

Algorithmic Trading Simulation & Comparison

Created for

FIN 6368 – Financial Information Analysis

Table of Contents

- 1. Introduction**
- 2. Data and Methodology**
- 3. Strategy**
 - a. Bollinger Bands**
 - b. Moving Average Convergence Divergence**
- 4. Why we used QuantConnect**
- 5. Algorithms**
- 6. Backtesting Results**
- 7. Limitations**
- 8. Detailed Results of Algo Backtest**
- 9. References**

1. Introduction

Algorithmic Trading, or algo trading or automated trading, is a method of choosing to buy or sell decisions and executing such trade decisions in the open market using computer programs and algorithms. The algorithms have pre-programmed trading rules or instructions set by the company operating the algorithm or by the person creating it. These algorithms make use of complex formulas, and mathematical and statistical models to make these decisions. There may be some instances where some form of human input may be required depending on the extraordinary circumstances the market may be going through, like the financial crisis for instance. But these instances are very rare in nature, and usually, it is these algorithms that decide and execute most of these trades. Traders and companies generally prefer to use such trading algorithms as these algorithms allow them to execute many trades with speed and efficiency that cannot be achieved by human interaction.

In recent years, algorithmic trading simulations have emerged as a powerful tool for these traders and investors to test and refine their trading rules in a risk-free environment. This is made possible by something called backtesting. Back testing are simulation where the trading algorithms are run through the historical price of data of multiple types of financial assets. The simulations offer an analytical view of how the algorithms perform in the general market and offer a chance to improve the algorithms further.

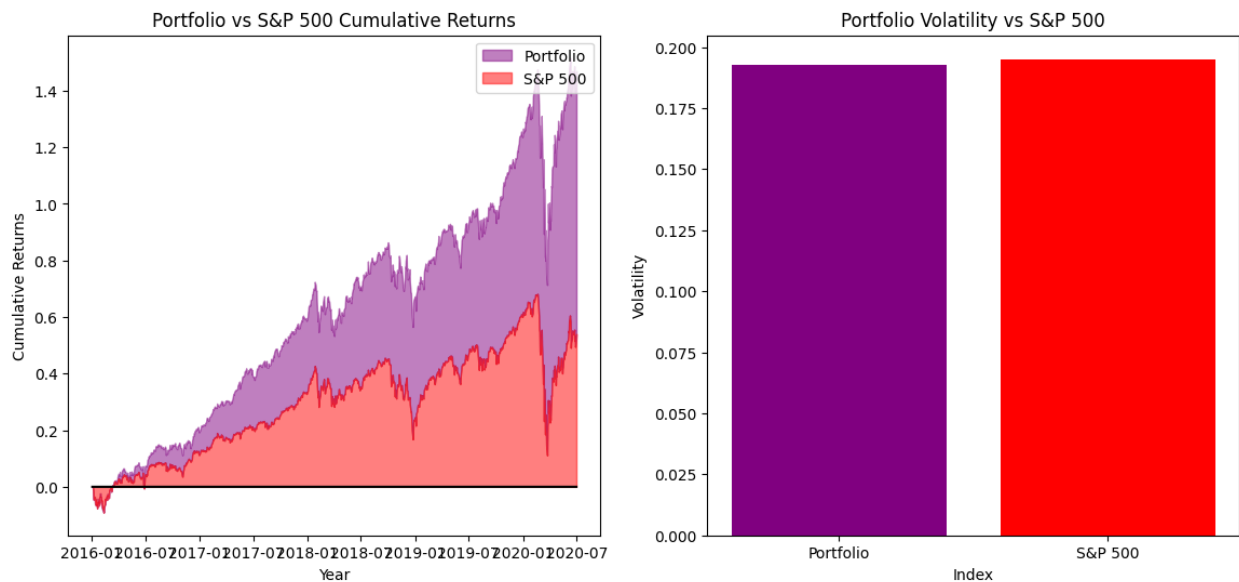
In recent years, the use of these algorithms has increased after computerized trading systems were introduced in American financial markets during the 70s. The NYSE introduced the Designated Order Turnaround (DOT) system for routing orders from traders to specialists on the exchange floor. In the following decades, exchanges enhanced their capabilities to accept electronic trading. By 2009, more than 60% of all trades in the United States were executed by some form of computer or computer system.

The advantages of deploying a trading algorithm far outweigh the disadvantages. Mainly, algorithms are deployed for their speed of executing a trade. A normal human cannot possibly execute trades in multiple classes of financial assets, like Equity, Fixed Income, FX, Cryptocurrency, etc., while also keeping an eye on current developments in the market that may affect a portfolio. In this new age of technological advancement, the high processing power of computers lets these algorithms make trade decisions very quickly. The issue of bias is also addressed by these algorithms. A rational human may not always be able to make the best decision, as emotions may play a part in some trades. That is never possible with algorithms, as the trades will be executed based on the rules and instructions set in the program. Another advantage is the ability to simulate the performance using historical prices of assets. That way, constant changes and improvements can be made for better performance and efficiency. Based on the performance, the algorithms could also be easily scaled to perform in high-volume trading, which is particularly risky and requires a high level of analysis.

The purpose of this project is to explore algorithmic trading simulations, through backtesting, and compare differing trading strategies. This report will examine the benefits and limitations of algorithmic trading simulations, as well as the factors that affect the performance of our strategies. It will also provide a comparative analysis of different types of strategies that can be deployed. Ultimately, this report aims to provide insights into the effectiveness of algorithmic trading strategies and the value of algorithmic trading simulations as a tool for traders and investors to boost their profits.

2. Data and Methodology

We chose a list of 25 stocks from the S&P500 universe. The stocks selected include Pepsi, Apple, Microsoft among others. All stocks could be considered as value stocks in their own regard, as they have consistently offered a better return than S&P500.



We chose QuantConnect as our tool to build the algorithms and for backtesting purposes. QuantConnect is an open-source, cloud-based algorithmic trading platform for equities, FX, futures, options, derivatives, and cryptocurrencies. QuantConnect serves over 100,000 quants from 170+ countries, with customers including hedge funds and brokerages, as well as individuals such as engineers, mathematicians, scientists, quants, students, traders, and programmers.

QuantConnect supports coding in Python and C# but also supports other languages through its open-source project, the Lean Algorithmic Trading Engine (LEAN). LEAN is an open-source algorithmic trading engine that allows users to do the same algorithm design, backtesting, and trading that they can do on the website. Once users code an algorithm, they can run a backtest on historical data, which provides a full breakdown of how it could have performed in the market in the past.

Since we used the QuantConnect technology from their website, there was no need to specifically download data for our stocks, since QC has specific packages for that purpose, which feed pre-processed data for the algorithm. However, for EDA purposes, QC could not be used. So, we relied on local Python applications like Anaconda Navigator and VS Code for EDA.

For our trading algorithms, we chose to assign one trading strategy for one algorithm each. The idea was to use the indicators to make the trading decisions for that specific algorithm. We created two algorithms using this idea. For the third version, we chose to combine the two trading strategies into the same algorithm. In this algorithm, for there to be a buy or sell decision, both indicators should be favorable for that strategy to work.

3. Strategies

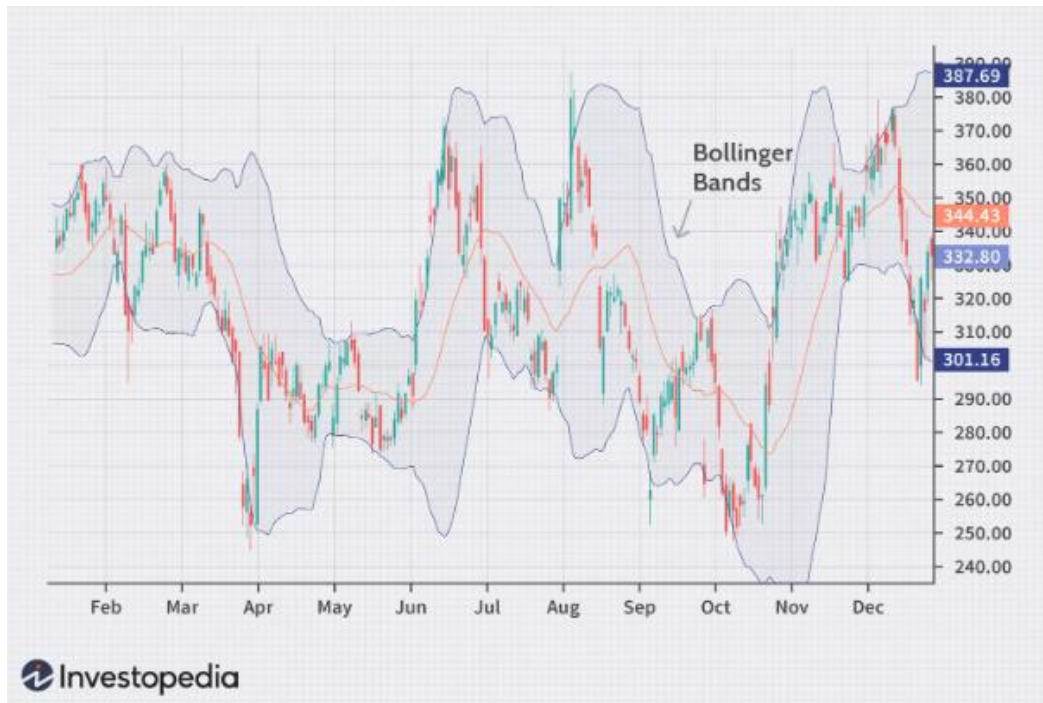
As mentioned above, we chose two trading strategies for our algorithms. These strategies are widely used within the industry by technical analysis traders to base their decisions. We explain each strategy in detail below –

a. Bollinger Bands Strategy (BB)

Bollinger bands strategy, specifically Bollinger Bands, is a type of statistical chart characterizing the prices and volatility over time of a financial instrument or commodity, using a formulaic method. The whole method, or strategy, was introduced by John Bollinger, a technical analyst in the 80s. Quants use these charts as a tool to make trading decisions or as a component of a much larger analysis. The chart is plotted as two standard deviations, both positively and negatively, away from a simple moving average of the asset price. However, this can be adjusted according to the preference of the company.

The three lines on the chart are the upper band, middle band, and lower band. The middle band is the simple moving average. Usually, this is a 20-day moving average, but as mentioned before, this could be changed. The upper band and lower bands are two standard deviations away from the middle band. This is both positive and negative.

These bands tell a quant whether the asset is overbought or oversold in the market. Many traders believe the closer the prices move to the upper band, the more overbought the market, and the closer the prices to the lower band, the more oversold the market. John Bollinger additionally has a set of 22 rules when using these bands in a trading system.



The signal to whether to buy or sell an asset can be interpreted using the price of the asset and the current SMA. When the current price converges to the SMA and crosses and passes above the SMA, this may mean an upward trend for the asset. At this point, when the price crosses above the SMA, the upper band price is the target price of the asset. The trader then buys the asset when the price exceeds the SMA. Conversely, when the price crosses below the SMA, this may mean a downward trend, a signal to sell.

b. Moving Average Convergence Divergence (MACD)

MACD, short for Moving Average Convergence Divergence, is a trading indicator used in the technical analysis of securities prices. It was created by Gerald Appel in the late 70s. It is designed to reveal changes in the strength, direction, momentum, and duration of a trend in a stock's price. Hence, we can say that

MACD is a trend-following momentum indicator. It shows the relationship between two Exponential Moving Averages (EMAs) of a security's price. The MACD line is calculated by subtracting the 26-day EMA from the 12-day EMA. A 9-day EMA of this MACD line is called the signal line. This line is plotted on top of the MACD line, which can function as a trigger for buy and sell signals.

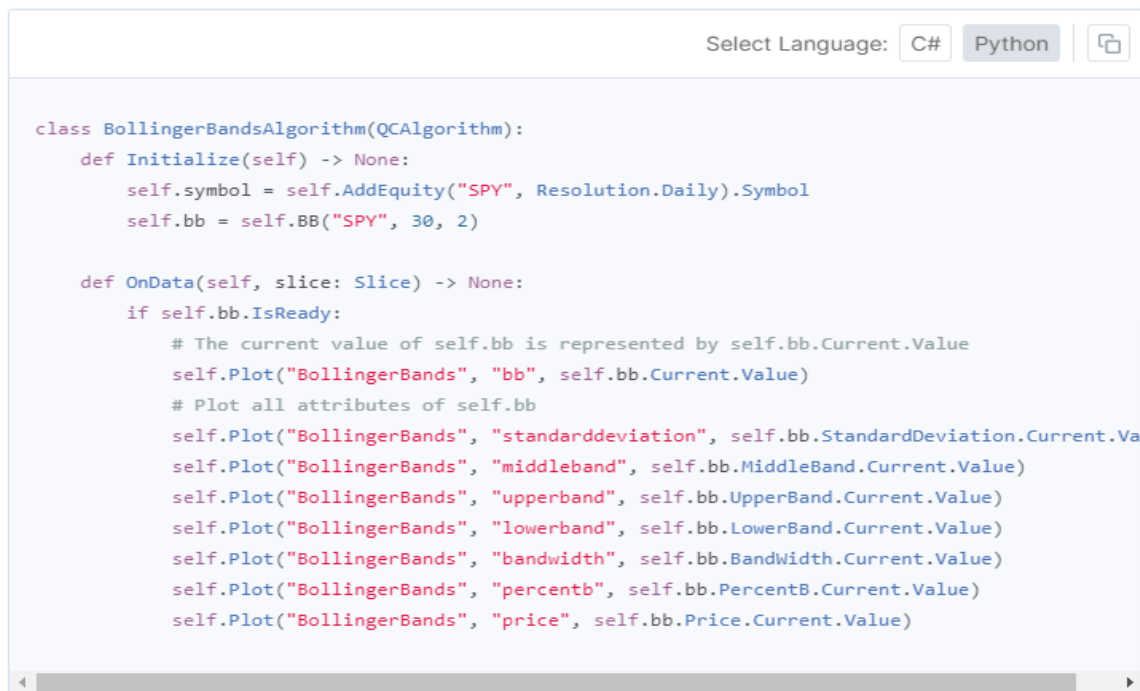


Quants and traders interpret these lines in the following way – when the MACD line crosses above the signal line, this is a signal to buy the asset. And conversely, the MACD line crossing below the signal is interpreted as a signal to sell. MACD indicators can be interpreted in many ways, the most common methods are crossovers, divergences, and rapid rise/falls.

We decided to use these strategies each on their own as a standalone strategy for an algorithm. That way, we can determine which strategy has a better methodology for more returns. The third algorithm then combines both these strategies.

4. Why did we use QuantConnect?

QuantConnect has a wide variety of resources, that can be run through Python, which supports data download, pre-processing, machine learning, and even indicators calculations. In local Python development, we would have to calculate the indicator values by ourselves. But QC supports the calculation of these indicator values with just a few lines of code.

A screenshot of a code editor interface. At the top right, there is a 'Select Language:' dropdown menu with 'C#' and 'Python' options, and a copy icon. The code is written in Python and defines a class 'BollingerBandsAlgorithm' that inherits from 'QCAAlgorithm'. The class has two methods: 'Initialize' and 'OnData'. 'Initialize' sets the symbol to 'SPY' and calculates the Bollinger Bands. 'OnData' plots various attributes of the Bollinger Bands, including the current value, standard deviation, middle band, upper band, lower band, bandwidth, percent B, and price.

```
class BollingerBandsAlgorithm(QCAAlgorithm):
    def Initialize(self) -> None:
        self.symbol = self.AddEquity("SPY", Resolution.Daily).Symbol
        self.bb = self.BB("SPY", 30, 2)

    def OnData(self, slice: Slice) -> None:
        if self.bb.IsReady:
            # The current value of self.bb is represented by self.bb.Current.Value
            self.Plot("BollingerBands", "bb", self.bb.Current.Value)
            # Plot all attributes of self.bb
            self.Plot("BollingerBands", "standarddeviation", self.bb.StandardDeviation.Current.Value)
            self.Plot("BollingerBands", "middleband", self.bb.MiddleBand.Current.Value)
            self.Plot("BollingerBands", "upperband", self.bb.UpperBand.Current.Value)
            self.Plot("BollingerBands", "lowerband", self.bb.LowerBand.Current.Value)
            self.Plot("BollingerBands", "bandwidth", self.bb.BandWidth.Current.Value)
            self.Plot("BollingerBands", "percentb", self.bb.PercentB.Current.Value)
            self.Plot("BollingerBands", "price", self.bb.Price.Current.Value)
```

5. Algorithms

For the first algorithm, we used the Bollinger bands to determine the signals to buy and sell an asset. As mentioned before, the algorithm will decide between 25 stocks to make the buy and sell decisions. First, we set the start and end date for our backtest data period. For our algorithm, we decided to start our period from January 2016 to July 2020. We then set the simulated virtual cash at the start of the period. The algorithm will backtest using this virtual cash.

We calculate the upper, middle, and lower band for all stocks using the self.BB function within QC.

The algorithm automatically calculates the band values for all stocks and assigns them to a dictionary. These values keep changing every day, as the 20-day SMA is used to determine the upper and lower bands. The buy and sell signals are indicated using the middle band and the close price of the stock. If, on any given day, the market price of the stock crosses above the 20-day SMA, the algorithm interprets this as a signal to buy the stock, as the indicators depict a possible upward trend in the price. The algorithm will first check if there is an existing open position in the stock. If the algorithm finds an open position, it will not buy any more than the current holdings. If it detects no holdings in the stock, it will buy the stock and invest in a way that the portfolio is equally weighted among all 25 stocks. Similarly, when the close price for any day crosses below the 20-day SMA, the algorithm interprets this as a signal to sell the holdings in the stock, assuming there is an open position in the stock.

```
7
8 class PensiveRedOrangeGoshawk(QCAlgorithm):
9
10     def Initialize(self):
11         self.SetStartDate(2016, 1, 1) # Set the start date
12         self.SetEndDate(2020, 7, 1) # Set the end date
13         self.SetCash(100000) # Set the starting cash balance
14         self.SetWarmUp(30)
15
16         # Adding stocks to our universe
17         self.stocks = ["AAPL", "MSFT", "AMZN", "NVDA", "GOOGL", "BRK.B", "GOOG", "META", "UNH", "XOM", "JNJ", "TSLA", "JPM", "PG", "V", "LLY", "MA", "HD", "MRK", "CVX", "ABBV", "PEP", "AVGO", "KO", "COST"]
18         for stock in self.stocks:
19             self.AddEquity(stock, Resolution.Daily)
20
21         # Defining the Bollinger Bands for each stock in our universe using self.BB and assigning values to a dictionary
22         self.bands = {}
23         for stock in self.stocks:
24             bb = self.BB(stock, 20, 2, MovingAverageType.Simple, Resolution.Daily)
25             self.bands[stock] = bb
26
27     def OnData(self, data):
28
29         # Checking if the indicators are ready for all stocks. If some data points are missing, the code will not do anything and return
30         for stock in self.stocks:
31             if not self.bands[stock].IsReady:
32                 return
33
34         # Looping through all the stocks that are in our universe
35         for stock in self.stocks:
36             # Checking if we have any open positions
37             holdings = self.Portfolio[stock].Quantity
38
39             # Checking if the symbol is present in the data object in case some data points are missing. Same approach as we took for indicators
40             if stock not in data:
41                 return
42
43             # Checking if the price is above the upper band. If it is above the upper band, that is a buy signal for us, assuming that we do not have any current holdings
44             if data[stock].Close > self.bands[stock].MiddleBand.Current.Value:
45                 # If we don't have any open positions, enter a long position, weighting the portfolio equally
46                 if holdings <= 0:
47                     self.SetHoldings(stock, 1.0/len(self.stocks))
48                     self.Debug("BUY {0} >> {1}".format(stock, self.Time))
49             # Checking if the price is below the lower band. If it is below the lower band, that signifies a downward trend, meaning a sell signal.
50             elif data[stock].Close < self.bands[stock].MiddleBand.Current.Value:
51                 # If we have an open long position, close it
52                 if holdings > 0:
53                     self.Liquidate(stock)
54                     self.Debug("SELL {0} >> {1}".format(stock, self.Time))
55
```

Figure 1 Algorithm #1 Syntax

Like Algorithm #1, the rules are set in #2 using the MACD indicators. Again, QC has built in functions to calculate the signal line of MACD. Using `self.MACD`, we calculate the 12-day EMA, 26-day EMA and 9-day EMA to calculate the signal line used in MACD. The algorithm then interprets the signal line and the current MACD line to come up with buy and sell decisions. On any given day, if the MACD line crosses above the signal line, the algorithm interprets this as a signal to buy the stock, again, assuming that there is no open position in the stock. Like Algo #1, Algo #2 also equally weighs the portfolio among all 25 stocks. Conversely, when the MACD line crosses below the signal line, the algorithm interprets this as a sell sign, and when there is an open position, the algo will liquidate the position.

The third version of our algorithm combines the indicators of both Bollinger bands and MACD to execute trades. For the algorithm to determine a buy signal, the close value needs to be above the middle band, and the MACD needs to be bullish. Also, for the sell signal, the close value needed to be below the middle band, and the MACD to be bearish.

6. Back testing Results



	#1 - BB	#2 - MACD	#3 - BBMACD
Total Trades	3235	2284	1464
Net Profit %	63.793%	63.17%	68.959%
Sharpe Ratio	1.025	1.027	1.114
Treynor Ratio	0.271	0.266	0.291
Win-Loss	37-63	45-55	51-49
Average Win	0.21%	0.23%	0.26%
Average Loss	-0.08%	-0.11%	-0.15%

Figure 2 Backtesting Results

We ran all algorithms through the backtesting feature of QuantConnect and got the following results. Algo #1 and #2 yielded almost identical returns and performance metrics, with only a marginal increase/decrease in some. While the BB strategy had a better return on paper, the MACD strategy had a higher Sharpe Ratio and a better Win-Loss Rate. A trade done by the MACD strategy has a better chance of being a profitable one, compared to a trade executed by BB. So, it can be concluded that deploying either of them in place of the other would have an incremental effect on the portfolio performance of a trader.

Algo #3, however, had significant incremental numbers on almost all metrics. Although it executed the least trades at 1464, which is understandable since there are two conditions to be met to make a buy/sell decision, it had a much better Net Profit, Sharpe Ratio, and, more importantly, a better Win-Loss Rate. On average, a trade executed by this algorithm had a 51% chance of being profitable. One metric that was significantly different from the first two algorithms was that the average loss was higher, at -0.15%.

In a way, we cannot say for sure which strategy is a more reliable one as a standalone strategy for an algorithm. Although BB's strategy has a better net profit on paper, it had a much worse win-loss rate, at 37-63 compared to 45-55. There is no significant difference in the average wins and losses between the two, as differences in wins are mitigated by differences in losses.

Combining the two strategies, however, gave us a much better win-loss rate, and higher Sharpe, Treynor, and net profit numbers. So, it seems that combining the two strategies is a much better option. However, there are some limitations in the strategy itself.

7. Limitations

According to the general market consensus between quants, traders, and investors – people who use these strategies daily, it is believed that Bollinger bands should not be used as a standalone tool for making decisions. This couldn't be truer, as John Bollinger himself has a set of 22 rules that are to be followed when using the bands' methodology. Instead, the bands should be used as an accompanying rule set along with other non-correlated indicators. One such indicator, according to the market, is the RSI – Relative Strength Index. Another limitation of this strategy is the fact that since we use a simple moving average, older price data is given the same weightage as newer data. This may result in some form of bias and might not be the best indicator for assessing trends.

As for MACD, it is said that the MACD indicators sometimes signal a reversal, but no such reversal occurs. In short, a false positive. But, since the algorithm is based on the signal value rule, it may execute a trade according to the signal. If no such reversal occurs, this could result in significant losses to a portfolio. Also, divergence doesn't forecast all reversals.

These limitations are seen clearly in the results of our backtests, as the standalone algorithms of BB and MACD show much lower win-loss rates as compared to the combined BBMACD algorithm.

But, in general, there were other limitations to this project. Some of them are as follows –

- a. QuantConnect's steep learning curve – Although QC has a wide range of built-in packages and tools for our convenience, it takes a lot of time to get used to them, as they are not quite as same as the general Python syntax that we use. So, getting used to it takes a lot of time, even though they provide documentation for every single function.
- b. The limitations of the standalone strategies may be why the combined strategy did not work as well as expected. Currently, both strategies assess trends. As mentioned before, we should have used another strategy in combination with one of these, as both have the same performance when run standalone.
- c. QuantConnect would not backtest post August 31st, 2020. We wanted to incorporate the data during post first COVID wave to see how the algorithms perform under periods of high volatility. But for some unexplained reason, a runtime error always occurred on 31st August. The backtest would run till that date but stop abruptly. We suspect that there was possibly a stock split or delisting of one of our stocks on that day, which may have led to this issue.

These limitations may have resulted in some numbers that could have been better under better circumstances.

8. Detailed Results of Algorithms

Overview	Report	Orders	Insights	Logs	Code	Share
Download Results						
PSR		48.601%			Sharpe Ratio	1.025
Total Trades		3235			Average Win	0.21%
Average Loss		-0.08%			Compounding Annual Return	11.578%
Drawdown		12.500%			Expectancy	0.306
Net Profit		63.793%			Loss Rate	63%
Win Rate		37%			Profit-Loss Ratio	2.57
Alpha		0.053			Beta	0.303
Annual Standard Deviation		0.08			Annual Variance	0.006
Information Ratio		-0.101			Tracking Error	0.128
Treynor Ratio		0.271			Total Fees	\$3283.44
Estimated Strategy Capacity		\$27000000.00			Lowest Capacity Asset	CHV R735QTJ8XC9X
Portfolio Turnover		7.78%				

Figure 3 Algorithm #1 Performance

Overview	Report	Orders	Insights	Logs	Code	Share
Download Results						
PSR		48.725%			Sharpe Ratio	1.027
Total Trades		2284			Average Win	0.23%
Average Loss		-0.11%			Compounding Annual Return	11.484%
Drawdown		12.100%			Expectancy	0.393
Net Profit		63.169%			Loss Rate	55%
Win Rate		45%			Profit-Loss Ratio	2.07
Alpha		0.052			Beta	0.305
Annual Standard Deviation		0.079			Annual Variance	0.006
Information Ratio		-0.108			Tracking Error	0.127
Treynor Ratio		0.266			Total Fees	\$2323.40
Estimated Strategy Capacity		\$20000000.00			Lowest Capacity Asset	CHV R735QTJ8XC9X
Portfolio Turnover		5.50%				

Figure 4 Algorithm #2 Performance

Overview	Report	Orders	Insights	Logs	Code	Share
Download Results						
PSR		56.661%			Sharpe Ratio	1.114
Total Trades		1464			Average Win	0.26%
Average Loss		-0.15%			Compounding Annual Return	12.350%
Drawdown		11.900%			Expectancy	0.409
Net Profit		68.959%			Loss Rate	49%
Win Rate		51%			Profit-Loss Ratio	1.78
Alpha		0.059			Beta	0.299
Annual Standard Deviation		0.078			Annual Variance	0.006
Information Ratio		-0.062			Tracking Error	0.127
Treynor Ratio		0.291			Total Fees	\$1488.92
Estimated Strategy Capacity		\$110000000.00			Lowest Capacity Asset	CHV R735QTJ8XC9X
Portfolio Turnover		3.53%				

Figure 5 Algorithm #3 Performance

9. References

<https://www.investopedia.com/articles/technical/102201.asp>

<https://www.investopedia.com/terms/m/macd.asp>

<https://www.quantconnect.com/docs/v2>

<https://www.youtube.com/watch?v=yuZBBX47xK0>

<https://www.youtube.com/watch?v=joXDV5eqOoY>

<https://medium.com/codex/algorithmic-trading-with-bollinger-bands-in-python-1b0a00c9ef99>

https://www.lean.io/docs/v2/lean-engine/class-reference/classQuantConnect_1_1Algorithm_1_1QCAAlgorithm.html

[Investopedia.com](https://www.investopedia.com)

[Wikipedia.com](https://www.wikipedia.com)

[Chat GPT for error debugging](#)