Research Project Report
Quantitative Long/Short Trading Strategy on BTC-USD

Student:	Danan Solal - Ecole Polytechnique
Company:	Enigma Securities
	1 av. Franklin D. Rooselvelt, 75008, Paris, France
Internship supervisor:	Jonathan Issan
Starting date:	2nd of April 2024
Ending date:	30th of August 2024

Acknowledgments

I would like to express my deepest gratitude to my internship mentor, Jonathan Issan, for his guidance and support throughout my time at Enigma Securities. His trust in my abilities allowed me the freedom to explore and experiment with various research directions independently. His thoughtful monitoring of my progress and insightful feedback were invaluable to my development. Furthermore, his generosity in sharing his knowledge enabled me to immerse myself in a passionate field. It was an absolute pleasure to collaborate with him.

I would also enjoy to extend my sincere thanks to the entire team at Enigma Labs—Clément Gamé, Lénaïc Bagnères, Léo Doucy, and the interns, Thomas Sanchez and Elie Mazelier. Their genuine interest in my work, willingness to share their knowledge, and creative ideas were greatly appreciated and significantly enriched my learning experience. Their expertise has been crucial in shaping my understanding of algorithmic trading.

Contents

1 Introduction

1.1 Company Overview

Enigma Securities is an institutional cryptocurrency liquidity provider, offering bespoke solutions through an electronic trading platform, API access, and an OTC desk. Enigma's proprietary trading and execution technology bridges the user experiences of Traditional Finance and Decentralized Finance. By combining **quantitative models** to backtest liquidity and price ladder construction, **ultra-low-latency architecture** to minimize market impact, ensure a high fill ratios and the most competitive pricing without slippage, and a **broad range of APIs** (FIX, REST, WebSocket) to request quotes and execute trades, Enigma Securities has created an extensive trading environment.

Enigma Securities is fully owned by Makor Group, an international FCA-regulated brokerage firm. Established over six years ago, Enigma Securities has rapidly become one of the most competitive participant in the cryptocurrency market. It operates across four countries, with a team of more than 40 professionals, including developers, sales/-traders, analysts, and quantitative researchers. The company serves institutional clients, primarily banks and quantitative hedge funds.

Unlike traditional brokerage firms, Enigma Securities operates as a market maker in the cryptocurrency space. Market making, a crucial component of financial markets, involves providing liquidity by quoting both buy (bid) and sell (ask) prices for a specific asset. High-frequency trading (HFT) has further transformed market making, enabling rapid submission and adjustment of orders to capture fleeting opportunities. Traditional brokers typically act as intermediaries, executing buy or sell orders for their clients without taking any market positions themselves. They earn revenue primarily through commissions or fees for this service. However, Enigma Securities takes on a more active role by using its own funds to buy and sell cryptocurrencies, thereby providing liquidity to the market. By aggregating prices from top-tier financial performers like Jane Street and other liquidity providers (LPs) they offer competitive pricing to their clients. Their dealing desk actively creates and distributes liquidity, executing trades that are hedged using various hedging strategies.

1.2 Research Project Presentation

Enigma Labs, the research arm of Enigma Securities, aims to develop a robust toolkit of statistical signals and predictive models to generate accurate, real-time market signals, enhancing their market-making strategy and risk management by:

- Dynamically adjusting bid/ask quotes based on real-time conditions to reduce spreads and attract more trading activity. For instance, if a model predicts an upward price trend, the strategy would involve more aggressive bid prices and more passive ask prices, capitalizing on the trend.
- Managing inventory imbalances. If market trends indicate a downturn, the strategy would reduce large inventory holdings, minimizing inventory risks and potential losses.

- Forecasting volatility, which enables the widening of spreads during high-volatility periods, thus mitigating adverse selection.
- Optimizing the market entries timing, in order to improve their hedging strategies and increase the overall profit and loss (PnL).

Given the high volatility and inefficiencies in the cryptocurrency market, my project at Enigma Labs involved focusing on this toolkit by creating a medium-frequency (minute-level) long-short trading strategy on BTC-USD Spot. While some details are confidential, I will give as much information as I can. As Michael Isichenko states in his book *Quantitative Portfolio Management, The Art and Science of Statistical Arbitrage*: "whether or not a particular formula or approach is expected to make money (or avoid losses) is not disclosed or opined upon, in part because any application success is data- and implementation-dependent."

2 Database Management

2.1 Time Series Candlestick (OHLCV)

Price-Volume data, including "OHLCV" are widely used in finance, where for a specific timeframe, you get the open price, close price, low price, high price, and the volume traded during the period. This data provides a summarized view of market activity over a set interval. It is useful for analyzing market trends, applying technical indicators. I needed several timeframes to help differentiate my entry signals. For example, a long-term timeframe for trend signals and a short-term timeframe for entry signals. My first mission was to create a PostgreSQL database with the OHLCV data of BTC-USD spot since March 2023 for the following timeframes: 1 minute, 5 minutes, 15 minutes, 1 hour, and 4 hours.

To achieve this, I used CoinAPI, a well-known provider of financial data. Using the REST API paradigm to collect Coinbase data, I coded a Python script that created this database and updates itself daily at midnight to include the most recent data. This database is useful for backtesting our systematic trading strategy, and for future deployment, I will need to use WebSocket to access live trading data. One of the main challenges in creating this database was to implement a robust looping process with logs and time sleep intervals to overcome API rate limits. Table 1 shows an example of the OHLCV data for the 4-hour timeframe of BTC-USD spot.

Table 1: Selected OHLCV Data for 4-Hour Timeframe (BTC-USD Spot)

time_start	Open	High	Low	Close	Volume
2023-03-01 00:00:00	23144.37	23496.14	23025.17	23443.06	1527.656754
2023-03-01 04:00:00	23443.07	23850.00	23433.95	23718.30	1710.324235

Data Validation: Several issues can occur while collecting data using an API, so it's essential to have a validation script. It is crucial for ensuring the quality and integrity of data before processing or analysis. It involves verifying that the data meets specific criteria and standards. Therefore, I established several rules to ensure my OHLCV data was consistent. For instance, I needed to ensure that consecutive records spanned the entire data range without missing periods, that each timeframe interval was respected, and that 'price_low' was less than or equal to 'price_open', 'price_high', and 'price_close', which were all less than or equal to 'price_high'. Additionally, it was important to ensure there were no duplicate entries.

It turned out that except for a few missing dates, all the other rules were respected by our database.

Data Preprocessing: Initially, I decided to use linear interpolation to fill in the date gaps and flag them with a boolean. However, during the backtesting of my strategies, I realized this was not a good idea as it resulted in multiple winning trades during the interpolated periods, which falsified my results. Therefore, I decided to use forward fill for these missing values, which successfully overcame this issue.

3 Backtesting Framework

Backtesting is a methodical approach where traders evaluate the effectiveness of a trading strategy using historical data to see how the trading strategy would have performed on the past. It is a very important part of the journey of a trader because it serves as a risk-free testing ground for strategies, offering insights that are crucial to identify the strengths and weaknesses of the approach, fine-tune parameters, and develop confidence in the strategy before applying it in real-time market scenarios.

3.1 Backtrader

Backtrader is a well-known trading framework that provides a solid foundation. Hence, I customized this framework to suit my specific needs. It allows for the implementation of trading functions, risk management (such as stop loss and take profit), and realistic order execution (including fee management and slippage). By using Backtrader, I was able to run my strategy on the previously collected data, reproducing conditions that are as close to real market conditions as possible.

3.2 Result Analysis

The framework was designed to facilitate robust and realistic backtesting of trading strategies, providing comprehensive logs, plots and reports to analyze and refine quantitative strategies. I added visual representations of key metrics such as plots of:

- **Broker:** This plot displays the cash available and the portfolio value over time, allowing traders to monitor the financial health and growth of their portfolio.
- **Trades:** The trades plot differentiates between winning and losing trades over time.
- **Drawdown:** This essential plot for evaluating the risk taken illustrates the drawdown periods, which represent the declines from the peak portfolio value.
- **Expired Order Observer:** This plot flags expired LIMIT orders, indicating orders that were not executed within the given parameters.
- **BTC-USD Spot Price:** This plot shows the executed buy and sell orders against the BTC-USD price. It gives insights into the timing and effectiveness of the trades.
- **Signal Plots:** These plots display the entry signals used by the strategy.

To gain a deep understanding of the strategy and intelligently analyze the results, I coded a Python class that generates detailed Excel reports containing my trading journal. This in-depth reports include:

- **Strategy Information**: Containing all the strategy's parameter (Strategy used, Start Date, End Date, Broker Commission, Slippage (%), Order Type...).
- **Strategy Results (KPIs)**: Profit and Loss (PnL), Sharpe Ratio, Sortino Ratio, Volatility, Max Drawdown, Cumulative Returns and more.
- **Trade Analysis**: Win/Loss Ratio, Long & Short Trade Analysis, Win & Loss Trade Analysis, Win & Loss or Long & Short Trade Lengths, PnL and more.

- **Logs**: Detailed logs of the backtest.
- **Daily Returns**: Performance on a daily basis including date, return (%), return in cash, number of transactions, starting & final portfolio value.
- **Daily Positions**: Daily held positions including date, nominal, cash, quantity.
- **Transactions**: All executed transactions including transaction_id, date, symbol, quantity, price, nominal, order type (Market, Limit, Stop, StopTrail ...), trade type (entry, close), reason (signal, stop loss, take profit...), broker commission.

This report is crucial for understanding the strategy's performance, identifying areas for improvement, and making informed decisions based on comprehensive data analysis.

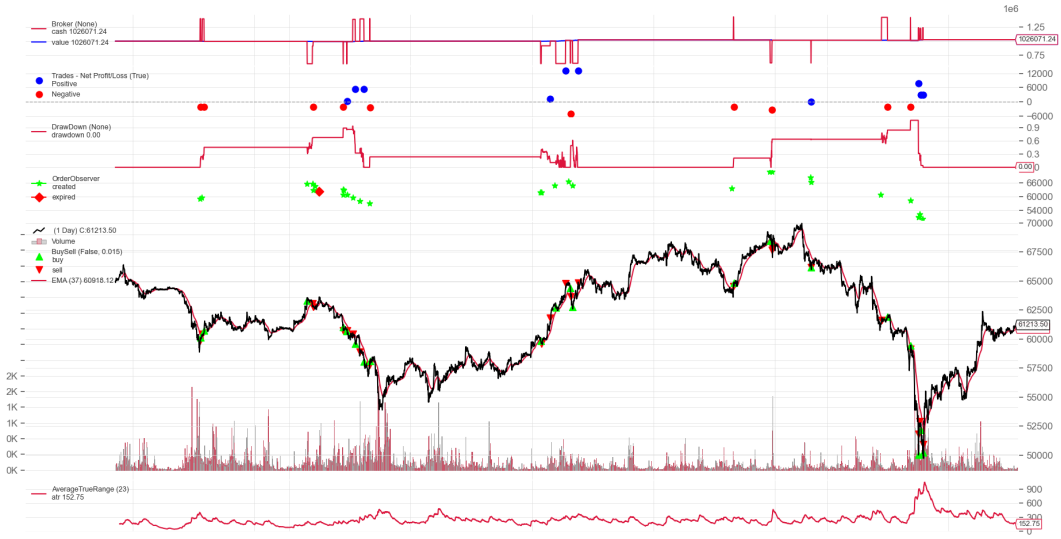


Figure 1: Backtesting plots of the strategy created for the research project.

3.3 Comparative Analysis

To enhance the strategy's performance analysis, I integrated the Quantstats packages into the backtesting framework, which generate a comprehensive HTML report. This report provide additional KPIs (time in market, calmar ratio, kelly criterion), many plots (underwater plot), and allow for an in-depth comparison between our trading strategy and a Hold Wait strategy on BTC-USD. The purpose of this comparison is to evaluate the effectiveness, risk management, and added value of our trading strategy against a simple, passive investment approach.

4 Risk Management

Risk management is a critical aspect of any successful trading strategy. Effective risk management can significantly enhance the overall performance of a strategy. By focusing on risk management, traders can protect their capital, optimize their profits, and navigate various market conditions more effectively. Given the high volatility of Bitcoin, we decided to prioritize advanced risk management techniques taking volatility into account to bolster our trading framework. This approach encompasses both money management and order management, aiming to increase profitability and mitigate potential losses across different market scenarios.

4.1 Money Management

Money management encompasses strategies and techniques to manage the allocation of capital in a trading account. For many traders, money management is the cornerstone of a successful trading strategy. Proper money management ensures that traders do not overexpose themselves to risk, preserving their capital and enabling sustained participation in the markets.

4.1.1 Position Sizing

Position sizing answers the crucial question of "how much to trade?" It determines the size of each trade, balancing potential rewards against the risks involved. Our strategy incorporates a Risk Adjusted Sizer, which adjusts the size of entry orders based on market volatility, measured by the Average True Range (ATR). This approach ensures that our position sizes are aligned with the current market conditions, allowing for more effective risk management.

Average True Range (ATR)

The Average True Range (ATR) is a technical indicator that measures market volatility, introduced by J. Welles Wilder. The ATR provides insights into price movement and helps traders assess the volatility of an asset.

The ATR is calculated as follows:

1. **True Range (TR)**: Calculate the True Range for each period:

$$\text{TR} = \max(\text{High} - \text{Low}, |\text{High} - \text{Previous Close}|, |\text{Low} - \text{Previous Close}|)$$

Where:

- High is the highest price of the current period.
- Low is the lowest price of the current period.
- Previous Close is the closing price of the previous period.

2. **Average True Range (ATR)**: The ATR is then calculated as the moving average of the True Range over a specified number of periods (e.g., n periods):

$$\text{ATR}_t = \frac{1}{n} \sum_{i=0}^{n-1} \text{TR}_{t-i}$$

Where:

- n is the number of periods used to calculate the ATR.
- TR_{t-i} is the True Range for the i-th period before time t.

Risk Adjusted Sizer Explanation

The Risk Adjusted Sizer is a tool used to determine how much of an asset to buy or sell in a trade, with the goal of managing risk effectively. Instead of blindly placing trades, this method adjusts the size of each trade based on the **amount of risk** the trader is willing to take and the current market conditions, especially **the volatility**.

For example, imagine you're trading stocks, and you want to make sure that no single trade can lose more than a certain percentage of your overall capital. The Risk Adjusted Sizer helps you calculate the appropriate number of shares to buy or sell, taking into account how much the stock price could potentially move. This approach is particularly useful in such a volatile market, where prices can change rapidly. By adjusting the trade size based on the market's volatility, the Risk Adjusted Sizer helps protect your capital, reducing your risk exposure when volatility is high while still allowing you to take advantage of trading opportunities.

The class includes several parameters such as the maximum risk per trade as a percentage of total capital, the number of bars used to calculate the ATR. These parameters have been carefully optimized to enhance the strategy's performance. This will be discussed in detail in Section 7.

4.1.2 Position Management

The trading strategy also includes a Position Management class designed to dynamically scale positions in and out based on market conditions, volatility, and momentum. This class uses signals, which will be discussed in Section 5 of the report, to either reinforce or reduce the entry position. By doing so, it helps to dynamically adjust positions, ensuring they are optimal for the current market environment. This dynamic adjustment is crucial for maintaining a balanced risk-reward ratio and capitalizing on market opportunities while managing potential downsides. Another idea, which is focused on managing positions using a grid trading strategy in low volatility scenarios, is still under development

4.2 Order Management

Order management is the process of managing and executing trading orders within a trading strategy. It is crucial because it directly impacts the execution quality, cost efficiency, and overall performance of the trading strategy.

To recap, the main types of orders are:

- **Market Orders**: An order to buy or sell immediately at the current market price. Market orders ensure execution but do not guarantee the execution price.
- **Limit Orders**: An order to buy or sell at a specified price or better. Limit orders provide control over the execution price but do not guarantee execution.
- **Stop Orders**: An order to buy or sell once the price reaches a specified level, known as the stop price. A stop order becomes a market order when the stop price is reached.

I decided to use limit orders for entering trades because they provide better control over the entry price, which is crucial for precise execution and managing slippage. Slippage refers to the difference between the expected price of a trade and the actual price at which the trade is executed. It is a common phenomenon in trading, especially in volatile markets, and can significantly impact trading performance. Effective order management strategies aim to minimize slippage, thereby ensuring better control over trade execution. To improve the backtesting realism, I used Backtrader to simulate a slippage of 1 basis point (bps) as it typically ranges between 0.5 and 2 bps.

4.2.1 Bracket Orders

Bracket orders are order types that include both a stop-loss order and a take-profit order. This means that when a entering position is executed, two additional orders are automatically created: one to close the position at a specified profit level (take profit) and another to close the position at a specified loss level (stop loss). Bracket orders help in managing risk and securing profits.

We optimized several parameters for entries and bracket orders, such as:

- `limit price`: Specifies the limit price for entering a long or short position, ensuring controlled entry.
- `order type`: Specifies the type of order for entries, stop loss and take profit.
- `validity period`: Defines how long the order remains active.
- `distance from entry price`: Determines the distance from the entry price for the stop loss and take profit orders, which is crucial for managing risk and reward.
- `risk-reward ratio`: Balances potential profit against potential loss.

The main challenge of order management lies in balancing execution speed, price control, and risk management. Ensuring that orders are executed at favorable prices without excessive delay, while also maintaining effective risk control, is essential for a successful trading strategy.

4.2.2 Trailing Strategy

No experienced trader would favor a strategy that results in several small winning trades but incurs severe losing trades. Conversely, a trader might prefer a strategy with a lower hit ratio but with significant winning trades.

We have built a Trailing Strategy within our framework that implements a dynamic trailing mechanism designed to safeguard profits and manage risks. It automatically adjusts stop-loss and take-profit levels based on two primary functionalities: initial threshold-based adjustments to secure profits and volatility-driven adjustments. It is designed to secure profits while allowing room for further potential gains in volatile conditions.

First Function: Basic Trailing Stop

This initial phase activates once the asset's price surpasses a predetermined threshold, which could be set as either a specific percentage above the entry price.

- **Trigger**: Activation occurs when the current price is $X\%$ above the entry price.
- **Action**: Moves the stop-loss to secure $Y\%$ of the profits gained so far.

Second Function: Dynamic Trailing Stop and Take Profit After the first function has been activated, this function is designed to adjust your stop-loss and take-profit levels as market conditions evolve, particularly in response to changes in volatility. This method draws inspiration from the Chandelier Exit strategy introduced by Chuck LeBeau in his book "Computer Analysis of the Futures Market." In this approach, the stop-loss and take-profit levels are not fixed; instead, they adapt based on the market's behavior. The key idea is to allow your positions to benefit from favorable price movements.

- **Dynamic Adjustment:**
  - **Stop Loss and Take Profit:**
    - **Formula for Long Positions:**

$$\text{Chandelier Stop} = \text{Highest High} - (\text{ATR} \times \text{ATR Multiplier})$$

$$\text{Chandelier TP} = \text{Highest High} + (\text{ATR} \times \text{ATR Multiplier})$$

      The stop-loss is then set to the maximum of the current stop-loss and this calculated Chandelier Stop, ensuring that the stop-loss does not drop below the initial protective level set by the first function. The take-profit is set to the maximum of the current take-profit and this calculated Chandelier TP, allowing for increased profit potential during high volatility.

    - **Formula for Short Positions:**

$$\text{Chandelier Stop} = \text{Lowest Low} + (\text{ATR} \times \text{ATR Multiplier})$$

$$\text{Chandelier TP} = \text{Lowest Low} - (\text{ATR} \times \text{ATR Multiplier})$$

      The stop-loss is set to the minimum of the current stop-loss and this calculated Chandelier Stop. The take-profit is set to the minimum of the current take-profit and this calculated Chandelier TP.

This strategy is particularly effective in volatile and trending markets where you want to maximize gains without exposing yourself to unnecessary risk. It seeks to balance securing accrued gains while permitting additional profit potential in an environment of high volatility.

During the development, implementation, and testing of these risk management features, the trailing strategy and risk-adjusted sizer proved to be highly efficient as they effectively captured market conditions. Across various market environments, the observed PnL (Profit and Loss) increased significantly. Specifically, the trailing strategy allowed for securing profits and capturing substantial winning trades. As a result, the absolute value of the average PnL of our winning trades was substantially greater than that of the losing trades. This demonstrates the effectiveness of these strategies in enhancing overall trading performance.

5 Trading Signal

A trading signal is an indicator or trigger for action like buy or sell a security. These signals can be based on various methods. Obviously, in this section, I can only provide an overview of the ideas used for the signals, without divulging specific details about the datasets or the exact methods employed to implement those signals.

5.1 Statistical Signal - Machine Learning Price Prediction

In a perfectly efficient market, the future price of a publicly traded asset is not statistically dependent on past prices; the price follows a 'random walk'. However, strategies used by hedge funds employ machine learning to generate alpha by identifying patterns, anomalies, or inefficiencies in the market that may not be apparent through traditional analysis. This motivates my research on machine learning signals, particularly in the cryptocurrency market, which is far less efficient than the US equity market.

5.1.1 Time Series Forecasting

Time series is a sequence of observations recorded at regular time intervals (hourly, daily, weekly, monthly, quarterly). Any time series can be split into the following four patterns: Base Level, Trend, Seasonality (and/or Cyclic) and Error.

A trend is observed when there is an increasing or decreasing slope in the time series. Seasonality is observed when there is a distinct, repeated pattern at regular intervals due to factors such as the month of the year, the day of the month, weekdays, or even the time of day. If the patterns do not have fixed calendar-based frequencies, they are cyclic, often influenced by business cycles and other socio-economic factors.

Classical decomposition of a time series considers it as an additive or multiplicative combination of the base level, trend, seasonal index, and residual.

- **Additive Time Series:** Value = Base Level + Trend + Seasonality + Error
- **Multiplicative Time Series:** Value = Base Level ×Trend × Seasonality × Error

Auto Regressive Integrated Moving Averag (ARIMA)

ARIMA is a class of models used for forecasting future values, that explains a given time series based on its own past values (lags) and the lagged forecast errors.

ARIMA models are characterized by three terms: p, d, and q:

- p: The order of the Auto Regressive (AR) term
- d: The number of differencing operations required to make the time series stationary
- q: The order of the Moving Average (MA) term

AR and MA Models

Auto Regressive (AR) Model:

A pure Auto Regressive (AR) model is one where Y_t depends only on its own lags:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \cdots + \beta_p Y_{t-p} + \epsilon_t$$

'p' - Auto Regressive (AR) Term: The order of the AR term refers to the number of lags of Y to be used as predictors.

Moving Average (MA) Model:

A Moving Average (MA) model is one where Y_t depends on lagged forecast errors:

$$Y_t = \alpha + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \cdots + \phi_q \epsilon_{t-q}$$

'q' - Moving Average (MA) Term: The order of the MA term refers to the number of lagged forecast errors that should go into the ARIMA Model.

ARIMA Model:

An ARIMA model combines the AR and MA terms after differencing the series d-times to make it stationary. The equation becomes:

$$Y_t = \alpha + \beta_1 Y_{t-1} + \beta_2 Y_{t-2} + \cdots + \beta_p Y_{t-p} + \epsilon_t + \phi_1 \epsilon_{t-1} + \phi_2 \epsilon_{t-2} + \cdots + \phi_q \epsilon_{t-q}$$

Order of Differencing (d)

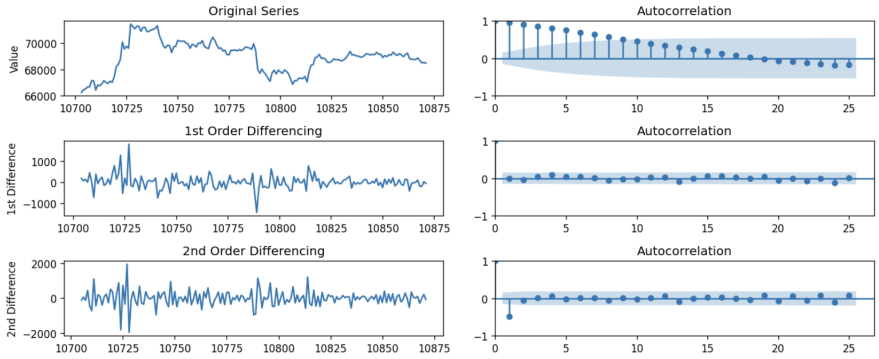
A stationary series is one where the values of the series do not depend on time, meaning that statistical properties like mean, variance, and autocorrelation are constant over time. Forecasting a stationary series is relatively easier and the forecasts are more reliable. The stationarity of a series can be tested using statistical tests called **'Unit Root Tests'**. The most commonly used test is the Augmented Dickey-Fuller Test (ADF) test, where the null hypothesis is that the time series possesses a unit root and is non-stationary. If the p-value in the ADF test is less than the significance level (typically 0.05), you reject the null hypothesis. The KPSS test, on the other hand, tests for trend stationarity, with the null hypothesis and p-value interpretation being the opposite of the ADF test.

Table 2: Unit Test Root on BTC-USD Close Price

Unit Root Test	Statistic	p-value
ADF	0.146190	0.969077
KPSS	14.422817	0.010000

Both tests suggested that the time series data is non-stationary. The ADF test indicates the presence of a unit root, and the KPSS test also confirms non-stationarity by showing trends or non-constant variance in the data.

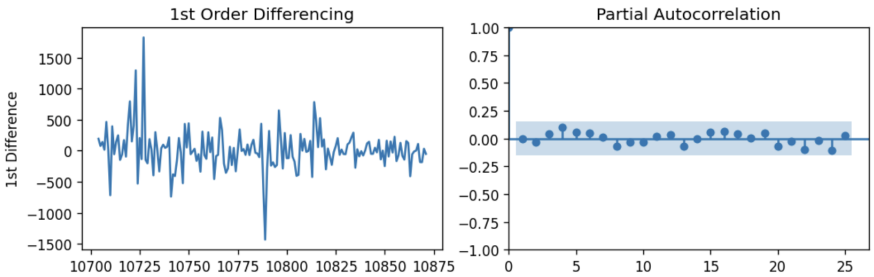
The first step to building an ARIMA model is to make the time series stationary. This is necessary because ARIMA use linear regression, which assume that the predictors (lags of the series) are independent of each other. The most common approach is differencing, where you subtract the previous value from the current value. The value of d is the minimum number of differencing operations needed to make the series near-stationary that fluctuates around a defined mean, and the ACF plot reaches zero fairly quickly. If the autocorrelations are positive for many lags, further differencing is needed. Conversely, if the lag 1 autocorrelation is too negative, the series is likely over-differenced.



For the above series, full stationarity is reached with two orders of differencing. However, the autocorrelation plot for the 2nd differencing suggests over-differencing, so the order of differencing is tentatively fixed at d=1, even though the series is not perfectly stationary (weak stationarity).

Order of the AR Term (p)

The number of AR terms required can be identified by inspecting the Partial Autocorrelation (PACF) plot. The PACF measures the pure correlation between a series and its lag, excluding contributions from intermediate lags. This helps in determining the direct effect of past values on the current value, without interference from correlations passed down through intermediate values. If the PACF shows a sharp cut-off after the first lag, an AR(1) model is typically a good fit.

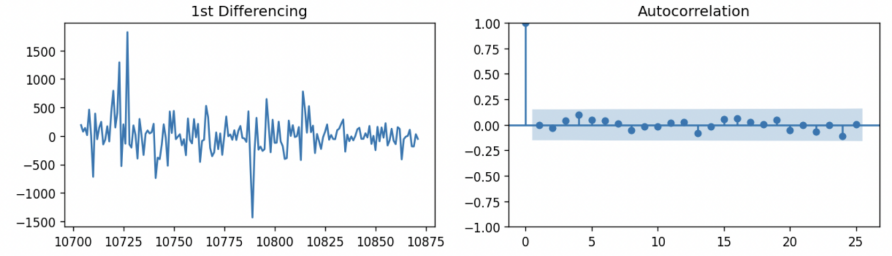


In this plot, lag 1 is significant, indicating that only the first lag is significantly correlated with the current value when the effects of all intermediate lags are removed (p=1).

Order of the MA Term (q)

The Autocorrelation Function (ACF) measures how a series is correlated with itself over different time intervals (lags). In simpler terms, it quantifies how the time series values at one time point relate to the values at previous time points (lags). Significant autocorrelation suggests that previous values of the series (lags) can help predict the current value. The ACF plot helps determine the number of Moving Average (MA) terms required to remove autocorrelation in the stationarized series.

To choose the appropriate value for q, we analyze the ACF plot of the stationarized time series. If the ACF plot shows significant spikes at lag 1 and decays quickly to zero, it suggests that an MA(1) model may be appropriate. If there are significant spikes at lags 1 and 2, an MA(2) model might be better. The key is to select q such that the autocorrelation is minimized in the residuals after fitting the model.



In the ACF plot above, the significant spike at lag 1 suggests that an MA(1) (q=1) model may capture most of the autocorrelation present in the series.

Model Performance

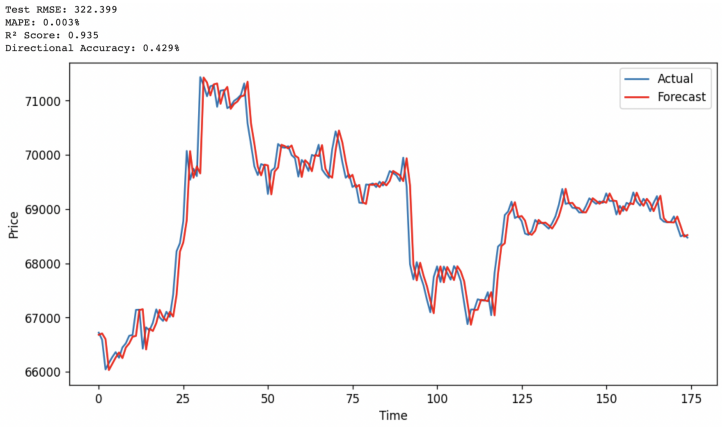


Figure 2: ARIMA Walk Forward Validation Performance

Through the previous analysis, I used ARIMA(1,1,1) for forecasting. While other values were tested, it showed the best results. Instead of predicting actual price points, the focus was on generating directional signals. Initially, ARIMA was used to predict a few hours ahead to verify if the general direction of the price was realized. However we found that a walk-forward validation approach led to more consistent results.

First, the R^2 value, which corresponds to the proportion of variance in the dependent variable that is predictable from the independent variables, indicated a good fit. In this context, the independent variable is the lagged value(s) of Bitcoin prices, and the dependent variable is the next Bitcoin close price. Additionally, the low Mean Absolute Percentage Error (MAPE) also indicated a good fit.

However, the directional accuracy achieved was only 42.5%, which is insufficient for inclusion in a trading strategy. This low directional accuracy may be seen in the decay of one period in the resulting plot.

Analysis and Potential Improvements

- **High Noise in Bitcoin Prices:** Bitcoin prices exhibit a high level of noise. In finance, the "low signal-to-noise ratio" is a metaphor borrowed from engineering and physics, where it describes the ratio between the level of useful signal and the background noise. It means that the data is dominated by random fluctuations rather than predictable patterns, making it harder to achieve reliable forecasts using traditional methods like ARIMA.
- **Exogenous Variables:** When autocorrelation is absent, exogenous variables should be considered. For instance, to predict Bitcoin prices in the short term, exogenous variables such as market sentiment (through NLP tweet analysis), the impact of key opinion leaders, or analysis of relevant news could be incorporated. The SARIMA/X model, which includes seasonality and exogenous variables, is a more promising adaptation of ARIMA. This could lead to more accurate predictions by capturing the underlying factors driving Bitcoin's price movements.
- **Memory vs. Stationarity Trade-off:** Differencing makes the data stationary, but with first-order differencing, we lose all the historical patterns of the data, thereby losing its memory, which is the predictive power of any model. A potential solution is fractional differencing, where instead of differencing with order 1, we perform differencing with a fractional value. This approach retains some of the information, potentially improving the model's predictive power.

5.1.2 Price Movement Classification

Previous attempts using traditional time series methods, such as ARIMA, did not yield promising results. Therefore, a different approach was adopted, focusing solely on price direction as a binary classification problem. This problem was formulated by creating a target variable that predicts the future price direction over a specified period. The target variable was generated by looking 'forward' a predetermined number of periods. The time to which we compared the price change, be it 1 bar, 2 bars or 5 bars was a source of optimisation and needs to be adapted accordingly to the strategy. If the price increased, the target variable was labeled as 1, indicating an upward movement; otherwise, it was labeled as 0, signaling a downward movement.

The distribution of this target variable in the dataset was balanced (no class imbalance), as illustrated by the pie chart in Figure 3.

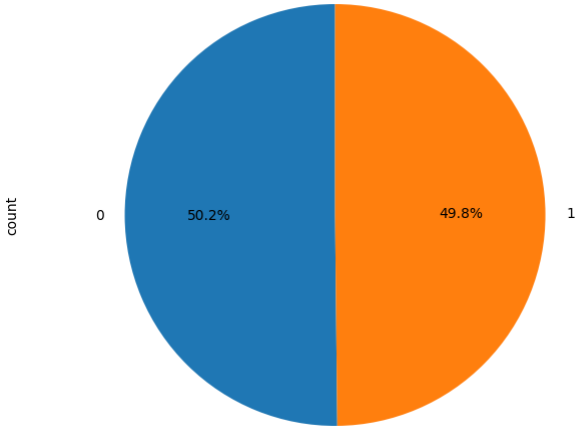


Figure 3: Distribution of Target Variable Classes (Upward vs. Downward Movement)

Feature Engineering

Feature engineering is the process of selecting, transforming, and creating new variables or "features" from raw data to improve the performance of machine learning models. In this context, feature engineering focused on the selection and creation of technical indicators that aim to reduce noise and uncover potential price patterns. These indicators are crucial for providing insights into various market dynamics such as momentum, volatility, trends, and market conditions (e.g., overbought or oversold states). Specifically, features such as the Rate of Change Ratio, Average True Range, Momentum, and Exponential Moving Averages were employed. Additionally, I used weighted moving averages of volume over the past X days, drawing a parallel to how previous lags are utilized in ARIMA models. For data distributions that exhibited significant skewness during Exploratory Data Analysis (EDA), a square root transformation was applied to normalize these features, thereby reducing skewness.

Feature Selection

To identify the most impactful features variable I used two approaches: feature importance scores from the ExtraTrees classifier and p-values from logistic regression. The ExtraTrees classifier ranks features based on their usefulness in predicting the target variable. Features with higher importance scores were prioritized. This is particularly useful in financial data, where understanding the impact of different features can provide insights into the driving factors behind price movements. Additionally, logistic regression was used to refine the selection by evaluating the statistical significance of each feature. In logistic regression, a p-value is calculated for each feature to assess whether it has a meaningful relationship with the target variable. The p-value is derived from a Wald test that evaluates the null hypothesis, which states that there is no association between the feature and the target variable. A low p-value (< 0.05) indicates that there is strong evidence to reject the null hypothesis, suggesting that the feature is significantly related to the target variable and should be included in the model.

To avoid the *'curse of dimensionality'*, I limited the number of features selected. It refers to the problems that arise when working with large dimensions' datasets. They include an increase in computational efforts and a loss of intuition and interpretability.

Data Preprocessing

Feature scaling was implemented through standardization (using a Standard Scaler), which is essential for many machine learning algorithms. This preprocessing step transforms the features to have a mean of zero and a standard deviation of one. This normalization is a crucial step because it ensures that each feature contributes equally to the distance calculations in models that are sensitive to the scale of the data such as KNN Classifier. Additionally, standard scaling helps to speed up the convergence of gradient-based optimizers by maintaining a consistent scale across all dimensions.

Model Selection

Starting from simple baseline classifiers to more complex ensemble methods, several models were tested to identify effective approaches for predicting price movements.

- **Baseline Classifier:** A simple dummy classifier was used as a baseline to evaluate the performance of more sophisticated models. This dummy classifier makes predictions without considering any underlying data patterns, often by random guessing. In our case, the dummy classifier achieved an accuracy of 0.49 on the testing set, which reflects the accuracy of a model with no domain knowledge or learning.
- **Extra Trees Classifier:** The Extra Trees Classifier is an ensemble method that builds multiple decision trees and aggregates their predictions to enhance accuracy and robustness. A decision tree is a model that makes decisions by splitting the dataset into branches based on feature values, leading to a final decision at each leaf node. In ensemble methods like Random Forests, multiple decision trees are created, and their predictions are combined to improve the overall performance of the model. Random Forests achieve this by constructing each tree from a different subset of the training data and using a random subset of features at each split. This randomness helps in reducing overfitting and improving generalization.

Extra Trees, short for Extremely Randomized Trees, take this a step further by introducing additional randomness in the choice of split points within the features. While Random Forests search for the best split among a subset of features, Extra Trees choose split points randomly within each feature, further reducing variance. This makes Extra Trees particularly effective on high-dimensional datasets like ours, where it helps in generalizing better to unseen data.

- **Logistic Regression:** Logistic Regression is a linear model commonly used for binary classification tasks. It models the probability that a given input belongs to a particular class by applying the logit function to a linear combination of the input features. Mathematically, the model predicts the probability $P(y = 1)$ as:

$$P(y = 1) = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x_1 + \cdots + \beta_n x_n)}}$$

where β_0 is the intercept, $\beta_1, \ldots, \beta_n$ are the coefficients, and $x_1, \ldots, x_n$ are the input features. Logistic Regression provides a clear and interpretable model, where the coefficients β represent the log odds of the outcome per unit change in the predictor. This interpretability makes Logistic Regression valuable not only for prediction but also for understanding the impact of each feature on the target variable.

- **XGBoost Classifier:** The XGBoost Classifier was integrated into our analysis framework due to its well-known efficiency and superior performance in handling complex datasets. XGBoost, which stands for eXtreme Gradient Boosting, is an implementation of the gradient boosting framework. It builds models in a sequential manner, where each new model attempts to correct the errors of the previous ones by minimizing a differentiable loss function. The key idea behind XGBoost is to combine the outputs of multiple weak learners, typically decision trees, into a single strong learner. Mathematically, the model aims to minimize the loss function by adding trees sequentially. In each iteration, it fits a new tree to the residuals of the current model:

$$\text{New Prediction} = \text{Old Prediction} + \eta \times \text{New Tree Predictions}$$

where η is the learning rate, a hyperparameter that scales the contribution of each new tree. During our experiments, XGBoost demonstrated robust performance, balancing speed and accuracy.

Stacking: A stacking ensemble technique was employed to combine the predictions from multiple models to produce a stronger final model. The advantage of stacking is that it leverages the strengths of different models, potentially improving predictive performance by allowing each model to specialize in different aspects of the data. In stacking, the predictions from the base models are used as input features for a higher-level model, often referred to as a meta-learner, which makes the final prediction. This meta-learner can identify patterns and relationships between the predictions of the base models, effectively combining their outputs in a way that enhances overall accuracy. However, there are some challenges associated with stacking. One significant drawback is that correlations between the base models can introduce bias, which may negatively affect the ensemble's generalization ability. If the base models make similar errors or rely on similar features, the meta-learner might inherit these biases, leading to suboptimal performance.

Hyperparameter Tuning

I used random search to optimize hyperparameters across the four models used. These included parameters such as the learning rate, maximum depth of the trees, number of trees, the type of regularization penalty (L1 or L2), and the regularization strength.

Cross-Validation

To ensure that the model's performance was not only high but also stable across various segments of the data, I applied K-fold cross-validation. This method enhances the reliability of our evaluation by mitigating the potential biases that could arise from a single, static split of the data into training and test sets. Specifically, the dataset was divided into K equal parts, with the model being trained on K-1 folds and validated on the remaining fold. This process was iterated K times, each time with a different fold held out for validation, providing a comprehensive assessment of the model's performance.

Model Evaluation

The evaluation of our predictive models incorporated several key performance metrics, with a particular emphasis on accuracy. Additionally, precision and negative predictive value were calculated to provide a more nuanced view of the models' performance. These metrics are given by the following formulas:

- **Precision**:

$$\text{Precision} = \frac{\text{True Positives (TP)}}{\text{True Positives (TP)} + \text{False Positives (FP)}}$$

  Precision measures the proportion of true positive predictions out of all positive predictions made by the model.

- **Negative Predictive Value (NPV)**:

$$\text{NPV} = \frac{\text{True Negatives (TN)}}{\text{True Negatives (TN)} + \text{False Negatives (FN)}}$$

  It measures how many predictions out of all negative predictions were correct. You can think of it as precision for negative class.

- **Accuracy**:

$$\text{Accuracy} = \frac{\text{True Positives (TP)} + \text{True Negatives (TN)}}{\text{Total Predictions}}$$

  Accuracy measures the proportion of correct predictions (both true positives and true negatives) out of all predictions made by the model.

The scores results after cross-validation were as follows:

Table 3: Model Performance on BTC-USD Close Price Prediction

Model	Precision	Negative Predictive Value	Accuracy
Logistic Regression	0.5227	0.5273	0.5251
ExtraTrees	0.5469	0.5157	0.5315
XGBoost	0.5278	0.5572	0.5403
Stacking	0.5383	0.5512	0.5446

The stacking model achieved reasonably good accuracy, it was incorporated as a signal in the overall trading strategy. However, this model is subject of endless further refinement and research to enhance its ability to generate alpha.

Potential Improvements

Currently, online learning is not implemented, which means that over time, the data on which the model was trained may become outdated. Implementing partial training at regular intervals (e.g., weekly or monthly) could help the model adapt to new data and maintain its predictive performance. Partial training refers to updating the model incrementally with new data rather than retraining from scratch, thus improving efficiency and responsiveness to market changes.

This classification model relies solely on OHLCV data, which, while useful, is relatively simplistic. Expanding the dataset to include diverse sources could enhance the overall accuracy of the model. Hedge funds often incorporate a wide range of datasets to capture alpha, including news data, social media sentiment (using NLP), macroeconomic indicators, and fundamental data. For instance, achieving similar goals using order book data and the Order Book Pressure (OBP) as a feature is more promising, but it presents database management challenges (this model is currently a work in progress). Additionally, incorporating blockchain-specific data could significantly improve predictive power, especially in the context of cryptocurrencies.

- **Block Size:** Larger average block sizes may indicate higher transaction volumes, potentially correlating with price movements.
- **Difficulty:** Changes in mining difficulty could indicate shifts in mining power, possibly affecting market sentiment.
- **Hash Rate:** A higher hash rate could indicate a more secure network and greater miner confidence.
- **Unique Addresses:** A higher number of unique addresses might indicate broader adoption and usage.

5.2 Probabilistic Signal - Technical Indicators

We improved the trading strategy by incorporating well-known technical indicators. These indicators are commonly categorized into two classical families used in technical analysis: momentum and mean reversion. Momentum indicators capitalize on the tendency of assets to continue moving in the same direction, aiding traders in identifying and following trends. In contrast, mean reversion indicators are based on the premise that prices will eventually return to their historical average, helping to identify potential reversal points. Each of these indicators involves several parameters that have been carefully optimized to enhance the strategy's performance, more details on the Section 7. Below, I present some of the technical indicators used in the strategy.

Bollinger Bands

Bollinger Bands are a mean reversion technical indicator that generates overbought and oversold signals using three bands. The bands expand and contract based on market volatility. When the price touches the upper band, it may be considered overbought; when it touches the lower band, it may be considered oversold. We decided to modify the formula of the bands to be more suited for our strategy.

- **Formula:**

$$\text{Middle Band} = \text{EMA}(n)$$

$$\text{Upper Band} = \text{EMA}(n) + \delta_{\text{Hi}} \cdot \sigma$$

$$\text{Lower Band} = \text{EMA}(n) - \delta_{\text{Lo}} \cdot \sigma$$

where $\text{EMA}(n)$ is the exponential moving average over n periods, σ is the standard deviation, and δ_{Hi} and δ_{Lo} are the upper and lower deviation coefficients, respectively. Conventionally, δ is the same for both bands (usually set to 2), but we implemented different values to better capture small dips with quick corrections and be less sensitive to potentially false uptrends. We used an EMA instead of an SMA as it gives more weight to recent data, making it more optimal for short-term trading.

Average Directional Index (ADX)

The Average Directional Index (ADX) is a momentum indicator that measures the strength of a trend but not its direction. It helps traders identify whether the market is trending or ranging. We use the ADX to confirm trends if it is above a certain threshold.

- **Formula:**

$$\text{ADX} = 100 \times \text{EMA}\left(\frac{|\text{+DI} - \text{-DI}|}{\text{+DI} + \text{-DI}}\right)$$

where +DI (Positive Directional Indicator) and -DI (Negative Directional Indicator) are calculated as the smoothed values of directional movement over a specific period.

Stochastic Oscillator

The Stochastic Oscillator is a momentum indicator that compares a particular closing price to a range of prices over a certain period. It is a momentum indicator because it measures the speed and magnitude of price movements relative to recent price history. It also serves as a mean reversion indicator by helping traders identify overbought or oversold conditions. Values above a certain threshold suggest an overbought condition, while values below a certain threshold suggest an oversold condition.

- **Formula:**

$$\%K = \frac{(C - L_n)}{(H_n - L_n)} \times 100$$

$$\%D = \text{SMA}(\%K, m)'$$

where C is the most recent closing price, L_n is the lowest price over the last n periods, H_n is the highest price over the last n periods, and $\%D$ is the m-period simple moving average of $\%K$.

I used a combination of these indicators, along with the previously introduced signals (Order Book Pressure and Price Movement Classification), on different timeframes (1-hour data for identifying trends and 15-minute data for entry points) to create the strategy's logic. The detailed methodology of the implementation is confidential.

6 Data Simulation

To avoid overfitting and ensure the robustness of our strategy, it is necessary to test our approach on out-of-sample data. An additional method is to test our strategy on simulated data, which allows us to evaluate performance under various market conditions and scenarios. Therefore, we have developed data simulation models that generate synthetic OHLCV data to support trading strategy testing and validation.

6.1 Random Walk Model

The first approach was to generate data using a Brownian Model, based entirely on stochastic processes and random walk. This approach simulates market cycles of random lengths along with randomly assigned market trends (bullish, bearish, neutral). It models price movements using random fluctuations, mirroring the unpredictability observed in real financial markets. We used the Augmented Dickey-Fuller (ADF) test to check for stationarity. However, this approach proved unsuitable because BTC-USD spot price does not follow a purely random walk and is not stationary, as revealed by our time series analysis.

6.2 Hybrid Model

Hybrid models take advantage of the strengths of different modeling techniques, making them particularly useful in complex systems. The hybrid model developed in this project combines actual market data with a random factor on the realized volatility to simulate realistic market conditions.

First, we calculate the BTC-USD logarithmic returns, which measure the rate of return of a security over a specific period. These are calculated as the logarithm of the closing price of the current period divided by the closing price of the previous period. The cumulative log returns are adjusted by the historical volatility. The historical volatility, which represents the standard deviation of the log returns, captures the variability of the asset's returns over time. By applying a random adjustment factor to this volatility, the model introduces additional randomness, simulating sudden market fluctuations.

The adjusted log returns, with added randomness, are then used to generate new prices using the formula:

$$\text{New\_Price} = \text{Previous\_Price} \times e^{(\text{Cumulative\_Log\_Returns})}$$

Finally, to maintain realism, the hybrid model targets correlations in the range of 60-70% with actual market data. This ensures that the simulated data closely resembles real market conditions, making the model useful for backtesting and strategy development. Figure 4 shows an example of data simulation using this model.

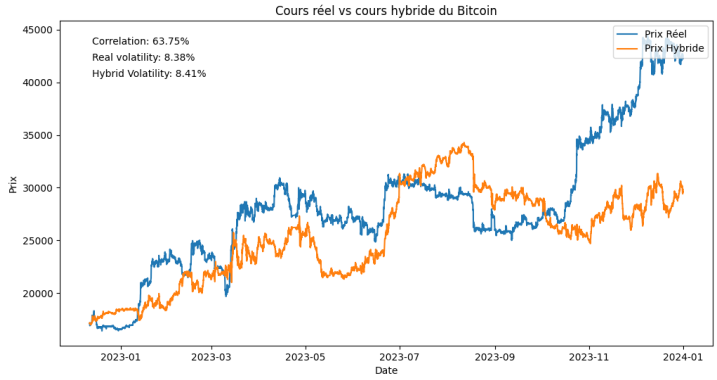


Figure 4: Hybrid Model Simulated Data & Actual Market Data

6.3 Crisis Simulation

The last idea of our analysis was to evaluate how the trading strategy would perform during crisis periods. To achieve this, we enhanced the data simulation model by incorporating crisis events. Specifically, we combined our previously developed Hybrid Model with data representing crisis scenarios. Using a randomly selected start date, the model applies percentage changes similar to those observed in the NASDAQ during the 2008 financial crisis to Bitcoin data. This simulation aims to approximate how Bitcoin might respond under comparable severe market conditions. While this third model could serves as a valuable tool for assessing the resilience of our strategy in the face of significant economic downturns, it operates under strong assumptions that do not mirror reality.

This comprehensive approach ensures that our trading strategy is rigorously tested against a broad spectrum of market conditions, including extreme events. This enhanced robustness and reliability are crucial as we move towards validating and potentially implementing our strategy in real-world scenarios.

Figure 5: Crisis Model Simulated Data & Actual Market Data

7 Optimization

Our strategy involves many parameters, including those for signals, order management, and money management. Finding the optimal hyperparameters is challenging and the subject of extensive research. To address this, I integrated an Python optimization module to apply some algorithms. All these algorithms aim to find the parameters that maximize the Sharpe ratio and the Sortino ratio, widely used KPIs in quantitative trading.

The Sharpe ratio measures risk-adjusted return and is defined as:

$$\text{Sharpe Ratio} = \frac{R_s - R_f}{\sigma_s}$$

where R_s is the return of the strategy, R_f is the risk-free rate, and σ_s is the standard deviation of the returns. A higher Sharpe ratio indicates better risk-adjusted performance.

The Sortino ratio focuses on downside risk and is defined as:

$$\text{Sortino Ratio} = \frac{R_s - R_f}{\sigma_d}$$

where σ_d is the standard deviation of negative returns. This ratio provides a more focused measure by penalizing only downside volatility, making it useful for strategies aiming to minimize risk.

While it is also possible to optimize for maximum PnL, incorporating risk metrics like the Sortino ratios into the optimization process provides better risk management.

7.1 Optimization Algorithms

Grid Search: Grid search systematically evaluates all possible combinations of specified hyperparameters within a defined grid. It ensures all potential combinations are tested and is straightforward to implement. However, it can be computationally expensive and time-consuming, especially with large parameter grids.

Random Search: Random search samples parameter combinations randomly from the search space. It is often more efficient than grid search, especially in high-dimensional spaces, as it requires fewer evaluations to find good solutions. However, it does not guarantee that all regions of the parameter space will be explored.

Bayesian Optimization: Bayesian optimization uses probabilistic models to find optimal parameters efficiently. It constructs a posterior distribution of the objective function based on past evaluations and uses this to suggest the next most promising points to evaluate. This is typically done using Gaussian processes, which are a type of statistical model.

1. **Define a prior**: A Gaussian process typically represents the objective function. A Gaussian process is a collection of random variables, any finite number of which have a joint Gaussian distribution. It is defined by a mean function and a covariance function (or kernel).

2. **Update the prior**: Incorporate new data to refine the model. Each new evaluation of the objective function updates the Gaussian process, improving its accuracy.
3. **Acquisition function**: Selects the next point to sample, balancing exploration (searching new areas) and exploitation (refining known good areas).

Bayesian optimization is particularly effective when the objective function is expensive to evaluate, as it requires fewer iterations to find good parameter values.

7.2 Avoiding Overfitting

One of the main challenges in backtesting a strategy is avoiding overfitting. It occurs when a trader repeatedly backtests the same historical period, adjusting trading rules based on previous backtest results. This leads to a strategy that performs well during the historical period but poorly on new, unseen data because it captures noise instead of underlying trends. To prevent overfitting, it's crucial to avoid over-optimizing parameters. I decided to use the following two methods for the strategy's parameter optimization.

In/Out Sample Validation: A common approach is to split the data into in-sample and out-of-sample datasets. Optimization is performed on the in-sample data, while the out-of-sample data is used to validate and test the strategy's consistency.

Walk Forward Validation: A more advanced method to avoid overfitting is Walk Forward Validation. In this approach, the historical data is divided into multiple segments, and the model is repeatedly trained on a rolling window of in-sample data, followed by testing on the subsequent out-of-sample data. For example, we might use an initial in-sample period of 3 months and an out-of-sample period of 1 month. The model is optimized on the first 3 months of data, then tested on the following month. The window is then shifted forward by one month, and the process is repeated. This method provides a robust measure of the strategy's performance by aggregating results from all out-of-sample periods, ensuring that the model generalizes well to new data and effectively reduces the risk of overfitting.

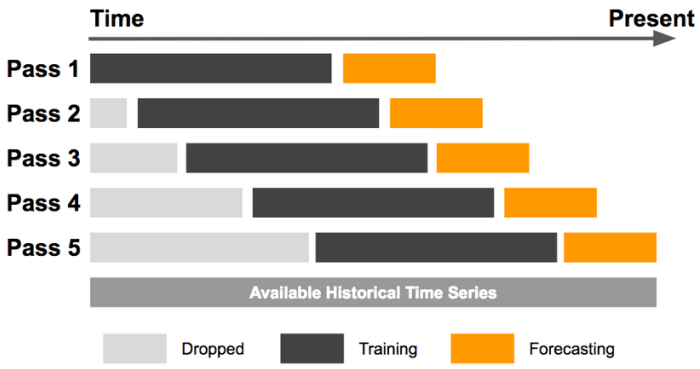


Figure 6: Illustration of Walk Forward Validation

7.3 Optimization Algorithms Comparison

The created optimization module returns a dictionary with the best parameters, key performance indicators (Sharpe and Sortino Ratios), time taken for each optimization process, and the number of iterations or evaluations performed. This data is crucial for comparing the efficiency and effectiveness of each optimization technique.

I compared Bayesian optimization with random search using an equal number of iterations to evaluate their performance. I found that both approaches showed promising results, but Bayesian optimization often reached better maxima in less time, demonstrating its efficiency in finding optimal parameters with fewer evaluations.

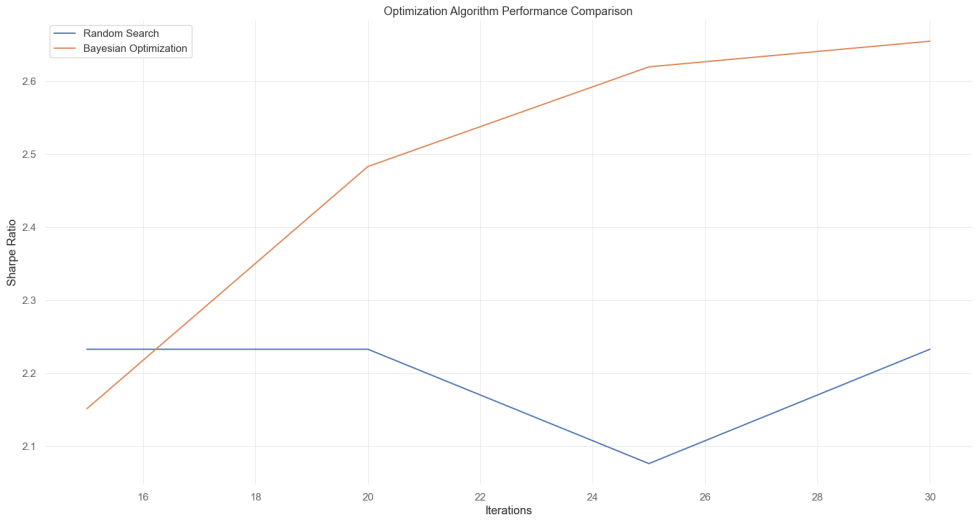


Figure 7: Comparison of Random Search and Bayesian Optimization

8 Strategy Performance Evaluation

Being overly optimistic and overlooking losses during a backtest can skew results. The goal of a backtest is not to "win" and realize paper profits but to develop a robust strategy for live trading. It is preferred to be pessimistic during backtesting to avoid ending up with a seemingly profitable strategy that fails during live trading, also called a false positive strategy.

Key Performance Indicators:

The following metrics were obtained from March 2023 to July 2024 using parameters obtained from the walk-forward validation and a starting cash of $1,000,000:

- **Final Portfolio Value**: $1,187,681.576
- **Cumulative Returns (%)**: 18.768%
- **Annualize Returns (%)**: 13.9%
- **PnL Net**: $187,681.576
- **Sharpe Ratio**: 2.461
- **Sortino Ratio**: 7.085

On paper, this looks very good (a Sharpe ratio over 2 can be considered as a very good strategy, balancing a good return with not too much risk taken). When evaluating backtest results, it is essential to focus on a few critical metrics to have a better understanding of the strategy's performance.

Hit Ratio: The hit ratio is a measure of the proportion of winning trades to the total number of trades. Although a trading strategy may be profitable during a backtest, a low hit ratio can be mentally challenging to trade. More losing trades can lead to mental mistakes. The strategy showed a hit ratio of 51%, which is slightly over 50%. However, the risk management modules and imposed risk-reward ratio ensure that the average winning trade ($6,381) is significantly greater than the average losing trade ($2,959). This indicates that having more losing than winning trades is not a problem, as the wins are larger.

Holding Time: Many traders struggle with holding onto winning trades. A strategy with a long holding time is harder to trade, increasing the likelihood of errors. I was looking for a strategy with a shorter holding time. Here, it has a 22% time in market, which is good as it reduces exposure and risk during adverse market conditions.

Losing Streaks: The maximum losing streak stood at six. Considering that we have slightly over 100 trades (or over 250 transactions) for this backtest, this is a bit too much. Long losing streaks can be a red flag, as they can lead to emotional trading errors.

Max Monetary Drawdown: Drawdown represents the peak-to-trough decline during a specific period of an investment. Here the max drawdown is at $27,380.867. This indicates the largest loss from any peak in the portfolio's value during the backtest period. An underwater plot, shown in Figure 8, visualizes drawdowns over time, highlighting the maximum drawdown and its duration. At the end, we see clearly when the max drawdown occurs, which is at the end of the out-of-sample data, suggesting a weakness of the strategy.

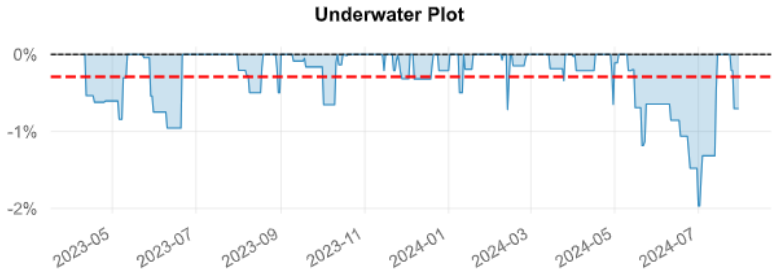


Figure 8: Underwater Plot

Consistency:

Consistency is crucial in backtesting a strategy and can be assessed using the cumulative return growth curve. Consistent performance comes with steady growth curve. Although the performance of a more volatile strategy may be slightly better, it is mentally harder to rely on due to longer sideways periods and sudden leaps. The following growth curve shows consistent results across the historical data but still shows on few strong winning trades outliers.

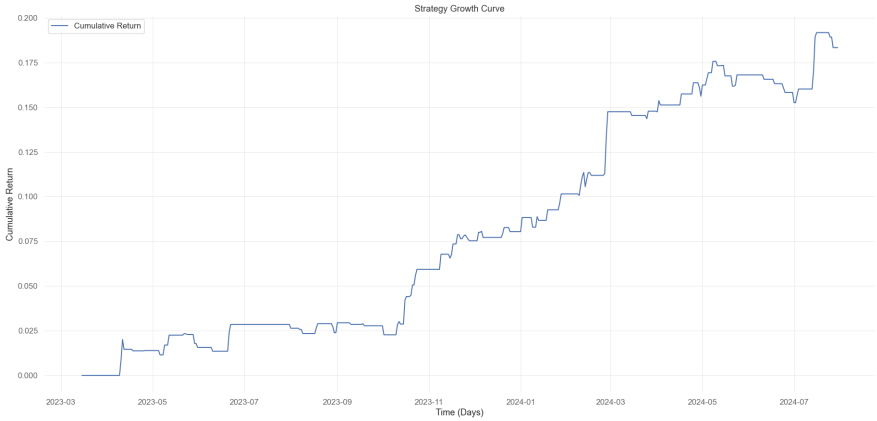


Figure 9: Growth Curve

Robustness:

I aimed to ensure maximum robustness by avoiding over-optimizing parameters and using walk-forward validation techniques. I also analyzed the strategy on simulated data. It did not perform as well on simulated data as on real market data, suggesting slight overfitting.

Statistical analysis is the backbone of backtesting and adds another layer of robustness, ensuring that a strategy is more likely to perform well in live trading. Statistical Tests like Bonferroni's are used to reduce the instance of a false positive by ensuring that the signals are not just a result of random variations. Techniques like Monte Carlo simulations enhance the reliability of backtesting results. Using the strategy's daily returns, I simulated the strategy performance 1,000 times over a year to obtain statistical properties that could help assess the robustness of the strategy.

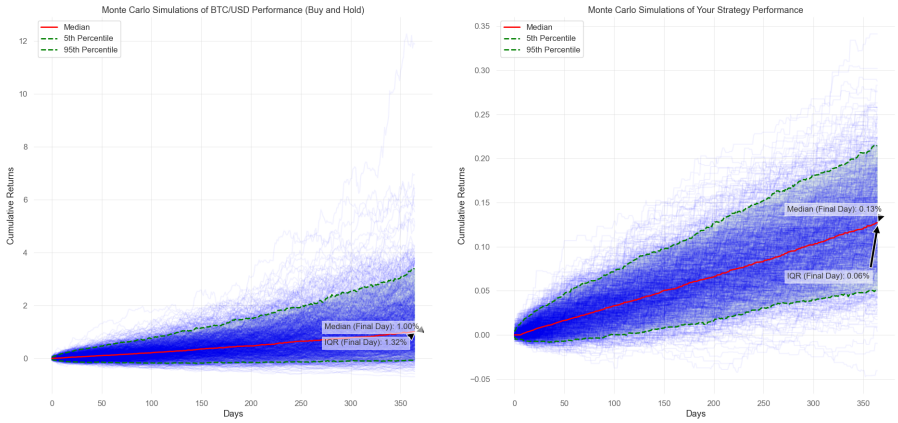


Figure 10: Buy and Hold vs Strategy Monte Carlo Simulation

Firstly, The median cumulative return of the 1,000 simulations of the strategy shows an annualized return of 13%, lower than the 14% obtained, indicating that the 18% return is likely too optimized and not achievable in live trading.

However, the interquartile range allows assessing the precision and reliability of the strategy. We aim to minimize it, proving that the strategy is not too volatile and is stable. Here, we can clearly see that this range is far little in the strategy than in the buy-and-hold benchmark. Indeed, our strategy is significantly less volatile. This can be seen in Figure 11, showing the cumulative return volatility-matched comparison plot against a benchmark.

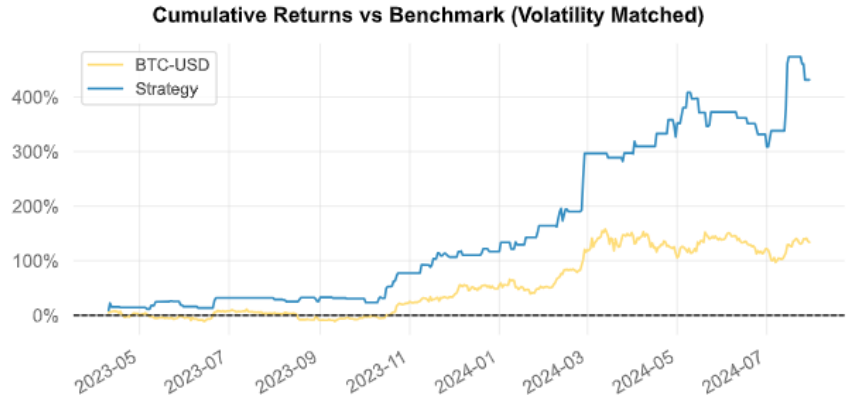


Figure 11: Strategy vs Buy and Hold Cumulative Returns (Volatility Matched)

This plot clearly shows a positive aspect of the strategy as it has very positive results compared to the risk taken. This explains why the Sortino and Sharpe ratios tend to be high.

Completing this analysis helps ensure that the backtested strategy is robust, reliable, and less likely to fail during live trading, providing confidence to traders and investors.

9 Conclusion

Summary of Key Findings

The results are strong and consistent, largely due to effective risk management features like the trailing stop and position sizing, which adapt to market volatility. Although some backtests of the strategy have more losing trades than winning ones, the profits from the winners significantly outweigh the losses. This underscores the importance of money management in our approach, which has been crucial but still offers room for improvement.

Our combination of signals shows promise, but there's potential to enhance performance by incorporating additional datasets. Relying solely on OHLCV data doesn't fully capture market complexity. Integrating order book data and news sentiment analysis could greatly improve the strategy, especially in the volatile cryptocurrency market. For example, during my internship, major events like Germany's large BTC liquidation or JumpTrading Crypto's ETH transfer had significant market impacts. Capturing such news with Natural Language Processing could better reflect market sentiment, such as Fear of Missing Out as well as panic selling, and enhance the strategy, preparing it for real-world application in live environments.

Internship Reflections

This research project was highly fascinating, allowing me to explore research papers, delve into quantitative trading, and meet passionate professionals. I'm grateful to Enigma Labs' team for their warm welcome. It has been a transformative experience, both academically and professionally. I am deeply grateful to my supervisor, Jonathan Issan, for his unwavering support, guidance, and trust throughout this journey.

This experience has fueled my desire to continue in quantitative research, however, with a stronger focus on data science, which is my primary interest. I had a solid introduction by integrating a machine learning model into my signals and by reading extensively on the industry, including the book by Michael Isichenko mentioned earlier. Moving forward, I aim to deepen my understanding of the field by working in a team dedicated to the R&D of trading strategies implemented in live environments, which I find particularly inspiring. As I embark on the next phase of my academic and professional career, I carry with me the knowledge, skills, and passion developed during this internship. There is still much more to explore, and I am eager to continue this journey.

References

[1] Ryad Belhakem. *Market Making in Limit Order Books*. 2024.

[2] John A. Bollinger. *Bollinger on Bollinger Bands*. Financial News Network, 2001.

[3] Dimitri Chemla, Ethan Goldberg, and Jonathan Issan. *Cryptocurrency Statistical Analysis for Dynamic Market Making Strategy*. 2023.

[4] Dimitri Chemla and Jonathan Issan. *OrderBook Pressure Market Microstructure Prediction*. 2023.

[5] Xinran Chen. "Stock Price Prediction Using Machine Learning Strategies". In: *ResearchGate* (2022).

[6] Ziad Francis. "Grid Trading with Python". In: *Medium* (2024).

[7] Peter I. Frazier. "A Tutorial on Bayesian Optimization". In: *arXiv:1807.02811v1* (2018).

[8] Ferat Sahin Girish Chandrashekar. "A survey on feature selection methods". In: *computers and electrical engineering* (2013).

[9] Alexandros Iosifidis, Dat Thanh Tran, and Juho Kanniainen. "How informative is the Order Book Beyond the Best Levels? A Machine Learning Perspective". In: *ResearchGate* (2022).

[10] Michael Isichenko. *Quantitative Portfolio Management: The Art and Science of Statistical Arbitrage*. Wiley Finance Series, 2021.

[11] Ishan Kapur, Dushyant Panchal, and Sajag Prakash. *Bitcoin Prediction Using Machine Learning*. 2022.

[12] Chuck LeBeau. *Computer Analysis of the Futures Market*. Hardcover, 1992.

[13] Xiaodong Li et al. "An intelligent market making strategy in algorithmic trading". In: *ResearchGate* (2014).

[14] Satoshi Nakamoto. *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2022.

[15] Ved Prakash. "Reasons Why Machine Learning Fails in Stock Prediction". In: *Medium* (2024).

[16] Daniel Rodriguez. *Backtrader: A Backtesting Framework*. 2006.

[17] Manthan Thakker and Bharat Vaidhyanathan. *Bitcoin Stock Price Prediction*. 2022.