_portfolio.

```
def buy_stocks(context, data):
    # Retreive all stocks that have been ordered, but not bought yet.
    open_orders = get_open_orders()
    stocks = context.stocks_in_portfolio
    for stock in stocks:
        # Prevents ordering stocks that have already been ordered.
        if stock not in open_orders:
            order_target_percent(stock, 0.045)
```

Figure 25: Buy_stocks method

In this figure, the method buys the stocks that have been defined in the global object: context.stocks_in_portfolio. The method tests to see if the stocks have already been ordered. If the orders have not yet been fulfilled and no check is done, the algorithm will place the orders to acquire the same amount again, resulting in a purchase of at least two times the amount of stocks required. This will result in some stocks having a higher weight in the portfolio and impacting its total value in a greater way. This also means that the algorithm will borrow money if it does not have enough, resulting in leverage. This exposes the trader to risks linked to credits and also nullifies the benchmarks. The targeted value for each stock is 4.5% (0.045). This means the algorithm will use 99% (22 * 4.5) of its available cash to buy the 22 stocks.

```
def rebalance(context, data):
    # Rebalance the stocks so that they all have the same weight in the
portfolio.
    stocks = context.stocks_in_portfolio
    portfolio_value = context.portfolio.portfolio_value
    target_stock_value = portfolio_value / len(stocks)
    for i in range(len(stocks)):
        stock = stocks[i]
        # Checks if the stock is still listed. If not, another one must be
bought.
        if not check_stock_is_listed(context, data, stock):
            swap_delisted_stock(context, data, stock, i)
        rebalance_stock(context, data, stock, target_stock_value)
```

Figure 26: Rebalance

The rebalance method is called once at the end of each month. It will recuperate the total value of the portfolio and divide it by the number of stocks in order to find the target value each position should have. It then buys the stocks that have had a decrease in price and sell those that have had an increase in price so that their values reach the target. This method also calls the check_stock_is_listed method to ensure the stocks are still trading. If the called method returns false, a call to another method called swap_delisted_stock is made. The method finishes by calling the rebalance_stock method to do the actual buying and selling.

```
def check_stock_is_listed(context, data, stock):
    # Returns true if the stock is still listed, and  false if the stock is
not listed anymore.
    stock_can_trade = data.can_trade(stock)
    if not stock_can_trade:
        return False
    return True
```

Figure 27: Check_stock_is_listed method

It is possible that stocks are not listed anymore. This can be due to companies buying back all their shares, being completely bought over by another company or going bankrupt. This method is called by the rebalance method and checks if the stock passed in parameters is still listed. The method will return true if it is the case and false if the stock is not listed anymore. This prevents our algorithm from having a decreased number of stocks. This method and the swap_delisted_stock (see below) were created after realising

that the algorithm always finished with 14 stocks instead of 22 for the testing period. Without these methods, the program is not fully autonomous since it can finish by having 0 stocks in the portfolio.

```
def swap_delisted_stock(context, data, stock, i):
    '''
    Uses the same strategy as the buy_stocks method.
    Finds a stock to replace the stock that has been delsited.
    '''
    sector_index = i/2
    sector_number = context.sectors[sector_index]

     #recuperate the pipeline
    context.output = pipeline_output('my_pipeline')

    # These are the securities that we are interested in trading each day.
    context.security_list = context.output.fillna(0)

    for stock in context.stocks_in_portfolio:
        if stock in context.security_list.index:
            context.security_list =
context.security_list[context.security_list.index != stock]

    #select the current sector
    current_sector = context.security_list[context.security_list['sector']
== sector_number]

    #sort the stocks in the sector by eps in descending order
    factor = current_sector.sort_values('factor', ascending=False)

    stocks_to_buy = []
    stock_to_buy = factor.head(n=1)
    stocks_to_buy.extend(stock_to_buy.index)
    context.stocks_in_portfolio[i] = stocks_to_buy[0]
    del stocks_to_buy[0]
```

Figure 28: Swap_delisted_stock method

This method uses the same logic as the define_portfolio method, with one difference: it only goes through the process of selecting one stock, while the define_portfolio method selects 22 stocks. Once the selection is done, it changes the global object context.stocks_in_portfolio. This object is an array and the method modifies the cell containing the delisted stock with the newly selected stock. The algorithm proceeds to buy that stock in the rebalance_stock method.

```
def rebalance_stock(context, data, stock, target_stock_value):
    # Proceeds to buy or sell the correct amount of a certain share to
rebalance the portfolio.
    open_orders = get_open_orders()
    if stock not in open_orders:
        order_target_value(stock, target_stock_value)
```

Figure 29: Rebalance_stock

In this figure, the method proceeds to buy the stock and its amount defined in the re-balance method. It also checks that the stocks are not already ordered, exactly like the buy_stocks method.

```
def my_record_vars(context, data):
    """
    Plot variables at the end of each day.
    """
    record(leverage = context.account.leverage)
    record(number_to_buy = len(context.stocks_in_portfolio))
    record(number_shares= len(context.portfolio.positions))
```

Figure 30: My_record_vars method

This method is called every day and plots three variables: leverage, number of stocks to buy and number of stocks in the portfolio onto a graph. This is used to check if the algo-rithm is following the requirements given by the developer.

I am not expecting anything from this version, since it is only setting a benchmark for the next versions. This shows how a raw algorithm performs, with a basic stock picking strat-egy.

The image below shows the results from the full backtest, which contains the returns, risks, leverage, number of shares to hold and number of shares held. The objective is to keep leverage as close as possible to 1. Under that value means that the algorithm is not using all the available cash in the portfolio and above means it is borrowing money. These situations would offset the results and nullify the comparisons. The two other plotted varia-bles must have the same values to ensure the algorithm has the correct number of shares in the portfolio. As mentionned before, companies can be removed from the stock market. In this case, the algorithm must look for other stocks to replace the delisted ones.

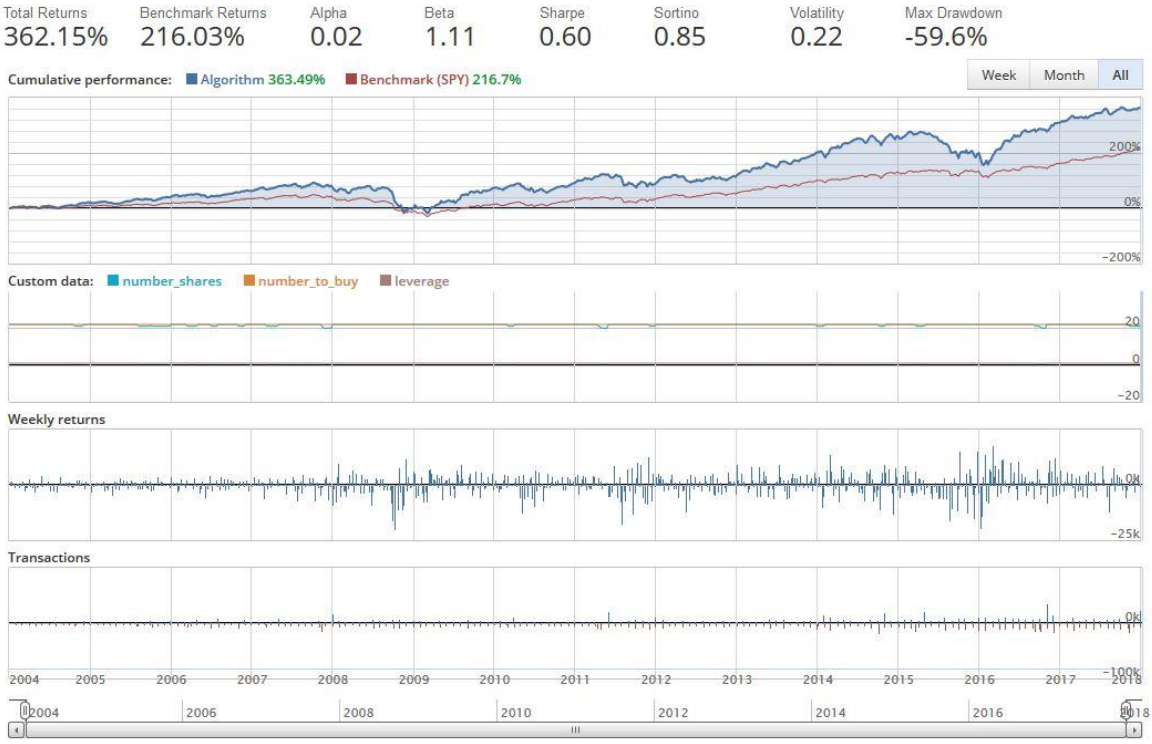

Figure 31: Algorithm 1 Version 0 results

3.5.2 Algorithm 1 – Version 1

Version 1 uses the same four factors as version 0, but instead of merging them, it uses each factor individually to narrow down the number of stocks. For instance, the algorithm looks for companies with the highest eps, then companies with a pe_ratio that is lower than 25, etc. It will continue this process for the four factors to narrow down the number of stocks. This is a stock picking strategy that I decided to implement because it represents how I invest “manually”, i.e. without using algorithmic trading.

```
def make_pipeline(context):
    """
    A function to create our dynamic stock selector (pipeline).
Documentation on
    pipeline can be found here: https://www.quantopian.com/help#pipeline-
title
    """
    #context.sector_mappings

    # factors
    basic_eps = Fundamentals.basic_eps_earnings_reports.latest
    pe_ratio = Fundamentals.pe_ratio.latest
    pb_ratio = Fundamentals.pb_ratio.latest
    total_debt_equity_ratio = Fundamentals.total_debt_equity_ratio.latest
    sector = Fundamentals.morningstar_sector_code.latest

    # universe definition
    universe = Q1500US()

    pipe = Pipeline(
        columns={
            'basic_eps':basic_eps,
            'pe_ratio':pe_ratio,
            'pb_ratio':pb_ratio,
            'debt_equity': total_debt_equity_ratio,
            'sector': sector
        },
        screen = universe)

    return pipe
```

Figure 32: Algorithm 1 Version 1 make_pipeline method

Like Versions 0, this method retrieves the required factors with the use of the Fundamentals package. The universe is then defined, and the pipeline's columns are specified based on the aforementioned factors. The key difference from Version 0 is that there are four distinct factors instead of one that is a combination of the four. Although the process is different, the factors stay the same between the two versions. The method finishes by returning the pipeline.

```
def define_portfolio(context, data, sectors):
    for sector in sectors:
        context.sectors.append(sector)
        two_securities = False
        # Initialise dataframes, so that they are not null when tested for
length
        pb_filter = context.security_list[context.security_list['sector']
== sector]
        debt_equity_filter =
context.security_list[context.security_list['sector'] == sector]

        #select the current sector
        current_sector =
context.security_list[context.security_list['sector'] == sector]

        #sort the stocks in the sector by eps in descending order
        stock_by_eps = current_sector.sort_values('basic_eps',
ascending=False)

        pe_filter = stock_by_eps[stock_by_eps['pe_ratio'] < 25]
```

Figure 33: Algorithm 1 Version 1 define_portfolio method part 1

The define_portoflio method is defines the portfolio. It is divided into two figures due to its size. The difference to the previous version is that the stocks go through a specific selection process. In the first figure, the method goes through all sectors. For each loop, the Boolean variable two_securities is put to False. This variable is used in the tests in the second part of the method. The algorithm then initialises the data frames that it's going to use. There is one data frame for each factor. It will then select all the stocks in the list that are part of the current sector and sort them by eps from highest to lowest. The last line of code in the method does another selection for all stocks that have a pe ratio under 25. Those that have values above are discarded.

```
        # filter pe
        if len(pe_filter.index) > 2 and two_securities == False:
            pb_filter = pe_filter[pe_filter['pb_ratio'] <= 1]
        else:
            two_securities = True
            #add securities to context.stocks_to_buy
            stocks = stock_by_eps.head(n=2)
            context.stocks_in_portfolio.extend(stocks.index)

        # filter pb
        if len(pb_filter.index) > 2 and two_securities == False:
            debt_equity_filter = pb_filter[pb_filter['debt_equity'] <= 2]
        elif two_securities == False:
            two_securities = True
            #add securities to context.stocks_to_buy
            stocks = pe_filter.head(n=2)
            context.stocks_in_portfolio.extend(stocks.index)

        # filter debt_equity
        if len(debt_equity_filter.index) >= 2 and two_securities == False:
            #add securities to context.stockt_to_buy
            stocks = debt_equity_filter.head(n=2)
            context.stocks_in_portfolio.extend(stocks.index)
            #print(debt_equity_filter.head(n=2))
        elif two_securities == False:
            two_securities = True
            #add securities to context.stockt_to_buy
            stocks = pb_filter.head(n=2)
            context.stocks_in_portfolio.extend(stocks.index)
```

Figure 34: Algorithm 1 Version 1 define_portfolio method part 2

This figure shows the rest of the define_portfolio method. Each factor has a condition attached to it. For example: the pe ratio (mentioned above) must be smaller than 25. This is an arbitrary solution and represents strategy decisions based on fundamental analysis. Each condition narrows down the number of stocks and puts them into a new dataframe. The conditional test will also check that the last data frame created has at least two stocks. If this condition is met, the filtering process continues. If not, the top two stocks of the previous data frame are added to the list of shares to buy. Once two stocks are bought, the Boolean variable two_securities is True. This indicates that there is no need to look for other stocks in the current sector. As soon as the algorithm passes on to the next sector, the variable is turned back to False, as seen in the first figure.

```
def swap_delisted_stock(context, data, stock, i):
    sector_index = i/2
    sector_number = context.sectors[sector_index]

     #recuperate the pipeline
    context.output = pipeline_output('my_pipeline')

    # These are the securities that we are interested in trading each day.
    context.security_list = context.output.fillna(0)

    for stock in context.stocks_in_portfolio:
        if stock in context.security_list.index:
            context.security_list =
context.security_list[context.security_list.index != stock]

    # Initialise dataframes, so that they are not null when tested for
length
    pb_filter = context.security_list[context.security_list['sector'] ==
sector_number]
    peg_filter = context.security_list[context.security_list['sector'] ==
sector_number]
    debt_equity_filter =
context.security_list[context.security_list['sector'] == sector_number]

    #select the current sector
    current_sector = context.security_list[context.security_list['sector']
== sector_number]

    #sort the stocks in the sector by eps in descending order
    stock_by_eps = current_sector.sort_values('basic_eps', ascending=False)

    pe_filter = stock_by_eps[stock_by_eps['pe_ratio'] < 25]

    stocks_to_buy = []
```

Figure 35: Algorithm 1 Version 1 swap_delisted_stock part 1

This method swaps the delisted stocks. It uses the same filtering process as the define_portfolio method. The key difference is that it does it for only one stock. The sector_index variable is equal to i / 2, where i is the index (cell) at which the stock is present in the context.stocks_in_portfolio array. Since there are 11 sectors and 22 stocks, by dividing the index by two, the algorithm will always end on a number between 0 and 10. Because the sector and the stocks related to that sector are added in the same order, it is possible to use this link to find the corresponding index in the sector array. The algorithm is also working with Integers: natural numbers, and any number that is not whole is

rounded towards the bottom. 9 / 2 will give 4.5 which rounds to 4. The first appendix: Appendix 1. Sectors and stocks arrays displays a paragraph and table explaining in more detail the link between the two tables.

```
    # filter pe
    if len(pe_filter.index) > 1:
        pb_filter = pe_filter[pe_filter['pb_ratio'] <= 1]
    else:
        #add securities to context.stocks_to_buy
        stock = stock_by_eps.head(n=1)
        stocks_to_buy.extend(stock.index)
        context.stocks_in_portfolio[i] = stocks_to_buy[0]

    # filter pb
    if len(pb_filter.index) > 1:
        debt_equity_filter = pb_filter[pb_filter['debt_equity'] <= 2]
    else:
        #add securities to context.stocks_to_buy
        stock = pe_filter.head(n=1)
        stocks_to_buy.extend(stock.index)
        context.stocks_in_portfolio[i] = stocks_to_buy[0]

    # filter debt_equity
    if len(debt_equity_filter.index) >= 1:
        #add securities to context.stockt_to_buy
        stock = debt_equity_filter.head(n=1)
        stocks_to_buy.extend(stock.index)
        context.stocks_in_portfolio[i] = stocks_to_buy[0]
    else:
        #add securities to context.stockt_to_buy
        stock = pb_filter.head(n=1)
        stocks_to_buy.extend(stock.index)
        context.stocks_in_portfolio[i] = stocks_to_buy[0]

    del stocks_to_buy[0]
```

Figure 36: Algorithm 1 Version 1 swap_delisted_stock part 2

The second part of this method contains the filtering process used in the define_portfolio method. After the method has found the stock to buy, it returns it.

I am hoping to beat the benchmark set in the previous version. I expect performance to be higher and risk to be lower since I am actively trying to pick the best stocks.

The picture below shows the results for this version. The ploted values are the same as for the other version. We can see that the risk metrics and returns are worse than version 0, which is suprising. These results are discussed in detail in the first algorithm comparison chapter (Algorithm 1 comparison: 3.6).

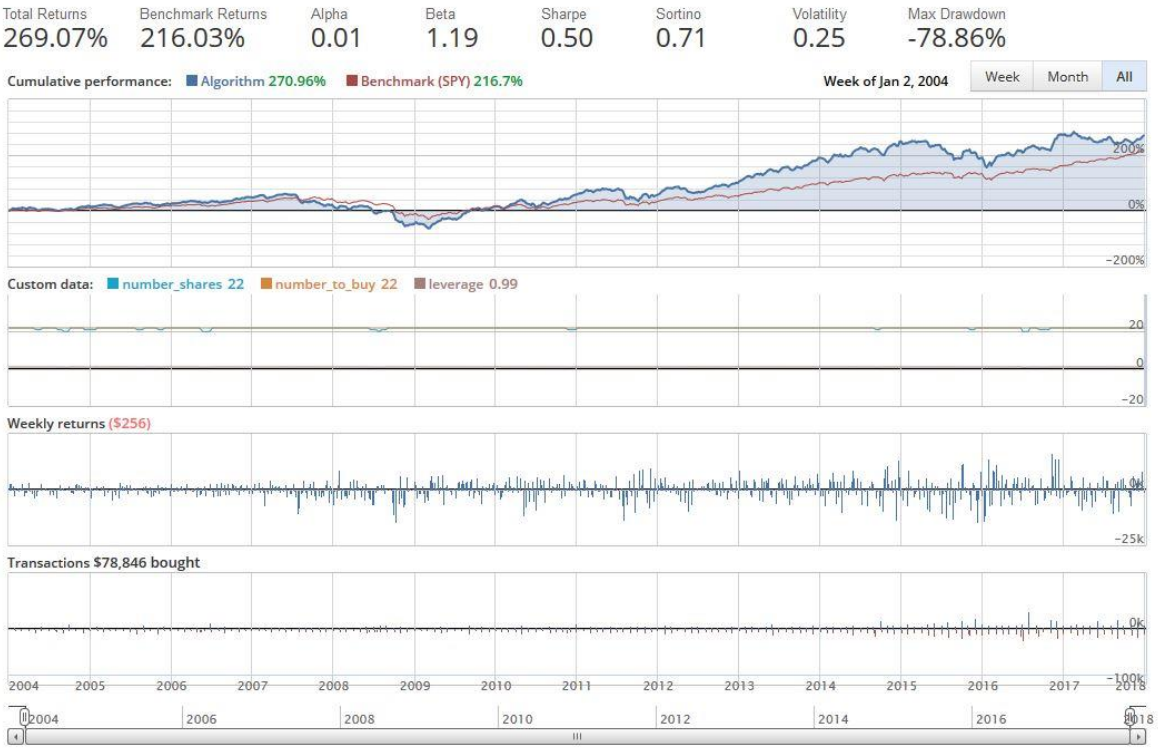

Figure 37: Algorithm 1 Version 1

3.5.3 Algorithm 1 – Version 2

Version 2 uses the selection principle seen in version 1 and shorts half of the portfolio's value when the index falls. Since the two first algorithms are highly correlated to the S&P500, it is a good idea to try a strategy that should prevent the portfolio's value from falling as much. This principle is a form of hedging: investing in an asset that is negatively correlated to reduce risk and volatility in the portfolio. This strategy uses simple moving averages (sma): indicator used in technical analysis to determine when to enter or exit trades (Banks 2010, 338; Bannock & Manser 2003, 182.). The sma determines when to short the S&P500 and go long the rest of the time.

```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    # boolean that tells us if the portfolio has been created or  not
    context.portfolio_created = False
    context.stocks_in_portfolio = []
    context.sectors = []
    context.etf_sp500 = sid(8554)

    # Define portfolio once at the start of the program
    schedule_function(my_define_portfolio, date_rules.month_start(),
time_rules.market_open(hours=1))

    # Rebalance every day, 1 hour after market open.
    #schedule_function(my_rebalance, date_rules.month_start(),
time_rules.market_open(hours=1))
    schedule_function(rebalance, date_rules.month_end(),
time_rules.market_open(hours=1))

    # Record tracking variables at the end of each day.
    schedule_function(my_record_vars, date_rules.every_day(),
time_rules.market_close())

    # short S&P500 depending on Simple Moving Averages
    schedule_function(long_short_sp500, date_rules.every_day(),
time_rules.market_open(hours=2))

    # Create our dynamic stock selector.
    attach_pipeline(make_pipeline(context), 'my_pipeline')
```

Figure 38: Algorithm 1 Version 2 initialize method

This method is very similar to the initialize methods in the other versions. The only differences are that the algorithm declares a new global variable: context.etf_sp500 and schedules a new method named long_short_sp500 which is called every day. The variable retrieves the ETF that our algorithm will long and short depending on the simple moving averages. The scheduled method implements the logic our algorithm will use to go long and short.

```
def buy_stocks(context, data):
    open_orders = get_open_orders()
    stocks = context.stocks_in_portfolio
    for stock in stocks:
        if stock not in open_orders:
            order_target_percent(stock, 0.022)
    if context.etf_sp500 not in open_orders:
        order_target_percent(context.etf_sp500, -0.5)
```

Figure 39: Algorithm 1 Version 2 buy_stocks method

This method follows the same logic as seen previously. The only exception is that it also shorts the S&P500. The target for the normal stocks are 2.2% (0.022) of the portfolio. This adds up to 48.4% (22 stocks * 2.2). The other 50% are ordered by the short position: -0.5. The minus in front of the targeted values means that the algorithm will short instead of go long.

```
def rebalance(context, data):
    stocks = context.stocks_in_portfolio
    portfolio_value = context.portfolio.portfolio_value
    target_stock_value = (portfolio_value / len(stocks)) / 2
    for i in range(len(stocks)):
        stock = stocks[i]
        if not check_stock_is_listed(context, data, stock):
            swap_delisted_stock(context, data, stock, i)
        rebalance_stock(context, data, stock, target_stock_value)
    order_target_value(context.etf_sp500, -(portfolio_value/2))
```

Figure 40: Algorithm 1 Version 2 rebalance method

This method is the same as the other versions. The separating element is that instead of calculating the target_stock_value on the whole value of the portfolio, only half of the portfolio is used. This leaves the other half of the available money for the algorithm to short or long of the S&P500.

```
def long_short_sp500(context, data):
    hist = data.history(context.etf_sp500, 'price', 50, '1d')
    sma_50 = hist.mean()
    sma_20 = hist[-20:].mean()
    open_orders = get_open_orders()
    if sma_20 > sma_50:
        if context.etf_sp500 not in open_orders:
            #1.0 = 100%
            #buy company
            order_target_percent(context.etf_sp500, 0.5)
    elif sma_50 > sma_20:
        if context.etf_sp500 not in open_orders:
            #short company
            order_target_percent(context.etf_sp500, -0.5)
```

Figure 41: Algorithm 1 Version 2 long_short_sp500 method

In this figure, the method recuperates the simple moving averages for the last 50 and 20 days. Each time their values cross each other, it gives an indication to long or short the S&P500. Each time the sma_50 is lower than the sma_20, the algorithm goes long. Each time the sma_50 is higher than the sma_20, it shorts the ETF.

Compared to the two last algorithms, this version should have less risk since it is shorting when the market is going down and going long when the market is rising. The trade-off is that the results should also be lower than the previous versions.

The picture below shows that the expected results are met. Although having risk metrics that are lower, it is still highly correlated to the index. In my opinion, the risks are not diminished enough to justify the losses in results. Here we can also see that the number of shares is 23 instead of 22. This is because we are using an extra asset to short and long the S&P500.

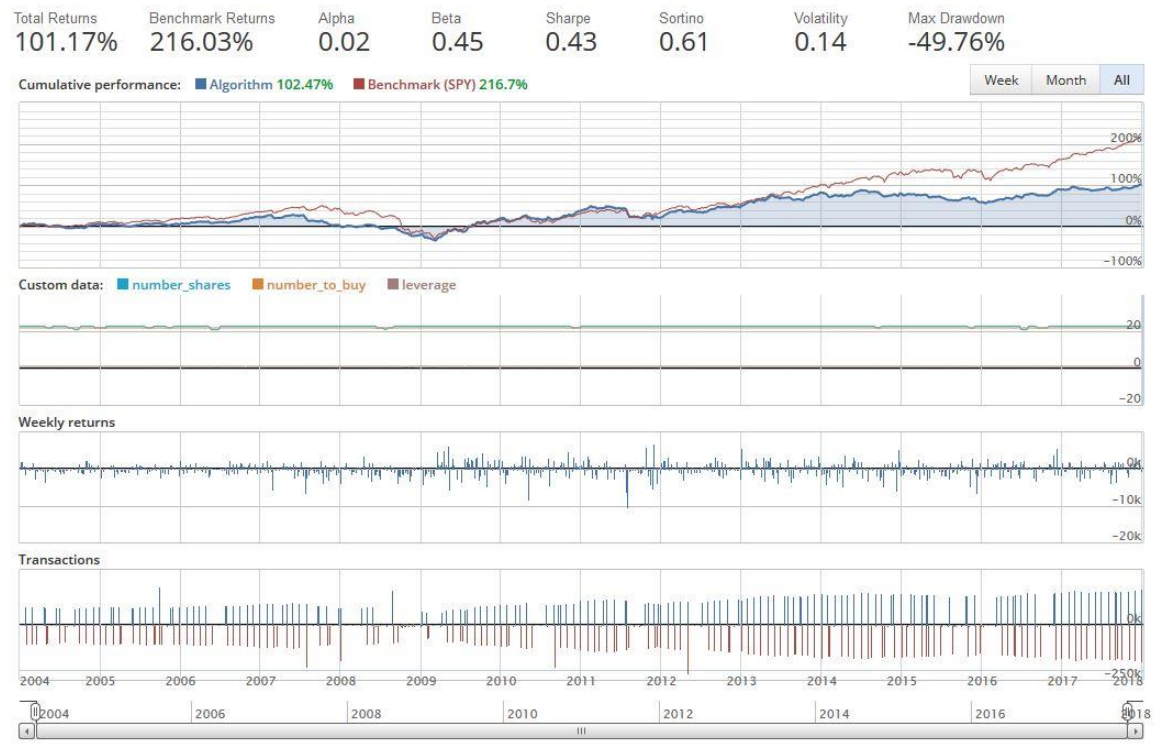

Figure 42: Algorithm 1 Version 2

3.5.4 Algorithm 1 – Version 3

Version 3 uses the same principle as version 2, but instead of deciding when to short or long half of the portfolio, it shorts it all the time. To the contrary of the second version, this is a full hedge since the algorithm is always balanced. This style of trading is interesting for people who are frightened of risk. Increases and decreases in the portfolio's value will be very close to 0. For that type of strategy, the returns come from dividends.

```
def initialize(context):
    """
    Called once at the start of the algorithm.
    """
    # boolean that tells us if the portfolio has been created or  not
    context.portfolio_created = False
    context.stocks_in_portfolio = []
    context.sectors = []
    context.etf_sp500 = sid(8554)

    # Define portfolio once at the start of the program
    schedule_function(my_define_portfolio, date_rules.month_start(),
time_rules.market_open(hours=1))

    # Rebalance every day, 1 hour after market open.
    #schedule_function(my_rebalance, date_rules.month_start(),
time_rules.market_open(hours=1))
    schedule_function(rebalance, date_rules.month_end(),
time_rules.market_open(hours=1))

    # Record tracking variables at the end of each day.
    schedule_function(my_record_vars, date_rules.every_day(),
time_rules.market_close())

    # Create our dynamic stock selector.
    attach_pipeline(make_pipeline(context), 'my_pipeline')
```

Figure 43: Algorithm 1 Version 3 initialize method

This method only differs from the last version's initialize method by the fact that it only has the global variable: context.etf_sp500 for the S&P500 ETF. There is no scheduled method since the algorithm is always shorting the ETF.

I am expecting the lowest returns and risk from this version. The permanent short position of half of the portfolio should result in the decrease of these two metrics, compared to the other versions. My prediction is that the returns will be close to 0.

The image below shows that risks and returns have decreased compared to the other versions. The number of stocks is still 23, like the past version. This is due to the algorithm using an extra asset to short the S&P500. This type of strategy is useful for people wanting to reduce risk as much as possible, due to being afraid of risk or needing the possibility to sell and recuperate cash rapidly without losing money.

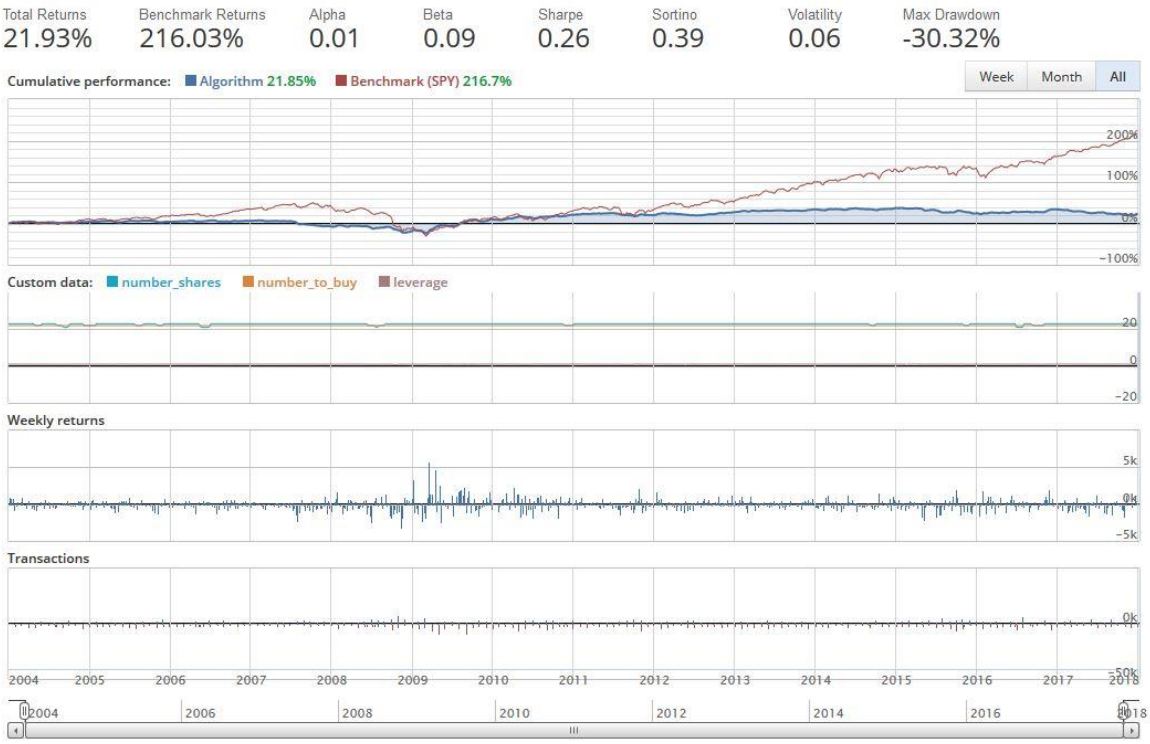

Figure 44: Algorithm 1 Version 3

3.6 Algorithm 1 comparison

The metrics have been divided into the two following tables:

- Portfolio returns metrics: We want these numbers to be the highest possible.
- Portfolio risk metrics: We want these numbers to be the lowest possible.

Table 6: Returns metrics

	Total Returns	*Banchmark Returns*	*Sharpe*	*Sortino*
Version 0	362.15%	216.03%	0.60	0.85
Version 1	269.07%	216.03%	0.50	0.71
Version 2	101.17%	216.03%	0.43	0.61
Version 3	21.93%	216.03%	0.26	0.39

In this table, we notice clearly that version 0 surpasses the other versions in every metric. This comes as a surprise, mostly compared to version 1, which I was expecting to do better. The sharpe and sortino ratios for versions 2 and 3 are lower, even though these two versions have less risk (see table 2). This is because the returns are a lot lower, and these ratios portray the risk-adjusted returns.

Table 7: Risk metrics

	Beta	*Volatility*	*Max Drawdown*
Version 0	1.11	0.22	-59.6%
Version 1	1.19	0.25	-78.86%
Version 2	0.45	0.14	-49.76%
Version 3	0.09	0.06	-30.32%

We notice clearly that the two last versions have less risk than the two first: this is in line with my expectations. Astonishingly, version 1 is the version that has the most risk. I was expecting this version to have less risk than version 0, since I am selecting stocks with specific values for each factor.

To wrap up the first algorithm, I was able to achieve the expected goal and predict the counter part for the last two algorithms: less risk and less returns. For people that need low volatility to be able to withdraw their savings at any time, or simply with minimal risk tolerance: these two types of strategies are good starting points. The difference in results between version 0 and 1 were unexpected. I was expecting opposite results: version 1 to have less risk and higher results. This shows that the stock picking strategy is not up to point and going with the raw selection from version 0 is a better option.

3.7 Algorithm 2

For the second algorithm, Alphalens and Quantopian's Notebook are used to try and find a combination of factors that will do better than the first algorithm's version 0. From now on, each time Algorithm 1 is mentioned, it refers to version 0. The method to determine which factors to test is more of an art than science. The first algorithm having four factors, it will be used as a benchmark and three groups of four factors will be selected to compare it against. Since the comparing process uses the results from the Quantopian notebook, the code is segmented into cells. There are four cells for each test. All of them are identical, except for the second one that is different for each group. The first group's description contains the four cells, and the other groups only contain the modified cell.

The table below shows the four groups: benchmark, cash flow, returns and earnings and valuation. Each group and its factors are related to a theme. The groups are discussed in the paragraphs following the table and a more detailed explanation for each factor is given in the glossary.

Table 8: Groups and factors

	Factor 1	*Factor 2*	*Factor 3*	*Factor 4*
Benchmark	Price to earnings ratio	Price to book ratio	Earnings per share	Total debt to equity
Cash flow	Financing cash flow	Free cash flow	Operating cash flow	Cash return
Returns	Roa	Roe	Roic	Revenue Growth
Earnings and Valuation	Enterprise value	Enterprise value to ebitda	Ebitda	Net margin

3.7.1 Benchmark group

The benchmark consists of the first algorithms factors. I chose them because these are the factors I use for my personal investments. I use these elements to pick the stocks I think are cheaper than the companies' intrinsic values. It will be interesting to see if these factors have good predictive values, and if I can find better factors that will enable me to make smarter decisions for my future investments.

```
from quantopian.pipeline import Pipeline
from quantopian.research import run_pipeline
from quantopian.pipeline.filters.morningstar import Q1500US
from quantopian.pipeline.data import Fundamentals
```

Figure 45: Quantopian's Notebook imports

The imports are similar to the ones in the first algorithm. The key difference is that the notebook requires run_pipeline instead of attach_pipeline and pipeline_output. This is due to Quantopian's different implementation of their algorithm environment and notebook environment.

```
def make_pipeline():
    # column definition
    basic_eps = Fundamentals.basic_eps_earnings_reports.latest
    pe_ratio = Fundamentals.pe_ratio.latest
    pb_ratio = Fundamentals.pb_ratio.latest
    total_debt_equity_ratio = Fundamentals.total_debt_equity_ratio.latest

    # universe definition
    universe = (Q1500US() & basic_eps.notnull() & pe_ratio.notnull() &
pb_ratio.notnull()
                & total_debt_equity_ratio.notnull())

    # rank by average
    basic_eps = basic_eps.rank(mask=universe, method='average')
    pe_ratio = pe_ratio.rank(mask=universe, method='average')
    pb_ratio = pb_ratio.rank(mask=universe, method='average')
    total_debt_equity_ratio = total_debt_equity_ratio.rank(mask=universe,
method='average')
    combined = basic_eps + pe_ratio + pb_ratio + total_debt_equity_ratio

    # create pipeline
    pipe = Pipeline(columns={
        'basic_eps':basic_eps,
        'pe_ratio':pe_ratio,
        'pb_ratio':pb_ratio,
        'total_debt_equity_ratio':total_debt_equity_ratio,
        'combined':combined},
        screen=universe)

    return pipe
```

Figure 46: Quantopian's Notebook, Benchmark group, make_pipeline

This figure shows the definition of the pipeline. In the universe definition, the algorithm selects the 1500 most liquid US stocks that do not have null values for any of the factors. Like the previous algorithms, the columns for the pipeline are created. Each group's pipeline contains five columns: four for the factors and one for the combined column. The latter sums the value of all factors for a given stock. The method returns the pipeline. This cell contains the factors for the benchmark group and is different for each group.

```
result = run_pipeline(make_pipeline(), start_date='2015-01-01',
end_date='2016-01-01')
'''
result is a dataframe
it has 2 indices: date and stock
print(result.index.levels[0].unique()) => dates
print(result.index.levels[1].unique()) => stocks
'''
assets = result.index.levels[1].unique()
# pricing = price for each stock each day
'''
We look at the signal 1 month before and 1 month after the pipeline dates
This is to give alphalens more data before and after
'''
pricing = get_pricing(assets, start_date='2014-12-01', end_date='2016-02-
01', fields='open_price')
```

Figure 47: Quantopian's Notebook result, assets and pricing

The lines of code above have a few roles. The result variable retrieves the pipeline. The asset variable fetches the stocks from the pipeline and sorts them to only have one of each. The pricing variable recuperates the stocks' prices each day from the 1st of December 2014 to the 1st of February 2016.

```
import alphalens

alphalens.tears.create_factor_tear_sheet(factor = result['combined'],
                                         prices = pricing,
                                         quantiles = 2,
                                         periods = (1, 5, 10))
```

Figure 48: Quantopian's Notebook alphalens

In this figure, Alphalens is imported and the method is used to retrieve and display the results calculated by the Alphalens package. The package analyses the factors' daily evolution and compares them to the stocks' daily pricing. It then gives a predictive value to each factor. These results are inserted into the tables and discussed in the groups comparison chapter (Groups comparison: 3.7.5).

3.7.2 Cash flow group

For the cash flow group, I chose four factors related to cash flow. This group gives us information about the amount of money transiting to and from the company. A high cash flow signifies that the company has enough liquidity to pay its debts. Sectors like pharmaceuticals have a lot of Research and Development fees, lowering their cash flow. This means that low cash flow is not necessarily a negative signal, since a new patent can vastly improve the company's earnings.

```
def make_pipeline():
    # column definition
    financing_cash_flow = Fundamentals.financing_cash_flow.latest
    free_cash_flow = Fundamentals.free_cash_flow.latest
    operating_cash_flow = Fundamentals.operating_cash_flow.latest
    ebitda = Fundamentals.ebitda.latest

    # universe definition
    universe = (Q1500US() & financing_cash_flow.notnull() &
free_cash_flow.notnull() & operating_cash_flow.notnull()
                & ebitda.notnull())

    # rank by average
    financing_cash_flow = financing_cash_flow.rank(mask=universe,
method='average')
    free_cash_flow = free_cash_flow.rank(mask=universe, method='average')
    operating_cash_flow = operating_cash_flow.rank(mask=universe,
method='average')
    ebitda = ebitda.rank(mask=universe, method='average')
    combined = financing_cash_flow + free_cash_flow + operating_cash_flow +
ebitda

    # create pipeline
    pipe = Pipeline(columns={
        'financing_cash_flow':financing_cash_flow,
        'free_cash_flow':free_cash_flow,
        'operating_cash_flow':operating_cash_flow,
        'ebitda':ebitda,
        'combined':combined},
        screen=universe)

    return pipe
```

Figure 49: Quantopian's Notebook, Cash flow group, make_pipeline

This figure shows the cell containing the make_pipeline method for the cash flow group. The point of this method, as for each group, is to define the columns and return the pipeline. The columns are different from the benchmark group since the test is being done on different factors. It is interesting to see if factors linked to cash flow can give good indications and provide better results when given to Alphalens. It is possible that this group does not perform as well as other groups since pharmaceutical or tech companies do not have good cash flow, due to research and development, but have good returns. This means that good investments could be discarded because of low cash flow. The rest of the method follows the same logic as the benchmark.

3.7.3 Returns group

The returns group focuses on different methods of calculating returns generated by firms. The first tree factors include the company's net income for their calculations, while the fourth one details the company's growth in revenue.

```
def make_pipeline():
    # column definition
    roa = Fundamentals.roa.latest
    roe = Fundamentals.roe.latest
    roic = Fundamentals.roic.latest
    revenue_growth = Fundamentals.revenue_growth.latest

    # universe definition
    universe = (Q1500US() & roa.notnull() & roe.notnull() & roic.notnull()
                & revenue_growth.notnull())

    # rank by average
    roa = roa.rank(mask=universe, method='average')
    roe = roe.rank(mask=universe, method='average')
    roic = roic.rank(mask=universe, method='average')
    revenue_growth = revenue_growth.rank(mask=universe, method='average')
    combined = roa + roe + roic + revenue_growth

    # create pipeline
    pipe = Pipeline(columns={
        'roa':roa,
        'roe':roe,
        'roic':roic,
        'revenue_growth':revenue_growth,
        'combined':combined},
        screen=universe)

    return pipe
```

Figure 50: Quantopian's Notebook, Returns group, make_pipeline

This cell contains the make_pipeline method for the returns group. It follows the same logic as the last two groups with the only differences being the factors and columns. I expect this group to do better than the previous group. Companies' returns give a good indication on the profitability. One big caveat is that these factors do not perform well when finding companies that are not profitable yet, but have very good products, market strategy and growth. For this reason, I don't expect this group to beat the benchmark, but still do better than the cash flow group.

3.7.4 Evaluation & earnings group

The last group's factors enable us to focus on the earnings, income and value of the company. They help us determine what the company is intrinsically worth and what its earnings and income are. A company with good earnings and income, coupled with a good intrinsic value gives a notion of solid company.

```
def make_pipeline():
    # column definition
    enterprise_value = Fundamentals.enterprise_value.latest
    ev_to_ebitda = Fundamentals.ev_to_ebitda.latest
    ebitda = Fundamentals.ebitda.latest
    net_margin = Fundamentals.net_margin.latest

    # universe definition
    universe = (Q1500US() & enterprise_value.notnull() &
ev_to_ebitda.notnull()
                & ebitda.notnull() & net_margin.notnull())

    # rank by average
    enterprise_value = enterprise_value.rank(mask=universe,
method='average')
    ev_to_ebitda = ev_to_ebitda.rank(mask=universe, method='average')
    ebitda = ebitda.rank(mask=universe, method='average')
    net_margin = net_margin.rank(mask=universe, method='average')
    combined = enterprise_value + ev_to_ebitda + ebitda + net_margin

    # create pipeline
    pipe = Pipeline(columns={
        'enterprise_value':enterprise_value,
        'ev_to_ebitda':ev_to_ebitda,
        'ebitda':ebitda,
        'net_margin':net_margin,
        'combined':combined},
        screen=universe)

    return pipe
```

Figure 51: Quantopian's Notebook, Evaluation & Earnings group, make_pipeline

The code above represents the make_pipeline method for the evaluation and earnings group. As said for the previous groups, the logic is the same and only the factors and columns change. The combined factor will be passed on to Alphalens, hoping that we can find results that are good enough for us to build an algorithm which can compete with the

version 0 of the first algorithm. The factors in this group give us information on the companies' values, earnings and margins. In my opinion, this gives us a better view of the financial health concerning the companies. I expect this group to do better than the first two and be able to compete with the benchmark.

3.7.5 Groups comparison

The results are displayed in four graphs, each representing a factor. The values are defined for a specific time-frame. The blue bars represent the values if we had held the stocks for one day, the orange bars for five days and the grey bars for ten days.
We want annualized alpha and information coefficient values to be high, and the other two values as close to zero as possible.

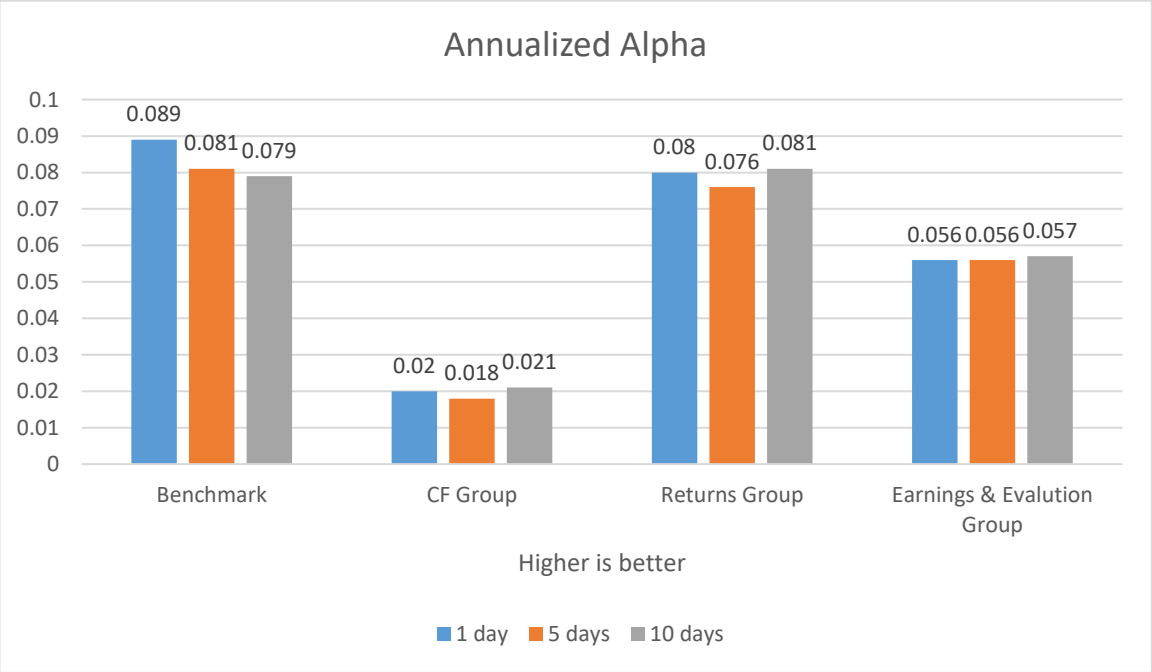

Figure 52: Annualized Alpha

Only two groups stand out for this factor: the benchmark and the returns group. The benchmark group does better for the first two periods, just losing by a small margin in the last period. The CF Group has the worst predicted returns in this case.

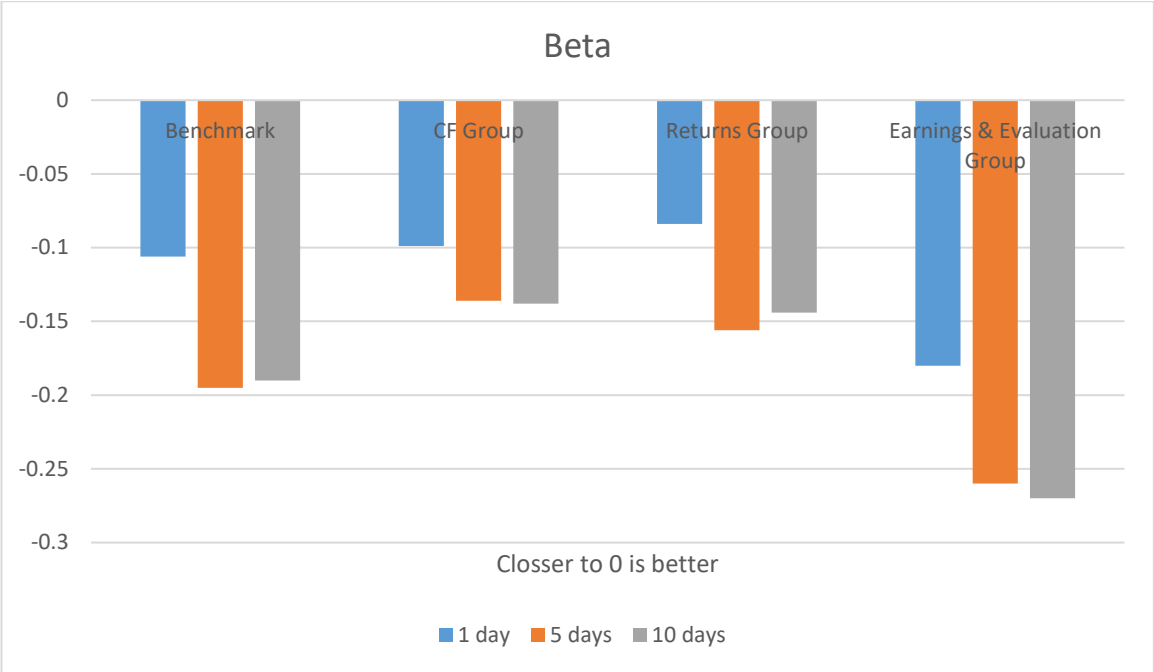

Figure 53: Beta

In this chart, we see that the three first groups are similar for the first period. The second and third period show that the CF Group is bit less risky than the Returns Group. The Earnings & Evaluation Group has by far the worst values.

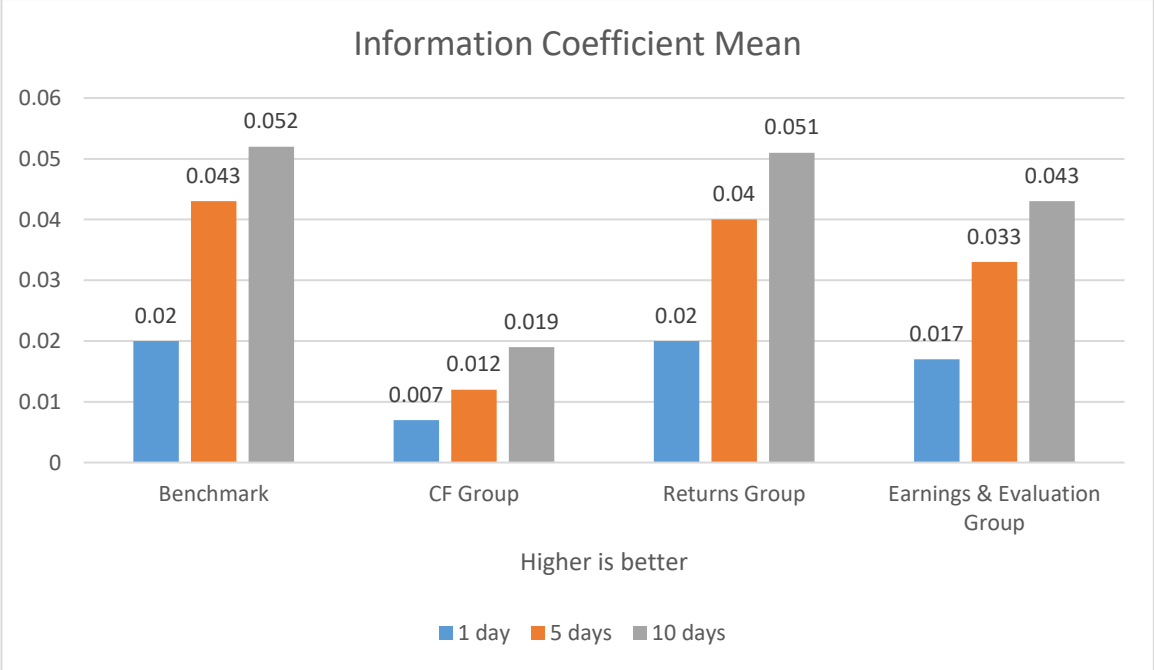

Figure 54: Information Coefficient Mean

In this graph we see that the CF Group is trailing behind again, like in the annualized alpha graph. The Benchmark and Returns Group are toppling the list within 0.01 from each other, while the Earnings and Evaluation Group is trailing behind.

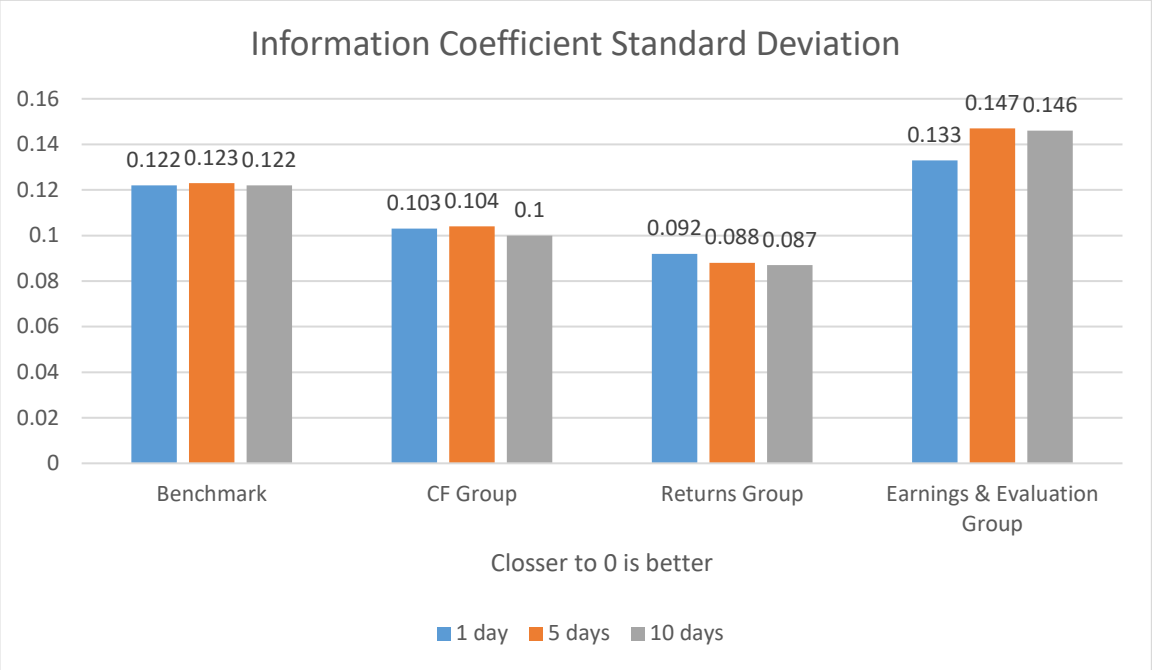

Figure 55: Information Coefficient Standard Deviation

In this chart, we see that the Returns group has the best values, with the CF Group following and the Earnings & Evaluation Group trailing behind. The Benchmark stayed very consistent, only moving by 0.01, while the rest moved more throughout the three different periods.

Based on the four criteria we used, we can assess that the Benchmark and Returns Group have the most potential. The latter has less risk, even though the trade-off is seen in the returns, specified by alpha, for the first two periods. The two other groups have less returns and arguably more risk. The second algorithm will use the Returns Group's factors to see if it can beat the first algorithm. When choosing the factors, I expected the earnings and evaluation group to do better. The results show that in this case, the returns factors are more indicative of better results than the other groups, except for the benchmark.

3.7.6 Algorithm 2 - Implementation

This algorithm is the same as the first algorithm. The only difference is that the factors have been switched. With the use of Alphalens, diverse groups of factors were tested against the factors used in the first algorithm. According to the results, the only group that has a possible chance of beating the first algorithm is the Returns group.

The figure below shows the make_pipeline method for the second algorithm. Only this method is showed since the other methods are identical to the first algorithm's version 0. The factors used to create the combined factor are the ones from the Returns group. The method recuperates each factor from the Fundamentals package. The universe is then defined to include only the 1500 most liquid US stocks. The factors are then ranked by average and added together. The pipeline is created with two columns: one for the combined factor and the other for the sectors. The pipeline is then returned, and the rest of the algorithm follows the same process as the first algorithm's version 0.

```
def make_pipeline(context):
    """
    A function to create our dynamic stock selector (pipeline).
Documentation on
    pipeline can be found here: https://www.quantopian.com/help#pipeline-
title
    """
    # columns
    roa = Fundamentals.roa.latest
    roe = Fundamentals.roe.latest
    roic = Fundamentals.roic.latest
    revenue_growth = Fundamentals.revenue_growth.latest
    sector = Fundamentals.morningstar_sector_code.latest

    # universe definition
    universe = Q1500US()

    # rank factors
    roa = roa.rank(mask=universe, method="average")
    roe = roe.rank(mask=universe, method="average")
    roic = roic.rank(mask=universe, method="average")
    revenue_growth = revenue_growth.rank(mask=universe, method="average")

    # adding factors
    factor = roa + roe + roic + revenue_growth

    pipe = Pipeline(
        columns={
            'factor':factor,
            'sector': sector
        },
        screen = universe)

    return pipe
```

Figure 56: Algorithm 2 make_pipeline method

3.7.7 Algorithm 2 - Comparison

The picture below shows the results for the second algorithm. The returns are better than the S&P500, but highly correlated to it. The number of shares, number of shares to buy and leverage are plotted. Like the algorithm 1 version 0, the two first have to be equal to 22 and leverage has to be as close as possible to 1.

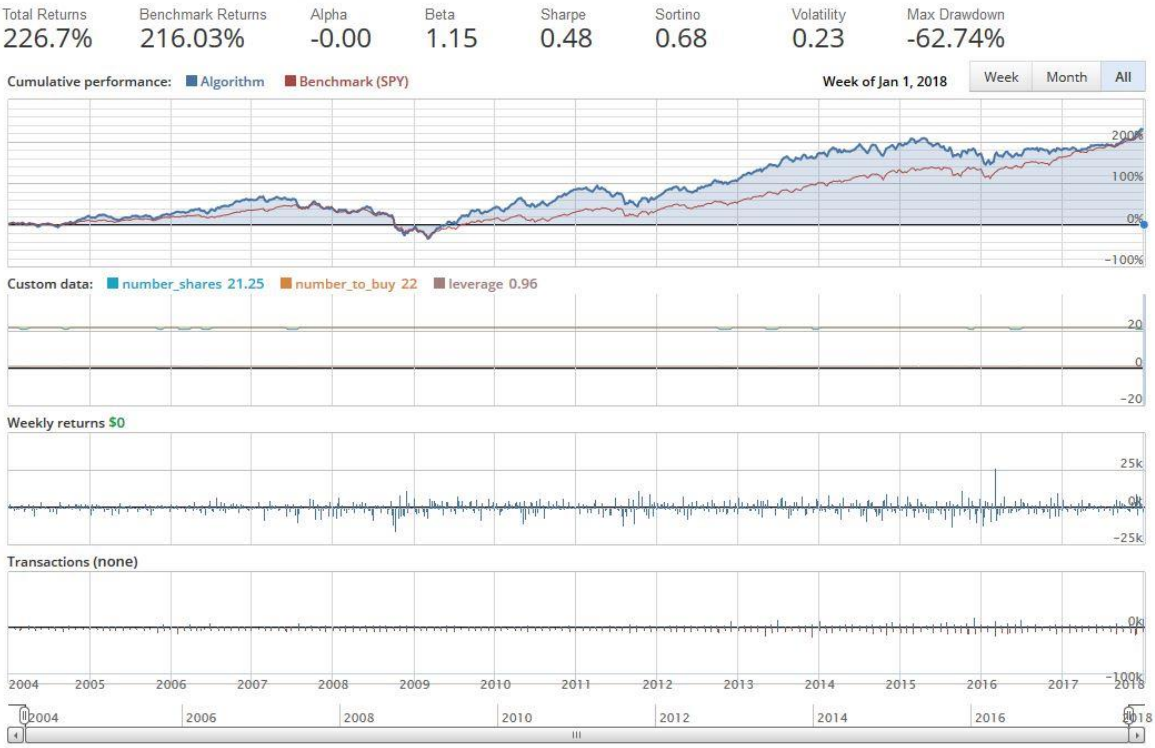

Figure 57: Algorithm 2 results

We will be looking at the same return and risk metrics that were used to compare the different versions of the first algorithm. We want the return metrics to have the highest possible values, and the risk metrics to have the lowest possible values.

Table 9: Returns metrics

	Total Returns	*Banchmark Returns*	*Sharpe*	*Sortino*
Algorithm 1	362.15%	216.03%	0.60	0.85
Algorithm 2	226.7%	216.03%	0.48	0.68

In this table, we see clearly that the first algorithm surpasses the second algorithm in every possible metric. Even though our second algorithm beats the S&P500, the first one's returns are over 130 percentage points higher.

Table 10: Returns metrics

	Beta	*Volatility*	*Max Drawdown*
Algorithm 1	1.11	0.22	-59.6%
Algorithm 2	1.15	0.23	-62,74

The table above shows the same story as the first table: the first algorithm is better in every way. The second one is riskier, and this is not compensated by higher returns.

Even though Alphalens' results suggested that the factors used for our second algorithm might have a chance at higher returns and lower risk, the results did not match the expectations. It was a surprise to see such a difference in results between the two algorithms. This does not negate the results given by the python package. Proving that Alphalens works, or not, would need a complete analysis with many single factors and combined factors of varied sizes. To add to that, the tests must be done for different periods and on a substantial amount of stocks. This said, in our case, the results show clearly that the results in Alphalens did not match the expected results once the algorithm was backtested. Finding the best factors is not a simple task: it requires a lot of testing within Alphalens and in complete algorithms. It necessitates going back to the drawing board and attempting to find the best combination of factors possible.

4 Discussion

Algorithmic trading being common in the world of finance, I decided to investigate creating my own algorithms. Selecting stocks to buy is a lengthy process and amateur managers do not necessarily have time to do complete analyses. Creating and using these algorithms can spare time on the data processing part and enable traders to focus on the strategies. The objective was to create two algorithms: the first one replicating a human style investment strategy, and the second using artificial intelligence. The second's objective was to find better factors and beat the first algorithm with the same strategy. After having seen machine learning tutorials, it was decided to abandon that part for a python package that provided factor evaluation process. The reason for the switch in direction was due to the accuracy of the artificial intelligence. Its accuracy was 49%, which is more luck than prediction.

It is clear that algorithmic trading with its huge processing power, capability to analyse large datasets and action times that are in the order of milliseconds improve vastly certain aspects of trading. Selecting 20 stocks among thousands is a question of seconds for a computer today. While humans would require days or even months to reach a result. It is important to note that not all elements are improved by machines. Qualitative analysis is not yet suitable for machines and is still only used by humans. With the creation of all ready solution like Quantopian, QuantConnect and Quantiacs, algorithmic trading has never been as attainable by amateurs. However, high frequency trading is still only feasible by big corporations.

4.1 Encountered problems and difficulties

The first difficulties encountered evolve around the learning process. Data science languages and programs have different syntax and logic than the programming I am used to doing. The biggest difficulty was working with data-frames, which I had very seldomly done before.

Another problem was the tutorials. There wasn't a lot of tutorials and they usually expected the viewer to have some knowledge of python, data science logic and algorithmic trading. This made it very hard to start writing my own algorithms with my own logic.

The documentation was very scarce for some subjects. For example, the Fundamentals package had just changed when I was starting to learn algorithmic trading. The old methods described in the documentation, which was not updated yet, did not work anymore, so

being able to call these methods was a feat in itself. Another example is Alphalens. Its documentation contained very vague explanations of how it worked, and I could only find two tutorials. To add to the difficulty, one of the tutorials was a person reading the (vague) documentation.

The algorithms also faced a decreasing number of stocks throughout the testing period. It would start by buying 22 stocks but finish with only 14. The last number varied depending on the time-frame. A big part of the actual code was added to face that issue and make the algorithm truly autonomous.

The last problem faced was that Quantopian did not have values for the price/earnings to growth ratio (peg) before 2011. This was not realised straight away and caused problems with the stock picking strategy. This ratio was replaced with the total debt to equity ratio. No answer was found in the documentation or on forums to why the values where not provided before the aforementioned date.

4.2 Consideration of results

As seen in the last chapter of the empirical part (Algorithm 2 – Comparison: 3.7.7), the second algorithm was not able to beat the first one. It still beat the S&P500 by 10.63 percentage points and had risk metrics very close to the first algorithm. Based on the fact that 94% of professional actively managed funds trailed behind the S&P500 index by 17.13% in 2016 (Efficient Market Hypothesis (EMH): 2.2.5), it was surprising to see that the two algorithms were actually able to beat the index. These results do not mean that the algorithms created for this paper can do better than these investors. They must be tested live to assess real performance on unknown data. The version 2 and 3 of the first algorithm also had interesting results. The idea behind them was not to seek the highest results but try to reduce risk. These types of strategies are useful for people who are concerned about risk, need to sell the stocks at a moment’s notice or have specific dates each year at which they need to sell them.

The results from the Alphalens package were very interesting. Based on the values returned by the python package, I was expecting the second algorithm to be on par with the first one. This means that more time must be spent evaluating the results and assessing how predictive the returned values are.

4.3 Quality or results

To start on a good note, it was a good idea to select a time-frame which included a major financial crisis (2007-2008). This enabled us to see how the algorithms fared during these unstable periods. As mentioned before in the testing chapter (Testing the strategies: 2.4), only backtesting a strategy is incomplete. Backtesting must always be followed by forward testing when any serious strategy is going to be used in the real world. Forward testing was dropped for this thesis because of the lack of time.

The other elements that lacked testing where the interactions with Alphalens. A lot more time can (and should) be spent on finding and comparing various factors. Instead of looking for combinations of four factors, the developer should seek factors that do well by themselves. Only then, should they be added together to form groups of diverse sizes. Merging factors that don't do that well by their own is another idea, since they might provide surprising results. To add to that, the different groups that were tested in Alphalens should be tested in the algorithm environment too. This will enable developers to see how accurate the predications are compared to the results. This would be an interesting subject, and someone looking to further their knowledge in this domain could do a lot of tests and research to find out if Alphalens really gives good predictions or not. Testing with this package is a lengthy process and the subject is vast enough to be a whole subject on its own. A lot of time should be spent testing the package and its results before going into real trading. Since this thesis is not oriented for professionals, the package and its use were demonstrated without going into detail. This would require too much time and would be too specific for the thesis and its intended audience.

4.4 Conclusions and suggestions for development or further work

Forward testing the algorithms is the first step that can be developed further. The lack of time when writing the thesis and developing the algorithms prevented testing them on live data. Now that the ground knowledge and algorithms have been set, it is easier for me to go into details and tweak the algorithms to find better results. The point of the thesis being the implementation and discussion around these algorithms, it is interesting to push the subject further and start doing more analysis and tests in order to optimize these algorithms. Another point is to look at the versions 2 and 3 of the first algorithm. The idea is to try other hedging strategies or improve the ones already created. These are the weaker algorithms since less time and effort were put into them. Spending more time to research and develop hedging strategies will vastly improve these algorithms and their results. This

means focusing on more advanced strategies in order to provide better returns with less risk.

Having a deeper look into Alphalens and testing various combinations of sizes and combined factors will also greatly improve these algorithms. Factors can be tested on their own and then combined in order to find the best results. Another important note is to test the factors in the algorithm environment. This will enable developers to see if Alphalens results are reliable or not. Proving this last point will require a lot of time and a lot of tests. In the case of this thesis, testing the four distinct groups and comparing them in full-fledged algorithms could provide surprising results.

The code written for this thesis is not perfect. Some optimisations and code cleaning have been done but it can still be improved.

4.5 Thesis process and self-learning

The research started in November, before the topic proposal. The reason for this decision is that I wanted to verify that there was enough substance for me to discuss. I also wanted to assure myself that I could learn enough to write the algorithms. After these two factors were confirmed, the topic proposal was sent. Once sufficient knowledge was acquired, the theoretical part was written. The algorithms were then created and tested. The third element that was focused on writing about the algorithms in the empirical part. After all these steps, the abstract, introduction and glossary were written.

Four meetings were arranged: Tuesday 27th of February, Wednesday 14th of March, Monday 9th of April and the last meeting on the Monday 23rd of April 2018. The most common topics discussed during these meetings were the academic rules concerning reports and the structure of the thesis. These were the most discussed topics because I was focusing on the technical aspects and not spending enough time reading about the regulations. One week before each meeting, an email was sent to the advisor containing the material required and the schedule for the next meeting. The minutes for each meeting were sent to the teacher, at the latest three days after the meeting, according to the specifications decided when writing the plan. The plan also contained the thesis' deadline, which was the Thursday 26th of April 2018 and the evaluation meeting was held on the Thursday 3rd of May 2018.

Learning about the different trading strategies and their implementation was difficult but interesting. The knowledge acquired from the theoretical part helped when writing the empirical part. New skills were learnt and developed in order to write the algorithms and test them. Testing provided an insight into how results can be misleading and how important it is to do different test-cases. For future development, looking into more advanced strategies and analytical technics is a good initiative. It will provide a more critical view of the basic strategies used in this paper and provide new tools to develop more complex processes.

4.6 Ethical viewpoint

Finance and IT are fields that are at the centre of many debates when discussing ethics. Mostly commodity trading is known to have dire impacts on farmers' lifes and raw materials' prices. Investing in certain industries also forces ethical questions concerning ecology, child labour, wages, working rights, exploitation in third world countries, and many more. For the thesis, the algorithms are not created to take these factors into account. The reason is that no actual trading with real money is done. However, once the testing process is done and the trader decides to take the algorithm to the market, it is crucial to keep these elements in mind and think of the impact on the world when investing.

Another point is the IT field. With artificial intelligence being at the centre of many stories today, it is important to take both sides into account. The first point of view concerns job replacements by machines, loss of human contact and decisions and faith put into machines that take more and more decisions themselves. On the other side however, people argue that jobs are created, humans are supervising and these processes only aim to help, not control.

In our western culture, the first situation clearly defines the immoral areas. Investing time and effort into learning about these conditions enables us to take actions that impact the world in a better way. The second however, is more balanced. It is harder to draw a line between the two points of views and define where the machines' limits should be. Learning about these subjects and discussing them with other people enables us to have better understanding of the subjects and their impact on peoples' lives.

Tables of figures

References

Akiwatkar, R. (2017, April 29). *The Most Popular Languages for Data Science*. Retrieved March 17, 2018, from DZone: https://dzone.com/articles/which-are-the-popular-languages-for-data-science

Alphalens. (2018). Retrieved April 16, 2018, from Github: https://github.com/quantopian/alphalens

Automated Trading. (2014, May 28). *Backtest, Forward Test, Live Trading*. Retrieved March 13, 2018, from Automated Trading: http://www.automatedtrading.com/2014/05/28/backtest-forward-test-live-trading/

Babypips. (2018). Retrieved March 17, 2018, from Drawdown and Maximum Drawdown Explained: https://www.babypips.com/learn/forex/drawdown-and-maximum-drawdown

Banks, E. (2010). *Dicitonnary of Finance, Investment and Banking.* Hampshire: Palgrave Macmillan.

Bannock, G., & Manser, W. (2003). *Iternational Dictionary of Finance.* (4. edition, Ed.) Suffolk: The Economist in association with Profile Books LTD.

Beattie, A. (2018, February 6). *The Birth of Stock Exchanges*. Retrieved January 9, 2018, from Investopedia: https://www.investopedia.com/articles/07/stock-exchange-history.asp

Blythe, B. (2017, June 28). *The Electronic Platform that Took Markets Global Turns 25*. Retrieved April 12, 2018, from Open Markets: http://openmarkets.cmegroup.com/12402/electronic-platform-took-markets-global-turns-25

Butler, M. (2016, January 14). *Mean Reversion | Everything You Need to Know*. Retrieved April 13, 2018, from dough: https://www.dough.com/blog/mean-reversion

Christopher, J. (2017, March 14). *Alphalens - a new tool for analyzing alpha factors*. Retrieved March 13, 2018, from Quantopian: https://www.quantopian.com/posts/alphalens-a-new-tool-for-analyzing-alpha-factors

Competent Hustling. (2014, May 26). *High Frequency Trading: Sneaking a Peek and Cutting the Line*. Retrieved March 17, 2018, from Competent Hustling: https://squeezingcompetence.wordpress.com/tag/high-frequency-trading/

Csiszar, J. (2017, August 9). *Understanding Delayed vs. Real-Time Stock Quotes*. Retrieved March 15, 2018, from Go Banking Rates: https://www.gobankingrates.com/investing/understanding-real-time-delayed-stock-quotes/

Das, R., Hanson, J. E., & Kephart, J. 0. (2001). *Agent-Human Interactions in the Continuous Double Auction.* New York: IBM T.J. Watson Research Center.

Demos, T. (2012, March 23). *Getco fined $450,000 for supervisory faults*. Retrieved April 13, 2018, from Financial Times: https://www.ft.com/content/ebb62d04-7466-11e1-9951-00144feab49a

Durden, T. (2011, August 25). *HFT Quote Stuffing Market Manipulation Caught In The Act*. Retrieved April 14, 2018, from Zerohedge: https://www.zerohedge.com/news/hft-quote-stuffing-market-manipulation-caught-act

Folger, J. (2018a). *Fundamental Analysis: Qualitative*. Retrieved January 11, 2018, from Investopedia: https://www.investopedia.com/university/beginner-trading-fundamentals/fundamental-analysis-quantitative-and-qualitative.asp

Folger, J. (2018b). *Paris Trading: Risks*. Retrieved January 18, 2018, from Investopedia: https://www.investopedia.com/university/guide-pairs-trading/pairs-trading-risks.asp

Galbraith, J. K. (2009). The Great Crash, 1929. Wilmington: Mariner Books; Houghton Mifflin Harcourt.

Gregory, P. (2014, August 18). *KDnuggets*. Retrieved March 17, 2018, from Four main languages for Analytics, Data Mining, Data Science: https://www.kdnuggets.com/2014/08/four-main-languages-analytics-data-mining-data-science.html

Hanson, T. A., & Hall, J. R. (2013). *Statistical Arbitrage Trading Strategies.* Ohio: Kent State University.

Hiltzik, M. (2014, October 7). *End of an era: The NYSE floor isn't even good for PR photos anymore*. Retrieved January 13, 2018, from Los Angeles Times: http://www.latimes.com/business/hiltzik/la-fi-mh-the-nyse-floor-20141007-column.html

Hulbert, M. (2017, May 13). *This is how many fund managers actually beat index funds* . Retrieved April 15, 2018, from MarketWatch: https://www.marketwatch.com/story/why-way-fewer-actively-managed-funds-beat-the-sp-than-we-thought-2017-04-24

Indran, S. (2017, April 4). *The 2008 Financial Crisis: Causes, Consequences and Lessons Learned*. Retrieved April 13, 2018, from TheMarketMogul: https://themarketmogul.com/financial-crisis-lessons/

Investopedia. (2018). *Qualitative Analysis*. Retrieved January 11, 2018, from Investopedia: https://www.investopedia.com/terms/q/qualitativeanalysis.asp

IPython. (2018). *The Jupyter Notebook*. Retrieved March 17, 2018, from IPython: https://ipython.org/notebook.html

Lati, R. (2009, October 7). *The Real Story of Trading Software Espionage*. Retrieved April 13, 2018, from WallStreetAndTechnology: http://www.wallstreetandtech.com/trading-technology/the-real-story-of-trading-software-espionage/a/d-id/1262125?

Levine, D. (2013, May 29). *A day in the quiet life of a NYSE floor trader*. Retrieved January 13, 2018, from Fortune: http://fortune.com/2013/05/29/a-day-in-the-quiet-life-of-a-nyse-floor-trader/

Levine, M. (2013, December 13). *Market-Making Is Making Markets*. Retrieved April 13, 2018, from Bloomberg: https://www.bloomberg.com/view/articles/2013-12-13/market-making-is-making-markets

Lewis, C., & Monett, D. (2017, April 26). *AI & Machine Learning Black Boxes: The Need for Transparency and Accountability*. Retrieved April 13, 2018, from KDnuggets: https://www.kdnuggets.com/2017/04/ai-machine-learning-black-boxes-transparency-accountability.html

Macrotrends. (2018, March 16). *Dow Jones - 100 Year Historical Chart*. Retrieved March 17, 2018, from Macrotrends: http://www.macrotrends.net/1319/dow-jones-100-year-historical-chart

Maidique, M. A. (2011, April 13). *Intuition Isn't Just about Trusting your Gut*. Retrieved January 13, 2018, from Harvard Business Review: https://hbr.org/2011/04/intuition-good-bad-or-indiffer

Mamudi, S. (2013, October 16). *Knight Capital Agrees to $12 Million Settlement for 2012 Errors*. Retrieved April 13, 2018, from Bloomber: https://www.bloomberg.com/news/articles/2013-10-16/knight-capital-agrees-to-pay-12-million-fine-for-2012-errors

Maverick, J. (2015, August 31). *How Hedge Funds Front-Run Index Funds to Profit*. Retrieved January 16, 2018, from Investopedia: https://www.investopedia.com/articles/active-trading/083115/how-hedge-funds-frontrun-index-funds-profit.asp

McCrank, J. (2015, September 30). *SEC fines high-speed trading firm Latour for rule violations*. Retrieved April 13, 2018, from Reuters: https://www.reuters.com/article/us-latour-sec-fine/sec-fines-high-speed-trading-firm-latour-for-rule-violations-idUSKCN0RU24L20150930

McFarlane, G. (2018). *High-Frequency Trading Regulations (ETFC)*. Retrieved March 13, 2018, from Investopedia: https://www.investopedia.com/articles/active-trading/041515/highfrequency-trading-regulations.asp

McLeod, A. S. (2013, July 22). *CFTC Fines Algorithmic Trader $2.8 Million For Spoofing In The First Market Abuse Case Brought By Dodd-Frank Act, And Imposes Ban*. Retrieved April 13, 2018, from Finiance Magnates:

https://www.financemagnates.com/forex/regulation/cftc-fines-algorithmic-trader-2-8-million-for-spoofing-in-the-first-market-abuse-case-brought-by-dodd-frank-act/

Meerman, M. (Director). (2013). *The Wall Street Code* [Motion Picture].

Mehta, R. (2012, September 5). *How Buffett RELIES on His Emotions*. Retrieved January 13, 2018, from The Emotionally Intelligent Investor: http://theemotionallyintelligentinvestor.com/?p=41

Milton, A. (2016, October 14). *Introduction to Scalping*. Retrieved April 13, 2018, from the balance: https://www.thebalance.com/introduction-to-scalping-1031052

Mitchell, C. (2014, September 15). *The Beginner's Guide to Pairs Trading*. Retrieved January 18, 2018, from TraderHQ.com: http://traderhq.com/trading-strategies/beginners-guide-to-pairs-trading/

Moffatt, M. (2017, September 18). *What Is Arbitrage?* Retrieved April 13, 2018, from ThoughtCo.: https://www.thoughtco.com/overview-of-arbitrage-1146194

Motif investing. (2015, November 20). *What Is Thematic Investing And Why Is It So Special?* Retrieved March 2, 2018, from Motif: https://www.motifinvesting.com/blog/thematic-investing-special

Oak Trading Systems - About. (2010). Retrieved April 12, 2018, from Oak Trading Systems: https://www.oaktrading.com/About/Default.aspx

O'Dell, J. (2010, October 7). *A Beginner's Guide to Integrated Development Environments*. Retrieved Arpil 14, 2018, from Mashable: https://mashable.com/2010/10/06/ide-guide/#uozHdGhZMiqp

Peavler, R. (2018, February 4). *Debt-to-Equity Ratio: How It's Calculated and What It Measures*. Retrieved April 16, 2018, from The Balance: https://www.thebalance.com/what-is-the-debt-to-equity-ratio-393194

QuantConnect. (2018). *Pioneering Tomorrow's Trading*. Retrieved March 15, 2018, from QuantConnect: https://www.quantconnect.com/

Quantiacs. (2018). *Code a trading algorithm. We connect it to capital. You profit.* Retrieved March 15, 2018, from Quantiacs: https://www.quantiacs.com/Home.aspx

Quantopian. (2018a). *Leveling Wall Street's Playing Field*. Retrieved March 15, 2018, from Quantopian: https://www.quantopian.com/

Quantopian. (2018b). *Curated financial data. Interactive coding.* Retrieved March 17, 2018, from Quantopian: https://www.quantopian.com/notebooks/survey

Quantopian Overview. (2018b). Retrieved March 20, 2018, from Quantopian: https://www.quantopian.com/help

Rodrigo, C. G. (2017, July 29). *Micro and Macro: The Economic Divide*. Retrieved April 12, 2018, from International Monetary Fund: http://www.imf.org/external/pubs/ft/fandd/basics/bigsmall.htm

Schaffler, R. (2015, July 8). *What It's Like on the Floor of the New York Stock Exchange*. Retrieved January 13, 2018, from TheStreet: https://www.thestreet.com/story/13212201/1/what-its-like-on-the-floor-of-the-new-york-stock-exchange.html

Scherbaum, C., & Shockley, K. (2015). Analysing Quantitative Data for Business and Management Students. Thousand Oaks: Sage.

Sethi, R. (2012, December 7). *Risk and Reward in High Frequency Trading*. Retrieved April 13, 2018, from RajivSethi: http://rajivsethi.blogspot.fi/2012/12/risk-and-reward-in-high-frequency.html

Shmuel, J. (2014, April 2). *The good and bad of high-frequency trading*. Retrieved April 13, 2018, from Financial Post: http://business.financialpost.com/investing/the-good-and-bad-of-high-frequency-trading

Soe, A. M., & Poirier, R. (2016). *SPIVA U.S. Scorecard.* Research Spiva.

Statistics Finland. (2015, April 1). *Half of all households had a net wealth of EUR 110,000 or more in 2013*. Retrieved March 8, 2018, from Statistics Finland: https://www.stat.fi/til/vtutk/2013/vtutk_2013_2015-04-01_tie_001_en.html

Syndicateroom. (2018). Retrieved March 27, 2018, from What are passive funds?: https://www.syndicateroom.com/crowd-investing/what-are-passive-funds

Teweles, R. J., & Bradley, E. S. (1998). The Stock market. New York: John Wiley & Sons, INC.

Thomsett, M. C. (2006). Getting started in Fundamental analysis. Hoboken: John Wiley & Sons, Inc.

Treanor, J. (2015, April 22). *The 2010 'flash crash': how it unfolded*. Retrieved April 13, 2018, from The Guardian: https://www.theguardian.com/business/2015/apr/22/2010-flash-crash-new-york-stock-exchange-unfolded

Twin, A. (2010, May 6). *Glitches send Dow on wild ride*. Retrieved April 16, 2018, from CNN Money: http://money.cnn.com/2010/05/06/markets/markets_newyork/index.htm?hpt=T1

Wiecki, T. (2017, June 23). *Machine Learning on Quantopian*. Retrieved March 15, 2018, from Quantopian: https://www.quantopian.com/posts/machine-learning-on-quantopian

Yager, F. (2012, April 11). *Riding Herd on Big Data, It's the "Algo-Jockey"—on Trading Desks Everywhere*. Retrieved April 13, 2018, from efinancialcarreers: https://news.efinancialcareers.com/us-en/92052/riding-herd-on-big-data-its-the-algo-jockey-on-trading-desks-everywhere

Yahoo Finance. (2017, October 17). *Traders say 1987-style crash could happen again*. Retrieved March 17, 2018, from Yahoo Finance:

https://finance.yahoo.com/video/30th-anniversary-1987-crash-then-142035511.html

Yu, R. (2017, June 8). *House passes bill to dismantle Dodd-Frank*. Retrieved March 13, 2018, from Usa Today: https://www.usatoday.com/story/money/2017/06/08/house-passes-bill-dismantle-dodd-frank/102643024/

Appendices

Appendix 1. Sectors and stocks array mappings

The sector array contains 11 cells numbered from 0 to 10. The stock array contains 22 cells numbered from 0 to 21. Each stock is linked to one sector and each sector has two stocks linked to it. Since the algorithm starts by adding the first sector into the corresponding array, then adds the two stocks into their corresponding array, the two arrays are always matched. By dividing the index of the stock array by two: because the array is twice bigger than the sector array, the algorithm will always land on to the corresponding cell in the sector array. Since the algorithm is working with Integers: natural numbers, any number that is not whole is rounded towards the bottom. 9 / 2 will give 4.5 which rounds to 4. The image below shows the link between the two arrays.

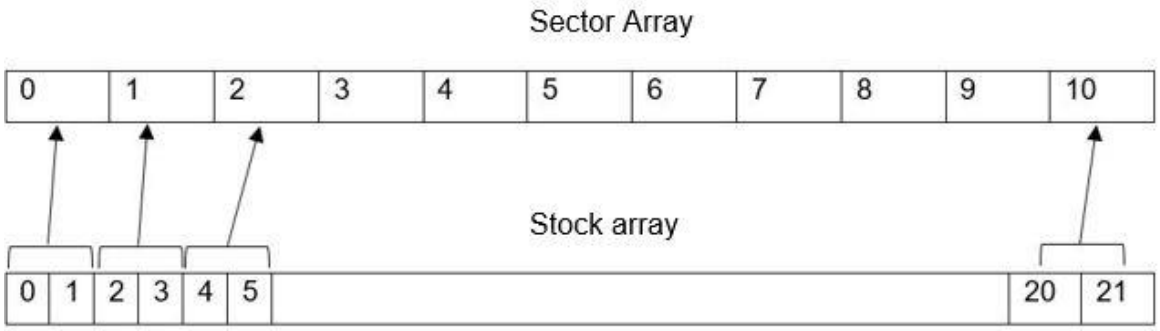

Figure 58: Stock and sector array

Appendix 2. Glossary

Alpha

Alpha is a measurement of the performance as opposed to the index. If the benchmark goes up by 50% from 10'000.- USD to 15'000.- USD. And our algorithm goes up by 100%, from 10'000.- USD to 20'000.- USD. The alpha will be 50% or 0.50. If our algorithm performs worse, this number will be negative.

Annual rate of return

The rate of return is a profit on an investment, expressed as a percentage, for a specific time-frame. In this case, a year.

Annualized alpha

Convert the alpha's value so that it reflects the annual rate.

Ask

Ask refers to the price an owner is willing to sell his asset for.

Assets

An asset can be financial or tangible:

- The first is non-physical and represents stock, bonds, bank deposits.
- The second is physical and represents real estate or commodities.

Even though the names do not suggest it, both are used in finance.

Benchmark returns

This shows the returns of an asset, generally an index like the S&P500, that is used to directly compare our algorithm with. The aim is to beat this benchmark.

Beta

This metric displays the volatility of a strategy compared to the benchmark. A beta of 1 means that the portfolio is completely correlated to the compared index. The higher the beta, the more volatile the portfolio is compared to the benchmark. A negative value signifies a negatively correlation.

Bid

A bid is the price a trader is willing to pay to buy an asset.

Bond

A bond is a debt that the issuer owes the holder. The issuer is obligated to either pay an interest or reimburse at a later date.

Cash flow

Total amount of money transiting in and out of a company.

Cash return

Measures how efficiently the company uses its capital.

Commodity

A commodity is a primary agricultural product or raw material. Examples of these products are: any type of cereal, any precious metal, oil, and more. These assets are bought and sold on specific markets.

Contract for differences (CFD)

CFD is an arrangement where the settlement of a futures contract is made by cash instead of the delivery of physical goods.

Correlation

Correlation is a statistic that represents a relationship between at least two assets. For example, two stocks that are highly correlated will move in the same direction (value increase or decrease). Correlation is measured as a value in between -1 and 1:

- A correlation of 1 means the two stocks will have the same value increase or decrease.
- A correlation of -1 is called a negative correlation: if stock A increases, stock B decreases by the same value.
- A correlation of 0 means there is no relationship.
- A correlation of 0.5 means that Stock B will only rise by 50% of the increase of Stock A.

Credit rating

Credit rating is an evaluation of the probability for a person, company or government to pay back their debts. The higher the rating, the higher the chance the debt will be paid.

Currency

Currency refers to money, under any form, that is used or is in circulation. In general, a currency is related to a specific geographic region, but this is not always the case. For example, cryptocurrencies are not linked to any specific geographic region.

Data centre

Data centres are huge facilities that house storage systems, telecommunications equipment and computer systems. Financial institutions build these structures close to stock markets or at strategic locations between several stock markets in order to have an increased access time to the information given by that stock market.

Drawdown

The drawdown is a value expressed in percentage that measures the decline from a historical peak, before the value increases again.

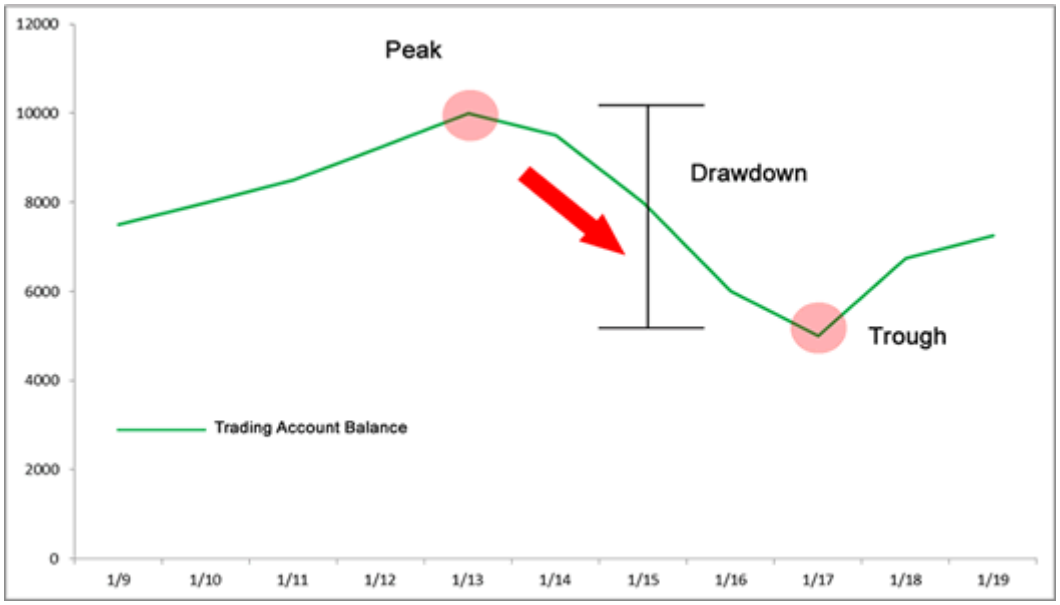

Figure 59: Drawdown (Babypips, 2018.)

Earnings per share (EPS)

EPS is the amount of a company's profit that is assigned to each share. An EPS of 3 means that the company is earning 3.- USD for each share.

Earnings before interest and taxes (EBIT)

EBIT is a measure of a firm's earnings before considering interests and taxes.

Earnings before interest taxes depreciation and amortization (EBITDA)

EBITDA is a measure of a firm's earnings before considering interests, taxes, depreciation and amortization.

Emerging markets

An emerging market refers to countries that meet some criteria of a developed market. India, China and Russia are examples of emerging markets.

Enterprise value

The enterprise value is the company's total value. It is calculated by adding the market capitalization, total debt and subtracting the investments and cash.

Enterprise value to EBITDA

Enterprise value to EBITDA is equals to enterprise value divided by EBITDA. This value is used to compare different companies.

Exchange traded fund (ETF)

An ETF is a fund that tracks and replicates an index fund, stock index, bond or other portfolio. Since this type of fund does not try to beat the underlying asset it is tracking, it has less fees.

Financing cash flow

Financing cash flow is a form of financing that differs from an asset-backed loan. The first backs a loan with its cash flow while the second backs the loan with its assets.

Free cash flow

Free cash flow is the valuation of the amount of cash a company generates after taking into account the spending for property, buildings, plants, equipment, etc.

Fund

A fund, also called investment fund enables several investors to participate in a common portfolio. This reduces transactions costs and increases diversification. These funds also hire professionals to manage the portfolios.

Futures contract (future)

A Futures contract is an accord to sell a financial asset at a specified date and determined price.

Index fund

An index fund follows the same strategies as ETFs. The core difference is in management fees. ETFS have very low management fees but have transaction costs. Index funds have les transaction costs but more management fees. Choosing one or another depends on the fees, transactions costs and the amount the investor has.

Information Coefficient (IC)

IC is a measure that evaluates how close a forecast matches the actual results. The value varies between 1 (completely correlated: the forecast matches the results) to -1 (no correlation at all).

Kevin O'Leary

Kevin O'Leary is a famous Irish and Canadian business man. He is known for his investment strategies where he only invests in secure companies that pay dividends. On the 14th of July 2014, he created and ETF using this investment strategy called O'Shares.

Leverage

Leverage is the action of borrowing money to invest. If a strategy has a leverage of 1, all capital is invested with no extra money borrowed. If the value is lower than 1, not all the capital is used and if it is over 1, money is being borrowed. It is important to keep the value as close as possible to 1 when benchmarking, because a higher value will bias the results.

Long (position)

Buying an asset in the hope that its price will appreciate is "going long" or "taking a long position" on that asset. In general, most non-professional traders go long. A long position has a maximum loss of the invested amount: if 4000.- USD are invested, only 4000.- USD can be lost. The possible returns are limitless since the prices can theoretically continue to increase without limit.

Market Timing

Market timing refers to the entry and exit of trades at specific moments defined by technical indicators, like simple moving averages.

Macroeconomics

Macroeconomics is a branch of economics that analyses the economy as a whole. The studies are made on different levels: international, national, regional, industrial and sectorial. Metrics such as Gross Domestic Product (GDP) and unemployment rates are used to examine this field.

Market capitalization (or market cap)

The market capitalization is the value of all the publicly traded stocks for a company. It is calculated by multiplying the number of stocks by their price. For instance: a company has 1000 shares worth 10.- USD each, the market cap will be 10'000 (10*1000).

Maximum drawdown

This is the drawdown with the biggest difference between a peak and its decline.

Microeconomics

Microeconomics studies the actions, interactions and behaviour of different actors regarding resource allocation and decision making.

Net margin

Net margin represents a company's revenue from sales after deducting all expenses.

New York Stock Exchange (NYSE)

The New York Stock Exchange is an American stock exchange. This is where many US and international companies get listed in order to be publicly traded. It is the biggest in the world by market capitalization.

Operating cash flow

Operating cash flow measures a company's income from its regular operations.

Optical fibre cable (or Fibre optic cable)

Optical fibre cable is a cable that carries light in order to transfer data at high speed between two distant points. This is used by financial institutions to get the information from stock markets as fast as possible.

Order to trade ratio

The order to trade ratio is calculated by dividing the filled orders by the number of posted orders. For instance, if a company posts 100 orders, but only 50 orders are executed, the order to trade ratio will be 50%. This means that half of the trades were not filled and are pending or rejected/cancelled.

O'Shares

O'Shares is an ETF created by Kevin O'Leary on July 14th, 2014, and respects the following rules:

- At least 100 stocks,
- Each stock must pay dividends,
- Each sector invested in must represent less than 20% of the portfolio,
- Each stock invested in must represent less than 5% of the portfolio.

Price to book (PB) ratio

The PB ratio measures the stock price compared to the company's book value. A high value means that investors expect the company to have good profits, but also means that the stock is expensive. A low value means that it is undervalued, or something is wrong with the company. This helps investors estimate if they are paying too much for an asset.

Price to earnings (PE) ratio

The PE ratio measures the share price relative to the earnings per share (EPS). This gives an indication of the company's value. The meaning of a high or low value is the same as the PB ratio.

Price earnings to growth (PEG) ratio

The PEG ratio is calculated by dividing the PE ratio by the growth rate of a company's earnings. It enables traders to assess the stock's value while taking the firm's earnings growth into account.

Returns

The returns represent the increase or decrease in value of the portfolio for a specific period. Investing 10'000.- USD with 3% returns results in a gain of 300.- USD.

Return on assets (ROA)

ROA measures a company's profits relative to its assets.

Return on equities (ROE)

ROE shows the profit a company makes with the shareholders invested money.

Return on invested capital (ROIC)

ROIC measures the return a company makes from its invested capital.

Risk

Risk metrics determine the probability the actual returns will meet the expected returns. The risk calculations also consider the possibility of decrease or loss of the original investment. Volatility is often used as a measurement of risk.

Revenue growth

Revenue growth represents the increase in sales for a company from one period to another.

Sharpe ratio

The Sharpe ratio is used to determine the performance compared to the associated risk. This enables traders to determine if one portfolio is doing better than another because of smart decisions and a good strategy, or due to more risk. The higher the ratio, the better, with a theoretical minimum of 0.

It compares the returns made as opposed to a “risk-free” investment: often the US Treasury or an AAA (see credit rating) bond. A high return associated with minimal risk results in a high Sharpe ratio, while the opposite results in a low value. Two different investments with same returns can be differentiated with this equation. This enables us to understand if the returns are due to a good strategy, or mostly due to a high risk.

Security

A security is another word for a stock.

Share

A share is another word for a stock.

Short (position)

A short position or “going short” is an investment against the market. This means that the more the asset’s price decreases, the higher the returns. To take this position, the trader sells a borrowed stock, and then reimburses the lender at a decided date in the future.
If the price of the asset decreases, the short is profitable, if the price increases, the investor loses money.

To the contrary of long positions, short positions have a maximum return which is 100%. If a share’s price loses all its value and decreases of 100%, the investor has made a 100% profit, doubling his investment. This is due to the fact that a share’s price cannot go below 0. Like it was mentioned for the long position, the value of an asset can increase indefinitely, making the loss limitless.

SIX Swiss Exchange

The SIX Swiss Exchange is based in Zürich, Switzerland and is the main Swiss stock exchange. Swiss and international public traded companies are listed on this stock exchange.

Slippage

Slippage occurs when there is a difference between the price of an ordered trade and the actual price of the trade. For example: if an investor buys a stock for 20.- USD and the

trade is executed at 20.05 USD, slippage has occurred. This is problematic for strategies relying on small spreads (see spread).

Sortino ratio

This ratio is a modified version of the Sharpe ratio that differentiates the upward volatility from the downward volatility. The upward volatility is not taken into account, since it is not considered “harmful” to the portfolio’s returns. Only the downward volatility is used, differentiating it from the Sharpe ratio. A high value represents a good strategy vs a higher negative risk taken.

Spread

The spread is the difference between the bid and ask price for an asset. If the bid price is of 20.-USD and the ask is 19.95 USD, the spread is 0.05 USD.

Standard deviation

A measure that represents by how much members of a group differ from the mean value.

Stock

A stock is the most common financial asset. A stock owner possesses a fraction of the company that has issued the stock. Also referred to as a shareholder, a stock owner has two rights:

- Participate in the company’s corporate election and big decisions by voting during the annual or special meetings,
- Claim part of the earnings under the form of dividends.

Stock market

The stock market is a non-physical entity that represents all owners of a certain type of asset. The stock market can be narrowed down to a geographical section but is different from a stock exchange which is a physical entity.

Stock market index (or stock index)

A stock index is the measurement of a part of the stock market. In general, the stock index represents a geographical, industrial or sectorial view of the market.

Trader

A trader is a person who buys and sells assets, for himself or for the account of another person, group of people or institution(s), with the aim of making a profit.

Total debt to equity ratio

This ratio is used to calculate a company's leverage. It is calculated by dividing the firm's debts by the shareholders equity.

Volatility

Volatility determines the risk associated with an asset. An asset with high volatility experiences frequent prices changes making it unstable, while a non-volatile asset will experience less fluctuation.

Warren Buffet

Warren Buffet is a fundamental investor that is renowned for his strategies and profits. For decades he has outperformed the S&P500 stock index, while this is considered close to impossible by many investors.