1. Introduction.

The authors (Frank Kornet and Harshad Hazariwala) have an interest in stock markets, and how machine learning can be used in trading stocks. One of the authors developed a trading machine learning system called "Stockie" (https://github.com/frkornet/Stockie) while attending a data science bootcamp. We used this system as a starting point and focused on improving it using what we learned in Georgia Tech courses, specifically Computational Data Analysis (ISyE 6740), Machine Learning for Trading (CSE 7646), and Financial Modeling (MGT 8813).

Figure 1 shows how the main Stockie components interoperate:

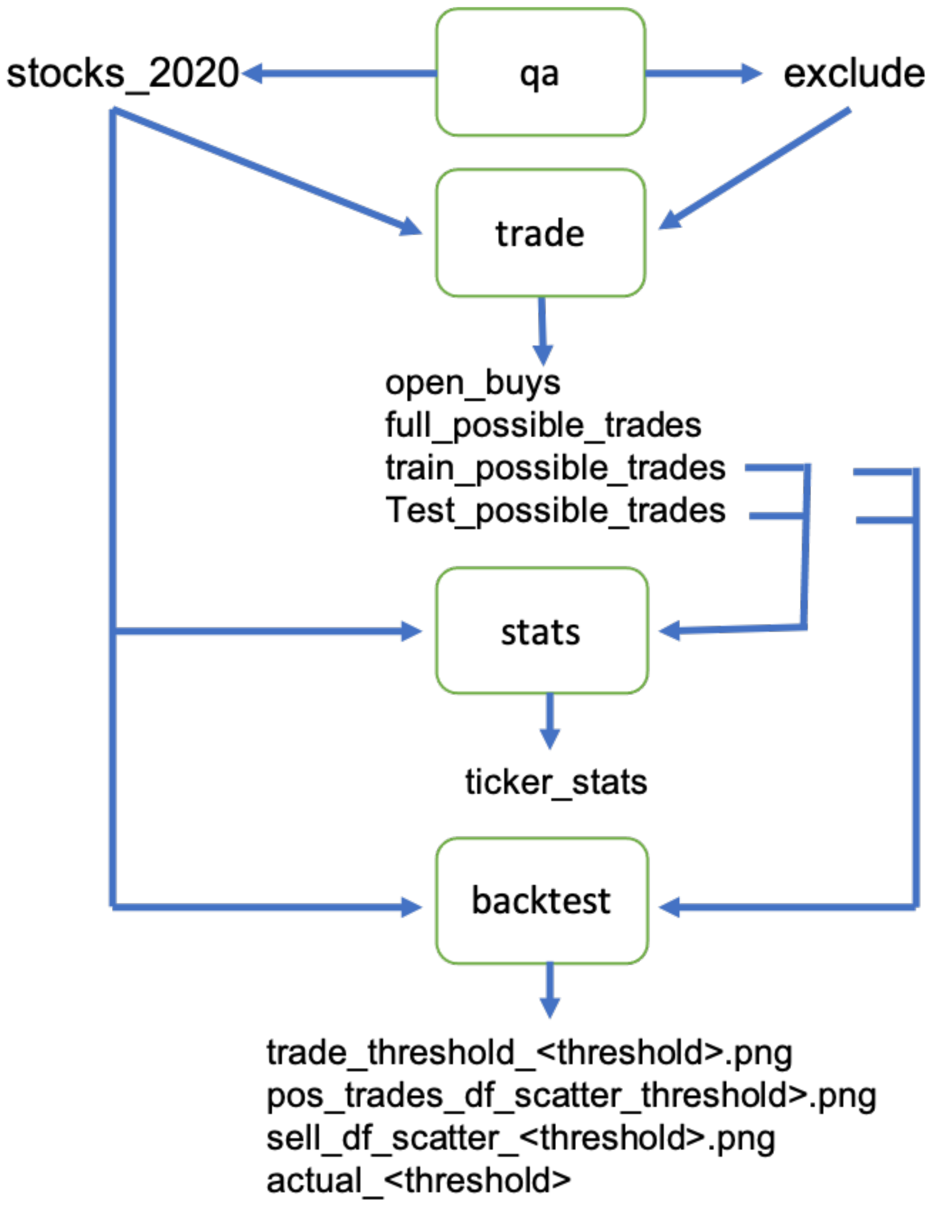

Figure 1: Main Components Stockie.

The files without extensions are CSV files. The key part of the machine learning trading system resides in trade.py where the possible trades are generated (a full buy and sell cycle). The program backtest.py uses heuristics to select trades to execute out of this pool of possible trades. To keep the scope manageable for the Computational Data Analysis manageable, we will focus on analyzing and improving trade.py.

We create a GitHub repository where we stored our code and solutions. You can find it at https://github.com/frkornet/CDA.

2. Stockie.

The trade.py program takes in the stock ticker symbols in stocks_2020.csv file. This file contains the list of stocks that Renaissance Technologies and Rockwell held as of May 2020. This list is compiled from the F-13 filings to the Stock Exchange Committee and was put together using web scraping. The assumption is that if those two large hedge funds consider it worthy to hold positions in these stocks, then these must be a good subset to start with. It avoids having to deal with all the stocks that are traded on the stock exchange in the US. It cuts down the subset from 13,000 symbols to 2,000. The exclude.csv contains stock symbols that need to be excluded for quality assurance reasons.

For each stock symbol, we build a smoothed price curve to reduce the noise that exists in the share price. This is then used to identify local minima and maxima using Numpy's argmin and argmax functions. These local minima and maxima are then used to train a logistic regression model to determine the buy and sell signals in the test data.

The buy and sell signals are combined into possible trades. The possible trades are stored as CSV files and are then processed by the backtester to determine which possible trades to execute. The backtester uses heuristics to decide which stocks to keep and which stocks to sell. It does that based upon the performance of the stocks to date. It aims to select the stocks that have historically shown the best performance in terms of daily returns.

3. Problem Statement.

In Figure 2, you can see a summary of the performance of the logistic regression model in trade.py. The performance shows the performance on training data and the performance on the test data. The training data is 7 years' worth of data and consists of 36,950 possible trades. The testing data consists of 3 years' worth of data and consists of 21,950 possible trades. The data spans a ten-year period (i.e., from 13 February 2014 to 13 February 2021). The possible trades are in 2,389 different stock symbols.

The charts in the top row of Figure 2 split trades into trades that have a positive return ("gain"), trades that have a negative return ("loss"), and trades that have a zero or close to zero return ("zero"). For each type we show the row count for each type.

The charts show that there are roughly an equal number of trades that have a positive or negative return in the training data. This is a problem as it potentially allows the logistic regression model to learn the “wrong” thing. So, as a minimum we need to eliminate the negative trades and use the positive trades to train the logistic regression model for each stock.

The top row in Figure 2 for testing data, shows a different pattern compared to that of the training data. The trades that have a positive return outweigh those that have a negative return, esp. up to around 50 trading days. This suggests that the logistic regression model is able to pick up signals for positive trades to a degree that is higher than what you would expect to see if the efficient market hypothesis were true.

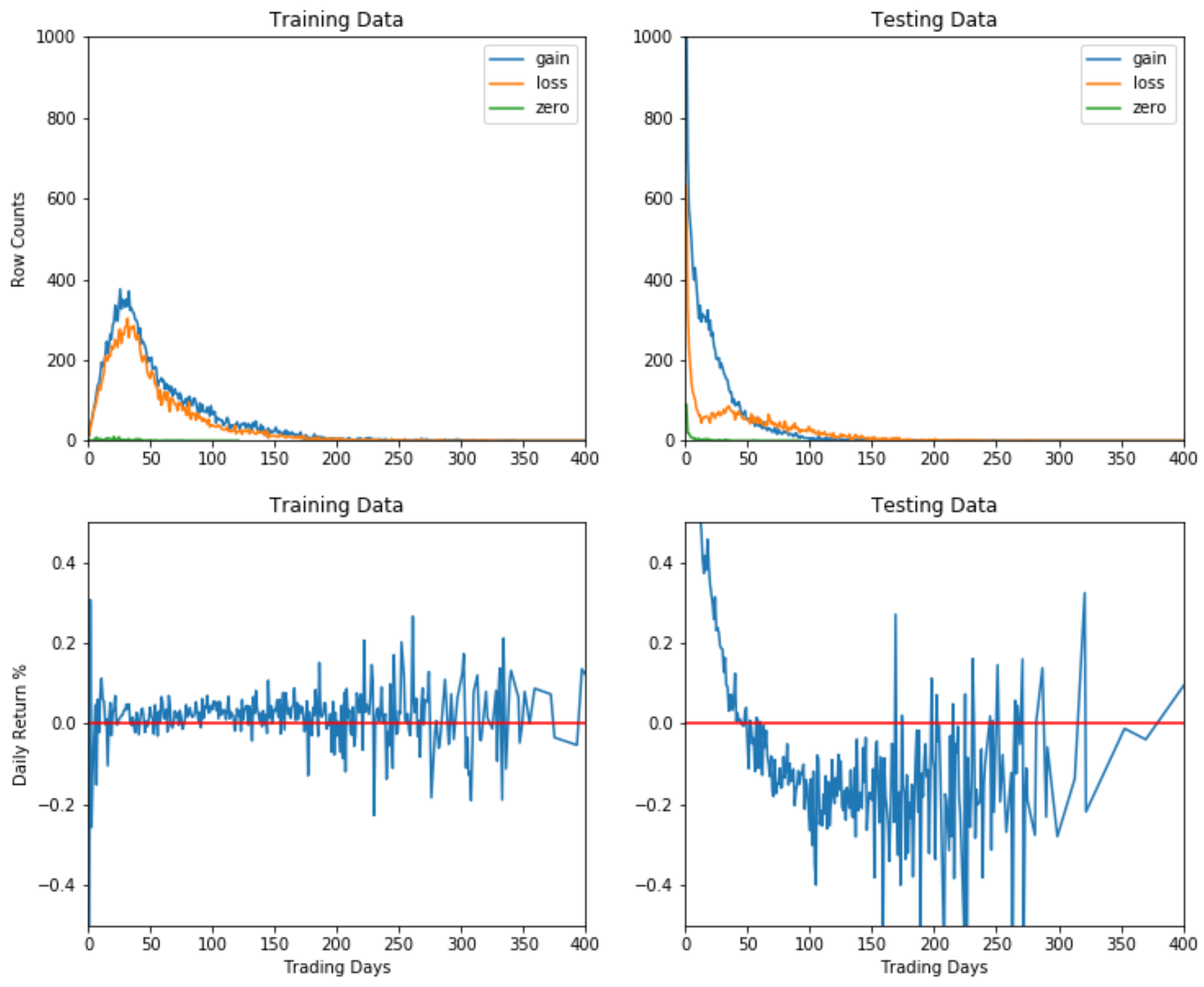

Figure 2: Performance of Logistic Regression Model

The bottom row of Figure 2 shows the overall daily return for each trading day. It executes all the trades for each trading day and then calculates an overall daily return for that particular trading day. This is done for each trading day and then plotted. This allows us to see how the daily return changes as the number of trading days for a trade increases. The plot on the left for training data is what you would expect if the number of possible trades have about an equal number of positive and negative returns.

The plot on the right for testing data shows a different picture and reflects the fact that the number of positive trades outweighs the negative trades up to about 50 trading days. Up to 50 trading days, the logistic regression model does better than on the training data. Beyond 50 trading days the logistic regression model does worse than on the training data since the daily returns seems to fluctuate around -0.2 % daily return (since there around 252 trading days per year this equates to an annual loss of 40 %).

The current implementation of trade.py does not implement fundamental trading or technical trading indicators. We want to determine whether we can improve the performance by adding technical and fundamental indicators. To implement the indicators, we will use open data sources on the web. As good proprietary data can cost millions of dollars and we simply don't have that money available. If we had, we'd prefer to use it to trade and make money that way! The other reason is that a lot of the good proprietary data (models) are used by the major stock market players making them more likely to have the similar models. We believe that having simple and unique models will allow us to outperform the market in the long run. It will require hard work to build these models.

4. Data Source.

As mentioned in the Problem Statement, we are using the python yfinance package to retrieve historical Yahoo Finance price and volume data. It is used to provide daily historical data. Yahoo Finance does not provide intraday historical volume and price data. It does provide intraday volume and price data for the ongoing trading day (i.e., as off the time of the call). It also does provide limited fundamental data (up to four years for annual reports and up to four quarters of the current finance year).

As part of the project, we downloaded the stock data into CSV files to reduce the need for network traffic. We wrote a dedicated Python program (yfload.py) to do, so. In the CSV files we store the daily Close, Open, Low, High, Volume, Stock Splits, and Dividends. We augmented the data with an extra field for the smoothed Close price. The smoothing was implemented using the savgol filter. The stock prices stored in the CSV files is from March 2011 to March 2021.

Since yfinance only provides limited fundamental data, we extended it using data stored in Edgar (SEC) using the XBRL interface. The XBRL (https://www.sec.gov/structureddata/osd-inline-xbrl.html) interface allows access to financial documents filed electronically since 2009.

We used this interface to store a local copy of all financial documents filed at SEC since 2009, including annual reports (K-10) and quarterly reports (Q-10). The 40 Gb data set was downloaded using by a dedicated python program (peload.py), and after downloading the tab separated files for each quarter, the data was converted to parquet data files to save space on disk and to allow quick access to the data.

Each quarter contains the files shown in Table 1. We work mostly with the SUB files to identify companies, and NUM to retrieve numerical data on the company (e.g., earnings per quarter).

File	Data Set	Description
SUB	Submission	Identifies each XBRL submission by company, form date, etc.
TAG	Tag	Defines and explains each taxonomy tag.
DIM	Dimension	Adds detail to numeric and plain text data.
NUM	Numeric	One row for each distinct data point in filing.
TXT	Plaintext	Contains all non-numeric XBRL fields.
REN	Rendering	Information on rendering on SEC website.
PRE	Presentation	Detail on tag and number representation in primary statements.
CAL	Calculation	Shows arithmetic relationships among tags.

Table 1: Edgar Tab Separated Files

To access the Yahoo finance data set you need a stock ticker symbol, e.g., AAPL for Apple. To access the Edgar data you need a company identifier called cik. The cik is a nine-digit number. Unfortunately, there is no easy way to link ticker symbols and cik identifiers. The only way was to use the company name stored in Yahoo finance and the company name used by Edgar. We used a simple one to one mapping. In that manner we were able to match 450 stocks out of the original 2650 stocks we have in the stocks_2020.csv file.

Since there is no easy solution to the problem, we restricted ourselves to the 450 stocks that we were able to match to Edgar data. One benefit is that limiting ourselves to 450 stocks allowed us to shorten the develop-test cycle for the code we had to write.

We use the Edgar files to implement a rolling four quarters price-to-earnings ratio for each stock in our 450 stock trading universe. Not all of these 450 stocks filed their quarterly reports electronically in 2009. Most of them did so, however, from 2014 onwards. The technical indicators are implemented using the Yahoo finance data stored in CSV files.

5. Methodology.

5.1 Model

The logistic regression model uses the following features: MACD<window>, PctDiff<window>, and StdDev_<window> where <window> is 3, 5, 10, 15, 20, 30, 45, or 60 days. The model uses 24 (=3 * 8) features.

Each stock that we trade will have two logistic regression models. One for generating buy signals and one for generating sell signals. The specific logistic regression model that we use, is a weighted balance of evidence score, and is typically used in credit scores to determine the default risk of applicants. It works well where the data is unbalanced and the class we are interested in is underrepresented. This is the case for trading, as trading signals are rare.

The model is implemented in Python using the WOEEncoder and KBinsDiscretizer in sklearn. A description of the weight of evidence model can be found at https://www.listendata.com/2015/03/weight-of-evidence-woe-and-information.html.

The local minima and local maxima are identified on the smoothed price curve. This is done using the SciPy argmin and argmax functions. The local minima are used as targets for the buying model and the local maxima are used as targets for the selling model. Figure 3 shows an example for Apple from June 1st, of 2020 to February 12th of 2021. It shows that the actual Close price bounces around the smoothed curves. Sometimes above it. Sometimes below it. There is a kind of randomness to whether the Close price is above or below the smoothed curve at a particular date.

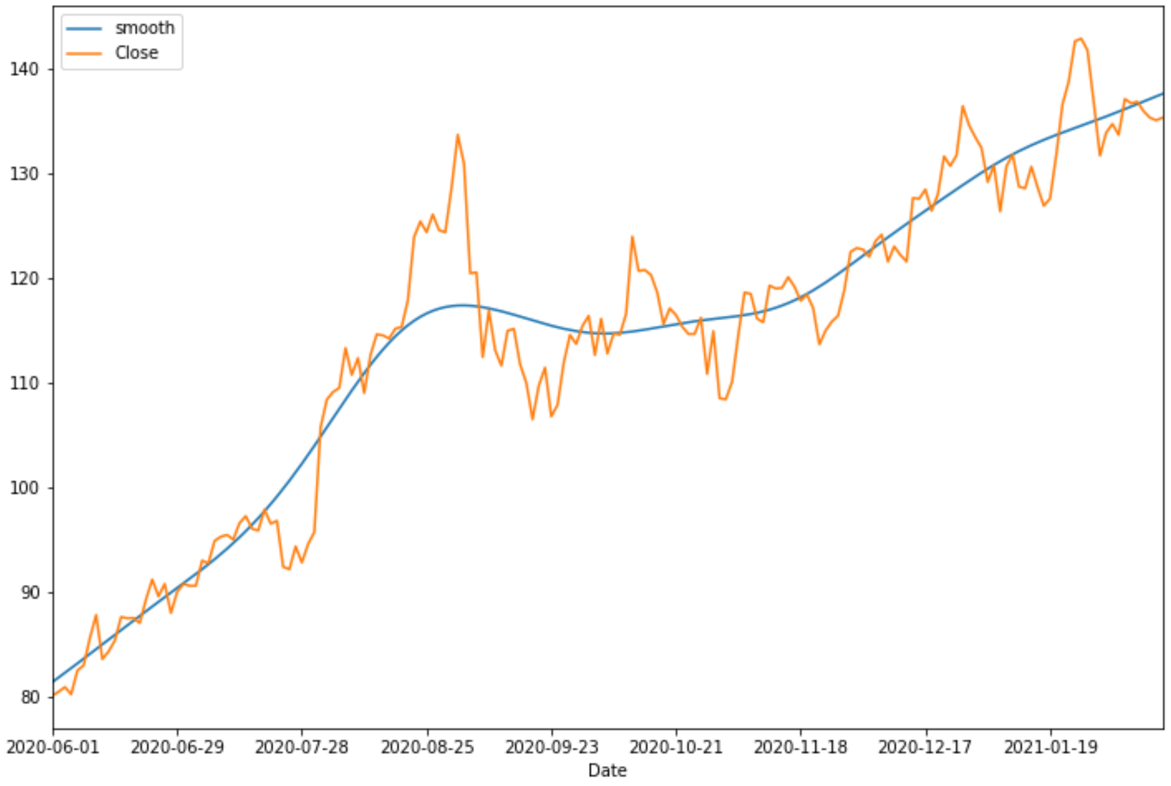

Figure 3: Smoothed versus Close Curves for Apple.

Since the buy and sell signals are generated by separate models, we need to pair the buy and sell signals. This is done by implementing a state transition diagram, see Figure 3. The states are shown as circles and the transitions as arrows. The label is the signal that is received. State 2 is used to lock the buy signal and state 4 is used to lock the accompanying sell signal that goes with the buy signal. The sell transition in state 1, the buy transition in state 2, the buy transition in state 3, the sell transition state 4, and sell transition in state 5 are used to eliminate unbalanced signals.

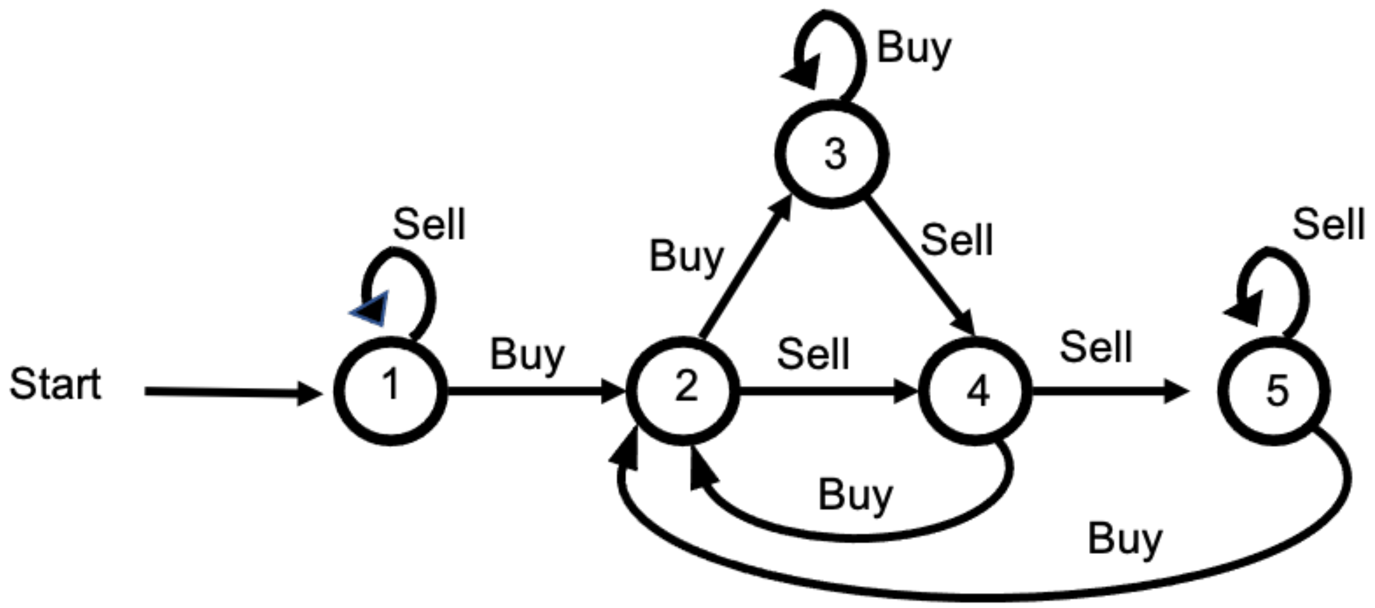

Figure 3: State Transition Diagram for Linking Buy and Sell Signals

The above logistic regression model is simple, easy to create for each stock, and relatively fast. The limitation is that it is a linear model and stock markets are clearly not linear. So, it will clearly have limits.

The model is trained on seventy percent of the data and tested against the remaining thirty percent. We retrieve stock data for the last ten years. So, about 7 years is used for training and creating indicators. The remaining 3 years is for testing.

Some stocks will not have a full ten year worth of data because they were not listed yet at the time or they were delisted at some point in time. To deal with this, we require a minimum of five trading years before the stock is included. A consequence of this is that there is not a single start date for testing data. It varies between stocks.

5.2 Remove Negative Trades

The original implementation does not filter out negative trades in the training data. As a result, the model may learn the wrong target as a result. This may occur due to the difference between Close price and the smoothed curve price. We address this by preprocessing the local minima and maxima pairs. If the pair is negative we try to shift the local maxima and minima date a few days back and forth to ensure the trade is positive. We, furthermore, demand that the trade achieves at least a daily return of 0.3% and that the number of trading days falls within five to fifty-five trading days.

A consequence of removing negative trades from the training data is that negative trades are no longer present in the training data while they are present in the test data. As a result, correlation is no longer a good measure to use to determine how well the system is performing.

We need a different measure to select the best performing model. We decided to use the percentage of positive trades with less than fifty-five trading days. The measure builds on the fact that shorter trades appear to have higher daily returns than longer lasting trades (see Figure 2 above for evidence).

5.3 Efficient Market and Random Walk

The stock market is deemed to be efficient and is the key underlying concept in the Capital Asset Pricing Model (CAPM) and option pricing models Black Scholes option pricing model. A consequence of the efficient market assumption is that it is not possible to predict the direction of the stock market or stock prices. The movement of stock prices is considered a random walk. Sometimes it goes up, and sometimes it goes down. The decision to go up or down is, however, random and each has a probability of fifty percent.

To get a sense of how the performance of the original version and final version compares we implemented a random walk simulation to get a benchmark. To simulate this, we randomly selected a) the stock to trade, b) the buy trading date, and c) the number of trading days to hold the stock before selling it. The stock is selected uniformly from our stock trading universe. We also select the trading date uniformly from the test data period for the stock. The trading day is selected based upon the trading day distribution of the original solution. This is done to get more resemblence between the random walk result and the original and final solution results.

The random walk is simulated a thousand time, and then the average is taken. Each simulation contains about 10,000 trades.

5.4 Technical and Fundamental Indicators

We selected to implement the following technical indicators Relative Strength Index (RSI), Money Flow Index (MFI), Williams R Percentage (WRP), and Bollinger Band Position (BBP). RSI is a well known indicator for capturing the momentum of a stock. The WPR is another momentum indicator. The RSI and MFI indicators aim to quantify how fast the stock price is increasing. The MFI is an indicator that gives insight whether or not a stock is oversold or overbought. The BBP gives an indication of how far away the stock price is from its mean. Both MFI and BBP have an underlying assumption that the stock price will sooner or later revert to its mean. The indicators are described below.

Relative Strength Index (RSI):

RSI is a momentum indicator used in technical analysis of stocks that measures the magnitude of recent price changes to evaluate overbought or oversold conditions in the price of a stock. The RSI oscillates between 0 to 100 where values above 70 indicate a stock maybe overbought and it may reverse trend and pullback in price. A value below 30 indicates the stock maybe oversold and it may reverse trend and appreciate in price.

RSI is a two-part calculation. Step 1:

$$RSI_{step\ one} = 100 - \frac{100}{1+\frac{Average\ gain}{Average\ loss}}$$

The standard is to use 14 periods to calculate the initial RSI value, but the number of periods can be changed. For our project, we played with different windows (ex. 10 periods, 20 periods etc) to find which one would produce the best results. The second step of the calculation smooths the results:

$$RSI_{step\ two} = 100 - \frac{100}{1+\frac{Previous\ avg\ gain\ \cdot\ (windows-1)\ +\ Current\ gain}{Previous\ avg\ loss\ \cdot\ (windows-1)\ +\ Current\ loss}}$$

William Percentage Range (WPR):

WPR is a type of momentum indicator that moves between 0 and -100 and measures overbought and oversold levels. A reading above -20 is considered overbought and it may be a good time to sell the stock. A reading below -80 is considered oversold and it may be a good time to buy the stock. WPR can be calculated as follows:

$WPR\ = \frac{Highest\ High\ -\ Closing\ Price}{Highest\ High\ -\ Lowest\ Low}$, where Highest High is the highest price in the last x days, e.g. 14 day period, and Lowest Low is the lowest price in the same period.

Money Flow Index (MFI):

MFI is a technical oscillator that uses price and volume data for identifying overbought or oversold signals in a stock. The oscillator moves between 0 and 100. Unlike conventional oscillators such as the Relative Strength Index (RSI), the Money Flow Index incorporates both price and volume data, as opposed to just price. For this reason, MFI is also known as volume-weighted RSI. MFI can be calculated as follows:

$MFI = 100 - \frac{100}{1\ +\ Money\ Flow\ Ratio}$ where

$Money\ Flow\ Ratio = \frac{(window-1)\ period\ Positive\ Money\ Flow}{(window-1)\ period\ Negative\ Money\ Flow}$.

$Money\ Flow = Typical\ Price \cdot Volume$, and

$Typical\ Price = \frac{High\ +\ +\ Low\ +\ Close}{3}$.

Bollinger Band Position:

A Bollinger Band is a technical analysis tool defined by a set of trendlines plotted two standard deviations (positively and negatively) away from a simple moving average (SMA) of a stock's price. The closer the prices move to the upper band, the more overbought the stock, and the closer the prices move to the lower band, the more oversold the stock.

The first step in calculating Bollinger Bands is to compute the period simple moving average, e.g. 20-day moving average. Next, multiply that standard deviation value by two and both add and subtract that amount from each point along the SMA. Those produce the upper and lower bands. Finally, subtract the lower band from the stock price and divide that by the difference between the upper band and lower band. That gives you the relative position to the Bollinger Band.

$$Upper\ Band\ =\ Period\ Moving\ Average + 2 \cdot \sigma$$

$$Lower\ Band\ =\ Period\ Moving\ Average - 2 \cdot \sigma$$

$$Bollinger\ Band\ Position = \frac{Closing\ Price - Lower\ Band}{Upper\ Band - Lower\ Band}$$

The implementation of RSI took the most effort. First of all, we struggled to figure out what the correct definition is and how it should be implemented. A lot of the implementation floating calculate it incorrectly using vectorization. The incorrect version assumes that you only need to calculate the moving gain and loss average. The correct version is more complicated and uses the previously calculated average gain and adds to that the current gain to determine the new average gain. The average loss is calculated in a similar manner.

Secondly, we needed to speed up the naive RSI implementation to allow us to use it with generating trades for 450 stocks. The correct solution cannot be fully vectorized and we tried different solutions until we had a version that is reasonable fast while still being correct. It also turned out that the RSI imlementation from ML4T does not conform with the official definition.

Price/Earning Ratio:

We implemented the Price/Earning (P/E) as the fundamental indicator. It is an essential fundamental indicator as it indicates how much people are paying for holding the stock. Normally, the P/E ranges from fifteen to twenty-five. P/E ratios above twenty-five indicate that investors believe that the stock has a lot of potential future growth in the coming years and are willing to pay more for the stock to hold it.

The P/E ratio is implementing using Edgar data that was downloaded. The Edgar data set allows us to access the quarterly earning using the 'EarningsPerShareDiluted' tag in the NUM file (see Table 1 above). The P/E ratio is then calculated by dividing the daily Close price by the latest filed earnings.

The implementation of the P/E ratio is slightly more complicated than it appears as it needs to deal with stock splits. To do so, we adjust the quarterly earnings for stock splits before we divide the Close price by the latest quartly earnings.

Note that the P/E ratio is a lagging indicator as it relies on the latest quarterly earnings. There is a different version of the P/E ratio that uses the expected next quarterly earnings. We were unable to implement this in this project as we do not have a proper model to predict the future expected earnings for a stock.

After implementing the described indicators, the final step was to integrate them into trade.py program. This took longer than expected as we ran into problems that we did not realize at first. The root cause of the problems turned out to be the fact that the indicators did not protect against division by zero. As a result, some of the indicator values became infinite for certain dates, and the logistic regression model in sklearn does not handle infinite values well. Once the true source of the problem was identified, it was easy to integrate and run tests with them to confirm all was working.

Since we were interested to see which features would perform best, we carried out tests with all possible combinations of the feature groups. We considered the following eight feature groups: MACD, PctDiff, StdDev, RSI, WPR, MFI, BBP, and P/E ratio. Our code had to consider $2^8 = 256$ combinations. We reduced the search space by only considering three or more feature combinations (219 combinations left).

Once we identified the best feature combination, we tested different window configurations for the features to determine which combination of days works best. When we determined the optimal days for that set we reran the job that considered all possible 219 combinations with the optimal day combination to make sure that it would not change the best feature combination.

5.5 Back Testing

We did not plan on using the backtester but we could not resist the temptation to try to see what the performance difference would be between the original model and the final model with the best feature and best window day combination. It turned out to be relatively easy to implement. We only had to modify some calls to make sure the downloaded data was used when running the backtest.

The backtester is restricted to holding five stocks at any point in time, and starts with an investment capital of USD 10,000. This is done to force the backtester to regularly buy and sell stocks. Furthermore, the backtester implements a stop loss at minus ten percent. Meaning, if a stock we are invested in loses more than twenty percent it will trigger an automatic sell. This is done to manage the downside risk. The backtester does not simulate short selling of stocks. Note that the system can easily be extended to handle short selling by reversing the order of buy and sell signals. This is, however, considered a higher risk strategy and we would only want to consider that once we have confirmed that the buy low and sell high trading strategy works in practice.

The backtester uses simple heuristics to determine every trading day which stock to sell and which stock to buy. It records for each stock the sample mean up to that point for daily return of all positive trades, all negative trades, all zero trades, and all trades combined. Each trading day it will first go through all sell signals for the day and sell any of the stocks with a sell signal we are invested in. We then go through all buy signals for the trading day and determine the top 5 best performing stocks with buy signals. We compare that top 5 against the performance of the stocks we are currently invested in. We determine the best performing 5 stocks overall and carry out sell and buy stocks for the day to achieve the best possible portfolio for that trading day.

The backtester has parameters that control what it does during the back test run. It can run multiple backtest simulations to give a sense of the range of possible outcomes. A short description of the key parameters is given below. The parameters evolved over time and need to be reviewed.

There are two types of daily return that can be considered by the backtester. The first one is the total daily return and the second is the daily return of the positive trades. The total daily return, typically is more conservative and results in a lower final return. The daily return of positive trades typically results in higher returns but is more volatility along the way. Which one is used is determined by the 'return_col' parameter.

The second key parameter controls whether the statistics collected are updated as trades complete during the run. There is typically a small difference in the final results. Updating statistics can sometimes improve results but does not always do so. If we have specified that we want to update the stock statistics, we can also specify how many months the want to keep the statistics for. By default, we keep it for the full period. We can, for example, specify to keep the statistics for the last three months or the last two years.

The third key parameter is whether statistics need to be batched before calculating new statistics. This is expressed as a percentage of total stocks in trading universe that have changed. This is done for performance reasons.

The final and fourth key parameter is the thresholds of good trades we want to have before we are allowed to trade in the stock. Good trades are considered trades that have more than 0.5 percent daily return. The threshold is a percentage of total trades. By default, it does not check the percentage of good trades. If the threshold value is larger than zero, it checks the percentage of good trades for the stock.

5. Final Results and Evaluation.

The section is broken down into a discussion of the original solution and the final solution. Then it goes into details for the backtester. And, finally, we cover cross validation and forward testing before we move to the next and final section on next steps.

6.1 The Original Solution

The market is deemed to be efficient and it is impossible to predict the direction of the stock market or stock prices. As a result, we carried out a random walk simulation as described above. Table 2 shows the result of the random walk simulation. The total does not add up to one hundred percent due to rounding errors.

	Pos. Trades	Neg. Trades	Zero Trades	Total
< 55 days	35.74%	28.95%	0.48%	65.17%
>= 55 days	18.92%	15.87%	0.03%	34.82%
	54.66%	44.82%	0.51%	99.99%

Table 2: Random Walk Simulation Results

You can see the positive trades are slightly above fifty percent and that is probably due to the trading day distribution we used from the original solution during the simulation. We did not check this, but we expect the result between positive and negative trades to be closer to fifty percent each if we use a uniform distribution for the number of trading days.

The corresponding result for the original solution is shown in Table 3. A key difference is that the percentage of positive trades goes up from about fifty-five percent to about sixty-seven percent, an increase of around twelve percent. There is roughly a four percent reduction in trades under fifty-five days between random walk and original solution.

	Pos. Trades	Neg. Trades	Zero Trades	Total
< 55 days	51.04%	9.92%	0.27%	61.23%
>= 55 days	15.79%	22.92%	0.06%	38.77%
	66.83%	32.84%	0.33%	100.00%

Table 3: Trades Breakdown for Original Solution

The fifteen percent increase for short term positive trades is significant as short trades typically have a higher return than short trades based upon the features we are using (see Figure 2.)

The table below summarizes the result from the backtest run for the original solution for some parameter combinations. The trading starts with an initial investment of USD 10,000. We see that the final result ranges from USD 27,000 to USD 46,000 with the majority in the USD 35,000 to USD 41,000 range. The Final Capital is calculated as USD 10,000 + Total Gains - Total Losses. The trading takes place over a three year period with the majority of the possible trades towards the second half. The daily return ranges from 0.16% to 0.18%.

Update Stats	Batch Update	Keep Stats	Return column	Threshold Percentage	Final Capital	Total Gains	Total Losses	Daily Return
Yes	5 %	0m	gain_daily_ret	0	$ 35k	$ 61k	$ 36k	0.16%
No	N/A	N/A	gain_daily_ret	0	$ 41k	$ 68k	$ 38k	0.18%
Yes	5 %	0m	gain_daily_ret	10	$ 38k	$ 65k	$ 38k	0.17%
No	N/A	N/A	gain_daily_ret	10	$ 46k	$ 76k	$ 39k	0.19%
Yes	5 %	0m	gain_daily_ret	20	$ 36k	$ 64k	$ 38k	0.16%
No	N/A	N/A	gain_daily_ret	20	$ 27k	$ 47k	$ 30k	0.13%

Table 4: Backtest Performance Results for Original Solution

Figure 4 shows how the final capital grows over the trading period for the second row of Table 4. The chart is produced by the backtester. You can see that it is mostly flat for two thirds of the time and then starts to climb materially. This is partly caused by the fact that testing period does not start the same data for all stocks. As a result, there are less trades to chose from at the beginning and when there are less trades there are typically not many 'good' trades to choose from. As we trade in more stocks, we have a better chance of finding a good stock to invest in. At this point, we are unable to quantify whether this is indeed the cause or whether there is something else going on. This is something we need to investigate to get more confidence in the system overall.

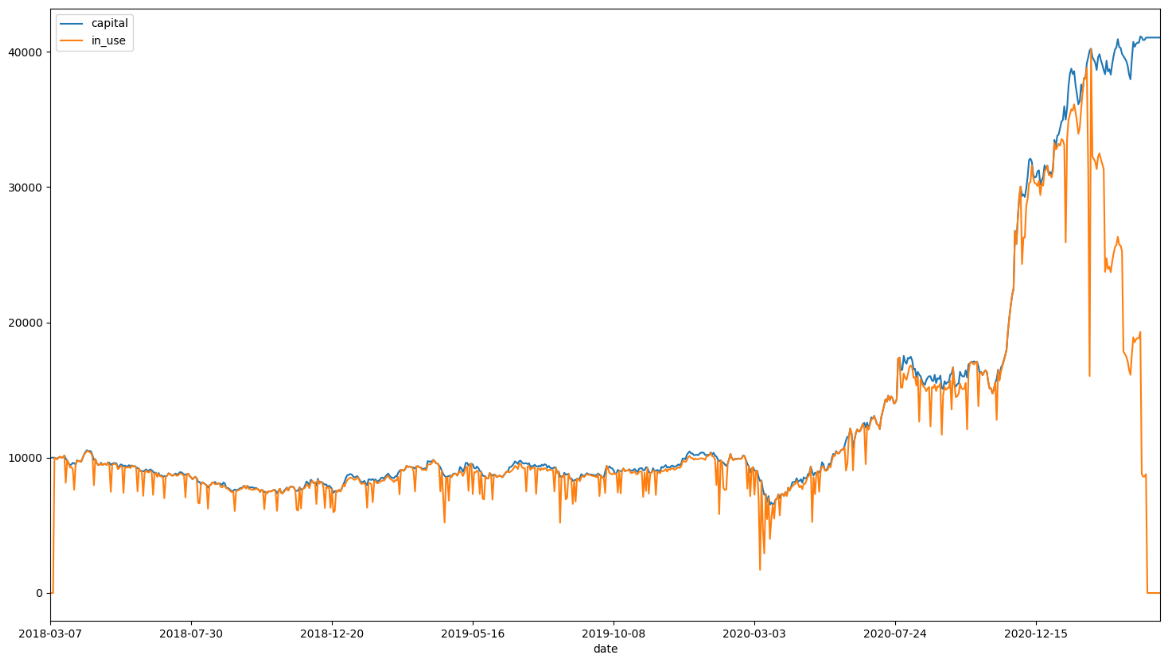

Figure 4: Original Performance Example

6.2 The Final Solution

With a sense of the performance of the original solution, we can now look at the performance of the final solution. We found that the best combination of feature groups consists of three feature groups, in particular the MACD, RSI and BBP groups performed best. These are well known technical indicators, and as such not a major surprise. We also saw that the inclusion of P/E ratio reduces performance. We suspect the reason for only three features groups is that there is correlation between the features and logistic regression is known to not handle correlation between features well.

We also found modifying the windows to calculate rolling indicators improved results. The original windows consisted of 3 days, 5 days, 10 days, 15 days, 20 days, 30 days, and 60 days. We found that by reducing the number of windows, we get better results. We suspect that this is due to reducing correlation between the features within a feature group and with other feature groups. We decided to take only three different trading days as that seemed to performed the best.

We, furthermore, found that by focusing on shorter trading day windows we could increase the percentage of trades under fifty-five days. In the end, we decided to go with 10 days, 20 days, and 30 days windows.

Table 5 shows the percentage break down of positive trades, negative trades, and zero trades. We see that the final solution increases the percentage of positive trades under fifty-five trading days by nineteen percent. That is quite an increase, esp. considering that the better trades reside in the beginning. The percentage of trades under fifty-five trading days increased by twenty-five percent overall. The gains come at the expecnse of a six percent increase for negative trades under fifty-five trading days.

	Pos. Trades	Neg. Trades	Zero Trades	Total
< 55 days	69.85%	16.07%	0.37%	86.29%
>= 55 days	2.66%	11.03%	0.02%	13.71%
	66.83%	27.10%	0.39%	100.00%

Table 5: Trades Breakdown for Final Solution

In the problem statement we showed a set of plots (see Figure 2) to visualize the problem. For completeness, we show the equivalent charts for the final solution in Figure 5.

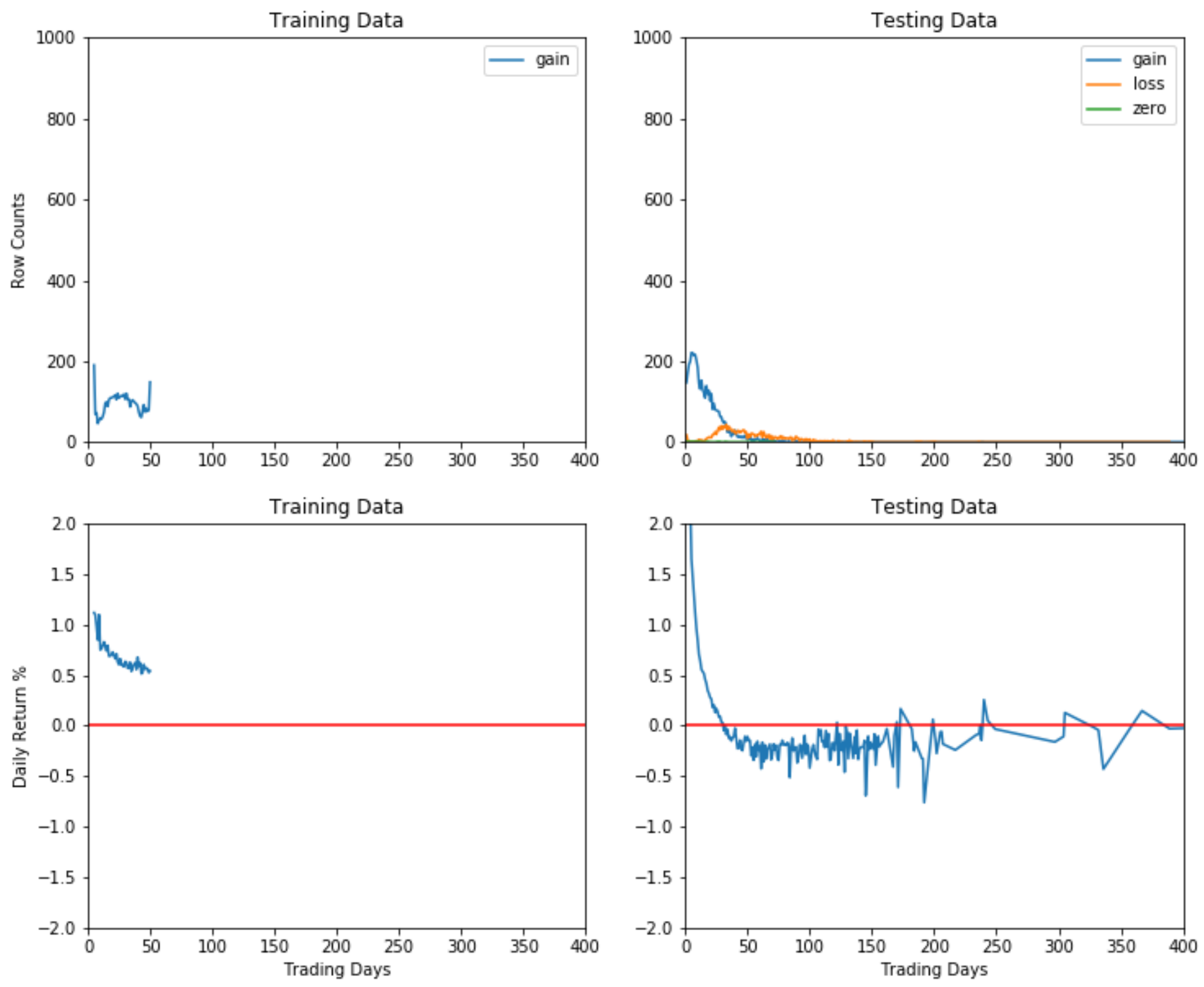

Figure 5: Performance of Final Solution

You can see in Figure 5 that the trades are cutoff beyond fifty-five trading days and positive trades are allowed. Hence, we see no loss or zero label for the top left top chart. For the testing data, we see that up to thirty-five trading days there are more positive trades. After that there roughly the same number of positive and negative trades. There is also a little hump at the beginning for positive trades. This is good as we prefer to trade between five and thirty-five trading days. The pattern of higher daily returns at the beginning is still similar to what we had for the original solution, see the bottom row of charts.

Table 6 shows the backtest performance of the final solution. The final solution's performance results range from USD 100,000 to USD 110,000. Between two and three times that of the original solution. The daily return ranges from 0.28 - 0.30 percent. This shows the positive effect of shifting longer lasting trades to shorter lasting trades. We show the growth of final capital over time for the second but last row in Figure 6. It shows a similar slow growth at the beginning, before increasing more steepily in the final year.

Update Stats	Batch Update	Keep Stats	Return column	Threshold %	Final Capital	Total Gains	Total Losses	Daily Return
Yes	5 %	0m	gain_daily_ret	0	$ 97k	$ 176k	$ 88k	0.28%
No	N/A	N/A	gain_daily_ret	0	$ 100k	$ 178k	$ 88k	0.29%
Yes	5 %	0m	gain_daily_ret	10	$ 105k	$ 188k	$ 92k	0.29%
No	N/A	N/A	gain_daily_ret	10	$ 101k	$ 178k	$ 88k	0.29%
Yes	5 %	0m	gain_daily_ret	20	$ 112k	$ 198k	$ 95k	0.30%
No	N/A	N/A	gain_daily_ret	20	$ 103k	$ 180k	$ 87k	0.29%

Table 6: Backtest Performance Results for Final Solution

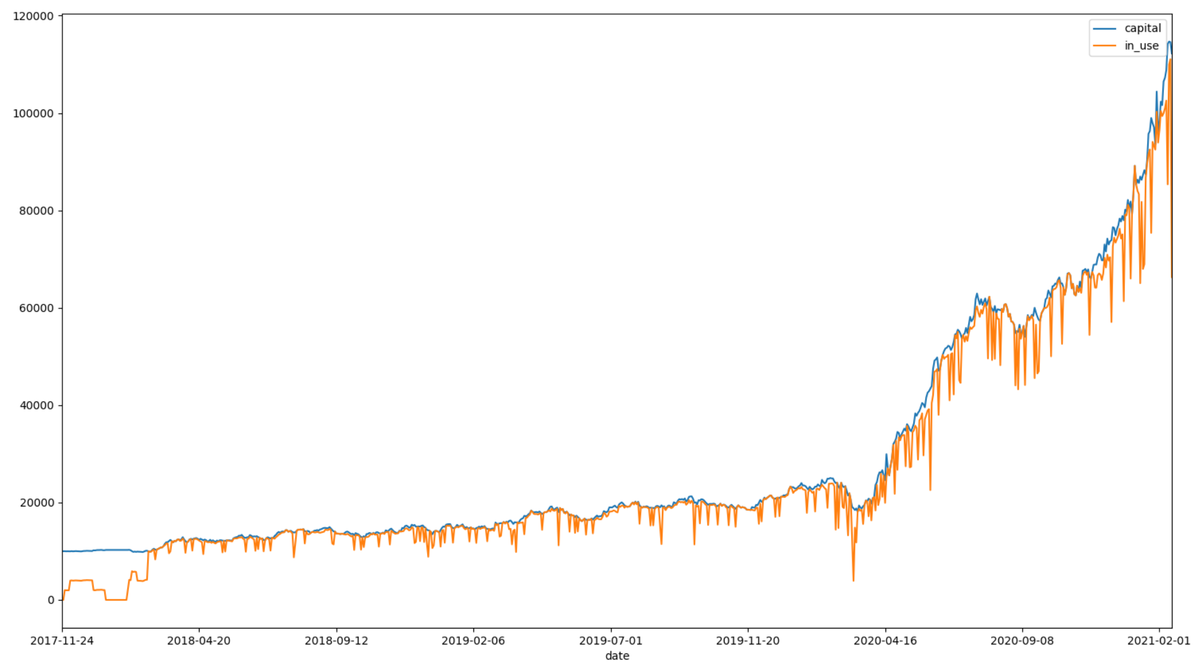

Figure 6: Final Performance of Trade.py with Backtester

Although, this seems to confirm that the final solution is better than the original solution, more testing is needed to confirm and quantify the improvement. Although we are happy to see that the results look much better, we are suspicious that the results are too good to be true, and spend a four - five days reviewing and analyzing what is happening inside the backtester. We go into more depth in the next section.

6.3 The Backtester

One source why the results of the backtester with the final solution may be too good to be true, is leaking of future information. This is a well-known source for problems with machine learning trading strategies. A lot of the leaking was removed in the original solution, but there may still be more sources of leaking. One source of leaking that the authors now realize after completing the project is the use of today's Close price in the indicators. We buy the stock on the day of identifying a local maximum. Strictly speaking, we do not know the Close price untill after we place a buy order and the trading day finishes. The question, though, is whether this is a material source of leaking, since there is a gap between the Close price and the smoothed price curve, see Figure 3 for an illustration.

We did identify and rectify one additional source of leaking. The Open, High, and Low stock prices were accidentally included as features. The features have been removed in the final solution. It reduces performance a little bit but not by much. The original solution still uses the Open, High, and Low stock prices as features.

It should be noted that the back testing returns do not include tax, and the return after tax will be lower as a result. The amount of tax, however, varies from person to person employing this trading strategy.

The backtester does not include transaction costs, and we modified it to support a transaction fee percentage per trade. Although investors are able to buy stocks without transaction fees (e.g. Robinhood), we wondered what the typical price would be for buying a stock given its Open, Close, Low and High prices. To find out we carried out a number of simulations. The typical price used in technical indicators is normally calculated as

$$\frac{Close+Low+High}{3}$$

The above expression can be modified to express the result as a factor of Close as follows

$$\frac{Close+Low+High}{3 \cdot Close}$$

We carried out simulations where we randomly picked a stock and a date and then we calculated the above ratio. We did this five hundred times, and then calculated a mean ratio for the five hundred stocks. This was stored in a list. We did this a thousand times, and then calculated basic statistics like mean, median, minimum, maximum, and standard deviation of the mean ratios. Table 7 shows the findings (round up to 7th digit behind the decimal).

Sample Statistics	Value
Mean	1.0001028
Median	1.0000905
Min	0.9970639
Max	1.0027413
Standard deviation	0.0007732

Table 7: Sample Statistics for Typical Price

The above results suggest that Close is a good estimator for the typical price a stock will be bought at. Assuming that the means are normally distributed, we can calculate an upper bound for the Close price (for 99.7% of the cases) as mean plus three standard deviation and that is 100.24 percent ($= 1.0001028 + 3(0.0007732) = 1.0024224$). This suggest setting the transaction fee at 0.25 percent for each buy and sell transaction.

We found that the outcome of the backtester is very sensitive to the stop loss percentage we use. The results go up by a significant amount as we make the stop loss smaller. Table 8 shows the outcome if we use a transaction fee of 0.5 percent:

Stop Loss Percentage	Range of Outcome
10%	$ 31k - 34k
5%	$ 54k - 58k
3%	$ 95k - 156k
2%	$ 153k - 236k
1%	$ 302k - 314k
0.5%	$ 526k - 607k

Table 8: Influence of Stop Loss Percentage on Backtester Outcome

The reduction of stop loss increases the number of trades carried out. More analysis is needed to fully understand what is causing the behavior we see.

We also implemented a maximum number of holding days for stocks to determine what the impact of that was on the performance. We found that setting it to thirty days resulted in the best performance. Setting it lower hurts performance and setting it much lower (say 5 days) deteriorates the performance. This suggest that some of the trades need more than 5 days to turn positive and cutting it off prematurely results in extra losses.

We also looked at why is the initial curve of the outcome typically flat or only slightly increasing up to March 2020. We did not find a clear answer, but we did notice that the volatility of the market increased significantly during that period as can be seen in Figure 7 that shows the VIX. A value of 30 or higher is considered high volatility and a value below 20 is considered low volatillity.

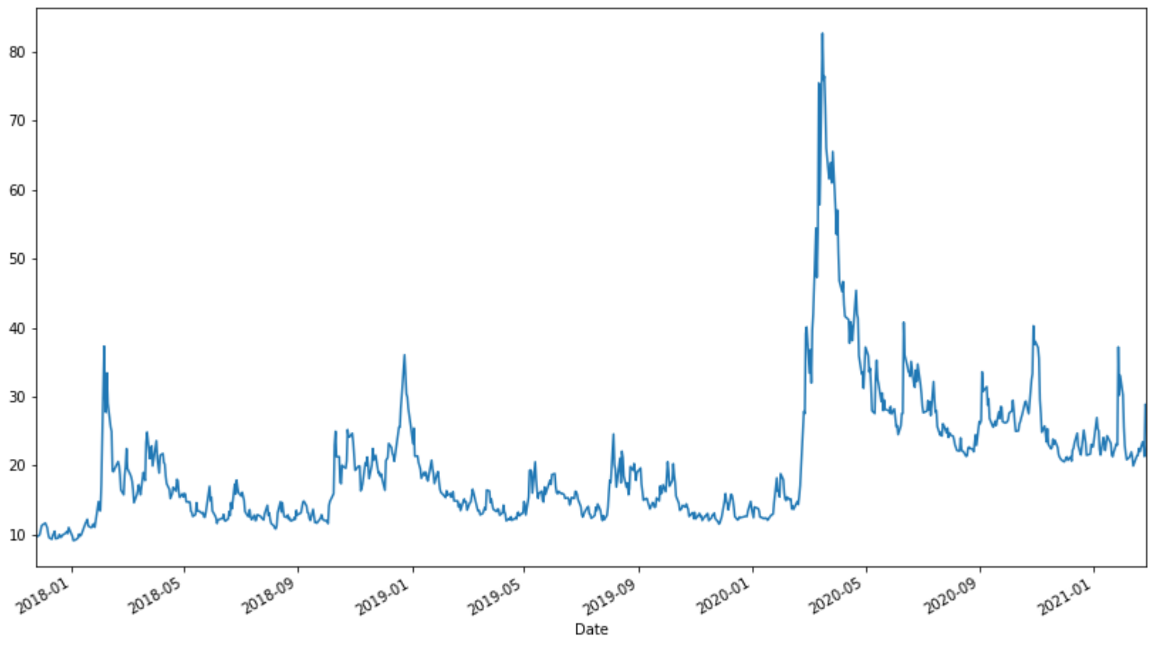

Figure 8: VIX Volatillity Index

This seems to suggest that the trading strategy works better with high volatility. It is unclear why that would be the case and that needs to be investigated in more depth to truly understand what is happening. It also may be pure coincidence.

6.4 Cross Validation and Forward Testing

We did not implement cross validation because we run out of time. However, we can to a degree achieve the same thing by increasing the transaction fee per buy and sell from say 0.25 percent to 0.5 percent. That at least dampens the expectations. Assuming we carry out 500 trades, we then need to earn at least 250 percent ($= 0.25 \cdot 500 \cdot 2$) on our initial investment before we turn a profit. In reality, we need to earn more than that, as hopefully, our investment grows over time.

The ultimate test lies in carrying out a forward test of the trading strategy we have built and confirming that it indeed earns money before we employ the trading strategy. We can then also identify any weaknesses that we may want to address before employing the trading strategy.

6.5 Alternative Models

In the original proposal, we planned to build alternative models and test them to determine the best model or sets of models. We did not have sufficient time to do that and this is what we still need to do. This was partly caused by the challenges we had implementing, integrating and testing the indicators. We also spend time on the backtester that we did not initially plan. We believe that the time we spend on the backtester improved the overall report and gives a better sense of the entire system instead of the logistic regression model in isolation.

7. Next steps

The following work will be carried out after the project, to improve the trading strategy and system, and to puyt in practice what we learned in the courses we took so far:

- Try other classification models/ensembles to see if we can improve results while keeping a simple and understandable model.
- Review and carefully analyze why the performance up to March 2020 is weak and then significantly improves. Is it truly related to volatility or is something else going on. Also, we currently work with only a subset of the 2,000 plus stocks. What happens to performance if we use full set again.
- Find out whether the standard classification models are able to determine which of the trades will turn out positive. Currently, seventy percent are positive trades in set of possible trades. Can you we increase that percentage with an extra machine learning model? We are skeptical but we want to try to make sure that this is indeed not possible.
- Try out different trading heuristics and assess impact on performance. This will then become a baseline for our RL agent to beat (see below).
- Implement forward simulation of latest stock trading model to confirm the trading performance still holds up.
- Replace trading heuristics with an RL/DL solution after taking these courses in OMSA.