Trading Strategy Back-Tester Creation

A Tutorial on Creating a Custom
Trading Strategy Back-Testing Algorithm

Ivan Silajev

Warwick Coding Society
ivan.silajev@warwick.ac.uk

Contents

1 Introduction

This **Warwick Coding Society** report outlines the theory and the creation of a trading strategy back-testing algorithm using the Python programming language. This document's approach to presenting the theory involves concisely explaining each topic before introducing the mathematics and Python code. We designed the code shown throughout the report to be flexible, so anyone who follows it to create their trading strategy back-tester would have an easier time changing it to their liking.

The topics covered in this report include:

- What is **trading** and **arbitrage**?
- What is a **trading strategy** and how use it?
- What is **portfolio optimisation** and how it's done?

The motivation for creating this report is to introduce programmers of any skill level to algorithmic trading and back-testing in a clear, concise manner, in line with Warwick Coding Society's goal of **making programming accessible**.

2 Trading Basics

2.1 What is Trading?

Trading is the exchange of value between two entities. A **market** is a field where entities trade. The most active markets today comprise financial asset markets, including the **New York Stock Exchange (NYSE)** and the **National Association of Securities Dealers Automated Quotations (NASDAQ)**.

Financial asset markets are ever-trending due to their increasing accessibility to the general public and the simplicity of trading on them. For this reason, this report will be primarily focusing on financial asset markets, though the methodology presented here applies to any market.

With the progress of technology and mathematics in the late 20th century, companies like **Renaissance Technologies** investigated algorithmic trading strategies to automate and optimise their trading activity in financial asset markets. Currently, anyone can access financial markets through the many internet brokers that exist, allowing anyone to use algorithmic trading methods to automate their trading activity.

There are two common ways to profit from trading in financial asset markets.

- In a **buy/long position**, one buys an asset on the market at an agreed price and time. Then, they sell the asset on the market when it increases in price, thus profiting from the increase in the price.
- In a **sell/short position**, one borrows assets from the broker and sells them on the market at an agreed price and time. Then, they buy back the asset when it decreases in price and returns it to the broker, thus profiting from the decrease in the price.

For more details on how brokers, financial markets, and assets work, the reader can resort to the resources mentioned in section 7 of this report.

2.2 What is Arbitrage?

Arbitrage involves exploiting inefficiencies across markets to trade more profitably. An **arbitrage opportunity** is the potential of investing a net-zero worth of capital into assets and profiting from them without risk. Simply put, an arbitrage opportunity can give something from nothing with no risk of loss.

One can find arbitrage opportunities using data that accurately reflects the current demand for different assets, allowing them to predict which assets will be profitable. Again, arbitrage is a general concept applicable to any set of markets.

Standard arbitrage opportunity detection methods include multivariate trend analysis, internet social trend analysis and portfolio optimisation methods, like the one explained in section 4.4 of this report.

3 Trading Strategy Back-Testing Theory

A **trading strategy** is a rule for when and how to trade an asset over time. **Back-testing** is the testing of a trading strategy's profitability with historical price data. We start by outlining what we expect from a trading strategy.

3.1 The Characteristics of a Good Trading Strategy

The aim of trade is to **profit**. One can trade in the financial market by opening long or short positions on financial assets.

Long positions are profitable when assets are bought at low prices and sold at higher prices. **Short positions** are profitable when assets are sold at high prices and bought at lower prices.

Therefore, a good trading strategy:

- Opens and holds long positions while the asset price trends upwards
- Opens and holds short positions while the asset price trends downwards
- Closes long positions when the asset price starts trending downwards
- Closes short positions when the asset price starts trending upwards

3.1.1 Mathematical Preliminaries

Define the set of times for which the price data of a given asset is observed as the countable set $T \subset \mathbb{R}$.

Brokers and financial websites provide data on the historical prices of assets, including the close, open, low and high prices for different time intervals. Define the close, open, low and high prices as follows:

- Close price series: $P_c : T \mapsto \mathbb{R}^+$
- Open price series: $P_o : T \mapsto \mathbb{R}^+$
- Low price series: $P_l : T \mapsto \mathbb{R}^+$
- High price series: $P_h : T \mapsto \mathbb{R}^+$

These series can be used for the construction of **indicator series**. See section 6 of this report for an example.

3.2 Indicator Series

A trading strategy involves using information about the asset price trend. This information is usually in the form of a time series, known as an **indicator series**, showing when the asset price is possibly increasing or decreasing. One can always transform price trend information not in the form of an indicator series to be one.

An indicator series is:

- Positive when the price trends upwards
- Negative when the price trends downwards
- Zero when the price trend is stable

3.2.1 Mathematical Description

Define an indicator time series $I : T \mapsto \mathbb{R}$ as a function of time.

An example of a trivial indicator series is the difference between the close and open prices of the asset traded:

$$I(t) = P_c(t) - P_o(t)$$

Another example of a trivial indicator series involves using the first order difference of the closed prices:

$$I(t) = P_c(t) - P_c(t - \Delta_t)$$

Where Δ_t is the difference between time t and the latest time before t in set T, or, more precisely:

$$\Delta_t = t - \sup\left(T \cap (\infty, t)\right)$$

Deriving reliable indicators is a separate science in which quantitative traders specialise.

3.3 Trade Signal Series

A **trade signal series** informs when a given trade type should be open according a given indicator series.

The long trade signal series is:

- Equal to one when the indicator is strictly positive
- Equal to zero otherwise

The short trade signal series is:

- Equal to one when the indicator is strictly negative
- Equal to zero otherwise

3.3.1 Mathematical Description

The **long trade signal series** $V_l : T \mapsto \{0, 1\}$ is defined as:

$$V_l(t) = \text{sign}(\max\{I(t), 0\})$$

The **short trade signal series** $V_s(t)$ is trivially derived by changing the sign of the indicator.

$$V_s(t) = \text{sign}(\max\{-I(t), 0\})$$

3.4 Action Series

An **action series** informs exactly when to open or close trades of a given type based on a given trade signal series.

The action series is:

- Equal to one when the trade must be opened
- Equal to zero when the position must be held (inaction)
- Equal to minus one when the trade must be closed

3.4.1 Mathematical Description

The **action series** $A : T \mapsto \{-1, 0, 1\}$, for a given trade signal series $V(t)$, is defined as:

$$A(t) = \begin{cases} V(t) & \text{if } t = \inf T \\ V(t) - V(t - \Delta_t) & \text{if } t > \inf T \end{cases}$$

3.5 Alpha Series

Finally, the **alpha series** shows the ratio of the value of the investment with its previous value at a given time under the use of a trading strategy.

When the trade is inactive, the alpha series will take the value one, since no capital is invested in the asset. When the trade is active, the alpha series will output the ratio change in the investment according to the close price series.

The alpha series of a given asset and trading strategy is what's used to evaluate the performance of the trading strategy for the asset.

3.5.1 Mathematical Description

The alpha series $\alpha : T \mapsto \mathbb{R}^+$, for a given trade signal series $V(t)$ and action series $A(t)$, is generated using the following algorithm:

Require: $T \wedge V(t) \wedge A(t) \wedge P_c(t) \wedge r$
Ensure: $\alpha(t)$
for $t = \inf T$ **to** $\sup T$ **do**
  if $A(t) = 1$ **then**
    $\alpha(t) \leftarrow 1$
    $P_0 \leftarrow P_c(t)$
  else if $A(t) = 0$ **then**
    if $V(t) = 0$ **then**
      $\alpha(t) \leftarrow 1$
    else if $V(t) = 1$ **then**
      $\alpha(t) \leftarrow (P_0 + r \cdot (P_c(t) - P_0))/(P_0 + r \cdot (P_c(t - \Delta_t) - P_0))$
    end if
  else if $A(t) = -1$ **then**
    $\alpha(t) \leftarrow (P_0 + r \cdot (P_c(t) - P_0))/(P_0 + r \cdot (P_c(t - \Delta_t) - P_0))$
  end if
end for

The capital ratio r is the proportion of capital dedicated to the trade. A positive r generates an alpha series for a long position strategy. A negative r generates an alpha series for a short position strategy.

3.6 Back-Test of an Alpha Series

Using the alpha series generating algorithm, one can generate the alpha series for a trading strategy consisting of:

- Long positions only, giving the **long alpha series** $\alpha_l(t)$
- Short positions only, giving the **short alpha series** $\alpha_s(t)$
- Both long and short positions, giving the **full alpha series** $\alpha_f(t) = \alpha_l(t) \cdot \alpha_s(t)$

The product of the terms in an alpha series is the ratio by which one's investment would change when following the underlying strategy. Section 6.2 showcases how to use alpha series to compare trading profits with the asset price and determine the underlying strategy's performance over time.

4 Portfolio Theory

4.1 Why are Portfolios Important?

A trading portfolio is an allocation of capital dedicated to a set of assets for trade. One's choice of portfolio could significantly determine the efficiency of their trading activity. Dedicating more capital into assets with volatile prices may be irrational if more stable assets with higher average returns exist.

The volatility of an asset price is its degree of variation. An asset price that deviates more from its trend than another is more volatile than the other. Usually, investors optimise their portfolios according to asset prices' volatility and average returns.

4.2 Portfolio Distribution Series

Since a portfolio is an allocation of capital, it is expressed as a vector function called a **portfolio distribution series** $\pi : T \mapsto [0,1]^n$ with entries summing to one. The ith entry of $\pi(t)$, is the proportion $\pi_i(t)$ of capital dedicated towards asset $i \in I$ for trade at time $t \in T$, where I is the index set for the assets.

Setting $\alpha_i(t)$ to be the alpha series of the ith asset, define the portfolio alpha series as follows:

$$\alpha_\pi(t) = \textstyle\sum_{i \in I} \alpha_i(t)\pi_i(t)$$

The **portfolio alpha series** is shows the ratio of the value of the total invested capital with its previous value at a given time.

4.3 The Equal-Weight Portfolio

The equal-weight/trivial portfolio uniformly allocates an investor's capital across all their traded assets. Letting $V_i(t)$ be the trade signal series of asset $i \in I$, the trivial portfolio is such that:

$$\pi_i(t) = \begin{cases} \dfrac{V_i(t)}{\sum_{i \in I} V_i(t)} & \text{if } \sum_{i \in I} V_i(t) \neq 0 \\ 0 & \text{if } \sum_{i \in I} V_i(t) = 0 \end{cases}$$

4.4 Markowitz Modern Portfolio Theory

Markowitz MPT (Modern Portfolio Theory) is one of the most successful and useful portfolio optimisation methods created and used to date. MPT optimises a portfolio by maximising its average returns and minimising its variance, used as a measure of volatility in this case. It utilises information on the returns of individual assets, including their average returns and returns covariances.

For convenience, assume all objects defined from hereon are functions of $t \in T$.

Let μ be the **vector of average returns** of all assets in I. So, μ_i is the average returns of asset i.

Let Σ be the **matrix of covariances between returns** of all assets in I. So, Σ_{ij} is the covariance between the returns of asset i and j.

Let d be the **wealth distribution**, showing the proportion of wealth to be invested in each asset. The wealth distribution is different to the portfolio distribution π, since the entries of d can be any real values all summing to one. d_i is the wealth distributed to asset i.

The entries of the portfolio distribution are derived from the wealth distribution as follows:

$$\pi_i = \frac{|d_i|}{\sum_{i \in I} |d_i|}$$

There exist two special wealth distributions.

- The **minimal variance risky wealth distribution**:

$$d^M = \frac{\Sigma^{-1} \cdot \mathbf{1}}{\mathbf{1}^T \cdot \Sigma^{-1} \cdot \mathbf{1}}$$

  With its corresponding average return $d^M \cdot \mu = \mu^M$.

- The **zero rate tangency wealth distribution**:

$$d^Z = \frac{\Sigma^{-1} \cdot \mu}{\mathbf{1}^T \cdot \Sigma^{-1} \cdot \mu}$$

  With its corresponding average return $d^Z \cdot \mu = \mu^Z$.

The **optimal wealth distribution** is derived by solving the following optimisation problem for some desired average return μ^0:

$$\begin{array}{ll} \text{minimise} & d^T \cdot \Sigma \cdot d, \quad d \in \mathbb{R}^n \\ \text{subject to} & d \cdot \mathbf{1} = 1, \\ & d \cdot \mu \geq \mu^0 \end{array}$$

The problem involves deriving the wealth distribution that minimises the variance of the total returns with a set of necessary constraints. The unique solution to the problem is given by:

$$d^0 = \left(\frac{\mu^Z - \mu^0}{\mu^Z - \mu^M} \right) d^M + \left(\frac{\mu^0 - \mu^M}{\mu^Z - \mu^M} \right) d^Z$$

Therefore, it is best to use the distribution d^0 for a desired average returns of μ^0.

5 Back-Testing Code Implementation

Before creating the necessary functions and code for back-testing, we load the following Python packages:

```
# 'numpy' used for manipulating arrays of data
import numpy as module_np

# 'pandas' used for manipulating tabular data sets
import pandas as module_pd
```

```
# 'datetime' used for working with 'datetime' data types
import datetime as module_dt

# 'matplotlib.pyplot' used for creating plots of data
import matplotlib.pyplot as module_plt

# 'yfinance' to extract asset price data from Yahoo Finance
import yfinance as module_yf
```

5.1 Trade Signal Series Code

Below is the Python code for creating a trade signal series from a given indicator series in the form of a list.

```
# Define the function 'function_tsig'
# which transforms indicator series
# into trade signal series
def function_tsig(
    list_indicator
):

    # Transform the entries of the
    # indicator series to create the
    # trade signal series 'list_sign'
    list_sign = [

        # Pass through 'sign' function
        module_np.sign(

            # Pass through 'maximum' function
            module_np.maximum(
                float_value,
                0
            )
        )

        # Iterate for all values in 'list_indicator'
        for float_value
        in list_indicator
    ]

    # Output 'list_sign'
    return list_sign
```

5.2 Action Series Code

Below is the Python code for creating an action series from a given trade signal series in the form of a list.

```
# Define the function 'function_action'
# which transforms a trade signal series
# into an action series
def function_action(
    list_sign
):
```

```
    # Shift 'list_sign' by one step
    # and fill the first value with zero
    list_shift = [0] + list_sign[:-1]

    # Subtract entries 'list_shift'
    # from 'list_sign' element-wise
    list_output = [

        list_sign[int_i] - list_shift[int_i]

        for int_i
        in range(
            0,
            len(list_sign)
        )
    ]

    # Return final action series
    # as a list
    return list_output
```

5.3 Alpha Series Code

Below is the code for generating the alpha series for either a long or short position. Long positions are tested with positive `float_ratio`. Short positions are tested with negative `float_ratio`.

```
# Define the function 'function_backtester'
# that generates the alpha series for a given
# price series, indicator series and
# capital ratio
def function_backtester(
    list_prices,
    list_indicator,
    float_ratio
):

    # Derive the trade signal series
    # from the indicator series in the form
    # of 'list_indicator'
    list_tsig = function_tsig(
        list_indicator = list_indicator
    )

    # Derive the action series from
    # the previously derived trade
    # signal series
    list_action = function_action(
        list_sign = list_tsig
    )

    # Create a list for storing the
    # generated alpha series entries
    list_alpha = []
```

```
    # Set the length of the list of
    # prices
    int_length = len(list_prices)

    # Carry out the algorithm to
    # generate the alpha series
    for int_i in range(0, int_length):

        if list_action[int_i] == 1:

            list_alpha.append(1)
            float_inprice = list_prices[int_i]

        if list_action[int_i] == 0:

            if list_tsig[int_i] == 1:

                list_alpha.append(
                    (float_inprice + (list_prices[int_i] -
   float_inprice) * float_ratio) / (float_inprice + (list_prices[int_i
    - 1] - float_inprice) * float_ratio)
                )

            if list_tsig[int_i] == 0:

                list_alpha.append(1)

        if list_action[int_i] == -1:

            list_alpha.append(
                (float_inprice + (list_prices[int_i] - float_inprice)
   * float_ratio) / (float_inprice + (list_prices[int_i - 1] -
   float_inprice) * float_ratio)
            )

    # Return the final alpha series
    return list_alpha
```

5.4 Price Extraction Code

To back-test a trading strategy, one requires historical price data on which they test their strategy. In this case, we extract historical price data from Yahoo Finance using the `yfinance` package.

In the code below, one can set the ticker symbol of the asset with which they want to back-test their trading strategy. In this case, the example uses the ticker symbol "BTC-USD" for Bitcoin against US Dollars.

```
# Set specified ticker symbol from
# which data will be extracted
ticker_symbol = module_yf.Ticker(
    ticker = "BTC-USD",
)
```

Extract the candlestick price data corresponding to the chosen ticker symbol. Use the `dropna()` method to remove observations with missing values.

```
# Extract historical price data for ticker
dataframe_ticker_data = ticker_symbol.history(
    interval = '1d',
    period = 'max'
).dropna()
```

Finally, implement an indicator series `list_indicator` into the main data frame. Section 6 of this report gives an example on creating an indicator series.

```
# Reset the indexing of the data frame, so new
# 'Date' column is generated
dataframe_prep = dataframe_ticker_data.reset_index()

# Add indicator to standard indexed dataframe
dataframe_prep['Indicator'] = module_pd.Series(list_indicator)

# Reassign 'Date' as index series for final dataframe
dataframe_ticker_data = dataframe_prep.set_index('Date')
```

5.5 Alpha Series Generation Code

Following price data extraction, select the observations falling within the time interval tested. In this case, the script below selects price data from the 1st of January 2021 to 1st of January 2022.

```
# Set the starting date for
# the back test
datetime_start = module_dt.datetime(
    year = 2021,
    month = 1,
    day = 1
)

# Set the final date for
# the back test
datetime_end = module_dt.datetime(
    year = 2022,
    month = 1,
    day = 1
)

# Select all price and indicator data
# falling within the chosen time bound
# and store it as 'dataframe_selected_data'
dataframe_selected_data = dataframe_ticker_data.loc[
    datetime_start:datetime_end
].reset_index()
```

Convert the close prices, indicator series and dates from `dataframe_selected_data` to lists.

```
# Create list of close prices
list_close = list(dataframe_selected_data['Close'])

# Create list for long indicator
list_long_indicator = list(dataframe_selected_data['Indicator'])
```

```
# Create list for short indicator
list_short_indicator = list(-dataframe_selected_data['Indicator'])

# Create list for dates
list_dates = list(dataframe_selected_data['Date'])
```

Lastly, use the `function_backtester` function defined earlier to generate the alpha series for the long and short versions of the utilised trading strategy.

```
# Create alpha series for long position
list_long_alpha = function_backtester(
    list_prices = list_close,
    list_indicator = list_long_indicator,
    float_ratio = 1
)

# Create alpha series for short position
list_short_alpha = function_backtester(
    list_prices = list_close,
    list_indicator = list_short_indicator,
    float_ratio = -1
)

# Create full alpha series
list_full_alpha = [

    # Multiply each element from
    # short alpha and long alpha
    # element-wise
    list_long_alpha[int_i] * list_short_alpha[int_i]

    for int_i
    in range(
        0,
        len(list_long_alpha)
    )
]
```

6 Back-Testing Example

6.1 The Heikin Ashi Indicator

As an example, we will test the performance of the Heikin Ashi indicator as a trading strategy. The Heikin Ashi indicator follows the iterative formula below:

$$I(t) = \bar{P}(t) - \bar{P}(t - \Delta_t) + \sigma I(t - \Delta_t)$$

Where σ is the strength of the indicator and $\bar{P}(t)$ is the average of the high, low, close and open prices at time t. The initial value of the indicator at time $k = \inf T$ is given by:

$$I(k) = \frac{P_l(k) + P_h(k) - P_c(k) - P_o(k)}{4}$$

The default strength of the Heikin Ashi indicator is $\sigma = 0.5$.

Below is a function for creating the Heikin Ashi indicator series from a given average price series $\bar{P}(t)$, an initial value $I(k)$ and strength σ.

```
# Create function that generates
# the Hekin Ashi indicator series
# with vaiable starting value and
# strength
def function_heikin_ashi_gen(
    list_values,
    float_initial,
    float_strength
):

    # Create list for recording
    # the indicator series
    list_output = [
        float_initial
    ]

    # Set the list of values to
    # iterate over
    int_length = len(
        list_values
    )

    # Use definition of Heikin Ashi
    # indicator to derive the rest of
    # the values
    for int_i in range(0, int_length - 1):

        # Create new indicator value
        float_new_ha = list_output[int_i] * float_strength +
   list_values[int_i + 1] - list_values[int_i]

        # Append generated indicator to
        # series
        list_output.append(
            float_new_ha
        )

    # Return the indicator series
    return list_output
```

The script below showcases how to use `dataframe_prep` from section 5.4 to extract the necessary data to create the Heikin Ashi indicator series `list_indicator`.

```
# Isolate the candlestick price data
dataframe_isolated = dataframe_prep[
    [
        'Open',
        'High',
        'Low',
        'Close'
    ]
]
```

```
# Create an intitial value for Heikin Ashi
float_initial = (
    dataframe_isolated.iloc[0][1] +
    dataframe_isolated.iloc[0][2] -
    dataframe_isolated.iloc[0][0] -
    dataframe_isolated.iloc[0][3]
) / 4

# Create list of Heikin Ashi closes from
# candlestick price data
list_hclose = list(module_np.mean(module_np.log(dataframe_isolated.T))
   )

# Create Heikin Ashi indicator list
list_indicator = function_heikin_ashi_gen(
    list_values = list_hclose,
    float_initial = float_initial,
    float_strength = 0.5
)
```

Upon deriving the indicator series, implement it into the main data set `dataframe_ticker_data` as shown in section 5.4.

6.2 Graphing Strategy Performance

To quickly compare the performance of the trading strategy used, one can graph the log of the cumulative product of each alpha series created.

```
# Set the size of the graph
module_plt.figure(
    num = 0,
    figsize = (8 * (16 / 9), 8)
)

# Plot the log cumulative product of the buy only
# alpha series generated before
module_plt.plot(
    list_dates,
    module_np.log(module_np.cumprod(list_long_alpha)),
    label = "Long only"
)

# Plot the log cumulative product of the sell only
# alpha series generated before
module_plt.plot(
    list_dates,
    module_np.log(module_np.cumprod(list_short_alpha)),
    label = "Short only"
)

# Plot the log cumulative product of the full
# alpha series
module_plt.plot(
    list_dates,
    module_np.log(module_np.cumprod(list_full_alpha)),
```

```
    label = "Long and short"
)

# Plot the log of the re-scaled close price
# series
module_plt.plot(
    list_dates,
    module_np.log(module_np.array(list_close) / list_close[0]),
    label = "No strategy"
)

# Label the x axis as the date
module_plt.xlabel("Date")

# Label the y axis as the log
# of the portfolios
module_plt.ylabel("log(Value) of portfolio")

module_plt.title("Plot comparison of variations of same strategy")

# Create a legend
module_plt.legend()

# Finally, show the graph
module_plt.show()
```

The final graph shows the performance of the long only (blue), short only (orange) and full (green) Heikin Ashi strategies against using no strategy (red).

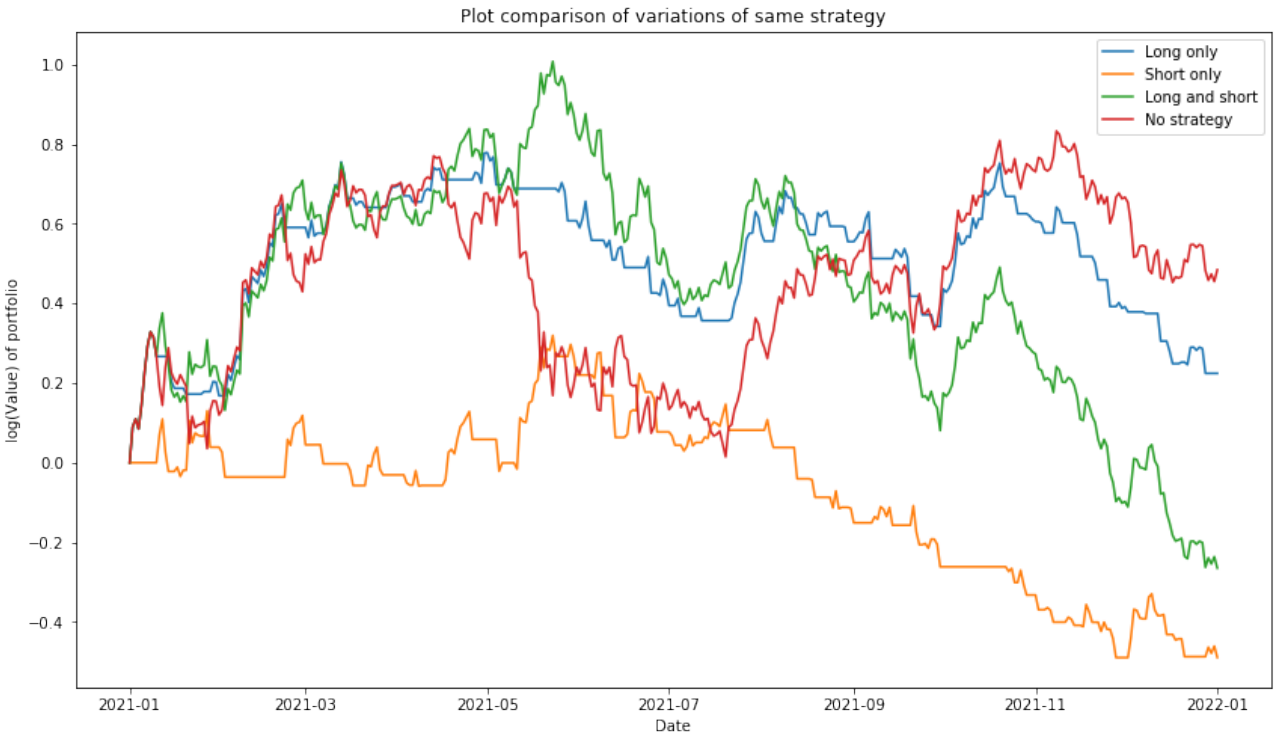

Figure 1: Performance of Heikin Ashi strategy over year 2021 for Bitcoin against US Dollars

In addition, visualising the Heikin Ashi indicator itself reveals the way it changes over time for the same period.

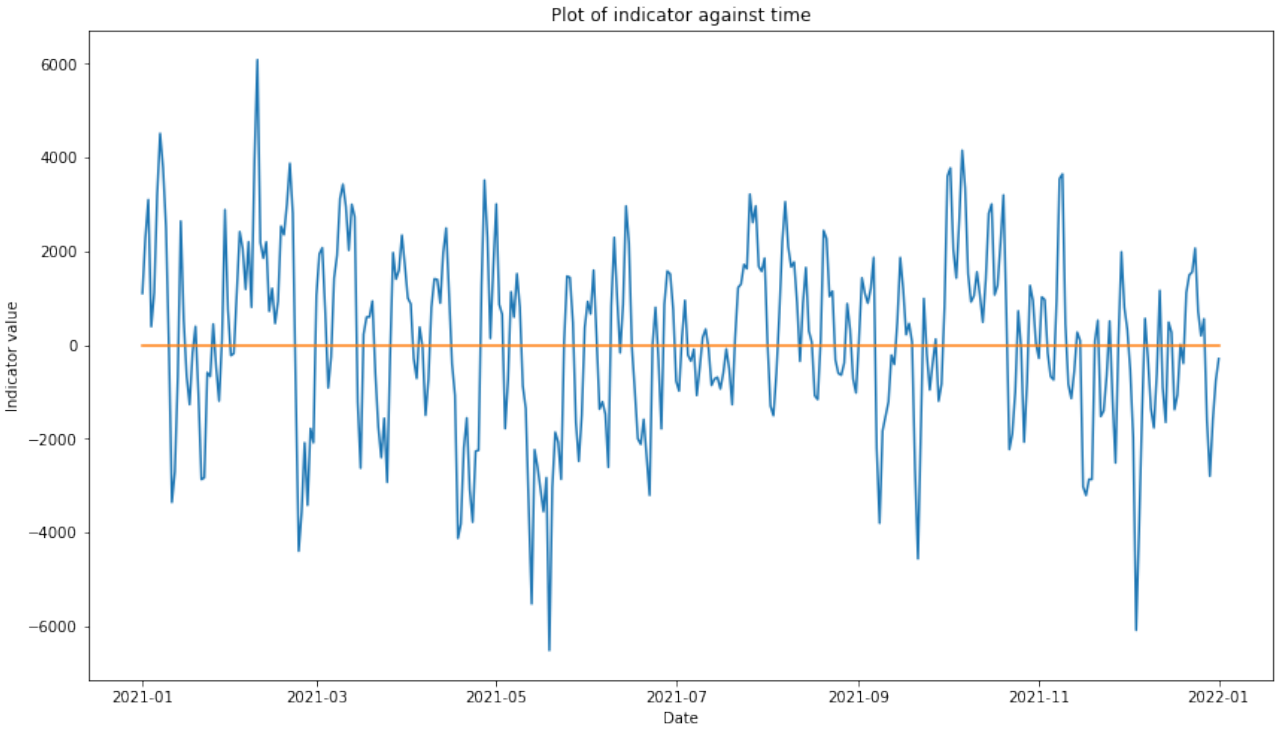

Figure 2: Heikin Ashi indicator series over year 2021 for Bitcoin against US Dollars

Evidently, following the Heikin Ashi indicator when trading Bitcoin was not a profitable strategy in 2021. This does not necessarily mean that Heikin Ashi under-performed for other assets.

7 Bibliography

1. R. Velu, M. Hardy, D Nehren, (2020), Algorithmic Trading and Quantitative Strategies, CRC Press, Taylor & Francis Group

2. A. Cartea, S. Jaimungal, J. Penalva, (2015), Algorithmic and High Frequency Trading, Cambridge University Press

3. M. J. Best, (2010), Portfolio Optimisation, CRC Press, Taylor & Francis Group

4. `https://www.investopedia.com/trading/heikin-ashi-better-candlestick/`

5. `https://finance.yahoo.com/`