Trading System Development

An Interactive Qualifying Project
submitted to the Faculty of
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfilment of the requirements for the
degree of Bachelor of Science

by
Adam Maier
Dylan Richardson
Manuel Gonsalves

Date:
Day Month Year

Report Submitted to: Michael Radzicki

ABSTRACT

The goal of this Interactive Qualifying Project (IQP) is to test whether a diverse system of automatic trading strategies can be profitable, ie., have a higher return than market indexes after costs. The second objective is to understand how a system of systems can improve return rates compared to a single system. In order to do this each member of the team has developed their own trading strategy for a specific market (Stocks and Forex). Each set of strategies will have rules for entry, exit and position sizing according to its unique trading patterns.

ACKNOWLEDGEMENTS

The Team would like to thank professor Michael Radzicki and professor Hossein Hakim for their guidance and help in finding the resources we needed to complete the project. We would also like to thank TradeStation and Backtrader for allowing us to use their platforms to create and test the trades for our systems.

TABLE OF CONTENTS

CHAPTER 1: Introduction

The goal of this IQP is to look at current answers that others have come up with to beating the market and expand on those ideas to create a few individual trading systems that could work together to beat the market.

Investing in one's future is something that everyone needs to do in order to be successful. As online trading becomes more and more available for everyday people it is important for people to know how they can develop their own systems to handle their own finances. This IQP could help people develop automatic trading strategies in scientific way that would allow them to have control over their financial futures.

The second trading system used an artificial neural network (ANN). There are three common ways to apply neural networks to trading systems; predicting an asset's future price, predicting the direction of change in an asset's future price, or predicting how/when to allocate assets within a portfolio. What this system did differently is that it predicted whether or not to simply buy or sell. Its output was a range from 1 (buy) to -1 (sell). So how does the model know when to buy/sell? An algorithm was developed to choose optimal buy/sell points on a historical time frame, meaning it could look forward in time. This algorithm produced the output to train the ANN to predict when to buy/sell based on technical indicators. When it came time to test the ANN, it did not need to look forward in time. It just used current and previous data as input.

The third trading system utilized the idea of a gap strategy. This idea is used based on the idea that a stock opening and closing price changes over night and if it does so significantly than the agent would wait and look for the first 30 minutes to an hour for a sign to enter the market. Normally traders would trust news news overnight or early morning scanners from other resources to find the stocks that they should be looking at. This strategy was based on finding stocks that were likely to gap over night from statistical data and trusting that they were more likely to cover the gap based on more data.

In the end the system of systems that was created was not successful. Not only did it not beat the market it also lost a fair amount of money along the way. This paper highlights the process that went into making the systems as well as some of the reasons as to why they did not work as well as they possibly could have.

CHAPTER 2: Trading and Investing

Trading vs Investing

There are two types of methods when in comes to managing one's finances in any kind of market: investing and trading. Investing is when an individual decides to become part of the market for a long period of time, while trading involves more short term buying and selling of items in the market [Folger]. Both of these general strategies apply to the 4 types of markets already previously mentioned and both have their own advantages and disadvantages.

Investors have a few advantages. Investing is generally safe and will lead to constant growth over a much longer period of time. For this section, the S&P 500 is a great index to look at to get a general idea of how a market changes over a long period of time. While taking inflation into account the S&P 500 has grown on average 10% every year since 1928. Inflation makes up roughly 3% of this growth, but 7% growth each year is very good [Maverick]. However, this is just an average, a good investor has to know when to enter the market and when to leave the market while the market is in a decline or not doing much. As an example from 1982 to 1999 there was only one year that the index had a negative return rate and had an average return rate of return of 15.54%, or in other words $100 would have become $1215 in that time. A large part of this growth is from dividends given by companies. On the other end of the spectrum, if an investor decides to invest in january 1966 and stayed in the market until 1978 the average rate of return was -.08% or $100 would have become $79. There were some good years from 66-78 like 1972 and 1975 but all and all it was a bad time to invest [Compound Annual Growth Rate]. This means that an investor also has the advantage of not having to keep super careful watch of the market day to day, but can look weekly or monthly and still probably remain safe.

One of the larger drawbacks though for an investor is that because they are not looking at it day to day there will be many times where the market does very poorly on the day. A study conducted by Javier Estrada called *Black Swans and Market Timing: How no to Generate Alpha* looked at the dow jones from december 31st 1899 to december 31st 2006. If an investor invested $100 on that day in 1899 and just let it sit there would have made $25,646 with an average annual rate of return of 5.3%. The study then looked at would have happened had the investor took their money out on the markets 100 best days and the investor would have lost $17 in that time, but if the investor managed to miss the 100 days the investor would have made $11,198,634 instead [Radzicki, The Big Picture]. A true investor would not take their money out though for any 1 specific day especially over 100 years so one of the downsides is that investors will have to take those heavy hits when the market crashes on any given day.

Traders have just about the inverse of the advantages and disadvantages an investor would have. A trader is someone that is going to be in and out of the market on a constant basis. The length that traders may hold onto a certain trade depends on the strategy and can range from a couple weeks all the way down to a few picoseconds. As just seen as above one of

the disadvantages of being a trader is not being able to avoid the bad days. A trader has the ability to stay out of the market on days they think will be bad or possibly take advantage of them.

There are many pitfalls that a trader has to be worried about than an investor does not. For one an investor is going to have pay an initial fee to buy into the market, but then there are no extra costs except for taking the money back out of the market; however, a trader because a trader is constantly buying in and out of the market they have to make sure that the money they make from the trade is greater than the cost it actually took to make the trade. A general commision now for most retail traders would be a $9 (both ends of the trade), so even if a trader has a strategy that wins 100% of the time but only makes $5 on average per trade they will on average lose $4 for every trade. As dividends are one the main ways to make profit for an investor they can be a major pain for traders. When traders short an equity they are technically borrowing the stock from someone else and selling it, so if a dividend is given out at that time they owe the person they borrowed the stock from that dividend as well.

Beating the Market

One benchmark that most retail traders look at is how an investment in the S&P 500 would have done in the time that the they were trading. The 2017 SPIVA US Scorecard shows some significant numbers that most professional traders cannot beat the market over a long period of time. While over 50% of professional traders beat the market in 2017, only 5% were able to beat it over 15 years. So it is possible for retail traders to beat the market and make their own fortune, but it is a very difficult task to do. There is a large debate with some people having very strong opinions that a retail trader can't beat the market so they should just be an investor and join the market while others believe it is very plausible for a retail trader to beat the market as long as they are careful.

Basic Market Trends

A very important part of being a trader or an investor is knowing when to enter the market, and knowing what to expect from the market after entering. Generally all markets and many other things like US unemployment, inflation, and GDP follow a similar trend which has a cycle of events. The first step is growth in the market or other area. This is the time that an investor would want to be in the market as all stocks are generally rising. The next step is either a general decline in the market or stagnation [Radzicki, Basic Macroeconomics]. An investor would not want to be in the market around this time, however for a trader as long as the market is moving up or down they are still in business. If the market does stagnate though a trader is going to be in for some rough times as well as they can't make money anywhere. After this general decline though the market will pick up again and start the cycle all over again.

An interesting part of this cycle however is the inverse relation between the private and public sector in the US. While one is on the rise the other is either on the decline or rising slowly. As long as the US has a net positive export the private sectors wealth will increase when the public sector runs a deficit. With this information a trader that is trading in multiple markets

can get a lot of information. If the government is currently running a deficit than the stock market is likely going to be on the rise as private companies are more likely doing better. At the same time though the US dollar is going to inflate lowering its value compared to other nations, so someone who is trading currencies would want to be paying attention to this as well.

It is the federal government's job to try and create a nice balance between the private and public sector. A government who has control over their currency has to choose from three factors to create this balance. The government can choose two of the factors as the other one will be changed by how the first two are changed. These three factors are controlling the flow of capital, having an independent monetary policy, and having a fixed exchange rate. It is the federal government's job to make policies that can directly affect two out of these 3 factors to keep a balance of inflation, unemployment, and GDP.

The Four Asset Classes and Inter-Market Analysis

While the federal government is taking care of the general economy traders and investors have to choose which areas they are going to want to participate in. The four Asset classes are Equities (stocks), currencies, bonds, and commodities [Frankenfield]. These asset groups are traded very differently from each other and each one has there advantages and drawbacks. The four asset groups tend to trend together though so a good trader should be paying attention to all of the markets to find when they should be investing [Corporate Finance Institute].

The basic trend in the US is that the dollar and commodity prices trend in opposite directions, commodity prices and bonds also trend in opposite directions, and stock prices and bonds trend in the same direction unless they decouple due do the fact the government is running a deficit. So as the value of the dollar goes up, commodity prices will go down, and bonds and equities will rise or as the value of the dollar goes up commodity prices and interest rates will go down and stock prices will go up [Radzicki, Inter-Market Analysis & Sector Rotation]. The systems used in this paper focus solely on equities and currencies this section will mostly focus on both of these asset classes.

There are again different advantages and disadvantages of being either a trader or an investor in each one of these markets. An investor in the equity market is going to have as little liquid assets in their portfolio as possible as the belief is that they should be part of the market. So an investors money in the equity part of their portfolio should be completely invested, because if it’s not then the money cannot grow at all. A trader though is likely to have quite a bit of liquid assets because they are waiting for some indicator to enter the market and don’t want to miss it because all their money is already wrapped up in the market.

Traders and investors can also buy equities on a margin. There are two margins that need to be followed, an initial margin requirement for entering the trade and also a maintenance margin requirement that has to be met during the trade. The initial margin requirement effects both traders and investors in about the same way while the maintenance margin requirement is much less likely to affect a trader as it is an investor. The initial margin

requirement for equities is 50% [Firstrade Securities Inc]. This means that how ever much of a stock a trader or investor buys at least 50% of that investment has to be capital that they put up themselves. As an example if an investor buys $5000 worth of stock XYZ than $2500 of the $5000 has to be there own capitol. A maintenance margin requirement allows for a stock to fluctuate up and down without removing an investor from the trade, however if the value of the invest drops by over 30% the trader will be removed from the trade. So in the example above if the value of the holdings in XYZ drops below $3500 than the amount borrowed on the margin remains $2500, but your holdings will drop to $1000 and the investor will have to leave the trade.

Taxes are a problem that everyone has to deal with and just how it affects people in different ways it also affects traders and investors differently. Investments that lasted for at least one year were taxed 15% less on average than trades that lasted for less than a year. For example someone who was in the 10% tax bracket would have to pay 10% on the total of short term trades and 0% in taxes on trades that lasted for over a year. The way the US tax code works it heavily favors those that are making long investments rather than those making smaller trades.

Traders and investors also have different rules for who can do each one. An investor just needs the capitol for the trade and they will be allowed to make the trade they are looking to meet. A day trader trading on larger margins though needs to meet a couple of prerequisites first. The first one being that they have $25,000 at the start of the day. They are allowed to margin up to four times the maintenance margin excess in the account as of the close of business of the previous day; However, if they exceed the day-trading buying power limitation than the firm will issue a margin call [Day-Trading Margin Requirements: Know the Rules]. The trader needs to meet this call in the next 5 days and can only trade at the regular margin rate. If this call is not met they will be restricted to trading with only capital for the next 90 days. These are just some of the regulations that apply to day traders and there are others that apply to both investors and other types of traders.

When considering the accounts and position sizes of traders and investors, there strategies tend to differ again. As was said above an investor is going to likely be investing all of their money they are putting into the stock market. So they have to use different strategies as to choose their stocks and then divide their money appropriately between all of the stocks as to maximize profit while reducing risk. Traders however have a different strategies as to how much they should invest in one trade. Sometimes these strategies will be based on the indicator they are using, and it may tell them it's a trade worth putting more into or it may be riskier so they shouldn’t put in as much. Other times though they may decide to invest based on the size of their portfolio so they may only lose some percentage of it with any given trade.

Trading in the forex market is quite similar to how a trader in the equity market would trade. A trader in the forex market is going to want to have quite a bit of liquid assets at any given time as to not miss out on any trades, and is going to likely also have more strict rules on their position sizes based on what they see. However, There are many differences between the two when it comes to taxes, trading on a margin and legal issues.

When it comes to taxes in trading with forex options they are traded at tax rate of 23%. The calculation for this is based on the a simple formula which is 60% long-term*15% max rate + 40% short-term rate * max income rate. The IRS looks at at gains made by an individual as 60% long term and the remaining 40% as short term [Hunt].

For trading on the margin it is very similar to trading on the margin in with equities. The trader must first set up an account with a broker and come to an agreement as to how much can be on the margin. The usual rate for these traders is that they will put up 1% or 2% of the money that will be traded in the account [Balasubramaniam]. So if the account has $10,000 in it they will only need to only contribute $100. While trading though if there is a loss equal to or greater than what the trader was initially trading a margin call can be made. When this happens the trader must either enter more money into the account to continue trading or exit all trades as to avoid further risk to the broker. If all the money borrowed from the broker is not returned by a certain date the broker can usually add interest to the what they borrowed or if there is a margin call made there can be interest incurred on how much money the trader lost of the brokers.

Sector Rotation in the Equity Market

The next part that is difficult for traders is knowing what market to be in in order to be making the most money possible. One way that traders in the equity market do this is by looking at interest rates set by the federal government, and the cycle that they tend to follow. Traders will likely want to invest in the market in the strongest industry groups in the current strongest sector while shorting the weakest stocks in the weakest industry groups in the weakest sector. The difficult part for traders is knowing which sectors are strong and weak at any given moment.

The sectors tend to follow a cycle that follows current interest rates and that state of the US as a whole. There are 4 periods in the cycle which are a full recession, early recovery, full recovery, and early recession. A full recession is when the market is at its worst and has bottomed out. An early recovery is when the market is starting to recover and is thus a bull market at the time and is then followed by a full recovery which is when the market is at its peak. An early recession is when the market is starting to go down and is a bear market.

During a full recession a trader is going to want to focus on the cyclicals and technology sectors, because these are going to be the strongest markets. As the cycle begins to enter the early recovery stage the sector beings to rotate out of the cyclicals sector and into the industrial sector. Once the market is completely in an early recovery stage the sectors continue to rotate towards the basic industries and Energy, and at a full recovery the main sectors that a trader should be focused on for buying are staples and services. Lastly, once the economy enters an early recession the strongest markets become utilities and financials. This is just one of the ways that a trader knows what markets are currently strong and where they should be looking, but there are other signs as well.

Breath of the Market

For a trader in the equity market they are going to be looking at the stock market. A stock exchange is a “place” that allows people to connect with companies to either buy or sell shares of that company. There are many stock exchanges all over the world. Some of the exchanges may host some of the same companies while some companies may only be found at certain exchanges. Some of the larger exchanges that exist are the New York Stock Exchange, NASDAQ, and the Japanese Stock Exchange. Once a trader knows what exchange they are going to be trading in they will want to know what is currently happening in the market.

There are many different measures that traders use to see the direction of the market. One way that people measure this is with trend lines. For creating an uptrend line the line will start at the lowest low in the period of time being looked at and then some more points will be made to connect the line to. These additional points are at the highest minor low point proceeding the highest high. A down trend line is going to be made almost in the exact same way except it will be starting at the highest high and going the lowest minor high point preceding the lowest low [Radzicki, The Breadth of the Market]. These trend lines are an easy way to get a simple idea if a market is up trending, down trending, or is mostly stagnant over a period of time.

These trend lines are generally applied to indices, but can also be applied to individual stocks. An indice a collection of stocks in one group. The big three indices looked at in the US that give a strong base line for how the market is doing in general are the S&P 500 (Standard and Poor’s 500), DIJA (Dow Jones Industrial Average), and the NASDAQ. The S&P 500 is thought of by many as “the market”. If the S&P 500 is doing good than the market as a whole is generally doing good. Following how these markets are doing is usually a good way of gauging where the market is whether it is a bull or bear market.

After knowing the type of market that the trader is in they will want to be able to measure the breadth of the market. The market breadth is the amount of force and the level behind moves in the market. The main factors to consider when measuring the breadth of the market are volume, mew highs, new lows, advancing stocks, and declining stocks. From these pieces of data there are different type of charts that can be made to measure the breadth of the market.

One of these measurements is called a bollinger band. The idea that there is an exponential moving average (EMA) of the market and two more lines that are created by looking at two standard deviations away from the average that was found. If most of the price bars for the indice or stock are between the middle and upper bound than the market is likely to be trending up. While if the market is between the middle and lower bound the market is likely declining.

Another way people look for these measurements is by looking at two EMAs calculated over a different time frame and comparing the two. Two popular ones to compare are the 13 week EMA with the 34 week EMA. If the 13 week EMA is greater than than the other than the

market is likely to be a bull market and if the 34 week EMA is greater the market is likely to be a bear market. Another popular pair of EMAs is the 50 day and 200 day EMA. Every bull market that has started when the close of the S&P 500 was above the 200 day EMA, but just because any given close is above the EMA does not guarantee a bull market. When the 50 day EMA becomes greater than the 200 day EMA the market is likely heading to a bull market while if the 50 day EMA crosses below the 200 day EMA the market is generally heading to a bear market. These are just a few of the indicators that a trader would use to determine if the market is either bearish or bullish. There are many more including the 100 and 400 day EMA , Advance-Decline Line, McClellan Oscillator, New 52 Week Highs and Lows, On Balance Volume, Arms Index, and a volatility Index.

CHAPTER 3: Trading Systems

A trading system is a collection of rules that determine what orders to place and when [Kuepper]. These orders can be for any one of the asset classes. The rules can be based on a variety of information about the asset being traded or the market in general. To be able to use a system in the actual market it must be developed using a trading platform. At its simplest, a trading platform is just a piece of software used to place orders through a broker [Chen]. Many platforms provide other features like data, charting, backtesting, and optimization. The two platforms used in this project are TradeStation and Backtrader.

All trading system rules rely on some sort of financial data as input. The source or feed of that data is provided by a vendor sometimes through the trading platform. Depending on the data vendor the type of information and the rate at which it is available can differ. Some provide real-time data, while others only provide intraday values or historical data. The information in the feed usually has the standard asset OHLC prices and might include additional metrics and indicators.

A broker is a financial firm acting as an intermediary between the trader and the market. They provide the ability to execute buy and sell orders that are generated by the trading system. For their services, they charge the customer a commision and possibly other fees [Kenton].

When developing a trading system, one of the first considerations to make is what time frame to trade on. The time frame is defined as the time between data points in the feed [Fundora]. It dictates the time scale of trends in the data. Time frames range from microseconds to days or months. In general, signal reliability is positively correlated with the time scale. In other words, on smaller time frames the data becomes noisier or harder to forecast.

Closely related to time frames are trading styles. The four main styles are scalp trading, day trading, swing trading, and position trading [Folger]. Each is defined in terms of time frame and holding period. Scalp traders frequently buy and sell throughout the day holding positions for seconds or minutes. This style relies on a high win percentage since the profit from each trade is relatively small so it is considered the riskiest. A similar style, day trading refers to entering and exiting positions within the same day but holding them for minutes to hours. Neither scalp nor day traders hold positions overnight. Swing trading positions are held for periods of days or weeks. It does not require constant attention like the first two styles. Lastly, position trading is based on the longest time frame. Positions are usually held for months to years. This style ignores short-term price fluctuations. Deciding a trading style is based on factors related to the trader such as their risk tolerance, experience, dedication, and funds.

Trading systems can be divided into two groups; manual and automatic. Manual systems rely on a person to submit the trade, while automatic does not. Generally, both employ the use of algorithms to find the right time to buy or sell. Who pulls the trigger on the order

determines the grouping though. Critics of auto trading worry that algorithms can miss information obvious to an experienced trader, ultimately leading to losing trades and less profit. Auto trading proponents enjoy the benefit of not having to monitor the market as that job has been shifted to the computer. Another reason why some choose auto trading is the ability to remove emotions from decision making. The system will make trades regardless of how the user feels about them.

As mentioned above, a system is a collection of rules based on financial data. This data tends to be broken down into two classes called fundamental and technical. Fundamentals are measurements of companies and economies that determine their intrinsic value [Kenton]. Examples can be quantitative like earnings, revenue, and growth or qualitative like news reports, management changes, and public sentiment. These kind of data are most often employed by manual traders. On the other side of the spectrum are technical indicators. These are defined as statistics of historical market data [Chen]. The goal of this approach is to uncover patterns and trends in the data to obtain a better forecast of the market.

The logic of a trading system is formalized in its collection of rules. The logical strategy of a system is the exploitation of some pattern found in historical data. For a simple example, one might notice that prices bounce between support and resistance and devise a strategy to buy at support and sell at resistance. The rules are executable code for determining when the exploitable pattern exists. With regard to the last example, the rules would need to detect when a price hits support and resistance. These are considered entry rules which are just conditions that must apply to an asset in order for the system to place an order on it. Other important rules exist such as position sizing and exit rules. Position sizing rules simply determine how much capital to place on an individual order. Exit rules, as the name implies, determine when to exit any given trade. They can be based on technical indicators, profit, loss, or time.

CHAPTER 4: Optimizing and Analyzing Trading Systems

Optimization "is the process of making a trading system more effective by adjusting the variables used for technical analysis" [Chen, Optimization]. Optimization provides several benefits for the effort needed to conduct it. The first is backtesting and historical fitting. Backtesting using historical data provides one with the closest possible scenario to what the system may see once it is deployed on the real market. While backtesting is powerful one must be careful of overfitting, where historical data is measured too closely, therefore relying on exact historical movements too much [Chen, Backtesting].

There are three types of data that are used in optimization; training, testing, and model selection. Training data is used to fit the system, the system is put through an optimization program with specific adjustable variables and this data and will produce the most profitable version of the system. Testing data is how the optimization is evaluated after the training. The model will not be adjusted off this data, it is only used to determine if the system is over or under fit. Finally, model selection data is used to determine which system is the most profitable. This data set is used to compare systems and provides the final judgement on which systems are the best according to the evaluator's criteria [Shah, About Train, Validation and Test Sets in Machine Learning].

One type of system optimization is walk-forward analysis. Walk-forward analysis is an optimization strategy that aims to reduce the chance of overfitting, or having too few degrees of freedom [Ruggiero]. Walk-forward analysis does this by dividing the data into in-sample and out of-sample. In-sample data is "used for the initial parameter estimation and model section" while out-of-sample data is "used to evaluate forecasting performance" [Eurostat]. Walk-forward analysis is conducted by taking an In-sample of data and optimizing a strategy with that data. The parameters of the optimization are then recorded. A new sample of data is selected consisting approximately 80% of the previous set and 20% of new data, or out-of-sample data. The strategy is again optimized and the parameters recorded. A visual representation can be seen in figure 4.1.

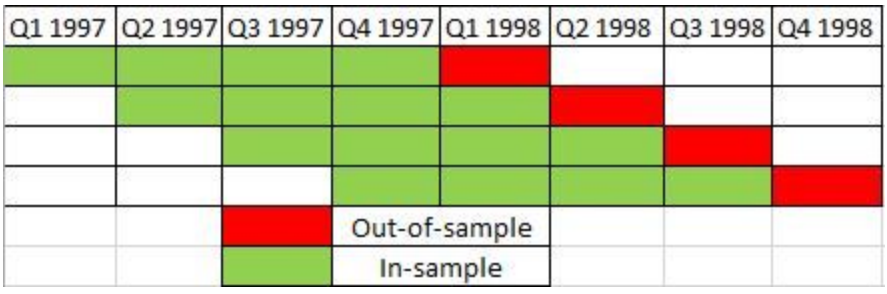


Figure 4.1

Once all the testing data has been used then the entire set is tested together, using the different parameters recorded though the walk forward tests. Once all the parameter sets have been tested they can be ranked and a best set can be determined through a predetermined ranking system [Ruggiero].

Those ranking systems can be a lot of different things, from average drawdown to number of trades per day. Three such ranking systems are expectancy, expectunity and system quality. Expectancy is the average amount the system win or loses on any given trade [Hind]. The formula is:

Expectancy = (Win Rate * Average Win Value) – (Loss Rate * Average Loss Value)

The win are loss rates are in decimal form, i.e. 70% = .70. The goal of expectancy is to determine if the system makes money on average [Hind]. Expectunity is how much a system is expected to make over a given length of time [Branscomb]. The formula is:

Expectunity = Expectancy * (Number of trades/Years the system ran)

System quality is "the representation of profit/loss per dollar risked relative to the total variability of the profit/loss per dollar risked" [Radzicki]. The formula is:

System Quality = Expectancy / (STDEV(R) * SQRT(Number of Trades)

R = (Profit or Loss) / Average Loss

With these three metrics a system can be analyzed in term of its average trade value, the average return over time and how much is being risked to achieve the former two.

Once a system is optimized it has to be monitored. There are three general types of rules for monitoring a system. They are suspension rules, reactivation rules and retirement rules. Suspension rules aim to stop a system when it is becoming unprofitable. These rules are very sensitive as a too strict rule might suspend a system on only a slight downturn while too loose rules will allow a system to lose more money before being suspended. Reactivation rules have a similar but inverse problem. These rules reactivate the system when their conditions are met, if they are too lose a system will reactivate while still unprofitable while too tight and they will miss out on potential profits. Finally system retirement rules are those that tell when a system should be totally shut down. When these conditions are meet the system will not look to reactivate and will cease trade completely.

CHAPTER 5: Literature Review

10 O'clock Bulls in FOREX

The 10 O'clock Bulls strategy is based around the opening minutes of the stock market. The theory behind it is that the opening 30 minutes, between 10:00 and 10:30, will determine the high and low of the market for the day. In this period the traders are reacting to news and analysis done between the close of the previous day and the opening of the current day (Bysshe). By trading within or above/below these limits, traders can use the 10 O'clock bulls strategy as a way to identify opportunity.

FOREX trading is the foreign exchange market. In this market currencies are traded and speculated on. In the FOREX market currencies are traded in pairs. These pairs are each denoted by a pair of three letter abbreviations, such as EUR/USD to represent the Euro/Dollar pair. When a FOREX trade occurs one of these currencies is traded for the other. The FOREX market runs as one of the longest hours markets in the world. The week's trading will start on Monday morning in the Asia-Pacific session and then close at the end of Friday in New York. This means the FOREX market is essentially 6 days a week and, because it is an international market, rarely closes for a holiday. Due to this market is split into 3 distinct sessions; the Asia-Pacific session, the European-London session and the North American session [Galant, 5].

This system aims to adapt the 10:00 Bulls strategy to the FOREX market. It system aims to trade during the North American session and liquidate by the end.

Artificial Neural Networks

An artificial neural network is a computational model inspired by the human brain, hence the name. The basic idea is that an ANN is given inputs that are known to be associated with certain outputs. During the training phase, the network guesses the output based on the input and the model's current parameters. Based on how close the network's guessed output is to the known output, changes are made to the model's parameters or weights. This process is iterated many thousands of times. Hopefully, in the end, the network is better at guessing the correct output for given inputs.

The structure of an ANN is made up of connected nodes or neurons. Each node has weighted inputs and a single output. The inputs are weighted to give more importance to particular inputs over others. The node applies what is known as an activation function to the sum of the weighted inputs. The result of this function is the output for the node. This output is then fed as input to the next layer of nodes. An ANN can have many layers of nodes. The first layer takes input externally, from the training data for example. The output from the last layer is the "answer" to the input. Each layer in between the first and last are called hidden layers. Their inputs and outputs are connected to adjacent nodes. The real process of "learning" in this

technique is the updating of input weights. The way this is done is through an algorithm called back propagation [Woodford]. It is complicated, but it suffices to say that the error between the guessed and actual output is accounted for by systematically changing the weights in the model. The whole process of training an ANN is analogous to learning from mistakes [Karn].

All traders look for an edge or some relatively unknown info that can aid there decision making. Some believe ANNs provide that edge. There are plenty of scholarly papers on just how ANNs have been used in the market and how effective they can be compared to more traditional methods [Voegt]. Let's take a look at a few.

In 1993, Kryzanowski, Galler, and Wright published their paper on using ANNs for investing. They tested the ability of neural networks using historical and current accounting and macroeconomic data to discriminate between stocks providing superior future returns and inferior returns. More precisely, the ANN would predict whether a stock would have positive, neutral, or negative returns for the next year. This is an example of a classification problem where there are distinct groups that the output could be. The input to the ANN was comprised of 88 pieces of data, including 5 yearly changes in 7 different macroeconomic factors, 2 trends of 14 financial ratios over 4 years, 4 years of 5 financial ratios compared to industry benchmarks and the volatility of these ratios over the time span. During their testing, they found the network correctly predicted the direction of stock returns 72% of the time. They didn't apply their ANN to an actual trading strategy so backtesting couldn't be performed. Nonetheless, this work was some of the first of its kind and gave much hope to utilizing AI in trading and investing.

Another study by Atlay and Satman in 2005 compared neural network performance to linear models of stock prices. The ANN inputs for a stock were daily closing price, quarterly book value, common shares outstanding, book to market value, and market capitalization. The output of the ANN was the stock price for the next time period. This kind of prediction is an example of regression where the output is a quantity selected from a continuous variable. Testing was executed on daily, weekly, and monthly time periods with three kinds of ANN models. Compared to three linear models, the ANNs all performed more accurately.

The last example is of another regression system by Wanjawa and Muchemi in 2016. Similar to Atlay and Satman, their ANN predicted stock closing prices for the next day. The input to the network was just the last five closing prices. Using the MAPE metric to evaluate the accuracy, they found their model to predict within 0.77% of the actual prices. While the authors claimed this to be a success, it appeared from their graphs that their predictions always lagged behind the actual price trend. Using this model in a trading strategy would surely not lead to profits.

Gap Strategy

The trading system used the basic idea of a gap strategy. When markets open in the morning there are trades that happened over night that change the opening price from what the closing price was the day before. Normally a trader trading with this strategy would create a

range for a stock that had a gap and then monitor it for the next hour. If the stock rose above the range that would signal the trader to buy the stock and if it fell below the range that would signal a short.

There are four main types of gaps that traders normally look for. Full gaps up or down and partial gaps up or down. A full gap up occurs when the opening price of a stock is higher than the previous days high, and a full gap down is just the opposite. A partial gap up is when the opening price is higher than the closing price but not higher than previous days high and a partial gap down is the the exact opposite [stockcharts, Gap Trading Strategies]. Normally traders will build rules depending on this opening price and observe the market for the first hour to create their range for the day, and then they would have certain buy and sell rules that would tell them what to do if the price of the stock leaves that range.

This system however works with a similar concept. It uses the idea of probability to find stocks that are likely to gap overnight and observe them immediately when the market opens. Instead of creating a range in the first hour the system enters a trade immediately if there is a large enough gap.

CHAPTER 6: 10 O'Clock Bulls in FOREX

The 10 O'clock Bulls in FOREX strategy was a day trading system. This strategy attempted to predict reversals in the market by trading between the High and Low as determined by the 9:30-10:00 period. The EUR/USD pair was picked for this as it was a major pair and both markets were open for the majority of the trading hours. The system had only one entry condition, buy once the low for the day was crossed. Position sizing was determined by a formulas taking into account how much the equity the trader was willing to risk per trade, how much they were willing to lose per trade and the size of the lot that is being traded. Finally, the system checked if a simple moving average predicted that the pair would rise in value before buying, this was to catch the upswing of a pair instead of buying before it crashed.

Following this several monitoring rules were setup to identify when to sell the pair. The first of these rules was to sell once the pair reached the day's high determined by the initial 9:30-10:00 period. The second monitoring rule was a preparation to close rule. As this system aims to be liquid by closing this rule would sell immediately if the trade had made a profit within two hours of closing. The third monitoring rule was a stoploss rule. This rule aimed to prevent catastrophic losses by selling as soon as the trade reached three standard deviations below the High/Low Average. Finally there was an end of the day rule. This rule sold at closing regardless of profit or loss. This was done to maintain the liquidity of the account and prepare for the next day.

This system was tested using two tools called TradeStation and Market Systems Analyzer. The first is a trading platform that was used to backest with three months of historical data from June 1st the October 12th with $10,000 of capital. This provided a series of trades for the period. These trades were then input into Market System Analyzer to evaluate them. In the period 147 trades were conducted with 81 winning and 66 losing trades. The biggest win and loss were $1,286.00 and $1,735.00 respectively. The maximum drawdown was $4002.70 and the average was $1,407.20. The Sharpe ratio of this system was -0.01830. The expectancy, expectunity and system quality were -10.07, -1487.10 and -0.0199 respectively. All these number show that in the long run this system will lose a significant amount of money and should not be traded. In figure 6.1 the Monte Carlo analysis is shown. This charts equity over the trades in the system. While most of the trades are in the outer bounds equity steadily declines as the number of trades increases. This shows that while this system may have lost of winning trades the smaller number of larger losing trades outweighs them heavily.

Future work that may make this system more profitable is an increase in the monitoring of the system. Either a tightening of the stoploss rule or a more rigorous assessment before entering a position may decrease the frequency and size of losing trades.

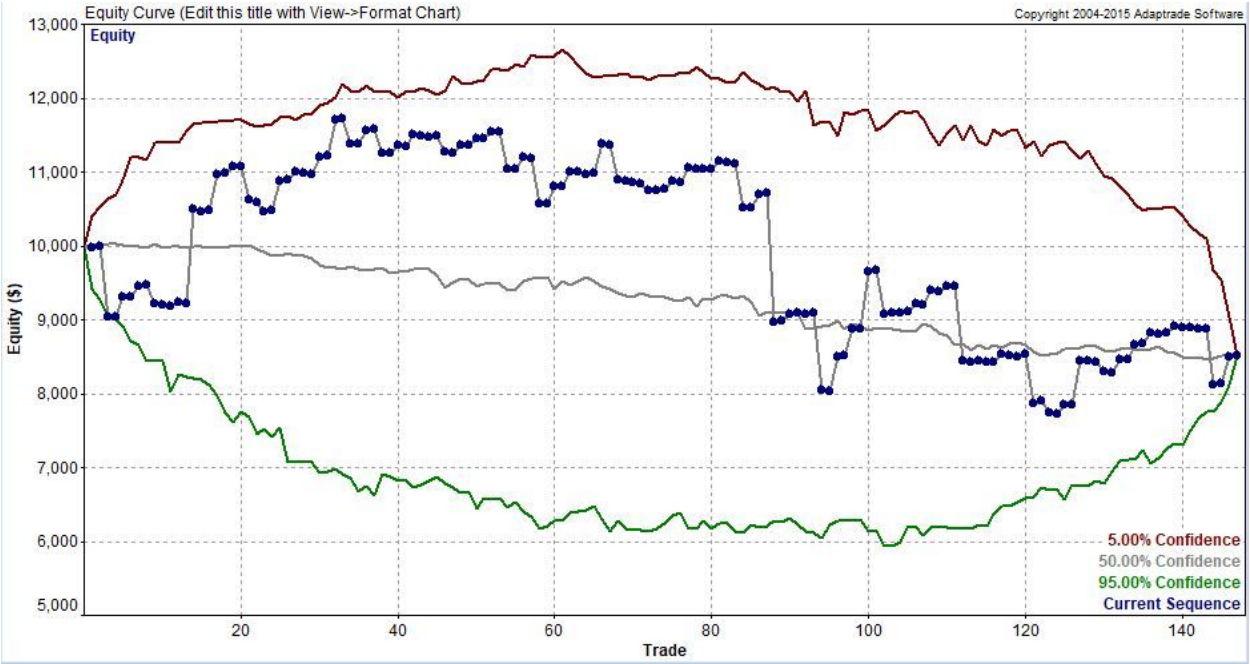


Figure 6.1

CHAPTER 7: Artificial Neural Network

This system is based on artificial intelligence. The intuition is that computers can find patterns in data where humans cannot. There are many techniques of applying AI. Probably the most popular today is the artificial neural network (ANN). This technique usually works best in situations with there is a lot of data for the ANN to find patterns in. Fortunately, trading markets are exactly this kind of situation. Vendors can have a variety of historical data for assets going back as far as 30 years.

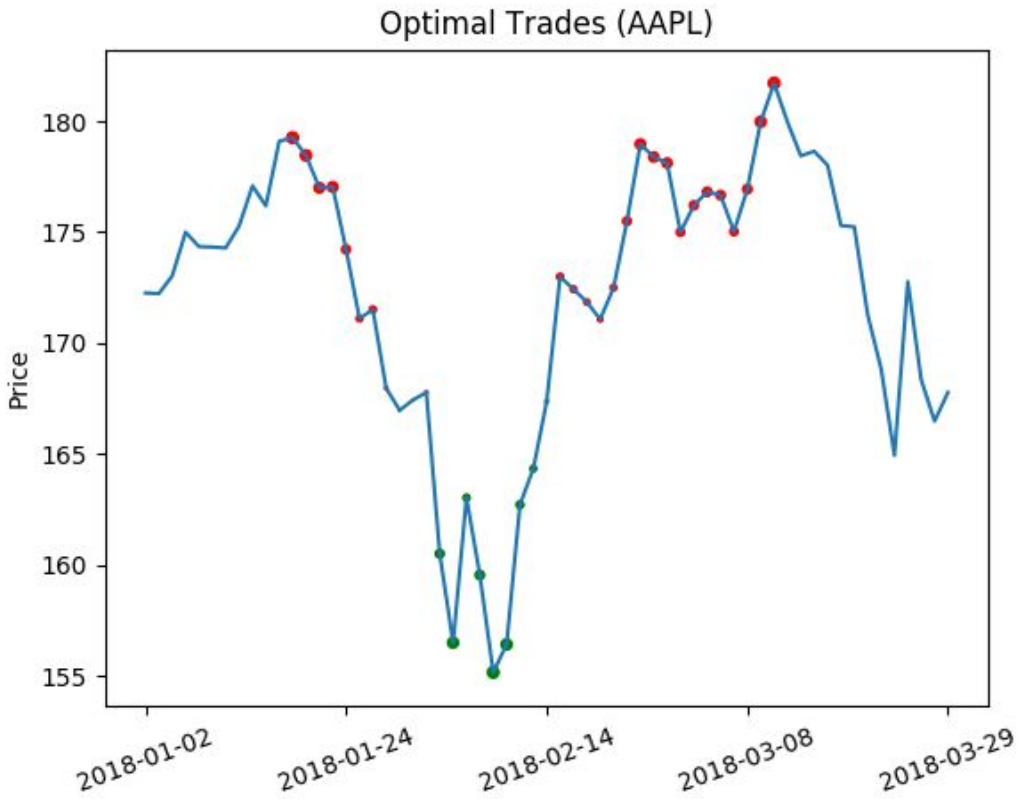
Using a neural network for trading is not a new idea. What differentiates this system from the rest is its signature or the type of input and output. The network is designed in such a way to predict whether to buy, sell, or hold an asset based on the raw technical data from that asset. The inputs are technical indicators. The output is a decimal ranging from -1 (sell) to 1 (buy). It is up to the calling strategy to interpret whether values such as 0.5 mean buy or hold. This flexibility allows strategies to optimize such a parameter to achieve better results.

Although the idea behind the system could be used in many trading scenarios, a specific area had to be chosen to do the development and testing. Stocks were favored as the asset class to trade simply due to the quantity of historical data that could be used for training. A time frame of one day was selected because of the reasons discussed earlier. Too small of a time frame causes noisy signals, but too large of a time frame would not allow enough trades to be generated for analysis. This kind of trading style would be classified as swing trading.

It was decided early in development to build the system in Python as opposed to TradeStation. The primary reason for this choice was due to library support for machine learning. Python has established itself as one of the best languages in this area while TradeStation with its Easy Language has very little support. The second reason was due to familiarity. Knowing this would be a large development endeavor, using a new platform was considered a major drawback. Using Python had its own shortcomings though. It is not equipped with a free data feed or backtesting tool, so third party services and libraries had to make due. Data was sourced using the REST API from AlphaVantange, providing not only OHLCV but also plenty of useful indicators. Backtesting was written with the help of the Backtrader library. The neural network library was called Kur, a promising new player in the field built on top of Google's TensorFlow with enhancements to usability and simplicity.

One of the first challenges in building this neural network was calculating its expected output. Based on the ANN signature, the output is supposed to be a value indicating how certain the strategy should buy or sell. Obviously, the bottom of a graph should give a buy value and the top a sell value, but what about in between? An algorithm was developed to solve this problem of finding optimal trades. An example of its results are shown below. Green circles represent positive values or buys and red circles represent negative values or times to sell. The size of the circle is the certainty of the algorithm that the action should be taken. As expected the largest green circles appear at the bottom of cycles and red circles at the top. The code for

this algorithm is given in the appendix under optimal.py. Some might think that this strategy is illegally looking ahead to future data. It is true that it looks ahead, but it should be noted that this algorithm is only run during the training phase of the ANN. When it comes time to backtest the strategy, only the most recent historical data per day is used as input to the ANN.



The next phase of development was preprocessing the data for the network. This was comprised of turning a list of symbols, indicators, and a start and end date into six matrices; one input and one output matrix for the three phases of training, validation, and evaluation. First, the start and end dates were partitioned into three parts for each phase of ANN training. These phases are used to increase the validity and determine the accuracy of the ANN model. The following is repeated for each phase. For each symbol, the indicator data was downloaded or retrieved from local storage. Then optimal trades for each symbol were calculated in the time period. Depending on a parameter to the model, the ANN was given a certain number of prior days of data as input also. The matrices for each symbol were concatenated together to give the final input and output form to train the model with. An example of what this might look like is given below using n symbols, m indicators, p dates, and D prior days. There are n*p rows of data to train with. For each row, there are m*(D+1) inputs and a single output to the ANN.

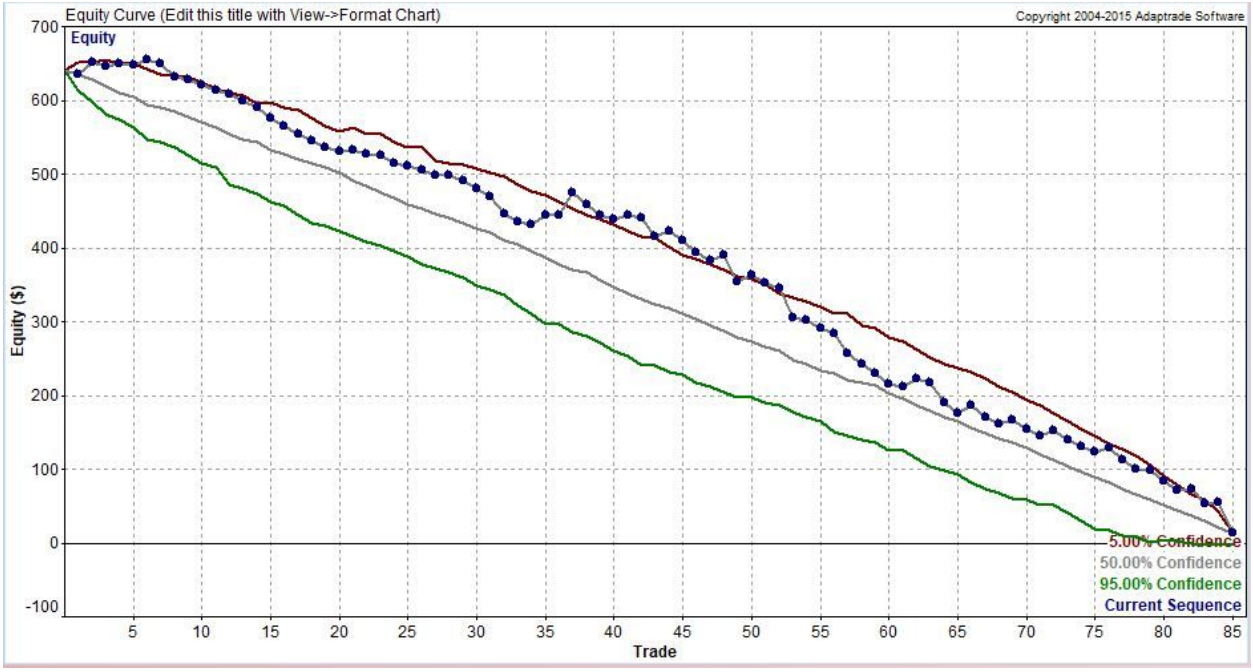
Symbols=S_1-S_n, Indicators=I_1-I_m(s,t), Dates=T_1-T_p, PriorDays=D, OptimalCertainty=Opt(s,t)

Input								Output
$I_1(S_1, T_1)$	...	$I_m(S_1, T_1)$	$I_1(S_1, T_{1-1})$	...	$I_1(S_1, T_{1-D})$	...	$I_m(S_1, T_{1-D})$	$Opt(S_1, T_1)$
...		...	...		...		...	...
$I_1(S_1, T_p)$	...	$I_m(S_1, T_p)$	$I_1(S_1, T_{p-1})$	...	$I_1(S_1, T_{p-D})$	...	$I_m(S_1, T_{p-D})$	$Opt(S_1, T_p)$
...		...	...		...		...	...
$I_1(S_n, T_1)$	...	$I_m(S_n, T_1)$	$I_1(S_n, T_{1-1})$	...	$I_1(S_n, T_{1-D})$	...	$I_m(S_n, T_{1-D})$	$Opt(S_n, T_1)$
...		...	...		...		...	...
$I_1(S_n, T_p)$	...	$I_m(S_n, T_p)$	$I_1(S_n, T_{p-1})$	...	$I_1(S_n, T_{p-D})$	...	$I_m(S_n, T_{p-D})$	$Opt(S_n, T_p)$

After preprocessing the data into a suitable format for the ANN, the next step was to build and train the ANN model. Doing this relied on selecting reasonable parameters for the model. These parameters include the number of epochs to train for, number of hidden layers, number of nodes per layer, the type of activation function, and the type of loss function. The values for these parameters were chosen through an optimization process, in which a single parameter was changed at a time until the evaluation metric of the model was maximized. This resulted in the following values: epochs=200, hidden layers=1, nodes=64, activation=tanh, loss=mean squared error. An AI researcher with more experience could have justified values for these parameters instead of using a simple optimization technique done by hand. The architectural design of the ANN is the main source for how the overall strategy performs. Therefore, it is also the best place for experimentation in an attempt to improve its results. The file neural.py contains this part of development.

The last step of building this system was to use the ANN to place trades in backtesting. This section of the code utilized the Strategy module of the python library Backtrader. A strategy was given a trained ANN, a symbol to trade, a time period to trade in, and a buy/sell threshold for optimization. This threshold determined whether the output from the neural network was significant enough to place an order. An example threshold might be 0.8, meaning that if the ANN output was above 0.8 or below -0.8 the strategy would buy or sell respectively. To use the ANN properly, the same type of inputs used in training had to be supplied. This meant retrieving specific indicator data for the symbol in question and formatting it appropriately. Once that was done, the backtesting could begin. For each day in the trading period, the symbol data on that day was fed as input into the ANN and a value between -1 and 1 was returned. If the output passed the threshold parameter a trade was placed. The exception to this was if there was already a position in the market. The strategy only took one short or long position at a time. Position sizing was not factored into the ANN so it was fixed at 10 shares per trade. Besides trades the ANN triggered, there was not an exit strategy to get out of a bad trade. The purpose of this was to test whether just the ANN alone was enough to make a system profitable. The code for this part is shown in strategy.py.

Once the system was developed, it was able to be tested. A portfolio of 10 stocks to trade was selected from Yahoo's screener for undervalued growth stocks. The neural network was trained for these stocks with 12 technical indicators: daily price, SMA, EMA, MACD, stochastic oscillator, RSI, ADX, CCI, Aroon, Bollinger bands, Chaikin A/D, and volume. Training was performed with data from 2007 to 2017 with a week of past data added to each row of the training matrix. The time period tested was the same as what was used for the other two systems, June 1st to October 12th. The code for this setup can be found in example.py. The results were not good. The system lost $626 from an initial pool of $10,000. Only 18 out of the 85 total trades were profitable. With a Sharpe ratio of -1.951 and a monthly return of -1.284%, it is safe to say that this system should not be traded. The max drawdown was basically the whole entire trading period since the system never reached a peak after the first 6 days. The max number of consecutive losses was 13 trades. For a trading system to make money in the long run, there are three important metrics that need to be positive - expectancy, expectunity, and system quality. For this trading system, these three metrics were -0.64, -151, and -8.55 respectively. Below is the monte carlo analysis showing that even in the top 5% of trade scenarios the equity steadily declines. Looking at just about any measure of performance would lead to the same conclusion that this system was a failure and should not be traded.



CHAPTER 8: Gap Strategy

The next strategy used was a type of gap strategy. The strategy was a day trading system that traded on equities. The first part of the strategy was to find stocks that would fit the strategy well. Instead of scanning the pre-markets or looking in the news to try and find some stocks that looked as if they were going to gap the next day, a scanner was used to find certain stocks that were statistically likely to gap and cover the gap as well. The scanner that was used to look for 4 distinct characteristics of an equity.

- Percentage of days the equity had a gap of at least 1%.
    - To find stocks that were likely to gap on that day.
    - To reduce the number of stocks in the portfolio to make it more manageable.
- Percentage of time the equity covered the 1% gap.
    - Looking for stocks that will cover the gap more often than not.
- The number of days the equity had a gap of 1% recently.
    - Looking to make the other percentages statistically significant so that the sample is not based on 5 days of data but at least 60 days.
- The percentage of times that the gap was extended by a certain amount.
    - Looking to reduce the amount of possible drawdown on a trade.
    - Looking to reduce the possible amount of money lost on a trade in a day

The idea with these criteria was to select a small portfolio of stocks that were proven to cover the gap a statistically significant amount of time.

This strategy looks for gaps in the closing and opening price of at least 1% and assumes that this gap is going to be covered at some point in the day. At the end of the day all open trades are closed and the system will take its loss for the day and start the next day on a blank slate.

This trading system was an auto trading system as it would be too difficult to manage multiple equities at the same time. As stated earlier the entry rule for the system is that there is at least a 1% difference between the opening and closing price. This difference can be positive or negative and a short or long position would be made accordingly. At the time the strategy was made there was no position sizing. It was a constant amount that would later be a parameter that would be used for optimizing the strategy. The exit rules were very simple for the strategy. The trade was ended when the gap was covered or at the end of the day. A stop

loss was tried for a little, but in the end just resulted in less profit because a significant number possible winning trades were being cut short.

To evaluate the system backtesting is what was used. The software used for trading was a software package called Trade Station. Trade Station has a lot of useful tools for trading, such as the scanner that was used to select the portfolio of stocks. They also have tool called portfolio maestro which is very useful for running back tests, optimizing, and running other experiments on a portfolio. For back testing purposes the software looked at 3 months of data from June first to October twelth with $10,000 of initial capital. Through back testing the scanner changed from what it was originally was and the strategy lost only a little money. In the beginning the scanner did not look for the 4th criteria of minimizing times the stock dropped 3% on a trade, but after this was added and the portfolio was changed the system did not lose nearly as much money. After the portfolio was selected, the strategy was optimized to look for the best position size possible for all the trades. However, because it was just the position size changing it recommended a very large position size when it was making money. The final decision on position sizing was 50% of the initial portfolio size. This minimized the cost of commissions as it allowed bigger wins to win more

All trades in this system were done using back testing however the system was not modified at all for the months that it that the results are based on so it can be assumed that those trades were live trades.

The final result of the system was not optimal by looking at almost every performance measure. In the end the system lost $2,256 of the original $10,000 from June 1st to October 23. Out of 2470 trades only 48.87% were profitable and as seen by the amount lost the losing trades were usually bigger than the winning trades.The system had a sharp ratio of -0.2977 and an average daily, weekly, and monthly return rate of 1.7%, -8.94%, and 25.13% with standard deviations of 21.57%, 65.41%, and 83.83% respectively. This goes to show that the system is wildly inconsistent and any thing could happen month to month or even day to day. The expectancy, expectunity and system quality for this system were -0.01, -53.28, and -3.24 respectively. The maximum drawdown from all of the trades was $1,379. The portfolio peeked on July 23rd at a liquid value of $18,260. The worst day the portfolio had was on August 15th when it lost $8,133, so just missing this one day would have had the portfolio making $4,201 instead. For a more general sense of how the trading went look at figure 8.1 below. There were also some monte carlo tests that were run on these trades. In only the bottom 15% of run throughs would the system not be profitable for average returns. Figure 8.1 also shows some confidence lines on how the system would do. You can see that the trades from the system went out of these bounds which shows that the system is very volatile and not what a system should look like. One thing to note which was interesting was that for every trade both entering and leaving there was a $4.50 commision fee. So after the 2470 trades there was a total commision fee of $22,279. Through all of these results which are basic measures of the quality of a system, and the fact that commissions are a part of every traders life the conclusion is the system is not reliable and should not be traded until modifications can be made.

Future work could go into the system to know more about what was happening to each symbol in recent days to better predict if it would cover or not. Also because a lot of the

symbols are in the same area of real world trade paying more attention to the news in which these symbols are based on could help prevent massive losses.

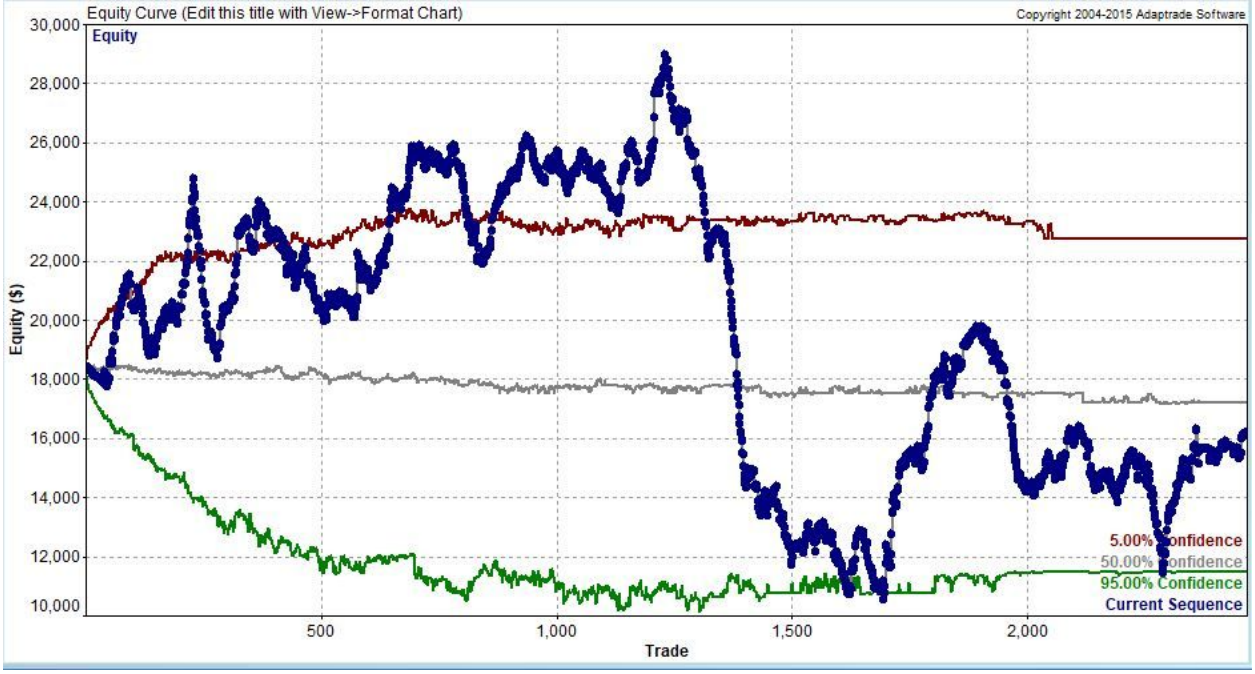
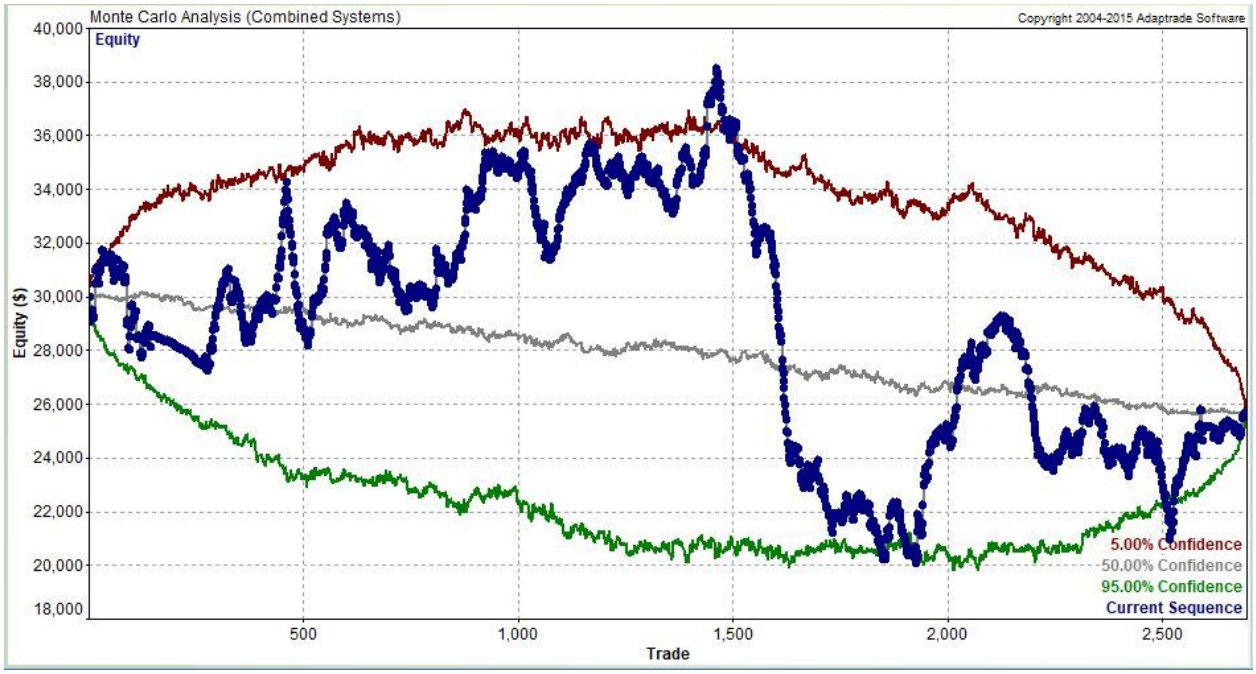


Figure 8.1

CHAPTER 9: System of Systems

As stated in the problem statement, one of the goals of this project is to understand the benefits of a diversified system. While an individual system can diversify by choosing varied stocks, another approach is to create a system of systems. The idea is that by using many systems in harmony, funds can be dynamically allocated to each system based on their performance and return rates become more stable. However, for this technique to work the individual trading systems should take advantage of increased funds. Both the neural network system and the gap strategy system did not alter its trading tendencies based on how much capital it had at its disposal. Because of this, diversification of the systems presented here will see less benefit than those that change trading behavior based on capital.

There are a few methods for aggregating multiple systems. The first is to just collect the data from each system and analyze it together. To achieve this method of aggregation, each of the three systems were given an initial fund of $10,000. The systems were backtested in the same conditions as outlined in earlier chapters. This produced identical individual results but the goal is to analyze them collectively. As a result, this naively aggregated system of systems lost $4,337 and produced an expectancy of -$1.10, a yearly expectunity of $8,135, and a system quality of -36. Given that each system individually lost money, it makes sense that each of these performance metrics are worse off when the systems are combined. The Monte Carlo analysis below shows a similar story. Only in the top 5% of runs and stopping before going negative, would the system make money.



The second method of producing a system of systems is to reward the better systems with more money. To do this, the system quality metric can be used to compare the performance of each system. In the order they were presented, the systems scored ---, -8.55, -3.24. Based on these values, it makes sense to give the largest portion of the funding to the last system and the smallest portion to the second. Unfortunately, due to how the systems were developed, allocating more or less money to any one system would not actually change the results. Ideally, a system would alter entry, exit, and sizing logic relative to its available equity. Another technique for improving a group of systems is to develop rules for activating, suspending, and retiring systems based on their performance.

CHAPTER 10: Summary and Conclusions

The results of the systems were not promising. None of the systems that were created were profitable in the long run.

The largest barrier in this project was the lack of familiarity the group had with trading and analysis software. No one in the group had previously used TradeStation or System Market Analyzer. There was a lot of extra time spent learning how to use the software rather than being able to utilize the software immediately.

Seeing that all the systems that were created were not profitable some more research could be done for all of the systems. For the gap strategy the lack of position sizing likely hurt the system. Running more tests as to whether or not there are indicators as to how large the trade should be would help the system. Also a more thorough scan of stocks from the screener to avoid stocks that are likely to be correlated with each other would reduce the risk for the system as a whole.

Possibly the one bright side to the neural network development is the number of possibilities to improve it. Some simple enhancements could be the addition of an exit strategy or position sizing rules. This would limit the size of losses and potentially increase the size of winning trades. Other ways to tune the system might be changing the method of evaluating ANN models or changing the calculation of ANN output for training. These two changes would have a large effect on the overall outcome since it is the core logic of how trades are determined. There could be a lot of experimentation with these factors. Lastly, what the ANN models lacked in this trading system were features or measurable characteristics. Instead of having the network turn a list of 20 indicators into a single number representing whether or not to trade, it would have been better to first have the ANN pick out features from the indicators such as trends. It would then turn a list of features into a single number. This process is called feature engineering and is considered one of the hardest yet most important aspects of developing a successful model [Brownlee].

REFERENCES

"Asset Class - Overview and Different Types of Asset Classes." *Corporate Finance Institute*, corporatefinanceinstitute.com/resources/knowledge/trading-investing/asset-class/.

Altay, Erdinç & Hakan Satman, M. (2005). "Stock Market Forecasting: Artificial Neural Network and Linear Regression Comparison in An Emerging Market". *Journal of Financial Management and Analysis*. 18.

Balasubramaniam, Kesavan. "How Does Margin Trading in the Forex Market Work?" *Investopedia*, Investopedia, 25 June 2018, www.investopedia.com/ask/answers/06/forexmargin.asp.

Brownlee, Jason. "Discover Feature Engineering, How to Engineer Features and How to Get Good at It." *Machine Learning Mastery*, 4 Apr. 2018, machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/.

Bysshe, Geoff. *Trading the 10 O'clock Bulls*. Dataview LLC., 2004.

Chen, James. "Technical Analysis." *Investopedia*, Investopedia, 13 Dec. 2018, www.investopedia.com/terms/t/technicalanalysis.asp.

Chen, James. "Trading Platform." *Investopedia*, Investopedia, 13 Dec. 2018, www.investopedia.com/terms/t/trading-platform.asp.

"Compound Annual Growth Rate (Annualized Return)." *CAGR of the Stock Market: Annualized Returns of the S&P 500*, www.moneychimp.com/features/market_cagr.htm.

"Day-Trading Margin Requirements: Know the Rules." *Types of Investments | FINRA.org*, 19 Sept. 2018, www.finra.org/investors/day-trading-margin-requirements-know-rules.

Firstrade Securities Inc. "Margin Requirements." *Margin Requirements | Initial & Maintenance Margin Requirements*, www.firstrade.com/content/en-us/education/margin/marginrequirements/.

Folger, Jean. "Different Styles For Trading." *Investopedia*, Investopedia, 15 Mar. 2018, www.investopedia.com/university/how-start-trading/how-start-trading-trading-styles.asp.

Folger, Jean. "What Is the Difference between Investing and Trading?" *Investopedia*, Investopedia, 9 Jan. 2019, www.investopedia.com/ask/answers/12/difference-investing-trading.asp.

Frankenfield, Jake. “Asset Class.” *Investopedia*, Investopedia, 13 Dec. 2018, www.investopedia.com/terms/a/assetclasses.asp.

Fundora, Joey. “Multiple Time Frames Can Multiply Returns.” *Investopedia*, Investopedia, 24 Nov. 2018, www.investopedia.com/articles/trading/07/timeframes.asp.

Galant, Mark, and Brian Dolan. *Currency Trading for Dummies*. John Wiley, 2011.

Karn, Ujjwal. “A Quick Introduction to Neural Networks.” *The Data Science Blog*, 10 Aug. 2016, ujjwalkarn.me/2016/08/09/quick-intro-neural-networks/.

Kenton, Will. “Broker.” *Investopedia*, Investopedia, 13 Dec. 2018, www.investopedia.com/terms/b/broker.asp.

Kenton, Will. “Fundamental Analysis.” *Investopedia*, Investopedia, 13 Dec. 2018, www.investopedia.com/terms/f/fundamentalanalysis.asp.

Kordos, Miroslaw & Cwiok, Andrzej. (2011). “A New Approach to Neural Network Based Stock Trading Strategy”. 429-436. 10.1007/978-3-642-23878-9_51.

Kryzanowski, Lawrence, et al. “Using Artificial Neural Networks to Pick Stocks.” *Financial Analysts Journal*, vol. 49, no. 4, 1993, pp. 21–27. JSTOR, JSTOR, www.jstor.org/stable/4479664.

Kuepper, Justin. “What Is A Trading System?” *Investopedia*, Investopedia, 5 Mar. 2018, www.investopedia.com/university/tradingsystems/tradingsytems1.asp.

“Gap Trading Strategies.” *StockCharts.com*, 9 Oct. 2017, stockcharts.com/school/doku.php?id=chart_school%3Atrading_strategies%3Agap_trading_strategies.

Hunt, David. “Forex Taxation Basics.” *Investopedia*, Investopedia, 24 May 2018, www.investopedia.com/articles/forex/09/forex-taxation-basics.asp.

Maverick, J.B. “What Is the Average Annual Return for the S&P 500?” *Investopedia*, Investopedia, 4 Jan. 2019, www.investopedia.com/ask/answers/042415/what-average-annual-return-sp-500.asp.

Radzicki, Michael. “Basic Macroeconomics.”

Radzicki, Michael. “Inter-Market Analysis & Sector Rotation.”

Radzicki, Michael. “The Big Picture.”

Radzicki, Michael. “The Breadth of the Market.”

Voegt, Theodore. (2017). “Artificial Neural Networks in Trading Systems”.

Vonko, Dima. “Neural Networks: Forecasting Profits.” *Investopedia*, Investopedia, 4 Jan. 2019, www.investopedia.com/articles/trading/06/neuralnetworks.asp.

Wanjawa, Barack & Muchemi, Lawrence. (2014). “ANN Model to Predict Stock Prices at Stock Exchange Markets”.

Woodford, Chris. “How Neural Networks Work - A Simple Introduction.” *Explain That Stuff*, 14 Mar. 2018, www.explainthatstuff.com/introduction-to-neural-networks.html.

Appendix A: Performance Reports

10 O'Clock Bulls in FOREX

Market System: MarketSystem4

Account Settings

Starting Equity: $10,000.00
Trading Vehicle: Stocks
Minimum Margin Requirement: 100.00%
Slippage per side: $0.00 per trade
Commissions and fees per side: $0.00 per trade

Input Data Settings

Profit/loss and risk calculated from P/L and risk inputs.
Trades File: Undef

Position Sizing Settings & Rules

Position Sizing Method: None
No. Shares: From input data

	All Trades	Long
Total Net Profit	($1,480.00)	($1,480.00)
Gross Profit	$10,259.80	$10,259.80
Gross Loss	($11,739.80)	($11,739.80)
Profit Factor	0.8739	0.8739
Pessimistic Return Ratio	0.6917	0.6917
Starting Account Equity	$10,000.00	
Highest Closed Trade Equity	$11,719.60	
Lowest Closed Trade Equity	$7,716.90	
Additions to Equity	$0.00	
Withdrawals From Equity	$0.00	
Final Account Equity	$8,520.00	
Return on Starting Equity	-14.80%	
Total Number of Trades	147	147
Number of Winning Trades	81	81
Number of Losing Trades	66	66
Trades Not Taken	0	0
Percent Profitable	55.10%	55.10%
Max Number of Shares	1	1
Minimum Number of Shares	1	1
Average Number of Shares	1	1
Largest Winning Trade	$1,286.00	$1,286.00
/Percent of Equity	13.96%	13.96%
Largest Winning Trade (%)	13.96%	13.96%
/Trade Value	$1,286.00	$1,286.00
Average Winning Trade	$126.66	$126.66
Average Winning Trade (%)	1.338%	1.338%
Average R-Multiple, Wins	0.000	
Max Number Consecutive Wins	6	6

Largest Losing Trade	($1,735.00)	($1,735.00)
/Percent of Equity	-16.21%	-16.21%
Largest Losing Trade (%)	-16.21%	-16.21%
/Trade Value	($1,735.00)	($1,735.00)
Average Losing Trade	($177.88)	($177.88)
Average Losing Trade (%)	-1.770%	-1.770%
Average R-Multiple, Losses	0.000	
Max Number Consecutive Losses	6	6
Average Trade (Expectation)	($10.07)	($10.07)
Average Trade (%)	-0.05786%	-0.05786%
Trade Standard Deviation	$307.07	$307.07
Trade Standard Deviation (%)	3.161%	3.161%
Win/Loss Ratio	0.7121	0.7121
Win/Loss Ratio (%/%)	0.7555	0.7555
Return/Drawdown Ratio	0.000	0.000
Modified Sharpe Ratio	-0.01830	-0.01830
Average Risk	$0.00	
Average Risk (%)	0.000%	
Average R-Multiple (Expectancy)	0.000	
R-Multiple Standard Deviation	0.000	
Average Margin	$0.00	
Average Margin (%)	0.000%	
Max Margin (%)	0.000%	
/Margin Value	$0.00	
Average Leverage	0.000	
Risk of Ruin	631200.%	

Closed Trade Drawdowns	All Trades	Long
Number of Drawdowns	4	4
Average Drawdown	($1,407.20)	($1,407.20)
Average Drawdown (%)	12.45%	12.45%
Average Trades in Drawdowns	34	34
Worst Case Drawdown	($4,002.70)	($4,002.70)
/Percent of Equity	34.15%	34.15%
Trade Number at Trough	124	124
Trades in Drawdown	114	114
Worst Case Drawdown (%)	34.15%	34.15%
/Equity Value	($4,002.70)	($4,002.70)
Trade Number at Trough	124	124
Trades in Drawdown	114	114

Annual Returns

Year	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac	Sharpe

Monthly Returns

Month	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac

Weekly Returns

Week	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac

Daily Returns

Day	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac

Dependency Analysis Results for All Trades

Dependency: Negative

Confidence Level: 29.53%
Total Number of Runs: 76
Number of Runs of Wins: 38
Number of Runs of Losses: 38
Average Length of Winning Runs: 2.1
Average Length of Losing Runs: 1.7

Significance Test Settings

Number of rules and/or restrictions in trading system or method: 0
Confidence level for confidence intervals around average trade: 95.00%

Significance Test Results

Number of Trades: 147
Number of Degrees of Freedom: 147
Average trade at 95.00% confidence: $-10.07 +/- 41.66
Worst-case average trade at 95.00% confidence: $-51.73
Probability that average trade is greater than zero: 34.85%

<System/method may not be profitable at specified confidence level>

Artificial Neural Network

Market System: MarketSystem1

Account Settings

Starting Equity: $10,000.00
Trading Vehicle: Stocks
Minimum Margin Requirement: 100.00%
Slippage per side: $0.00 per trade
Commissions and fees per side: $2.50 per trade

Input Data Settings

Profit/loss and risk calculated from P/L and risk inputs.
Trades File: Undef

Position Sizing Settings & Rules

Position Sizing Method: None
No. Shares: From input data

	All Trades	Long	Short
Total Net Profit	($626.53)	($370.77)	($255.75)
Gross Profit	$142.35	$61.20	$81.15
Gross Loss	($768.88)	($431.97)	($336.91)
Profit Factor	0.1851	0.1417	0.2409
Pessimistic Return Ratio	0.1261	0.08126	0.1354
Trading Period	6/6/2018 to 10/11/2018 (129 days 0 min)		
Starting Account Equity	$10,000.00		
Highest Closed Trade Equity	$10,000.00		
Lowest Closed Trade Equity	$9,373.47		
Additions to Equity	$0.00		
Withdrawals From Equity	$0.00		
Final Account Equity	$9,373.47		
Return on Starting Equity	-6.265%		
Total Number of Trades	85	47	38
Number of Winning Trades	18	9	9
Number of Losing Trades	67	38	29
Trades Not Taken	0	0	0
Percent Profitable	21.18%	19.15%	23.68%
Max Number of Shares	10	10	10
Minimum Number of Shares	10	10	10
Average Number of Shares	10	10	10
Largest Winning Trade	$30.62	$16.22	$30.62
/Percent of Equity	0.3088%	0.1636%	0.3088%
Largest Winning Trade (%)	0.3088%	0.1636%	0.3088%
/Trade Value	$30.62	$16.22	$30.62
Average Winning Trade	$7.91	$6.80	$9.02
Average Winning Trade (%)	0.08144%	0.06989%	0.09299%
Average R-Multiple, Wins	0.000		

Average Length of Wins	14 days 16 hours	12 days 8 hours	17 days 0 min
Max Number Consecutive Wins	2	2	2
Largest Losing Trade	($41.11)	($35.71)	($41.11)
/Percent of Equity	-0.4350%	-0.3691%	-0.4350%
Largest Losing Trade (%)	-0.4350%	-0.3691%	-0.4350%
/Trade Value	($41.11)	($35.71)	($41.11)
Average Losing Trade	($11.48)	($11.37)	($11.62)
Average Losing Trade (%)	-0.1183%	-0.1169%	-0.1202%
Average R-Multiple, Losses	0.000		
Average Length of Losses	8 days 21 hours	7 days 8 hours	10 days 23 hours
Max Number Consecutive Losses	13	8	11
Average Trade (Expectation)	($7.37)	($7.89)	($6.73)
Average Trade (%)	-0.07602%	-0.08110%	-0.06975%
Trade Standard Deviation	$11.36	$9.85	$13.10
Trade Standard Deviation (%)	0.1175%	0.1012%	0.1362%
Win/Loss Ratio	0.6891	0.5981	0.7762
Win/Loss Ratio (%/%)	0.6883	0.5981	0.7733
Return/Drawdown Ratio	0.000	0.000	0.000
Modified Sharpe Ratio	-0.6469	-0.8015	-0.5120
Sharpe Ratio	-1.951		
Average Risk	$0.00		
Average Risk (%)	0.000%		
Average R-Multiple (Expectancy)	0.000		
R-Multiple Standard Deviation	0.000		
Average Margin	$1,603.03		
Average Margin (%)	16.49%		
Max Margin (%)	29.39%		
/Margin Value	$2,914.87		
Date of Max Margin	6/22/2018		
Average Leverage	0.02824		
Risk of Ruin	-1.#I%		
Average Annual Profit/Loss	($1,773.95)		
Ave Annual Compounded Return	0.000%		
Average Monthly Profit/Loss	($147.83)		
Ave Monthly Compounded Return	0.000%		
Average Weekly Profit/Loss	($32.25)		
Ave Weekly Compounded Return	0.000%		
Average Daily Profit/Loss	($4.86)		
Ave Daily Compounded Return	0.000%		

Closed Trade Drawdowns	All Trades	Long	Short
Number of Drawdowns	1	1	2
Average Drawdown	($626.53)	($370.77)	($152.19)

Average Drawdown (%)	6.265%	3.708%	1.521%
Average Length of Drawdowns	127 days 0 min	127 days 0 min	73 days 12 hours
Average Trades in Drawdowns	85	47	18
Worst Case Drawdown	($626.53)	($370.77)	($278.48)
/Percent of Equity	6.265%	3.708%	2.783%
Date at Trough	10/11/2018	10/11/2018	10/3/2018
Trade Number at Trough	85	85	79
Length of Drawdown	127 days 0 min	127 days 0 min	129 days 0 min
Trades in Drawdown	85	47	34
Worst Case Drawdown (%)	6.265%	3.708%	2.783%
/Equity Value	($626.53)	($370.77)	($278.48)
Date at Trough	10/11/2018	10/11/2018	10/3/2018
Trade Number at Trough	85	85	79
Length of Drawdown	127 days 0 min	127 days 0 min	129 days 0 min
Trades in Drawdown	85	47	34
Longest Drawdown	127 days 0 min	127 days 0 min	129 days 0 min
Start of Drawdown	6/6/2018	6/6/2018	6/4/2018
End of Drawdown	10/11/2018	10/11/2018	10/11/2018
Percent of Equity	6.265%	3.708%	2.783%

Annual Returns

Year	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac	Sharpe
2018	($626.53)	$0.00	$9,373.47	-6.265	6.265	85	21.18	0.1851	-0.5953
Ave	**($626.53)**	**$0.00**	**$9,373.47**	**-6.265**	**6.265**	**85.00**	**21.18**	**0.1851**	**-0.5953**
SD	**$0.00**	**$0.00**	**$0.00**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**

Monthly Returns

Month	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac
10-2018	($37.53)	$0.00	$9,373.47	-0.3988	0.3988	8	25.00	0.3241
9-2018	($201.12)	$0.00	$9,411.00	-2.092	2.092	17	23.53	0.03994
8-2018	($94.48)	$0.00	$9,612.12	-0.9734	1.055	20	35.00	0.3623
7-2018	($170.55)	$0.00	$9,706.61	-1.727	1.727	21	14.29	0.08312
6-2018	($122.84)	$0.00	$9,877.16	-1.228	1.228	19	10.53	0.2760
Ave	**($125.31)**	**$0.00**	**$9,596.07**	**-1.284**	**1.300**	**17.00**	**21.67**	**0.2171**
SD	**$64.14**	**$0.00**	**$209.34**	**0.6581**	**0.6495**	**5.244**	**9.630**	**0.1461**

Weekly Returns

Week	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac
10/7/2018	($14.57)	$0.00	$9,373.47	-0.1552	0.3462	5	40.00	0.5527
9/30/2018	($22.97)	$0.00	$9,388.04	-0.2440	0.2440	3	0.000	0.000
9/23/2018	($130.17)	$0.00	$9,411.00	-1.364	1.364	7	14.29	0.002941
9/16/2018	$4.50	$0.00	$9,541.17	0.04719	0.000	1	100.0	100.0
9/9/2018	($42.77)	$0.00	$9,536.67	-0.4464	0.4641	6	33.33	0.07529
9/2/2018	($32.69)	$0.00	$9,579.43	-0.3401	0.3401	3	0.000	0.000
8/26/2018	($31.86)	$0.00	$9,612.12	-0.3304	0.5299	6	33.33	0.3921
8/19/2018	$4.85	$0.00	$9,643.99	0.05035	0.03519	3	66.67	2.429

8/12/2018	($39.63)	$0.00	$9,639.13	-0.4095	0.4095	4	25.00	0.2211
8/5/2018	($28.97)	$0.00	$9,678.76	-0.2984	0.3567	5	20.00	0.1633
7/29/2018	($29.73)	$0.00	$9,707.73	-0.3053	0.3169	6	16.67	0.2117
7/22/2018	($23.26)	$0.00	$9,737.46	-0.2383	0.2383	5	40.00	0.3362
7/15/2018	($28.29)	$0.00	$9,760.73	-0.2890	0.2890	4	0.000	0.000
7/8/2018	($35.71)	$0.00	$9,789.02	-0.3635	0.4009	5	20.00	0.09343
7/1/2018	($52.43)	$0.00	$9,824.73	-0.5308	0.5308	3	0.000	0.000
6/24/2018	($34.20)	$0.00	$9,877.16	-0.3450	0.7010	9	22.22	0.5780
6/17/2018	($52.55)	$0.00	$9,911.35	-0.5274	0.5274	5	0.000	0.000
6/10/2018	($25.67)	$0.00	$9,963.90	-0.2570	0.2570	4	0.000	0.000
6/3/2018	($10.43)	$0.00	$9,989.57	-0.1043	0.1043	1	0.000	0.000
Ave	**($32.98)**	**$0.00**	**$9,682.39**	**-0.3395**	**0.3924**	**4.474**	**22.71**	**5.529**
SD	**$28.22**	**$0.00**	**$185.77**	**0.2945**	**0.2937**	**1.954**	**26.32**	**22.88**

Daily Returns

Day	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac
10/11/2018	($22.15)	$0.00	$9,373.47	-0.2357	0.2357	2	0.000	0.000
10/10/2018	($2.43)	$0.00	$9,395.62	-0.02589	0.1108	2	50.00	0.7665
10/9/2018	$10.02	$0.00	$9,398.05	0.1067	0.000	1	100.0	100.0
10/4/2018	($3.40)	$0.00	$9,388.04	-0.03625	0.03625	1	0.000	0.000
10/3/2018	($19.56)	$0.00	$9,391.44	-0.2079	0.2079	2	0.000	0.000
9/28/2018	($55.84)	$0.00	$9,411.00	-0.5898	0.5898	2	0.000	0.000
9/26/2018	($13.80)	$0.00	$9,466.84	-0.1455	0.1455	1	0.000	0.000
9/25/2018	($39.31)	$0.00	$9,480.63	-0.4129	0.4169	2	50.00	0.009675
9/24/2018	($21.23)	$0.00	$9,519.94	-0.2225	0.2225	2	0.000	0.000
9/19/2018	$4.50	$0.00	$9,541.17	0.04719	0.000	1	100.0	100.0
9/13/2018	($6.58)	$0.00	$9,536.67	-0.06899	0.06899	1	0.000	0.000
9/11/2018	($25.47)	$0.00	$9,543.25	-0.2661	0.2848	2	50.00	0.06550
9/10/2018	($10.72)	$0.00	$9,568.72	-0.1119	0.1296	3	33.33	0.1367
9/6/2018	($9.63)	$0.00	$9,579.43	-0.1004	0.1004	1	0.000	0.000
9/4/2018	($23.06)	$0.00	$9,589.06	-0.2399	0.2399	2	0.000	0.000
8/31/2018	($5.15)	$0.00	$9,612.12	-0.05358	0.05358	1	0.000	0.000
8/30/2018	($26.56)	$0.00	$9,617.28	-0.2754	0.2754	1	0.000	0.000
8/27/2018	($0.15)	$0.00	$9,643.83	-0.001576	0.2018	4	50.00	0.9927
8/23/2018	($3.40)	$0.00	$9,643.99	-0.03519	0.03519	1	0.000	0.000
8/21/2018	$8.25	$0.00	$9,647.38	0.08557	0.000	2	100.0	100.0
8/17/2018	($35.71)	$0.00	$9,639.13	-0.3691	0.3691	1	0.000	0.000
8/16/2018	$11.25	$0.00	$9,674.84	0.1164	0.000	1	100.0	100.0
8/15/2018	($5.20)	$0.00	$9,663.59	-0.05378	0.05378	1	0.000	0.000
8/13/2018	($9.97)	$0.00	$9,668.79	-0.1030	0.1030	1	0.000	0.000
8/9/2018	$5.65	$0.00	$9,678.76	0.05845	0.000	1	100.0	100.0
8/7/2018	($26.23)	$0.00	$9,673.11	-0.2704	0.2704	3	0.000	0.000
8/6/2018	($8.40)	$0.00	$9,699.33	-0.08653	0.08653	1	0.000	0.000
8/3/2018	($6.86)	$0.00	$9,707.73	-0.07058	0.07058	1	0.000	0.000
8/2/2018	$7.98	$0.00	$9,714.59	0.08224	0.000	1	100.0	100.0
7/30/2018	($30.86)	$0.00	$9,706.61	-0.3169	0.3169	4	0.000	0.000
7/25/2018	($14.55)	$0.00	$9,737.46	-0.1492	0.1492	1	0.000	0.000
7/24/2018	$5.98	$0.00	$9,752.01	0.06138	0.05951	3	66.67	2.031
7/23/2018	($14.70)	$0.00	$9,746.03	-0.1506	0.1506	1	0.000	0.000
7/18/2018	($28.29)	$0.00	$9,760.73	-0.2890	0.2890	4	0.000	0.000
7/13/2018	$3.68	$0.00	$9,789.02	0.03761	0.000	1	100.0	100.0

7/12/2018	($13.57)	$0.00	$9,785.34	-0.1384	0.1384	1	0.000	0.000
7/10/2018	($16.75)	$0.00	$9,798.90	-0.1706	0.1706	2	0.000	0.000
7/9/2018	($9.07)	$0.00	$9,815.65	-0.09236	0.09236	1	0.000	0.000
7/3/2018	($22.69)	$0.00	$9,824.73	-0.2304	0.2304	1	0.000	0.000
7/2/2018	($29.75)	$0.00	$9,847.41	-0.3011	0.3011	2	0.000	0.000
6/29/2018	($18.49)	$0.00	$9,877.16	-0.1868	0.1868	2	0.000	0.000
6/28/2018	($17.38)	$0.00	$9,895.64	-0.1754	0.1754	1	0.000	0.000
6/27/2018	($4.04)	$0.00	$9,913.03	-0.04077	0.04077	1	0.000	0.000
6/26/2018	($29.82)	$0.00	$9,917.07	-0.2998	0.2998	2	0.000	0.000
6/25/2018	$35.54	$0.00	$9,946.89	0.3585	0.1138	3	66.67	4.145
6/22/2018	($8.06)	$0.00	$9,911.35	-0.08122	0.08122	1	0.000	0.000
6/21/2018	($40.29)	$0.00	$9,919.41	-0.4045	0.4045	3	0.000	0.000
6/20/2018	($4.20)	$0.00	$9,959.70	-0.04215	0.04215	1	0.000	0.000
6/14/2018	($7.70)	$0.00	$9,963.90	-0.07722	0.07722	1	0.000	0.000
6/12/2018	($14.60)	$0.00	$9,971.60	-0.1462	0.1462	2	0.000	0.000
6/11/2018	($3.37)	$0.00	$9,986.19	-0.03378	0.03378	1	0.000	0.000
6/7/2018	($10.43)	$0.00	$9,989.57	-0.1043	0.1043	1	0.000	0.000
6/4/2018	$0.00	$0.00	$10,000.00	0.000	0.000	0	0.000	0.000
Ave	**($11.82)**	**$0.00**	**$9,703.46**	**-0.1219**	**0.1493**	**1.604**	**20.13**	**13.36**
SD	**$15.61**	**$0.00**	**$181.93**	**0.1611**	**0.1289**	**0.9060**	**36.30**	**34.13**

Dependency Analysis Results for All Trades

Dependency: Positive

Confidence Level: 9.85%
Total Number of Runs: 29
Number of Runs of Wins: 14
Number of Runs of Losses: 15
Average Length of Winning Runs: 1.3
Average Length of Losing Runs: 4.5

Significance Test Settings

Number of rules and/or restrictions in trading system or method: 0
Confidence level for confidence intervals around average trade: 95.00%

Significance Test Results

Number of Trades: 85
Number of Degrees of Freedom: 85
Average trade at 95.00% confidence: $-7.37 +/- 2.05
Worst-case average trade at 95.00% confidence: $-9.42
Probability that average trade is greater than zero: 200.00%

<System/method may not be profitable at specified confidence level>

Gap Strategy

Market System: MarketSystem2

Account Settings

Starting Equity: $10,000.00
Trading Vehicle: Futures
Initial Margin: $0.00
Round-turn slippage per contract: $0.00
Round-turn commissions and fees per contract: $0.00

Input Data Settings

Profit/loss and risk calculated from P/L and risk inputs.
Trades File: C:\Users\Adam Maier\Desktop\June_1st_to_October_12.csv

Position Sizing Settings & Rules

Position Sizing Method: None
No. Contracts: From input data

	All Trades	Long
Total Net Profit	($2,256.85)	($2,256.85)
Gross Profit	$139,489.53	$139,489.53
Gross Loss	($141,746.39)	($141,746.39)
Profit Factor	0.9841	0.9841
Pessimistic Return Ratio	0.9295	0.9295
Starting Account Equity	$10,000.00	
Highest Closed Trade Equity	$20,553.95	
Lowest Closed Trade Equity	$2,103.23	
Additions to Equity	$0.00	
Withdrawals From Equity	$0.00	
Final Account Equity	$7,743.15	
Return on Starting Equity	-22.57%	
Total Number of Trades	2,465	2,465
Number of Winning Trades	1,191	1,191
Number of Losing Trades	1,273	1,273
Trades Not Taken	1	1
Percent Profitable	48.34%	48.34%
Max Number of Contracts	5,000	5,000
Minimum Number of Contracts	9	9
Average Number of Contracts	824	824
Largest Winning Trade	$1,493.80	$1,493.80
/Percent of Equity	12.08%	12.08%
Largest Winning Trade (%)	29.62%	29.62%
/Trade Value	$751.07	$751.07
Average Winning Trade	$117.12	$117.12
Average Winning Trade (%)	1.429%	1.429%
Average R-Multiple, Wins	0.000	
Max Number Consecutive Wins	18	18

Largest Losing Trade	($1,162.42)	($1,162.42)
/Percent of Equity	-7.892%	-7.892%
Largest Losing Trade (%)	-19.47%	-19.47%
/Trade Value	($729.72)	($729.72)
Average Losing Trade	($111.35)	($111.35)
Average Losing Trade (%)	-1.298%	-1.298%
Average R-Multiple, Losses	0.000	
Max Number Consecutive Losses	20	20
Average Trade (Expectation)	($0.92)	($0.92)
Average Trade (%)	0.02007%	0.02007%
Trade Standard Deviation	$171.49	$171.49
Trade Standard Deviation (%)	2.503%	2.503%
Win/Loss Ratio	1.052	1.052
Win/Loss Ratio (%/%)	1.101	1.101
Return/Drawdown Ratio	0.000	0.000
Modified Sharpe Ratio	0.008018	0.008018
Average Risk	$0.00	
Average Risk (%)	0.000%	
Average R-Multiple (Expectancy)	0.000	
R-Multiple Standard Deviation	0.000	
Average Margin	$0.00	
Average Margin (%)	0.000%	
Max Margin (%)	0.000%	
/Margin Value	$0.00	
Average Leverage	0.000	
Risk of Ruin	-1.#I%	

Closed Trade Drawdowns	All Trades	Long
Number of Drawdowns	25	25
Average Drawdown	($1,484.25)	($1,484.25)
Average Drawdown (%)	8.342%	8.342%
Average Trades in Drawdowns	95	95
Worst Case Drawdown	($18,450.73)	($18,450.73)
/Percent of Equity	89.77%	89.77%
Trade Number at Trough	1,694	1,694
Trades in Drawdown	1,234	1,234
Worst Case Drawdown (%)	89.77%	89.77%
/Equity Value	($18,450.73)	($18,450.73)
Trade Number at Trough	1,694	1,694
Trades in Drawdown	1,234	1,234

Annual Returns

Year	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac	Sharpe

Monthly Returns

Month	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac

Weekly Returns

Week	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac

Daily Returns

Day	Net Profit	Add/With	End Equity	Return(%)	Drawdown(%)	Trades	Wins(%)	P Fac

Dependency Analysis Results for All Trades

Dependency: Positive

Confidence Level: 99.9978%
Total Number of Runs: 1127
Number of Runs of Wins: 563
Number of Runs of Losses: 564
Average Length of Winning Runs: 2.1
Average Length of Losing Runs: 2.3

Significance Test Settings

Number of rules and/or restrictions in trading system or method: 0
Confidence level for confidence intervals around average trade: 95.00%

Significance Test Results

Number of Trades: 2464
Number of Degrees of Freedom: 2464
Average trade at 95.00% confidence: $-0.92 +/- 5.68
Worst-case average trade at 95.00% confidence: $-6.60
Probability that average trade is greater than zero: 39.57%

<System/method may not be profitable at specified confidence level>

APPENDIX B: 10 O'Clock Bulls in FOREX Code

```
Inputs: startTime (0930), endTime (1000), accountSize(100000), riskPercentage(.01),
stopLoss(100),
alotSize(10000), AverageLength(7);
vars: storedLow(5), storedHigh(0), lotSize(0), HighDate(0), HighTime(0), LowDate(0),
LowTime(0),
SMA(0), SMAsum(0), counter(0), CurrentBarAve(0), SD(0), Middle(0), Leave(0), positionSize(0),
pipMovement(0);


Switch (Time)
   Begin
        Default:
                storedLow = 5;
                storedHigh = 0;
        Case 0930 to 1701:
                If L <= storedLow and T <= 1000 and T >= 0930 then
                        Begin
                                storedLow = L;
                                LowTime = T;
                                LowDate = D;
                        End;
                If H >= storedHigh and T <= 1000 and T >= 0930 then
                        Begin
                                storedHigh = H;
                                HighDate = D;
                                HighTime = T;
                        End;

   {This function takes the stored highs and lows in order to create trendlines
   for a visual representation of what is being recorded}

        If Currentbar > 1 then
                begin
                Value1 = TI_new (HighDate, HighTime, storedHigh, D, T, storedHigh);
                Value2 = TI_new (LowDate, LowTime, storedLow, D, T, storedLow);
```

```
        end;

    {This is the buy portion. It takes the stored low and if it is crossed then
    buys the next bar at market}
    If Low crosses below storedLow and time < 1600 then
                Begin
    {This portion determines if there is support using the Simple Moving Average}

                For counter = 0 to AverageLength - 1
        Begin
                {This sums the average price of each bar }
                SMAsum = SMAsum + ((High + Low + Close)/3)[counter];
        end;

        {This function divides the sum by length to get the SMA}
        If AverageLength <> 0 then
                SMA = SMAsum / AverageLength;

        {This function checks if there is support for the reversal with the SMA}
        If SMA > Low then

        {If support is determined to exist then the buy order is executed}
        {This is the buy order and posiiton sizing rules. The position size is determied through
        the maximum risk the trader wishes to have for their account.}
        {Risk Percentage is the decimal form of the percentage of an account that the trader is
willing
        to risk. ex: 1% = .01}
        {stopLoss is how many pips does the trader wish to risk.}
        {pipMovement is how much each pip movement is worth. Microlot = .1, Minilot = 1,
Standardlot = 10}
                If alotSize = 100000 then
                        Begin
                        pipMovement = 10;
                        end;
                If alotSize = 10000 then
                        Begin
                        pipMovement = 1;
                        end;
                If alotSize = 1000 then
                        Begin
                        pipMovement = .1;
                        end;
```

```
                positionSize = ((accountSize * riskPercentage)/(stopLoss*pipMovement))*
alotSize;
                        Buy ("Entry") positionSize contracts next bar at market;
        end;

   {This is the ideal sell funciton. When the pair reaches the stored high the position is
reversed.}
   If High crosses over storedHigh then
        Begin

        For counter = 0 to AverageLength - 1
                Begin
                SMAsum = SMAsum + ((High + Low + Close)/3)[counter];
                end;

        If AverageLength <> 0 then
                SMA = SMAsum / AverageLength;

        If SMA > High then
                Sellshort ("Exit") next bar at market;
        End;
   {This is the preparation for the end of the day function. If a position has been held for more
than
   2 hours and it is after 3pm then the position will sell if the postion is profitable.}
   If time > 1600 and Barssinceentry > 24 then
        Begin
        If Entryprice < Close then
                Begin
                Sell ("Preping for Close") next bar at market;
                End;
        end;
   {This is the stoploss function. It checks whenever the Low goes below the stored low and
sells}
   If Low < storedLow then
        Begin
        CurrentBarAve = (H + L + C)/3;
        SD = (storedHigh - storedLow)/2;
        Middle = storedLow + SD;
        Leave = Middle - (3 * SD);
                If CurrentBarAve < Leave then
                        Sell next bar at market;
        end;
   {This is the end of the day function. If it is past 5pm then the position will be sold to close out
```

```
the day.}
If Time > 1700 then
        Begin
        Sell ("Timeout") next bar at market;
        end;

End;
```

APPENDIX C: Artificial Neural Network Code

analysis.py

```
import numpy as np

def get_accuracy(output, tolerance=0.5):
    if output:
        diff = np.abs(output['truth']['out'] - output['result']['out']) < tolerance
        correct = diff.sum()
        total = len(diff)
        return correct / total


def get_average_distance(output):
    if output:
        diff = np.abs(output['truth']['out'] - output['result']['out'])
        total_diff = diff.sum()
        total = len(diff)
        return total_diff / total
```

data.py

```
from utility import *
import traceback
import os


class DataException(Exception):
    pass


class Data:

    def __init__(self, **params):
        self.params = params
        self.path = None
        self.data = None
```

```
    self.get_path()
    self.make_path()
    self.write_params()
    self.data = self.get_data()

def data_error_msg(self):
    return 'Failed to get data for ' + str(self.params)

def get_data(self):
    try:
        if self.data is None:
            self.data = self.read_data()
            if self.data is None:
                self.data = self.get_new_data()
                if self.data is None:
                    raise DataException(self.data_error_msg())
                self.write_data()
        return self.data
    except Exception:
        raise DataException(traceback.format_exc() + '\n' + self.data_error_msg())

def get_base_path(self):
    if not self.path:
        file_name = shorten_path(encrypt_dict(self.params))
        cwd = os.getcwd()
        self.path = os.path.join(cwd, PARAMS['data_folder'], self.get_folder(), file_name)
    return self.path

def get_path(self, *paths):
    return os.path.join(self.get_base_path(), *paths)

def make_path(self):
    make_path(self.get_params_path())

def get_params_path(self):
    return self.get_path('params.pkl')

def write_params(self):
    write_pickle(self.get_params_path(), self.params)

@classmethod
def load(cls, path):
    params = read_pickle(os.path.join(path, 'params.pkl'))
```

```
        return cls(**params)

    def get_new_data(self):
        raise NotImplementedError()

    def get_folder(self):
        raise NotImplementedError()

    def read_data(self):
        raise NotImplementedError()

    def write_data(self):
        raise NotImplementedError()
```

example.py

```
from symbol import get_options_list
from strategy import Strategy
from neural import NeuralNetwork
from preprocess import stratify_parts
from graph import *
import csv

# yahoo screener stocks top 10

undervalued_growth_stocks = ['F', 'MU', 'NFX', 'CMCSA', 'MDR', 'KGC', 'VALE',
                    'XOM', 'FLEX', 'ON']


# dates

start = '2007-01-01'
end = '2018-01-01'

# indicators

indicators = ['daily', 'daily_adjusted', 'sma', 'ema', 'macd', 'stoch', 'rsi',
                    'adx', 'cci', 'aroon', 'bbands', 'ad', 'obv']

# additional params

days = 7
```

```
tolerance = 0.05

epochs = 200

threshold = 0.9

# ANNs


neural1 = NeuralNetwork(**stratify_parts(undervalued_growth_stocks, [0.25]*3, start,
        end), options_list=get_options_list(indicators),
        days=days, tolerance=tolerance, epochs=epochs)


# strategies

strat_start = '2018-06-01'
strat_end = '2018-10-12'

strategies = []
for symbol in undervalued_growth_stocks:
        strategies.append(Strategy(neural=neural1, start=strat_start, end=strat_end,
                symbol=symbol, threshold=threshold))

# save trades

trades = [ { 'date': d['date'], 'price': d['price'], 'size': d['size'],
                        'symbol': s.symbol } for s in strategies for d in s.get_data() ]

with open('trades.csv', 'w') as fh:
        f = csv.writer(fh)
        f.writerow(list(trades[0].keys()))
        for trade in trades:
                f.writerow(list(trade.values()))
```

graph.py

```
import optimal
import symbol
import matplotlib.pyplot as plt
from matplotlib.ticker import LinearLocator
```

```
import neural
from data import Data
from optimal import OptimalTrades
from symbol import SymbolData, SymbolCloseData
from utility import *

PT_SIZE = 20

class Graph(Data):

    def get_folder(self):
        return 'graph'

    def get_fig_path(self):
        return self.get_path('graph.pkl')

    def get_pic_path(self):
        return self.get_path('graph.png')

    def read_data(self):
        return read_pickle(self.get_fig_path())

    def write_data(self):
        fig = self.get_data()
        fig.savefig(self.get_pic_path())
        write_pickle(self.get_fig_path(), fig)

    def get_figure(self):
        return self.get_data()

    def get_new_data(self):
        return self.make_figure()

    def show(self):
        plt.show()

    def make_figure(self):
        raise NotImplementedError()


class SymbolDataGraph(Graph):

    def __init__(self, **params):
```

```
        self.symbol = params['symbol']
        self.options_list = params['options_list']
        self.start = params.get('start', None)
        self.end = params.get('end', None)
        super().__init__(**params)

    def make_figure(self):
        log('Making new symbol data graph...')
        return graph_symbol_data(self.symbol, self.options_list, self.start, self.end)


class OptimalTradesGraph(Graph):

    def __init__(self, **params):
        self.symbol = params['symbol']
        self.tolerance = params.get('tolerance', 0.01)
        self.start = params.get('start', None)
        self.end = params.get('end', None)
        super().__init__(**params)

    def make_figure(self):
        log('Making new optimal trades graph...')
        return graph_optimal_trades(self.symbol, self.start, self.end, self.tolerance)


class StrategyTradesGraph(Graph):

    def __init__(self, strategy=None):
        self.strategy = strategy
        super().__init__(strategy=self.strategy.params)

    def make_figure(self):
        log('Making new strategy trades graph...')
        return graph_strategy_trades(self.strategy)


def graph_symbol_data(symbol, options_list, start, end):
    data = SymbolData(symbol=symbol, options_list=options_list, start=start,
end=end).get_data()
    columns = get_columns(data)
    data_list = [extract_column(col, data) for col in columns]

    xys = dicts_to_xys(data_list)
```

```
    fig = plt.figure()
    for x, y in xys:
        plt.plot(x, y)

    plt.title('Symbol Data (%s)' % symbol)
    plt.ylabel('Indicator Value')
    plt.xlabel('Date')

    add_dates(sorted(data.keys()))
    plt.gca().get_yaxis().set_major_locator(LinearLocator(numticks=10))

    return fig


def graph_optimal_trades(symbol, start, end, tolerance):
    prices = SymbolCloseData(symbol=symbol, start=start, end=end).get_data()
    trades = OptimalTrades(symbol=symbol, start=start, end=end,
                           tolerance=tolerance).get_data()

    buy_sizes = {k: PT_SIZE * v for k, v in trades.items() if v > 0}
    buy_prices = {k: prices[k] for k in buy_sizes}
    sell_sizes = {k: -PT_SIZE * v for k, v in trades.items() if v < 0}
    sell_prices = {k: prices[k] for k in sell_sizes}

    (xp, yp), (xb, ybp), (_, ybs), (xs, ysp), (_, yss) = dicts_to_xys([
        prices, buy_prices, buy_sizes, sell_prices, sell_sizes
    ])

    fig = plt.figure()
    plt.plot(xp, yp)
    plt.scatter(xb, ybp, s=ybs, c='g')
    plt.scatter(xs, ysp, s=yss, c='r')

    plt.title('Optimal Trades (%s)' % symbol)
    plt.ylabel('Price')
    plt.xlabel('Date')
    add_dates(sorted(prices.keys()))

    return fig


def graph_strategy_trades(strategy):
```

```
    prices = SymbolCloseData(symbol=strategy.symbol, start=strategy.start,
                             end=strategy.end).get_data()
    trades = strategy.get_data()

    buy_prices = { t['date']: t['price'] for t in trades if t['buy'] }
    sell_prices = { t['date']: t['price'] for t in trades if not t['buy'] }

    (xp, yp), (xb, ybp), (xs, ysp) = dicts_to_xys([
        prices, buy_prices, sell_prices
    ])

    fig = plt.figure()
    plt.plot(xp, yp)
    plt.scatter(xb, ybp, c='g')
    plt.scatter(xs, ysp, c='r')

    plt.title('Strategy Trades (%s)' % strategy.symbol)
    plt.ylabel('Price')
    plt.xlabel('Date')
    add_dates(sorted(prices.keys()))

    return fig


def get_symbol_data_graphs(symbols, options_list, start, end):
    graphs = {}
    log(end)
    for symbol in symbols:
        graphs[symbol] = SymbolDataGraph(symbol=symbol, options_list=options_list,
                                         start=start, end=end).get_figure()
    return graphs


def get_optimal_trades_graphs(symbols, start, end, tolerance):
    graphs = {}
    for symbol in symbols:
        graphs[symbol] = OptimalTradesGraph(symbol=symbol, tolerance=tolerance,
                                            start=start, end=end).get_figure()
    return graphs


def add_dates(dates):
    num_locs = 4
```

```
    locs = [ i for i in range(0, len(dates), len(dates) // num_locs) ]
    labels = [ dates[i] for i in locs ]
    plt.xticks(locs, labels, rotation=20)


def add_args(parser):
    parser.add_argument('data', type=str, help='data to graph',
                        choices=['data', 'optimal', 'neural', 'strategy'])
    neural.add_args(parser)


def handle_args(args, parser):
    if args.data == 'neural':
        neural.handle_args(args, parser)
    elif args.data == 'optimal':
        optimal.handle_args(args, parser)
    elif args.data == 'data':
        symbol.handle_args(args, parser)


def get_neural_network_graph():
    print('neural network graph not implemented')


def get_strategy_graph():
    print('strategy graph not implemented')


def main():
    args = parse_args('Load a graph.', add_args, handle_args)
    data = []
    if args.data == 'data':
        data += get_symbol_data_graphs(args.symbols, args.options_list, args.start, args.end)
    if args.data == 'optimal':
        data += get_optimal_trades_graphs(args.symbols, args.start, args.end, args.tolerance)
    if args.data == 'neural':
        data += get_neural_network_graph()
    if args.data == 'strategy':
        data += get_strategy_graph()
    if args.print or PARAMS['verbose']:
        plt.show()
    if args.path:
        [log(d.get_path(), force=args.print) for d in data]
```

```
if __name__ == '__main__':
    main()
```

indicators.py

```
def daily():
    return {
        'function': 'TIME_SERIES_DAILY',
        'columns': ['open', 'high', 'low', 'close', 'volume'],
        'outputsize': 'full'
    }


def daily_adjusted():
    return {
        'function': 'TIME_SERIES_DAILY_ADJUSTED',
        'columns': ['adjusted close', 'dividend amount', 'split coefficient'],
        'outputsize': 'full'
    }


def sma(period=30):
    return {
        'function': 'SMA',
        'columns': ['SMA'],
        'interval': 'daily',
        'time_period': period,
        'series_type': 'close'
    }


def ema(period=20):
    return {
        'function': 'EMA',
        'columns': ['EMA'],
        'interval': 'daily',
        'time_period': period,
        'series_type': 'close'
    }
```

```
def macd(fast=12, slow=26, signal=9):
    return {
        'function': 'MACD',
        'columns': ['MACD_Signal', 'MACD_Hist', 'MACD'],
        'interval': 'daily',
        'series_type': 'close',
        'fastperiod': fast,
        'slowperiod': slow,
        'signalperiod': signal,
    }


def stoch(fastk=5, slowk=3, slowd=3, kma=0, dma=0):
    return {
        'function': 'STOCH',
        'columns': ['SlowD', 'SlowK'],
        'interval': 'daily',
        'series_type': 'close',
        'fastkperiod': fastk,
        'slowkperiod': slowk,
        'slowdperiod': slowd,
        'slowkmatype': kma,
        'slowdmatype': dma
    }


def rsi(period=14):
    return {
        'function': 'RSI',
        'columns': ['RSI'],
        'interval': 'daily',
        'series_type': 'close',
        'time_period': period
    }


def adx(period=14):
    return {
        'function': 'ADX',
        'columns': ['ADX'],
        'interval': 'daily',
        'series_type': 'close',
```

```
        'time_period': period
    }


def cci(period=14):
    return {
        'function': 'CCI',
        'columns': ['CCI'],
        'interval': 'daily',
        'series_type': 'close',
        'time_period': period
    }


def aroon(period=14):
    return {
        'function': 'AROON',
        'columns': ['Aroon Up', 'Aroon Down'],
        'interval': 'daily',
        'time_period': period
    }


def bbands(period=14, ndev=2, ma=0):
    return {
        'function': 'BBANDS',
        'columns': ['Real Middle Band', 'Real Upper Band', 'Real Lower Band'],
        'interval': 'daily',
        'time_period': period,
        'series_type': 'close',
        'nbdevup': ndev,
        'nbdevdn': ndev,
        'matype': ma
    }


def ad():
    return {
        'function': 'AD',
        'columns': ['Chaikin A/D'],
        'interval': 'daily',
    }
```

```
def obv():
    return {
        'function': 'OBV',
        'columns': ['OBV'],
        'interval': 'daily',
    }
```

model.yml

```
model:
  - input:
      shape: (SHAPE,)
    name: in
  - dense: NODES
  - activation: ACTIVATION
  - dense: 1
  - activation: ACTIVATION
    name: out

train:
  data:
    - pickle: TRAINING
  epochs: EPOCHS
  weights: WEIGHTS
  log: LOG

validate:
  data:
    - pickle: VALIDATION
  weights: WEIGHTS

evaluate:
  data:
    - pickle: EVALUATION
  weights: WEIGHTS
  destination: OUTPUT

loss:
  - target: out
    name: LOSS
```

neural.py

```
import logging
from subprocess import call
from kur.loggers import BinaryLogger
from kur import Kurfile
from kur.engine import JinjaEngine
from kur.utils import DisableLogging
from sys import stdout
import preprocess
from analysis import get_accuracy, get_average_distance
from utility import *
from data import Data


class NeuralNetwork(Data):

    def __init__(self, **params):
        self.training = params['training']
        self.validation = params['validation']
        self.evaluation = params['evaluation']
        self.options_list = params['options_list']
        self.days = params.get('days', 0)
        self.tolerance = params.get('tolerance', 0.01)
        self.epochs = params.get('epochs', 50)
        self.nodes = params.get('nodes', 64)
        self.activation = params.get('activation', 'tanh')
        self.loss = params.get('loss', 'mean_squared_error')
        self.part_data = None
        super().__init__(**params)

    def get_folder(self):
        return 'neural'

    def read_data(self):
        return read_pickle(self.get_data_path())

    def write_data(self):
        write_pickle(self.get_data_path(), self.get_data())

    def get_data_path(self):
        return self.get_path('data.pkl')
```

```
    def get_model_path(self):
        return self.get_path('model.yml')

    def get_new_data(self):
        log('Training neural network...')
        self.make_model()
        return train_neural_network(self.get_path())

    def make_model(self):
        make_path(self.get_model_path())
        with open(self.get_model_path(), 'w') as fh:
            fh.write(self.get_model())

    def get_part_data(self):
        if not self.part_data:
            self.part_data = preprocess.NeuralNetworkData(training=self.training,
                validation=self.validation, evaluation=self.evaluation,
                options_list=self.options_list, days=self.days, tolerance=self.tolerance)
        return self.part_data

    def get_model(self):
        folder = self.get_path()
        part_data = self.get_part_data()
        training = part_data.get_part_path('training')
        validation = part_data.get_part_path('validation')
        evaluation = part_data.get_part_path('evaluation')
        return make_model(training, validation, evaluation, folder, self.epochs,
                          self.nodes, self.activation, self.loss, part_data.get_shape())

    def predict(self, data):
        kurfile = Kurfile(self.get_model_path(), JinjaEngine())
        kurfile.parse()
        model = kurfile.get_model()
        with DisableLogging(logging.WARNING):
            model.backend.compile(model)
        model.restore(self.get_path('weights'))
        pdf, metrics = model.backend.evaluate(model, data={'in': np.array([data])})
        prediction = pdf['out'][0][0]
        return prediction


def make_model(training, validation, evaluation, folder, epochs, nodes, activation, loss, shape):
```

```
    with open('model.yml', 'r') as fh:
        model = fh.read()
        model = model.replace('TRAINING', training)
        model = model.replace('VALIDATION', validation)
        model = model.replace('EVALUATION', evaluation)
        model = model.replace('WEIGHTS', os.path.join(folder, 'weights'))
        model = model.replace('LOG', os.path.join(folder, 'log'))
        model = model.replace('OUTPUT', os.path.join(folder, 'output.pkl'))
        model = model.replace('EPOCHS', str(epochs))
        model = model.replace('NODES', str(nodes))
        model = model.replace('ACTIVATION', activation)
        model = model.replace('LOSS', loss)
        model = model.replace('SHAPE', str(shape))
        return model


def train_neural_network(folder):
    model_path = os.path.join(folder, 'model.yml')
    log_path = os.path.join(folder, 'log')
    output_path = os.path.join(folder, 'output.pkl')
    std_out = None if PARAMS['verbose'] else 'out'
    call(['kur', 'train', model_path], stdout=stdout)
    call(['kur', 'evaluate', model_path], stdout=stdout)
    output = read_pickle(output_path)
    return {
        'training_loss': get_loss(log_path, 'training_loss_total'),
        'validation_loss': get_loss(log_path, 'validation_loss_total'),
        'output': output,
        'accuracy': get_accuracy(output),
        'average_distance': get_average_distance(output)
    }


def get_loss(log_path, path):
    return BinaryLogger.load_column(log_path, path)


def add_args(parser):
    preprocess.add_args(parser)
    parser.add_argument('-e', '--epochs', type=int, default=50,
                        help='number of epochs to train for')
    parser.add_argument('-n', '--nodes', type=int, default=64,
                        help='number of nodes per layer')
```

```
    parser.add_argument('-a', '--activation', type=str, default='tanh', choices=['tanh'],
                        help='type of activation layer')
    parser.add_argument('--loss', type=str, default='mean_squared_error',
                        help='type of loss function', choices=['mean_squared_error'])


def handle_args(args, parser):
    preprocess.handle_args(args, parser)


def main():
    args = parse_args('Create a neural network.', add_args, handle_args)
    data = NeuralNetwork(**args.parts, options_list=args.options_list, days=args.days,
                         tolerance=args.tolerance, epochs=args.epochs, nodes=args.nodes,
                         activation=args.activation, loss=args.loss)
    log(data.get_data(), force=args.print)
    if args.path:
        log(data.get_path(), force=args.print)


if __name__ == '__main__':
    main()
```

optimal.py

```
from __future__ import (absolute_import, division, print_function, unicode_literals)
from data import Data
from utility import *
from symbol import SymbolCloseData, add_symbol_args, handle_symbol_args,
handle_date_args

BUY = 1
SELL = -1


class OptimalTrades(Data):

    def __init__(self, **params):
        self.symbol = params['symbol']
        self.start = params.get('start', None)
        self.end = params.get('end', None)
        self.tolerance = params.get('tolerance', 0.01)
```

```
        super().__init__(**params)

    def get_new_data(self):
        log('Calculating optimal trades...')
        trades = get_optimal_trades(self.symbol, self.start, self.end, self.tolerance)
        if trades == {}:
            raise Exception('Could not calculate optimal trades. Try increasing the period or
decreasing the tolerance.')
        return trades

    def get_folder(self):
        return 'optimal'

    def get_data_path(self):
        return self.get_path('data.pkl')

    def read_data(self):
        return read_pickle(self.get_data_path())

    def write_data(self):
        write_pickle(self.get_data_path(), self.get_data())


def get_optimal_trades(symbol, start, end, tolerance):
    data = SymbolCloseData(symbol=symbol, start=start, end=end).get_data()
    return calc_trades(data, tolerance=tolerance)


def calc_trades(data, tolerance):
    dates = sorted(data)
    prices = [data[date] for date in dates]
    trades = optimize_trades(prices, tolerance)
    trades = smooth_trades(trades, prices)
    trade_data = {dates[key]: val for key, val in trades.items()}
    return trade_data


def smooth_trades(trades, prices):
    if len(trades) < 2:
        return trades

    ordered_trades = sorted(trades.items())
```

```
    for i, (date, trade) in enumerate(ordered_trades[1:]):
        last_date = ordered_trades[i][0]
        if trade == BUY:
            buy_price = prices[date]
            sell_price = prices[last_date]
        else:
            buy_price = prices[last_date]
            sell_price = prices[date]
        for j, price in enumerate(prices[last_date + 1:date]):
            trades[last_date + 1 + j] = smooth_trade(price, buy_price, sell_price)

    return trades


def smooth_trade(price, buy_price, sell_price):
    return 1 - 2 * (price - buy_price) / (sell_price - buy_price)


def optimize_trades(prices, tolerance):
    if len(prices) < 2:
        return {}

    # determine whether to buy or sell first
    buying = should_buy_first(prices, tolerance)

    delay = 0
    trades = {}

    # determine when to buy and sell
    for index, price in enumerate(prices[1:]):
        index -= delay  # index is behind by one = index - 1
        price_diff = (price - prices[index]) / prices[index]

        if buying:  # looking to buy
            if 0 <= price_diff <= tolerance:
                delay += 1
            else:
                delay = 0
                if price_diff > 0:
                    trades[index] = BUY
                    buying = False
        else:  # looking to sell
            if -tolerance <= price_diff <= 0:
```

```
            delay += 1
        else:
            delay = 0
            if price_diff < 0:
                trades[index] = SELL
                buying = True

    return trades


def should_buy_first(prices, tolerance):
    delay = 0

    for index, price in enumerate(prices[1:]):
        index -= delay  # index is behind by one = index - 1
        price_diff = (price - prices[index]) / prices[index]

        if 0 <= price_diff <= tolerance:
            delay += 1
        else:
            delay = 0
            if price_diff > 0:
                return True
        if -tolerance <= price_diff < 0:
            delay += 1
        else:
            delay = 0
            if price_diff < 0:
                return False


def get_optimal_trades_dict(symbols, start, end, tolerance):
    trades = {}
    for symbol in symbols:
        trades[symbol] = OptimalTrades(symbol=symbol, start=start, end=end,
tolerance=tolerance)
    return trades


def add_args(parser):
    add_symbol_args(parser)
    parser.add_argument('-t', '--tolerance', type=float, default=0.01,
                        help='tolerance to use in algorithm')
```

```
def handle_args(args, parser):
    handle_symbol_args(args, parser)
    handle_date_args(args, parser)


def main():
    args = parse_args('Load optimal trades.', add_args, handle_args)
    data = get_optimal_trades_dict(args.symbols, args.start, args.end, args.tolerance)
    [log(k, v.get_data(), force=args.print) for k, v in data.items()]
    if args.path:
        [log(k, v.get_path(), force=args.print) for k, v in data.items()]


if __name__ == '__main__':
    main()
```

params.py

```
import os
from dotenv import load_dotenv, find_dotenv
load_dotenv(find_dotenv())
from indicators import *


ALPHAVANTAGE = 'alphavantage_key'
INTRINIO_USERNAME = 'intrinio_username'
INTRINIO_PASSWORD = 'intrinio_password'
DATA_FOLDER = 'data'


def check_credentials():
    for var in [ALPHAVANTAGE, INTRINIO_USERNAME, INTRINIO_PASSWORD]:
        if not os.environ.get(var):
            not_found(var)


def not_found(var):
    raise Exception(var + ' not found in environment')
```

```
check_credentials()


PARAMS = {

    'verbose': os.environ.get('verbose', False),

    'credentials': {
        'alphavantage': os.environ.get(ALPHAVANTAGE),
        'intrinio': {
            'username': os.environ.get(INTRINIO_USERNAME),
            'password': os.environ.get(INTRINIO_PASSWORD)
        }
    },

    'data_folder': os.environ.get('data_folder', DATA_FOLDER),

    'screeners': {
        'yahoo': [
            'undervalued_growth_stocks',
            'day_gainers',
            'day_losers',
            'most_actives',
            'growth_technology_stocks',
            'undervalued_large_caps',
            'aggressive_small_caps',
            'portfolio_anchors',
            'solid_large_growth_funds'
        ],
        'intrinio': {
            'undervalued': {
                'conditions': [
                    ['pricetoearnings', '<=', 20],
                    ['pricetoearnings', '>', 0]
                ]
            }
        }
    },

    'data_options': {
        'daily': daily,
        'daily_adjusted': daily_adjusted,
        'sma': sma,
```

```
        'ema': ema,
        'macd': macd,
        'stoch': stoch,
        'rsi': rsi,
        'adx': adx,
        'cci': cci,
        'aroon': aroon,
        'bbands': bbands,
        'ad': ad,
        'obv': obv
    }

}
```

preprocess.py

```
from argparse import Action
from data import Data, DataException
from symbol import SymbolData, handle_options_args
import symbol
from optimal import OptimalTrades
from utility import *
from screener import get_symbols

DATA_PARTS = ['training', 'validation', 'evaluation']
NUM_PARTS = len(DATA_PARTS)


class NeuralNetworkData(Data):

    def __init__(self, **params):
        self.training = params['training']
        self.validation = params['validation']
        self.evaluation = params['evaluation']
        self.options_list = params['options_list']
        self.days = params.get('days', 0)
        self.tolerance = params.get('tolerance', 0.01)
        validate_parts([params[p] for p in DATA_PARTS])
        super().__init__(**params)

    def get_folder(self):
        return 'preprocess'
```

```
    def get_part_path(self, part):
        return self.get_path(part + '.pkl')

    def read_data(self):
        return read_preprocess(self.get_path())

    def write_data(self):
        write_preprocess(self.get_path(), self.get_data())

    def get_new_data(self):
        log('Preprocessing neural network data...')
        return {p: self.get_new_data_part(self.params[p]) for p in DATA_PARTS}

    def get_new_data_part(self, part):
        return get_data_part(part['symbols'], self.options_list, part['start'],
                             part['end'], self.days, self.tolerance)

    def get_shape(self):
        return self.get_data()[DATA_PARTS[0]][0].shape[1]


def read_preprocess_part(path):
    data = read_pickle(path)
    if data:
        return data['in'], data['out']


def read_preprocess(folder):
    data = {k: read_preprocess_part(os.path.join(folder, k + '.pkl')) for k in DATA_PARTS}
    if None not in data.values():
        return data


def write_data_part(folder, data, part):
    path = os.path.join(folder, part + '.pkl')
    make_path(path)
    matrix_in, matrix_out = data[part]
    write_pickle(path, {'in': matrix_in, 'out': matrix_out})


def write_preprocess(folder, data):
    [write_data_part(folder, data, p) for p in DATA_PARTS]
```

```
def get_symbol_part(symbol, options_list, start, end, days, tolerance):
    symbol_data = SymbolData(symbol=symbol, options_list=options_list).get_data()
    trades = OptimalTrades(symbol=symbol, start=start, end=end,
tolerance=tolerance).get_data()
    data_in, data_out = filter_matching(symbol_data, trades)
    data_in = add_prior_days(data_in, days, symbol_data)
    new_in = json_to_matrix(data_in)
    new_out = json_to_matrix(data_out)
    return new_in, new_out


def get_data_part(symbols, options_list, start, end, days, tolerance):
    matrix_in = None
    matrix_out = None
    for symbol in symbols:
        new_in, new_out = get_symbol_part(symbol, options_list, start, end, days, tolerance)
        if matrix_in is None:
            matrix_in = new_in
            matrix_out = new_out
        else:
            matrix_in = np.concatenate((matrix_in, new_in))
            matrix_out = np.concatenate((matrix_out, new_out))
    return matrix_in, matrix_out


# add data from prior days to the data for the current date
def add_prior_days(data, days, full_data):
    if not data:
        return data
    new_data = {}
    date_list = list(enumerate(sorted(data)))
    full_data_sorted = sorted(full_data)
    full_date_list = list(enumerate(full_data_sorted))
    start_date = full_data_sorted.index(date_list[0][1])
    end_date = full_data_sorted.index(date_list[-1][1])
    for current, date in full_date_list[start_date:end_date + 1]:
        new_data[date] = {}
        for prior in range(days + 1):
            prior_data = full_data[full_date_list[current - prior][1]]
            for col in list(prior_data):
                new_data[date][str(col) + str(prior)] = prior_data[col]
```

```
    return new_data


def validate_parts(parts):
    [validate_part(p) for p in parts]
    if False in [len(parts[0]['symbols']) == len(p['symbols']) for p in parts[1:]]:
        raise Exception('A neural network must be trained, validated, '
                        'and evaluated on the same number of symbols')


def validate_part(part):
    if type(part['symbols']) is not list:
        raise Exception('symbols must be a list')
    if part['start'] is not None and type(part['start']) is not str:
        raise Exception('start must be a date string')
    if part['end'] is not None and type(part['end']) is not str:
        raise Exception('end must be a date string')


def get_part_order(args):
    order = []
    for s in args.symbol_order:
        if s.find('training') > -1:
            order += ['training']
        if s.find('validation') > -1:
            order += ['validation']
        if s.find('evaluation') > -1:
            order += ['evaluation']
    return remove_duplicates(order)


def make_parts(training_symbols, validation_symbols, evaluation_symbols, start, end):
    return {
        'training': {
            'symbols': training_symbols,
            'start': start[0],
            'end': end[0]
        },
        'validation': {
            'symbols': validation_symbols,
            'start': start[1],
            'end': end[1]
        },
```

```
        'evaluation': {
            'symbols': evaluation_symbols,
            'start': start[2],
            'end': end[2]
        }
    }


def get_parts(part_order, args):
    start = get_order_specific(args.start, part_order)
    end = get_order_specific(args.end, part_order)
    return make_parts(args.training_symbols, args.validation_symbols,
                      args.evaluation_symbols, start, end)


def get_order_specific(l, order):
    if not len(l):
        return [None] * NUM_PARTS
    elif len(l) == 1:
        return [l[0]] * NUM_PARTS
    elif len(l) == NUM_PARTS:
        return (l[order.index('training')],
                l[order.index('validation')],
                l[order.index('evaluation')])
    else:
        raise Exception


def stratify_parts(symbols, percentages, start, end):
    start = to_date(start or '2002-01-01')
    end = to_date(end or Date.today().strftime('%Y-%m-%d'))
    duration = end - start
    starts = [None] * NUM_PARTS
    ends = [None] * NUM_PARTS
    starts[0] = start
    starts[1] = starts[0] + duration * percentages[0]
    starts[2] = starts[1] + duration * percentages[1]
    ends[0] = starts[1]
    ends[1] = starts[2]
    ends[2] = end
    starts = [d.strftime('%Y-%m-%d') for d in starts]
    ends = [d.strftime('%Y-%m-%d') for d in ends]
    return make_parts(symbols, symbols, symbols, starts, ends)
```

```
class SymbolAction(Action):
    def __call__(self, parser, args, values, option_string=None):
        args.symbol_order = getattr(args, 'symbol_order', []) + [self.dest]
        setattr(args, self.dest, values)


def add_args(parser):
    symbol.add_args(parser)
    parser.add_argument('--percentages', type=float, nargs='+', default=[0.5, 0.25, 0.25],
                        help='relative size of each data part')
    parser.add_argument('--training_symbols', type=str, nargs='+', action=SymbolAction,
                        help='symbol(s) to train with')
    parser.add_argument('--training_screener', type=str, action=SymbolAction,
                        help='name of Yahoo screener to train with')
    parser.add_argument('--validation_symbols', type=str, nargs='+',
                        help='symbol(s) to validate with', action=SymbolAction)
    parser.add_argument('--validation_screener', type=str, action=SymbolAction,
                        help='name of Yahoo screener to validate with')
    parser.add_argument('--evaluation_symbols', type=str, nargs='+', action=SymbolAction,
                        help='symbol(s) to evaluate with')
    parser.add_argument('--evaluation_screener', type=str, action=SymbolAction,
                        help='name of Yahoo screener to evaluate with')
    parser.add_argument('-t', '--tolerance', type=float, default=0.01,
                        help='tolerance to use in optimal trades algorithm')
    parser.add_argument('-d', '--days', type=int, default=0,
                        help='number of prior days of data to use as input per day')


def handle_symbols(args, parser):
    if not ((args.symbols or args.screener)
            or ((args.training_symbols or args.training_screener)
                and (args.validation_symbols or args.validation_screener)
                and (args.evaluation_symbols or args.evaluation_screener))):
        parser.error('(-s/--symbols or -y/--screener) or '
                     '((--training_symbols or --training_screener) and '
                     '(--validation_symbols or --validation_screener) and '
                     '(--evaluation_symbols or --evaluation_screener)) is required')
    if args.symbols or args.screener:
        if len(args.percentages) != NUM_PARTS:
            parser.error('Exactly %s --percentages required' % NUM_PARTS)
        elif not (0.9999 < sum(args.percentages) < 1.0001):
```

```
            parser.error('--percentages must sum to 1')
        else:
            args.symbols = get_symbols(args.symbols, args.screener, args.limit)
            if len(args.start):
                args.start = args.start[0]
                args.end = args.end[0]
            else:
                args.start = None
                args.end = None
            args.parts = stratify_parts(args.symbols, args.percentages, args.start, args.end)


def handle_dates(args, parser):
    if len(args.start) != len(args.end):
        parser.error('number of --start and --end must match')


def handle_parts(args, parser):
    if not args.symbols:
        args.training_symbols = get_symbols(args.training_symbols, args.training_screener,
args.limit)
        args.validation_symbols = get_symbols(args.validation_symbols, args.validation_screener,
args.limit)
        args.evaluation_symbols = get_symbols(args.evaluation_symbols,
args.evaluation_screener, args.limit)
        part_order = get_part_order(args)
        try:
            args.parts = get_parts(part_order, args)
        except:
            parser.error('Either 0, 1, or %s --start and --end is required' % NUM_PARTS)


def handle_args(args, parser):
    handle_dates(args, parser)
    handle_symbols(args, parser)
    handle_parts(args, parser)
    handle_options_args(args, parser)
    log(';', args.start, args.end)


def main():
    args = parse_args('Preprocess neural network data.', add_args, handle_args)
    data = NeuralNetworkData(**args.parts, options_list=args.options_list, days=args.days,
```

```
                        tolerance=args.tolerance)
    log(data.get_data(), force=args.print)
    if args.path:
        log(data.get_path(), force=args.print)


if __name__ == '__main__':
    main()
```

screener.py

```
from __future__ import (absolute_import, division, print_function, unicode_literals)

from argparse import ArgumentParser
from urllib.parse import quote_plus
import requests
from base64 import b64encode
from pyquery import PyQuery as pq
from utility import *


# get data from Yahoo predefined screeners
def yahoo(screener):
    d = pq(url='https://finance.yahoo.com/screener/predefined/%s' % screener)
    elements = d("td.Va\\(m\\) > a.Fw\\(b\\)")
    return [a.text for a in elements]


def get_symbols(symbols, screener, limit):
    symbols = symbols or []
    if screener:
        symbols += yahoo(screener)
    return symbols[:limit]


# AAII screener 'table > tbody > tr:nth-child(2n+1) > td:nth-child(2) > a'
# needs authentication


USERNAME = PARAMS['credentials']['intrinio']['username']
PASSWORD = PARAMS['credentials']['intrinio']['password']
```

```
# get data from Intrinio custom screeners
def request(conditions):
    params = encode_conditions(conditions)
    url = 'https://api.intrinio.com/securities/search?conditions=%s' % quote_plus(params)
    auth = 'Basic %s' % b64encode(('%s:%s' % (USERNAME, PASSWORD)).encode()).decode()
    headers = {
        'Authorization': auth
    }
    return requests.get(url, headers=headers).json()


def encode_element(element):
    if element == '>':
        return 'gt'
    elif element == '>=':
        return 'gte'
    elif element == '<':
        return 'lt'
    elif element == '<=':
        return 'lte'
    elif element == '=':
        return 'eq'
    else:
        return str(element)


def encode_condition(condition):
    return "~".join(map(encode_element, condition))


def encode_conditions(conditions):
    return ",".join(map(encode_condition, conditions))


def add_args(parser):
    parser.add_argument('screener', type=str, help='name of Yahoo screener')
    parser.add_argument('-l', '--limit', type=int, help='take the first l symbols')


def handle_args(args, parser):
    pass
```

```
def main():
    args = parse_args('Screen for symbols.', add_args, handle_args)
    data = ' '.join(get_symbols(None, args.screener, args.limit))
    log(data, force=args.print)


if __name__ == '__main__':
    main()
```

strategy.py

```
import datetime
import os.path
import sys
import backtrader as bt
from data import Data
from symbol import SymbolData, SymbolCloseData
from preprocess import add_prior_days
from utility import *

COMMISION = 0.001
SLIPPAGE = 0.01
INITIAL_FUNDS = 10000
STAKE = 10

class Strategy(Data):

    def __init__(self, **params):
        self.neural = params['neural']
        self.start = params['start']
        self.end = params['end']
        self.symbol = params['symbol']
        self.threshold = params.get('threshold', 0.8)
        self.options_list = self.neural.options_list
        self.days = self.neural.days
        super().__init__(neural=self.neural.params, start=self.start,
                end=self.end, symbol=self.symbol, threshold=self.threshold)

    def get_folder(self):
        return 'strategy'
```

```
def get_data_path(self):
    return self.get_path('data.pkl')

def read_data(self):
    return read_pickle(self.get_data_path())

def write_data(self):
    write_pickle(self.get_data_path(), self.get_data())

def get_new_data(self):
    log('Backtesting strategy...')
    self.setup_input_data()
    strategy = self.backtest()
    return strategy.get_results()

def setup_input_data(self):
    # download OHLCV and specified indicators
    # OHLCV is used for backtesting stats, not as input to ANN
    SymbolData(symbol=self.symbol, options_list=[DAILY_OPTIONS],
            start=self.start, end=self.end)
    symbol_data = SymbolData(symbol=self.symbol, options_list=self.options_list,
            start=self.start, end=self.end)
    # get the path to the data
    self.symbol_path = symbol_data.get_symbol_path()
    symbol_data = symbol_data.get_data()
    # format data for input to neural network
    self.input_data = add_prior_days(symbol_data, self.days, symbol_data)

# remove dates from feed that don't have any data
def data_filter(self):
    def filter(data_feed):
        date = from_date(data_feed.datetime.date())
        if date not in self.input_data:
            data_feed.backwards()
            return True
        return False
    return filter

def get_data_feed(self):
    # get OHLCV column indices in CSV
    headers = get_csv_headers(self.symbol_path)
    daily_indices = {}
    daily_crypt = get_daily_crypt()
```

```
        for key in daily_crypt:
            daily_indices[key] = headers.index(daily_crypt[key])
        # create BT data feed from saved symbol data in CSV
        data_feed = bt.feeds.GenericCSVData(
            dataname=self.symbol_path,
            fromdate=to_date(self.start),
            todate=to_date(self.end),
            dtformat=('%Y-%m-%d'),
            openinterest=-1,
            **daily_indices)
        # remove dates without data
        data_feed.addfilter(self.data_filter())
        return data_feed

    def get_cerebro(self):
        cerebro = bt.Cerebro()
        # add strategy based on given neural network
        cerebro.addstrategy(BTStrategy, symbol_data=self.input_data,
                    neural=self.neural, threshold=self.threshold)
        data_feed = self.get_data_feed()
        cerebro.adddata(data_feed)
        # add drawdown observer
        cerebro.addobserver(bt.observers.DrawDown)
        # set funds, commision, slippage, and position sizer
        cerebro.broker.setcash(INITIAL_FUNDS)
        cerebro.broker.setcommission(commission=COMMISION)
        cerebro.broker.set_slippage_perc(SLIPPAGE)
        cerebro.addsizer(bt.sizers.FixedSize, stake=STAKE)
        return cerebro

    def backtest(self):
        cerebro = self.get_cerebro()
        # log(cerebro.broker.getvalue())
        strategy = cerebro.run()[0]
        # log(cerebro.broker.getvalue())
        return strategy


class BTStrategy(bt.Strategy):

    params = (
        ('symbol_data', None),
        ('neural', None),
```

```
    ('threshold', None)
)

def __init__(self):
    self.symbol_data = self.params.symbol_data
    self.neural = self.params.neural
    self.threshold = self.params.threshold
    if not self.symbol_data:
        raise Exception('A Backtrader strategy expects a symbol_data parameter')
    if not self.neural:
        raise Exception('A Backtrader strategy expects a neural parameter')
    if not self.threshold:
        raise Exception('A Backtrader strategy expects a threshold parameter')
    # keep track of pending orders
    self.order = None
    # keep track of trades
    self.trades = []

def get_close(self):
    return self.datas[0].close[0]

def get_date(self):
    return from_date(self.datas[0].datetime.date())

# def notify_trade(self, trade):
#     if not trade.isclosed:
#         return

#     log(self.get_date(), 'OPERATION PROFIT, GROSS %.2f, NET %.2f' %
#             (trade.pnl, trade.pnlcomm))

def notify_order(self, order):
    # reset order status if order finished
    if order.status not in [order.Submitted, order.Accepted]:
        self.order = None
        if order.status == order.Completed:
            self.trades.append({
                'date': self.get_date(),
                'buy': order.isbuy(),
                'price': order.executed.price,
                'value': order.executed.value,
                'commission': order.executed.comm,
                'size': order.executed.size
```

```
            })


    def is_in_market(self):
        return len(self.trades) % 2 > 0

    def is_long(self):
        return self.is_in_market() and self.trades[-1]['buy']

    def is_short(self):
        return self.is_in_market() and not self.trades[-1]['buy']

    def next(self):
        # check for pending order
        if self.order:
            return
        # format input data for neural network
        input_data = json_to_matrix(self.symbol_data[self.get_date()])
        # use neural network
        prediction = self.neural.predict(input_data)
        # buy/sell based on neural network prediction and position in market
        if not self.is_long() and prediction > self.threshold:
            self.order = self.buy()
            # log(self.get_date(), 'buy', self.get_close())
        elif not self.is_short() and prediction < -self.threshold:
            self.order = self.sell()
            # log(self.get_date(), 'sell', self.get_close())

    def get_results(self):
        return self.trades
```

symbol.py

```
from __future__ import (absolute_import, division, print_function, unicode_literals)
from urllib.parse import urlencode
import requests
import csv
from time import sleep
from data import Data
from utility import *
from screener import get_symbols
```

```
class SymbolData(Data):

    def __init__(self, **params):
        self.symbol = params['symbol']
        self.options_list = params['options_list']
        self.start = params.get('start', None)
        self.end = params.get('end', None)
        self.all_data = {}
        super().__init__(symbol=self.symbol)
        self.params = params
        self.write_params()

    def get_folder(self):
        return 'symbol'

    def get_symbol_path(self):
        return self.get_path(self.symbol + '.csv')

    def write_data(self):
        write_symbol_data(self.get_all_data(), self.get_symbol_path())

    def get_all_data(self):
        if not self.all_data:
            self.get_data()
        return self.all_data

    def read_data(self):
        self.all_data = self.read_all_data()
        self.refresh_data()
        if self.all_data:
            data = self.filter_data(self.all_data)
            return data

    def filter_data(self, data):
        return filter_data(data, self.options_list, self.start, self.end)

    def read_all_data(self):
        return read_symbol_data(self.get_symbol_path())

    def get_new_data(self):
        data = download_symbol_data(self.symbol, self.options_list)
        dict_merge(self.all_data, data)
```

```
        return self.filter_data(data)

    def refresh_data(self, update_old=False):
        missing_columns = get_missing_columns(self.all_data, self.options_list)
        if update_old:
            missing_columns += get_old_columns(self.all_data)
        missing_options = columns_to_options(missing_columns)
        if missing_options:
            new_data = download_symbol_data(self.symbol, missing_options)
            dict_merge(self.all_data, new_data)
            self.write_data()


class SymbolCloseData(SymbolData):

    def __init__(self, **params):
        self.data = None
        super().__init__(options_list=[DAILY_OPTIONS], **params)

    def get_data(self):
        if not self.data:
            data = super().get_data()
            self.data = filter_close(data)
        return self.data


def download_symbol_datum(symbol, options):
    options = {
        key: value for key, value in options.items() if value is not 'columns'
    }
    log('Downloading %s data for %s...' % (options['function'], symbol))
    data = request({**{
        'symbol': symbol,
        'apikey': API_KEY
    }, **options})
    if type(data) is str:
        # raise Exception(data)
        print('sleeping 10s...')
        sleep(10)
        print('trying again')
        return download_symbol_datum(symbol, options)
    data = sanitize_data(data)
    data = convert_data(data, options)
```

```
    return data


def download_symbol_data(symbol, options_list):
    data = {}
    for options in options_list:
        new_data = download_symbol_datum(symbol, options)
        dict_merge(data, new_data)
    return data


def request(options):
    url = 'https://www.alphavantage.co/query?%s' % urlencode(options)
    data = requests.get(url).json()
    if 'Error Message' in data:
        raise Exception(data['Error Message'])
    data = next(data[key] for key in data.keys() if key != 'Meta Data')
    return data


def sanitize_data(data):
    return {date[:DATE_LENGTH]: sanitize_datum(data[date]) for date in data}


def sanitize_datum(datum):
    return {(key[3:] if key[1:3] == ". " else key): val for key, val in datum.items()}


def convert_data(data, options):
    columns = encrypt_options(options)
    return {
        date: {
            column_hash: data[date][column]
            for column, column_hash in columns if column in data[date]
        } for date in data.keys()
    }


def json_to_csv(data, date, headers):
    return [date] + list(map(lambda col: data[date].get(col, ""), headers[1:]))


def csv_to_json(datum):
```

```
    date = datum['Date']
    del datum['Date']
    return {date: datum}


def columns_to_options(columns):
    options_list = [decrypt_dict(column) for column in columns]
    for options in options_list:
        del options['column']
    return remove_duplicates(options_list)


def get_old_columns(data):
    dates = sorted(data, reverse=True)
    if len(dates) == 0:
        return []
    # check latest data against date
    if dates[0] != get_latest_weekday():
        return get_columns(data)
    # check missing data
    columns = set()
    for date in dates:
        old_columns = [k for k, v in data[date].items() if v == '']
        if len(old_columns) > 0:
            columns.update(old_columns)
        else:
            break
    return list(columns)


def get_missing_columns(data, options_list):
    present_columns = get_columns(data)
    columns = list(map(lambda c: c[1], encrypt_options_list(options_list)))
    missing_columns = list_subtract(columns, present_columns)
    return missing_columns


def get_portfolio_data(symbols, options_list, start, end, refresh):
    data = {}
    for symbol in symbols:
        data[symbol] = SymbolData(symbol=symbol, options_list=options_list, start=start,
end=end)
        data[symbol].refresh_data(update_old=refresh)
```

```
    return data


def write_symbol_data(data, path):
    with open(path, 'w') as outfile:
        csv_file = csv.writer(outfile)
        columns = ['Date'] + get_columns(data)
        csv_file.writerow(columns)
        for date in sorted(data):
            csv_file.writerow(json_to_csv(data, date, columns))


def read_symbol_data(path):
    try:
        with open(path, 'r') as csv_file:
            reader = csv.DictReader(csv_file)
            data = {}
            for row in reader:
                new_data = csv_to_json(row)
                dict_merge(data, new_data)
            return data
    except FileNotFoundError:
        return {}


def filter_data(data, options_list, start, end):
    columns = list(map(lambda c: c[1], encrypt_options_list(options_list)))
    data = filter_columns(columns, data)
    if start and end:
        data = filter_dates(data, start, end)
    data = filter_incomplete(data)
    return data


def add_symbol_args(parser):
    parser.add_argument('-s', '--symbols', type=str, nargs='+', help='symbol(s)')
    parser.add_argument('-y', '--screener', type=str, help='name of Yahoo screener')
    parser.add_argument('-l', '--limit', type=int,
                        help='take the first l symbols')
    parser.add_argument('--start', type=str, action='append', default=[],
                        help='start date of data')
    parser.add_argument('--end', type=str, action='append', default=[],
                        help='end date of data')
```

```
def add_args(parser):
    add_symbol_args(parser)
    parser.add_argument('-o', '--options', type=str, nargs='+',
                        help='indices of data_options in params.py')
    parser.add_argument('-r', '--refresh', action='store_true', help='refresh the data')


def handle_symbol_args(args, parser):
    if not args.symbols and not args.screener:
        parser.error('At least one of -s/--symbols or -y/--screener is required')
    args.symbols = get_symbols(args.symbols, args.screener, args.limit)


def handle_options_args(args, parser):
    args.options_list = get_options_list(args.options)


def handle_date_args(args, parser):
        args.start = first(args.start)
        args.end = first(args.end)


def handle_args(args, parser):
    handle_symbol_args(args, parser)
    handle_options_args(args, parser)
    handle_date_args(args, parser)


def main():
    args = parse_args('Load symbol data.', add_args, handle_args)
    data = get_portfolio_data(args.symbols, args.options_list, args.start, args.end, args.refresh)
    log({k: v.get_data() for k, v in data.items()}, force=args.print)
    if args.path:
        [log(k, v.get_path(), force=args.print) for k, v in data.items()]


if __name__ == '__main__':
    main()
```

utility.py

```
import pickle
from argparse import ArgumentParser
from hashlib import sha1
from collections import Mapping
from itertools import filterfalse
import json
from datetime import datetime, timedelta, date as Date
import numpy as np
from binascii import hexlify, unhexlify
import os
import csv
from Crypto.Cipher import AES
from params import PARAMS

DATE_LENGTH = 10
DT_FORMAT = '%Y-%m-%d'
CRYPT_KEY = '1234567890123456'
API_KEY = PARAMS['credentials']['alphavantage']
DAILY_OPTIONS = PARAMS['data_options']['daily']()


def log(*args, force=False, **kwargs):
    if PARAMS['verbose'] or force:
        args = list(args)
        for i, arg in enumerate(args):
            if type(arg) is dict:
                try:
                    args[i] = json.dumps(arg, indent=4, sort_keys=True)
                except:
                    pass
        print(*args, **kwargs)


def set_verbosity(verbose):
    PARAMS['verbose'] = verbose or PARAMS['verbose']


def shorten_path(path):
    return str(sha1(path.encode('utf-8')).hexdigest())
```

```
def make_path(path):
    dir_name = os.path.dirname(path)
    if not os.path.exists(dir_name):
        os.makedirs(dir_name)


def write_pickle(path, data):
    with open(path, 'wb') as fh:
        fh.write(pickle.dumps(data))


def read_pickle(path):
    try:
        with open(path, 'rb') as fh:
            return pickle.loads(fh.read())
    except (FileNotFoundError, EOFError):
        return


def parse_args(description, add_args, handle_args):
    parser = ArgumentParser(description=description)
    add_args(parser)
    parser.add_argument('-p', '--print', action='store_true', help='print the data')
    parser.add_argument('-v', '--verbose', action='store_true', help='enable logging')
    parser.add_argument('--path', action='store_true', help='print the data path')
    args = parser.parse_args()
    set_verbosity(args.verbose)
    handle_args(args, parser)
    return args


def filter_incomplete(d):
    return {k: v for k, v in d.items() if complete(v)}


def filter_matching(d1, d2):
    d1 = filter_incomplete(d1)
    d2 = filter_incomplete(d2)
    d1 = {k: v for k, v in d1.items() if k in d2}
    d2 = {k: v for k, v in d2.items() if k in d1}
    return d1, d2
```

```
def complete(datum):
    if type(datum) is dict:
        for _, v in datum.items():
            if v is None or v == '':
                return False
    return True


def json_to_matrix(data):
    if type(data) is dict:
        return np.array([json_to_matrix(data[k]) for k in sorted(data)])
    return float(data)


def encrypt_dict(d):
    e = AES.new(CRYPT_KEY, AES.MODE_CFB, CRYPT_KEY)
    s = json.dumps(d, sort_keys=True)
    return hexlify(e.encrypt(s)).decode('utf-8')


def decrypt_dict(crypt):
    e = AES.new(CRYPT_KEY, AES.MODE_CFB, CRYPT_KEY)
    s = e.decrypt(unhexlify(crypt))
    return json.loads(s.decode('utf-8'))


def dict_merge(dct, merge_dct):
    for k, v in merge_dct.items():
        if (k in dct and isinstance(dct[k], dict)
                and isinstance(merge_dct[k], Mapping)):
            dict_merge(dct[k], merge_dct[k])
        else:
            dct[k] = merge_dct[k]


def merge_data(datum_list):
    data = {}
    for datum in datum_list:
        dict_merge(data, datum)
    return data
```

```
def list_subtract(l1, l2):
   return list(filterfalse(lambda x: x in l2, l1))


def remove_duplicates(l):
   return [i for n, i in enumerate(l) if i not in l[:n]]


def encrypt_options(options):
   return [(column, encrypt_dict({**options, **{'column': column}}))
         for column in options['columns']]


def encrypt_options_list(options_list):
   return [column for options in options_list for column in encrypt_options(options)]


def get_daily_crypt():
   return dict(encrypt_options(PARAMS['data_options']['daily']()))


def get_close_crypt():
   return get_daily_crypt()['close']


def filter_columns(keep, data):
   return {
      date: {
         column: val
         for column, val in columns.items() if column in keep
      } for date, columns in data.items()
   }


def extract_column(column, data):
   return {date: columns[column] for date, columns in data.items()}


def get_latest_weekday():
   today = Date.today()
   latest_day = today - timedelta(max(4, today.weekday()) - 4)
   return from_date(latest_day)
```

```
def to_date(date):
    return datetime.strptime(date, DT_FORMAT)


def from_date(date):
    return date.strftime(DT_FORMAT)


# inclusive
def date_between(date, start, end):
    date = to_date(date)
    start = to_date(start)
    end = to_date(end)
    return start <= date <= end


def filter_dates(data, start, end):
    return {
        date: val for date, val in data.items() if date_between(date, start, end)
    }


def filter_close(data):
    close_hash = get_close_crypt()
    return {date: float(columns[close_hash]) for date, columns in data.items()}


def get_columns(data):
    cols = set()
    for date in data.keys():
        if len(date) == DATE_LENGTH:
            cols.update(data[date].keys())
    return list(cols)


def get_csv_headers(path):
    try:
        with open(path, 'r') as csv_file:
            reader = csv.reader(csv_file)
            headers = next(reader)
            return headers
    except FileNotFoundError:
```

```
        return []

def dicts_to_xys(dicts):
    keys = set()
    for d in dicts:
        keys.update(d.keys())
    keys = list(enumerate(sorted(keys)))
    xys = []
    for d in dicts:
        x = []
        y = []
        for i, v in keys:
            if v in d:
                x.append(i)
                y.append(d[v])
        xys.append((x, y))
    return xys


def get_options(options_str):
    paren = options_str.find('(')
    if paren == -1:
        return PARAMS['data_options'][options_str]()
    name = options_str[:paren]
    args = options_str[paren + 1:-1].split(',')
    return PARAMS['data_options'][name](*args)


def get_options_list(options_strs):
    return [get_options(options_str) for options_str in options_strs]


def first(l):
    return l[0] if len(l) else None
```

APPENDIX D: Gap Strategy Code

```
{ Helpful instructions on the use of EasyLanguage, such as this, appear below and are
  contained within French curly braces {}.  There is no need to erase these
  instructions when using EasyLanguage in order for it to function properly, because
  this text will be ignored. }

{ STEP 1 OF 2: Replace <CRITERIA> with the criteria that will trigger a Buy at the open
  of the next bar using a market order. }

Inputs:
GapSize(.01),
FudgeFactor(10000);

Variables:
        BullMarket(false),
        BearMarket(false),
        NewDay(true),
        NumShares(1),
        isGap(false);

If (Date <> Date[1]) then begin
        NewDay = True;
End;

NumShares = FudgeFactor / open;
BullMarket = Open < closeD(1); {going to be buying}
BearMarket = Open > closeD(1); {going to be shorting}

isGap = AbsValue(openD(0) - closeD(1)) > closeD(1)*GapSize;


Condition1 = isGap and BullMarket;
Condition2 = isGap and BearMarket;

{ STEP 2 OF 2: Replace "Entry Name" (leaving the quotes) with a short name for the
  entry.  The entry name will appear on the chart above/below the trade arrows and in
  the trade by trade performance report. }
```

```
If NewDay then begin
	If condition1 or condition2 then begin
		if Condition1 then Begin
			Buy ( "found long Gap" ) NumShares shares on the next bar at market ;
		End;

		If Condition2 then Begin
			Sell Short ( "foudn short Gap") NumShares shares on the next bar at
market;
		End;

		NewDay = false;

	End;
End;

If BullMarket  then Begin
	If highest(High, 1) > (Open +closeD(1)*GapSize) then Begin
		Sell this bar;
	End;
end;

If BearMarket then Begin
	If lowest(Low, 1) < (Open -closeD(1)*GapSize) then Begin
		Buytocover this bar;
	end;
end;

If t = 1555 then Begin
	If BullMarket  then Begin
		Sell this bar;
	end;

	If BearMarket then Begin
		Buytocover this bar;
	end;
End;
```