Master's Thesis

Backtesting of Algorithmic Cryptocurrency Trading Strategies

Jan Frederic Spörer

First supervisor: Prof. Dr. Philipp Sandner

Second supervisor: Vahe Andonians

Submitted by: Jan Frederic Spörer[1]

Frankfurt, April 28, 2020

Abstract

This thesis presents a tool for backtesting algorithmic trading strategies for cryptocurrencies. The tool, called `quantbacktest`, provides a convenient way to automatically run comparisons of multi-dimensional parameter spaces for algorithmic trading strategies. The tool supports any algorithmic strategy to be simulated and any parameter spaces to be tested and optimized with minimal adjustments. Also, arbitrary trading frequencies can be tested, from intraday to long-term strategies. Many standard return metrics, risk metrics, and robustness test functionalities come out-of-the-box in CSV format and as diagrams. Users can provide signals and price data via CSV or Excel files. Signal processing does not require a technical (code-level) understanding of the backtesting tool on behalf of the user.

Keywords: Distributed Ledger Technology, Blockchain, Cryptocurrency, Algorithmic Trading, Backtesting
JEL classification: G12, G17

[1]Master in Applied Data Science student at Frankfurt School of Finance & Management | Address: Hermann-Löns-Straße 5, 57250 Netphen, Germany | Student ID: 8405672 | Email: jan@spoerer.me | Cell number: +49 171 5395666 | linkedin.com/in/janspoerer

1 Acknowledgment

First, I would like to thank my first thesis advisor Prof. Dr. Philipp Sandner[2], the head of the Frankfurt School Blockchain Center. Naturally, he gave me domain-specific guidance; but even more importantly, he connected me with blockchain practitioners that used the software presented here. The momentum that these projects caused was vital for me to stay focused and develop `quantbacktest` quickly.

Second, I am grateful to Vahe Andonians[3], Senior Lecturer at Frankfurt School and an expert on software development and AI, for agreeing to be the second supervisor for this thesis. He gave crucial technical guidance concerning time complexity optimizations of `quantbacktest` and helped me to make critical design decisions. He drew on his professional experience in algorithmic trading for a hedge fund.

Third, I would like to thank Jian Guo[4], a fellow student of mine from the Master in Applied Data Science at Frankfurt School, for his practical contributions to the source code. He supported me with urgently needed analyses for Immutable Insight GmbH. Also, he set up the time complexity monitoring for this software and presented it at a code review event for feedback from other software developers. In addition to that, he implemented various financial metrics, notably the maximum drawdown function, capitalizing on his background in the hedge fund industry.

Fourth, I express my gratitude to Immutable Insight GmbH[5], a startup in the field of blockchain technology, and its founders Katharina Gehra[6] and Dr. Volker Winterer[7]. They assigned me the task of performing backtests and hyperparameter scans on one of their proprietary trading algorithms that will become part of the Blockchainfonds initiative. I am thankful for the trust and commitment that they put into this project already at the very beginning of my research.

[2] frankfurt-school.de/en/home/research/staff/Philipp-Sandner, linkedin.com/in/philippsandner.
[3] andonians.com, frankfurt-school.de/en/home/research/staff/Vahe-Andonians.
[4] linkedin.com/in/jian-guo-57390981.
[5] blockchainfonds.com.
[6] linkedin.com/in/katharinagehra.
[7] linkedin.com/in/volker-henning-winterer-17b93ab.

Lastly, I would like to thank everyone who was involved with proofreading and technical advice: Leon Berghoff[8], Tobias Burggraf[9], Frederic Herold[10], Nicholas Herold[11], and Sascha Wildgrube[12].

[8] linkedin.com/in/leon-berghoff-753b52128.

[9] linkedin.com/in/tobias-burggraf-966001138.

[10] linkedin.com/in/frederic-herold-182034148.

[11] linkedin.com/in/nicholas-h-7518aa128.

[12] wildgrube.com, linkedin.com/in/sascha-wildgrube-4135b64.

2 List of Figures

3 List of Code Snippets

4 Contents

5 Introduction

5.1 Motivation

5.1.1 Practical Motivation and Relevance. As the cryptocurrency market is still young, investors hope to find opportunities for extraordinarily high risk-adjusted returns by exploiting mispricings. While a high number of retail investors characterizes the market, cryptocurrencies already have high enough market capitalizations to be attractive to many investors. The market has a relatively low share of sophisticated (institutional) investors and is in an early stage in terms of academic research. It is considered a fact that as the academic coverage for asset pricing improves, observed market inefficiencies vanish (McLean & Pontiff, 2016). Price movement patterns and influence factors on cryptocurrencies are not yet sufficiently understood, and the adoption of elaborate trading strategies is lower than in traditional fields of quantitative trading. This thesis will serve to ease the process of investigating these potential opportunities in a statistically sound way. Built-in test sample capabilities mitigate overfitting risks.

5.1.2 Academic Motivation and Relevance. As previously mentioned, cryptocurrencies are still a lesser covered research field than other asset classes. `quantbacktest` provides a way for researchers to test established theories from traditional finance and see if they apply to the cryptocurrency space. Of course, researchers can also develop entirely novel concepts for cryptocurrencies that are unrelated to anything known in more traditional financial domains. Using this tool, researchers can focus their efforts on formulating, refining, and iterating on interesting research hypotheses and can minimize the time spent on data collection, cleaning, and processing. Besides, the modular and transparent nature of this tool makes its results transparent and easy-to-verify for outsiders. Relying on tested software can increase trust in the researchers' results.

5.1.3 Comparison to Today's Technical Solutions.

5.1.3.1 Introduction to the Current Gap Among Technical Solutions

Investors today have a wide range of options to perform backtests for their strategies. The problem is that most realistic simulation tools only work with more traditional asset classes that are available as tradeable securities on common security exchanges. Adding customized price data can be cumbersome or impossible, especially for cryptocurrencies. To overcome this problem, it is also not always an option for investors and researchers to set up backtesting tools from scratch as this is time-consuming and requires deep financial and technical expertise. Sophisticated backtesting using an open-source programming language can be technically challenging, and simple software solutions, such as Excel, can be too inflexible and inefficient. In the following, I briefly introduce some current alternatives for backtesting and why I think that the tool presented here is better suited for some specific use cases. I shortlisted[13] the number of discussed backtesting tools; I only present the most promising ones here.

5.1.3.2 Excel Excel is a good tool for managing tabular data. In this case, however, the price data is three-dimensional: It has two identifying properties (`datetime` and `identifier`) and one value (`price`). Multi-dimensional data is generally hard to manage in Excel directly as Excel consists of tables; tables are two-dimensional. Excel's `index-match()` and `vlookup()` functions are also able to handle multi-criteria queries, but they are inefficient and likely to crash. Using Excel's native language VBA and its `array` data type would be a possible solution to this, but then the advantages that Excel has (i.e., a user-friendly interface that most people in finance know how to use) would vanish. In summary, Excel is not a feasible solution for this problem, and Excel VBA is a downgrade in comparison to Python, which offers more functionality for handling multi-dimensional data than VBA does.

[13] Some backtesting libraries that are not reviewed in detail here are: `PyAlgoTrade` (gbeced.github.io/pyalgotrade/docs/v0.17/html/), `pybacktest` github.com/ematvey/pybacktest, `bt` github.com/pmorissette/bt, `backtrader` github.com/mementum/backtrader, `ProfitPy` code.google.com/archive/p/profitpy/, `pinkfish` github.com/fja05680/pinkfish, `QSTrader` github.com/mhallsmoore/qstrader, `pysystemtrade` github.com/robcarver17/pysystemtrade, `QTPyLib` https://github.com/ranaroussi/qtpylib, `Prophet` github.com/Emsu/prophet, `Backtesting` pypi.org/project/Backtesting/.

5.1.3.3 Brokerage/OTC-related Software Many large trading/brokerage platforms offer client software or online technical analysis tools with built-in backtesting capabilities. Some of them come with proprietary programming languages[14] and *application programming interfaces* (APIs). Tradesignal and Oanda are examples of trading platforms that also offer access to such technical tools. The problem with those tools is that they serve to ease the use of their respective trading platforms. These trading platforms, however, are designed for foreign exchange trading or trading of other traditional asset classes such as stocks, bonds, commodities, futures, and derivatives. Therefore, those platforms have no incentive to facilitate the backtesting of products that are not available on their platforms; and they do not provide much help for cryptocurrency investors. To the best of my knowledge, I am not aware of software solutions offered by cryptocurrency exchanges.

5.1.3.4 Quantopian and Zipline Quantopian is one of the first platforms that come to mind when thinking about state-of-the-art backtesting of algorithmic trading strategies. It is a hosted backtesting and deployment platform with a large proportion of closed-source codebase. The community consists of individuals that compete for the best out-of-sample portfolio compositions[15]; Quantopian awards real funding to the best strategies. Zipline is one open-source library that Quantopian uses for backtesting. I have considered Quantopian when deciding about a technical platform and thought about ways to achieve the goals of this study within the technical framework that Quantopian provides.

I have decided against using Quantopian for the following reasons: Quantopian runs backtests on its servers. As a consequence, they offer a relatively low-effort, reliable, sufficiently documented, and actively maintained platform for backtesting and deploying quantitative trading strategies. Similar to other proprietary solutions, one issue is that Quantopian's asset universe consists of U.S. stocks that are tradeable on the platform. So cryptocurrency data is not yet available on Quantopian, at least not natively. There are workarounds to get cryptocurrency data into

[14] `Equilla` is Tradesignal's language.

[15] See the website for detailed algorithm requirements of Quantopian: www.quantopian.com/contest.

Quantopian[16], but I do not expect Quantopian to fully integrate cryptocurrencies into their platform as they are known for only using a tightly limited asset universe.

5.1.3.5 Crypto Wizards Crypto Wizards[17] aim to solve all of the problems mentioned above. So far, the focus of the project is promising for the use case presented. At the time of writing this thesis, the project is still in an early phase, with the project's start dating only to 2019. When the preparatory work for the thesis started, I did not even know that this project existed. In the explanation videos of Crypto Wizards, the developer hints that using the project is free now, but he implied that it might not be anymore in the future. What also makes us concerned is that Crypto Wizards' website is still in a nascent stage and contains suspicious customer reviews, raising some doubts about the project's integrity[18]. These doubts about the project's integrity, combined with the for-profit nature of the project, make this tool unacceptable for most users. Time will tell if the project develops into something worth considering for the use case presented here.

5.1.3.6 QuantConnect QuantConnect's cryptocurrency module is currently in its alpha phase. QuantConnect is a backtesting and order routing platform that already supports traditional asset classes. The business model of QuantConnect is similar to that of Quantopian. Quants can allow others to mirror their strategies and receive a profit share for this.[19]

5.1.3.7 Python The Python libraries `pandas`, `NumPy`, and `TA-Lib` are commonly used in research. They are particularly good for cash-neutral strategies that are tensor-based. One can perform tensor-based calculations with moderate effort and

[16] See quantopian.com/posts/cryptocurrency-data, quantopian.com/posts/crypto-currencies-backtest, quantopian.com/posts/how-to-trade-cryptocurrencies-with-zipline.

[17] The video gives a short introduction to the features of the Crypto Wizards backtesting tool youtube.com/watch?v=shvRlciJlZ0; this video goes more into depth youtube.com/watch?v=TkbwdPq3uWQ; this is their website: cryptowizards.net/.

[18] Here is a printout of the website cryptowizards.net/ from February 21, 2020: drive.google.com/open?id=19QeMAbAiMCkrFy3LVc3e3eqTxM0Y9DVn.

[19] See quantconnect.com/docs/data-library/crypto and algo-trading.capitalmarketsciooutlook.com for an initial overview of QuantConnect.

coding knowledge. As pointed out in sections 5.2 *Scope and Approach* (page 6) and 10.3 *Implications for Research* (page 45), tensor-based calculations as used by Jegadeesh and Titman, 2001 will probably continue to dominate research in financial markets and continue to exhibit unique advantages over the strictly realistic simulation approach that is used by `quantbacktest`. Many researchers will stick with the mentioned Python libraries instead of using `quantbacktest`. Practitioners and researchers with a more applied focus, however, will most likely not want to make the considerable effort that is involved in simulating trades by just using common mathematical Python libraries. I can tell from my work on this study and professional experience that it is not a trivial task to develop realistic backtesting software. For use cases that require a close resemblance of reality (e.g., cash balance modeling), I expect many people to prefer using the tool presented here instead of starting from zero. Starting from zero will likely compromise quality, budget, and time.

5.1.3.8 Open-source Programming Languages Other Than Python

Other popular, freely available programming languages such as R also allow for complex calculations and are commonly used in research and practice. However, they come with the same issues that were previously presented when discussing basic Python libraries: it is too much work to build a backtesting tool from scratch with only general-purpose libraries. In addition to these problems, there is another issue: Python has grown to be the most used programming language in data science. Python is more popular among developers as a primary programming language, not R or other languages[20]. Therefore, a Python-based module is desirable over other languages.

[20] See spectrum.ieee.org/computing/software/the-top-programming-languages-2019 and tiobe.com/tiobe-index/ for popularity rankings of programming languages.

5.2 Scope and Approach

This thesis aims at making backtests of cryptocurrency trading strategies easily accessible. Unlike existing approaches, `quantbacktest` can:

1. flexibly and realistically reflect market impact/slippage, bid-ask spread, and other transaction costs;
2. process flexible data inputs even from the researchers' self-sourced data;
3. take realistic constraints into account;
4. provide, in addition to financial metrics, insights into the internal calculation logic of the backtests to facilitate strategy optimizations based on individual trades;
5. provide users with commonly-known input and output formats that can be inspected quickly (CSV and Excel).

The advantages of `quantbacktest` are especially profound for cash strategies and non-perfect cash-neutral strategies. Simple tensor (matrix) operations are sufficient to simulate cash-neutral strategies. Any data science-ready programming language can perform such calculations. In Python, the `NumPy` package is capable of performing tensor operations. Tensor operations pose advantages in terms of speed and complexity over the backtesting tool presented here. As a result, this tool may not be the first choice for researchers that are looking into cash-neutral strategies. Also, some absolute return hedge funds may prefer performing initial backtests with tensors as tensors are less computationally expensive.

The objective is to create a reliable and easy-to-use tool that speeds up the process of backtesting algorithmic trading strategies for cryptocurrencies. The main contribution of this thesis is not its empirical results – the focus is on the software that produced those results. Ideally, this thesis paves the way for even more fruitful future research on cryptocurrency markets.

`quantbacktest` aims to simulate the processing of trading signals (buy and sell) just like a real order execution system would. It takes into account feasibility constraints such as cash and margin requirements. To a limited extent, it also considers risk constraints. Furthermore, it computes risk and return metrics out-of-the-box. In addition to these basic requirements, it eases the workflow of researchers and practitioners alike by allowing the user to test several settings (i.e., strategy settings and risk constraints) in one run.

5.3 Remarks on the Format and Notation

As this thesis contains many technical terms, I would like to introduce the notation that I will use throughout the text. A minimum of formatting conventions makes it easier for the reader to distinguish purely technical terms from conceptual terms.

First, Python objects and file paths are formatted in `monospace font`, as this is the font that closely resembles the fonts of development environments (text editors). Many people associate code and software with this type of font. For example, the table that contains the price data is denoted as `df_price_data`.

Second, any callable Python function has a suffix consisting of one left and one right bracket, like so: `name_of_the_function()`. The usage of open-close-bracket suffixes is a standard convention in software documentations. For example, the order execution function is denoted as `execute_order()`.

Third, internal references to other sections are written in *italics* and contain the section number. For example, a reference to the data section looks like this: "Please refer to section 7 *Data*".

All uncommon terms are introduced according to APA style rules. For example, the abbreviation "NVT ratio" is introduced as: "*network value to transactions* ratio (NVT ratio)". After introducing a term once, the normal font will be used. Further reference can be found in *Publication manual of the American Psychological*

Association, 2001, p. 100.

6 Theory and Literature Review

As the number of high-quality academic research articles in the field of cryptocurrency trading and investing is limited, I am also making use of inferences from other fields in finance, especially from the domain of stock markets. There are overlaps between stock markets and cryptocurrency markets. Some research from stock markets can be used analogously in cryptocurrency markets, other research can be repeated in cryptocurrency markets with similar methods, and some stock market research does not have any methodological overlap with cryptocurrency markets. Figure 1 shows the differences between equity market and cryptocurrency market research.

Many risk factors from traditional finance can be studied in cryptocurrency markets as well. For that reason, I start by reviewing the literature from traditional finance and then summarize which of the traditional patterns also hold in cryptocurrency markets.

The early work of Markowitz, 1952 and Lintner, 1965a, 1965b; Mossin, 1966 is concerned with the role of variance and market risk in financial asset markets. The arbitrage pricing theory by Ross, 1976 is a controversial generalization of established methods; it opened the door for more flexible approaches to asset pricing and more accommodating factor models beyond market risk considerations. Asness et al., 2018; Banz, 1981; Fama and French, 1992, 1993, 1996 study the size factor (based on market capitalization); Carhart, 1997; Jegadeesh and Titman, 1993 study the momentum factor; Ang et al., 2009 study the volatility factor. Sentiment, albeit being a broad concept and not a strictly defined risk factor, also plays a vital role in equity markets, as Jiang et al., 2019 exemplify.

The equity market risk factors presented in the previous paragraph were already covered in cryptocurrency markets, as Sovbetov, 2018, Shen et al., 2019, and Li and Yi, 2019 summarize. Despite the short time of operation of cryptocurrency

markets, there is rich coverage for each of these individual factors.

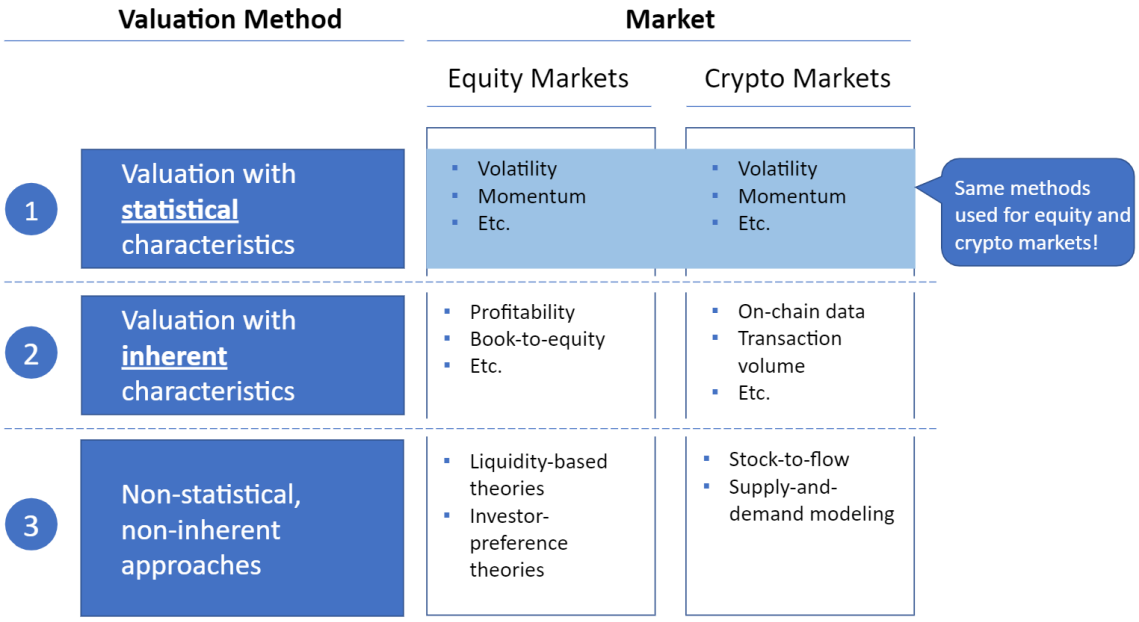

Figure 1. Overview of research coverage of different asset pricing factors for equity markets and cryptocurrency markets. The overview aims to compare research coverage in equity markets and cryptocurrency markets in the three different research categories *statistical characteristics*, *inherent characteristics*, and *non-statistical, non-inherent approaches*. Own visualization created with Google Slides.

Risk factors that stem from the inherent properties of the securitized/tokenized asset (such as investment and profitability, Fama and French, 2015, 2017) are harder to establish in cryptocurrency asset pricing research as direct analogies are more challenging to draw. Those factors are called fundamental accounting factors. Profitability was presented by Basu, 1977 and later adjusted in other models such as the Fama-French five-factor model to reflect not current, but reported future profitability. As stock markets are concerned with trading equity capital of cash-generating businesses, the profitability factor is an inherent factor of the underlying asset. The profitability factor, along with the investment factor, cannot be observed in cryptocurrency markets as it simply does not exist there. Other fundamental factors face researchers and investors with similar applicability problems for cryptocurrency markets. Asness et al., 2013; Fama and French, 1998 study the value factor, which is based on companies' accounting information and, therefore, not applicable to cryptocurrencies. Bhandari, 1988 studies the debt factor, which is again part of financial accounting. Direct analyses of the blockchain networks are a probable path forward to replace factors that are present in the stock market, but

not in the cryptocurrency market.

Hubrich, 2017 draws an analogy between the book value of equity and the on-chain transaction value and thereby introduces a cryptocurrency-specific value factor. Similarly, Woo, 2017 utilizes on-chain transaction volume by creating a growth factor that assumes that a cryptocurrency's value arises from the transaction activity on the network. The growth factor is thereby expressed as the *Network Value to Transactions Ratio* (NVT) and is coined "a PE ratio for Bitcoin" (p. 1) by the author. The NVT is the fraction of market capitalization over on-chain transactions. The NVT calculation can be smoothed by using a moving average of the last n days, like so:

$$\text{NVT} = \frac{\text{Market capitalization (network value)}}{\text{Average daily transaction value over the last } n \text{ days}}$$

When thinking of cryptocurrencies' primary function as being a medium of exchange and as being a payment processing system, NVT-based valuation makes intuitive sense. Berghoff, 2020 integrates the NVT into a factor model with size, NVT, and mean reversion factors. The Bachelor's thesis of Fürst, 2019 provides a method for extracting on-chain information that can be interesting for cryptocurrency-inherent factors like the NVT. Minability (the ability to mine cryptocurrencies, for example, through utilizing computing power to solve proof-of-work challenges) is another potentially interesting factor that Kakushadze, 2018 introduce into a cryptocurrency factor model. Analogous to equity markets (Moskowitz & Grinblatt, 1999), cryptocurrency markets may exhibit industry-specific differences that can be captured via cryptocurrency classifications such as those provided by ITSA e.V.

Cryptocurrencies have, similar to equity securities, occasional and regular events that lead to price distortions. In the case of stocks, stock splits and dividends often cause massive price movements that should be considered by backtesting tools. Research by Desai and Jain, 1997 indicates that events like stock splits may cause return anomalies. In the cryptocurrency space, forks may have a related role.

Equity markets are subject to regular portfolio rebalancings, especially by institutional investors. Therefore, they show some special characteristics that may

also be present in cryptocurrency markets on the grounds of processes that are comparable with rebalancing intervals. In equity markets, for example, Thaler, 1987 observes abnormal seasonal trading activity in January. Thaler observes monthly returns in January of 3.5% versus 0.5% in other months[21]. One of the many possible explanations for this considerably large market anomaly is taxation. Taxes also play a role for cryptocurrency investors, making seasonal excess returns a promising field for cryptocurrency research.

Methodologies for finding fair values for cryptocurrencies are less well understood than the same methodologies in traditional asset classes. In traditional asset classes, discounted future cash flows are usually the underlying driver of asset prices, making future cash flows and their riskiness the two key variables of consideration. Risk factors are often a reflection of risk and return. A similarly easy answer does not exist in the cryptocurrency domain, making direct modeling of supply and demand of cryptocurrencies (as opposed to price modeling) an interesting field of study. The supply of Bitcoin, for example, can be modeled by using the fraction of its cost to mine over the block reward. Hayes, 2015 propose three supply-based metrics for predicting the Bitcoin price and achieved an R^2 of 0.84. The idea behind Hayes' model is that the Bitcoin price should have a fundamental connection to the cost of electricity per unit of physical work (W) performed, the efficiency of mining as measured by W per unit of mining effort, the market price of Bitcoin (as measured by USD per Bitcoin), and the difficulty of mining (as measured by the complexity of the hashing problem). For the demand of Bitcoin, Athey et al., 2016 propose an approach to measure user adaption and provide a link between user adaption and Bitcoin price.

One completely different approach to valuing cryptocurrencies is treating them like commodities. In the case of Bitcoin, one can do so by comparing it with gold. McGlone, 2020 argues that Bitcoin stayed relatively stable during the equity sell-off in February and March 2020, with the cryptocurrency exhibiting gold-like charac-

[21] One needs to keep in mind that Thaler's article is not up-to-date anymore and the mentioned monthly returns probably changed.

teristics. Some investors try to determine gold's fair value using the stock-to-flow ratio, which is a metric that expresses the fraction of available stock/inventory of gold in the world over the yearly amount of mined gold:

$$\text{Annual stock-to-flow ratio} = \frac{\text{Inventory (stock) of already mined (and not in industrial use) gold}}{\text{Annual volume (flow) of mined gold}}$$

The stock-to-flow ratio is useful for commodities that have little industrial applications. Therefore, the stock-to-flow ratio should be used for assets whose primary economic function is a store of value, such as gold. The prices of other rare metals such as silver are too heavily influenced by industrial demand to be modeled reliably with the stock-to-flow ratio. One can argue that the analogy to gold in the cryptocurrency space is Bitcoin: Bitcoin is not the most efficient means of payment, but Bitcoin is inherently secure, making it useful as a store of value. The stock-to-flow ratio became so popular in the Bitcoin community that there are already several websites that offer live monitoring of Bitcoin's stock-to-flow ratio[22].

To conclude, cryptocurrency pricing factors based on price data (momentum, volatility, market capitalization) are already well-studied, but there is a gap in studying inherent cryptocurrency characteristics and the relation of those to cryptocurrency returns. The key difference between cryptocurrencies and stocks is that cryptocurrencies do not securitize a cash-generating right, such as company equity. Consequently, I have ventured into fields that resemble cryptocurrency markets more closely by seeing a cryptocurrency as a payment processing platform (NVT ratio) or as a value-storing commodity (stock-to-flow). Another promising field is direct supply and demand modeling as a proxy for price generation.

7 Data

7.1 Trade-offs Between Using Different Price Data Frequencies

The choice for the optimal price data frequency is mostly dependent on the requirements of the strategies that one wants to backtest. Day traders have different

[22]These websites show the historical and the current stock-to-flow ratios for Bitcoin: digitalik.net/btc/ and lookintobitcoin.com/charts/stock-to-flow-model/.

requirements than longer-term investors. For people that are undecided about the frequency to use, I list some trade-offs between using higher and lower frequency data here.

Higher frequencies are associated with more storage requirements, higher memory usage, more computational load, and are often harder to acquire. Lower frequencies may necessitate longer time horizons to be tested for reliable backtest results.

For this thesis, I think that a relatively low frequency (daily) is the right choice. Strategies that work with daily frequencies can be configured to be easily understandable while still being economically sensible. Also, many research articles use daily data, and experience shows that daily rebalancings are often the maximum sensible rebalancing frequency that does not incur prohibitive trading costs. `quantbacktest` is not designed for arbitrage trading and high-frequency approaches, as discussed in section 10.4.3 *Arbitrage Trading*.

`quantbacktest` can also handle any other frequency, but I will consistently use daily data throughout this study.

7.2 ITSA TOKENBASE

In all test runs, the *ITSA TOKENBASE* price data was used. *ITSA e.V.*[23] (ITSA) is a non-profit organization that provides the *distributed-ledger technology* (DLT) community with standardized identifiers and with a unified classification framework for cryptocurrencies. ITSA gathered their data from CoinGecko[24]. The columns from the ITSA TOKENBASE CSV file need to be renamed as `quantbacktest` expects specific column names for the price data input: `token_date` is renamed to `datetime`, `token_itin` is renamed to `id`, and `token_price` is renamed to `price`. The price column contains mid prices; bid and ask prices are not listed separately.

[23] itsa.global.

[24] coingecko.com.

8 Method (Trading Strategy and Backtesting)

8.1 Essential Objects and Terms

8.1.1 Scenario. I define a *scenario* to simply be one simulation of a trading strategy with one particular set of hyperparameters and settings. The concept of running multiple scenarios at once is described in the following paragraph.

8.1.2 Batch Processing. When using the term *batch processing*, I mean that the user tests multiple backtests with different scenarios in a single execution of the program. Scenarios can differ in many possible ways; notably, they can assume varying minimum cash constraints, time intervals, rebalancing frequencies, or variations in other parameters. The advantage of this batch processing functionality is that the user gets summarized results for all scenarios in a single `df_performance_metrics` table (explained in section 8.1.5 `df_performance_metrics`). As a visualization of this summary, the user gets out-of-the-box heatmaps for time-varying robustness of the results (as an example, figure 9 is given in section 12.1 *Additional Visualizations*). Similarly, the user gets a heatmap that shows the strategy's robustness to variations in hyperparameters (please refer to figure 10 in section 12.1 *Additional Visualizations* for an example).

Thus, batch processing allows the user to configure a complete set of scenarios. The user does not need to keep track of which scenarios were already tested and receives immediate visual robustness results. The risk and return metrics are conveniently stored in a single CSV file along with the respective input parameters. Batch processing helps to minimize errors and saves time.

8.1.3 `df_prices`. `df_prices` is a `Python pandas DataFrame` object and consists of two identifiers and a `price` column. Further columns may be added to the loaded CSV file or at runtime if further information is useful for the strategy. For example, one could create a moving average column at runtime to build a momentum strategy, as shown in section 9.2 *Momentum – Simple Moving Average*.

The input price data can be in CSV or Excel format. The table should at minimum have an asset identifier (usually a `string`), a date or datetime (usually a `string` or an `integer` that represents a `Unix epoch`), and a price (usually a `float`). The asset identifiers are assumed to appear in a column called `id`, the timestamps are assumed to be in a column called `datetime`, and the prices are assumed to be in a column called `price`.

8.1.4 `df_trading_journal`. `df_trading_journal` stores all transactions (or trades) that occur in each backtested scenario. `df_trading_journal` is saved as a CSV at the end of each scenario run. When batch processing is used, multiple CSV files with `df_trading_journal` will be saved to disk after the program finished.

`df_trading_journal` not only contains transactions, but also records account information. The fields are:

```
'datetime',
'Cash',
'Cash before',
'Asset',
'Buy or sell',
'Number bought',
'Price (quote without any fees)',
'Value bought',
'Portfolio value',
'Dict of assets in portfolio',
'Absolute fees (as absolute)',
'Current equity margin',
'Exposure (in currency)',
'Exposure (number)',
'Gross exposure',
'Interest paid',
'Money spent',
'Relative fees (as absolute)',
'Relative fees (as relative)',
'Strategy ID',
'Total exposure',
'Total fees (as absolute)',
'Total fees (as relative)'
```

Code Snippet 1: Fields of the `pandas DataFrame df_trading_journal`.

8.1.5 `df_performance_metrics`. The performance metrics are listed in the code block below. Please refer to Sharpe, 1964, 1966, 1994 as references for the

Sharpe ratio. Some of these metrics are used in section 9 *Results*, starting at page 38 to evaluate the exemplary strategies. The metrics are:

```
'Strategy metadata ---->',
'Strategy ID',
'Strategy label',
'Trading info ---->',
'Begin time of tested interval',
'End time of tested interval',
'Duration of the tested interval',
'Duration of the tested interval (in days)',
'Average cash',
'Average ticket size',
'Number of trades',
'Number of unique assets traded',
'Total transaction cost',
'Return metrics ---->',
'USD annualized ROI (from first to last trade)',
'Cryptocurrency annualized ROI delta (from first to last trade)',
'Ending benchmark value (first to last trade)',
'Initial budget',
'Ending portfolio value',
'Risk metrics ---->',
'Holding period volatility',
'Annual volatility',
'Monthly volatility',
'Weekly volatility',
'Beta relative to benchmark',
'Maximum drawdown',
'Maximum drawdown duration',
'Maximum drawdown peak date',
'Maximum drawdown trough date',
'Other metrics ---->',
'Alpha',
'Sharpe ratio (holding period)',
'Sharpe ratio (yearly)',
'Beginning benchmark value (first to last trade)',
'Other info ---->',
'Start time',
'End time',
'Parameter 1',
'Parameter 2',
'Comments',
'Benchmark return metrics ---->',
'Benchmark USD annualized ROI (from first to last trade)',
'Benchmark cryptocurrency annualized ROI delta (from first to last trade)',
'Benchmark ending benchmark value (first to last trade)',
'Benchmark initial budget',
'Benchmark ending portfolio value',
```

```
'Benchmark risk metrics ---->',
'Benchmark holding period volatility',
'Benchmark annual volatility',
'Benchmark monthly volatility',
'Benchmark weekly volatility',
'Benchmark Beta relative to benchmark',
'Benchmark maximum drawdown',
'Benchmark maximum drawdown duration',
'Benchmark maximum drawdown peak date',
'Benchmark maximum drawdown trough date',
'Benchmark other metrics ---->',
'Benchmark Sharpe ratio (holding period)',
'Benchmark Sharpe ratio (yearly)'
```

Code Snippet 2: Fields of the `pandas DataFrame df_results_metrics`.

8.2 Step-by-step Program Flow

8.2.1 General Program Flow. The backtest starts with a data loading process. Only CSV format and Excel format are allowed. The CSV file with price data will be loaded into memory as a `pandas DataFrame`, `df_price_data`.

Then, `df_price_data` is loaded into the primary backtesting function along with the user-defined constraints and parameters. The function then triggers the rest of the process and outputs the performance table as a `DataFrame`, `df_results_metrics` in memory and as a CSV file on disk. Old CSV output will not be overwritten as file names are automatically versioned.

The steps are described in more detail in the figures 2 and 3 below.

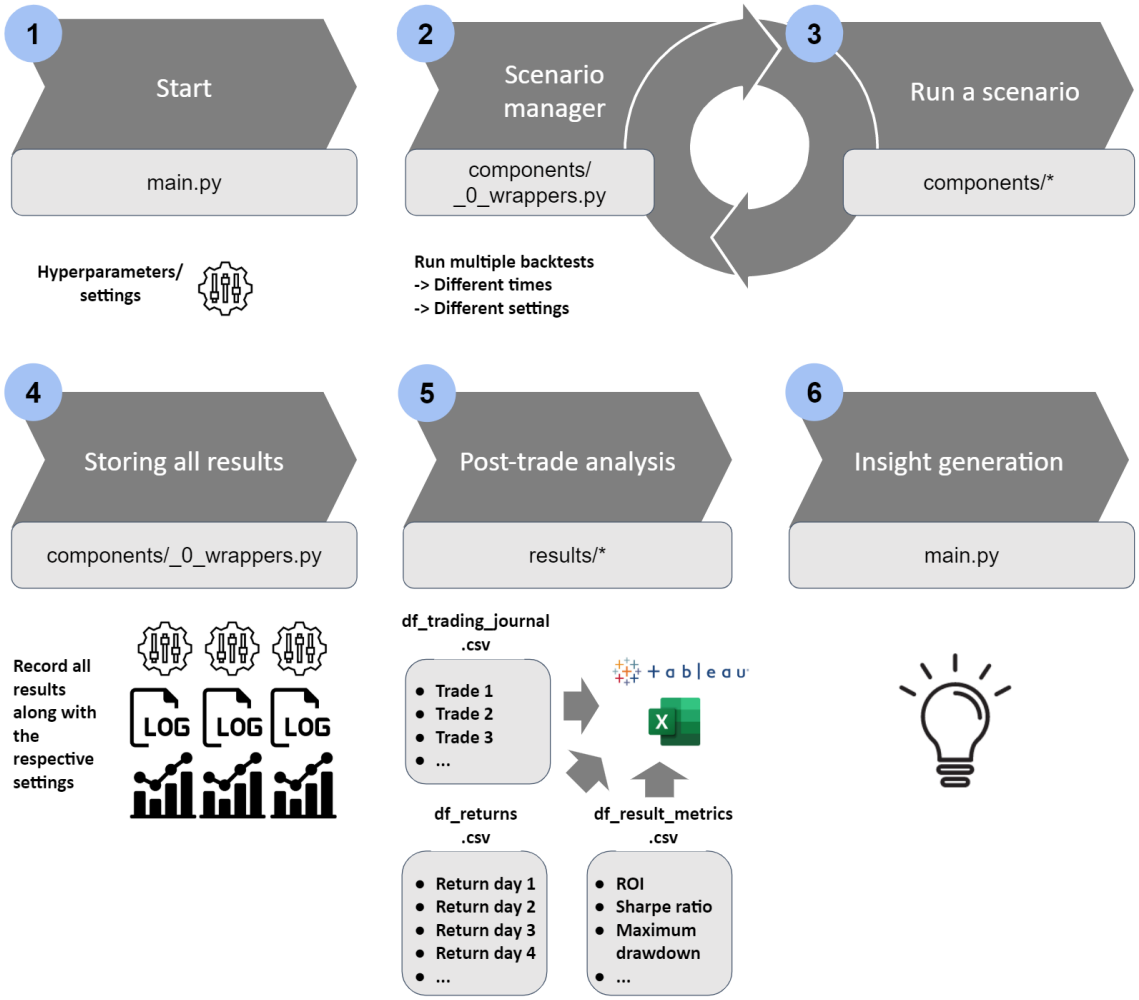

Figure 2. The high-level backtesting workflow in six steps. This overview is relevant to all users. First, the user determines which hyperparameters and settings to use (column 1). Then, the specified scenario(s) will be executed separately (column 2). The run of a single scenario (column 3) is explained in more detail in figure 3. All results are stored to disk in CSV format and in image format (column 4). The user can open the results and choose to use additional software such as Tableau and Excel for further analyses (column 5). The generated insights can be used to decide to take a strategy online (i.e., trade real money) or to change the strategy (column 6). Own visualization created with Google Slides.

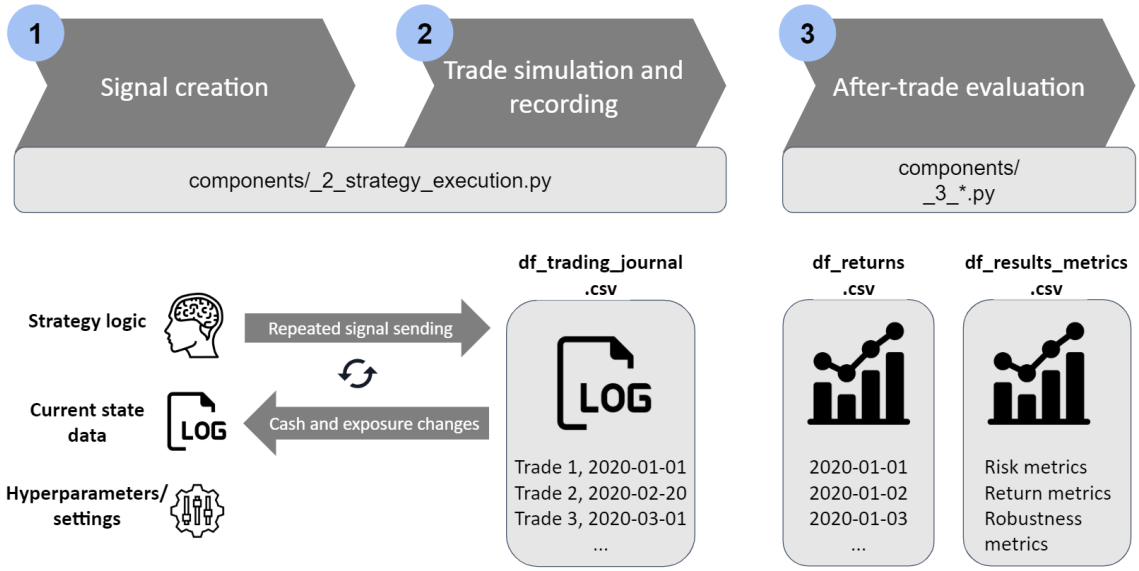

Figure 3. The low-level backtesting workflow in three steps. First, the user chooses strategy hyperparameters, such as optimization constraints and machine learning hyperparameters, and other settings, such as the number of rounding decimal places, warnings handling, and price data gap tolerance settings (see "Hyperparameters/settings" in column 1). Then, `quantbacktest` will run the scenario(s) specified by the user (repeating process between columns 1 and 2). Trades change the account balance and exposure states in `df_trading_journal`, and the trades create a feedback loop to the strategy. The strategy has access to the current portfolio state through `df_trading_journal`. For example, the current cash position is recorded there. The strategy may behave differently under certain circumstances (e.g., if certain exposure limits are already crossed, the strategy may decide to divest certain assets), making the feedback loop essential. After every trade has been recorded in `df_trading_journal`, it is converted to a standardized, fixed-step format. This format is called `df_returns` and is shown in column 3. It contains returns and portfolio values for fixed, user-defined intervals. An interval can be hourly, daily, weekly, or any arbitrary frequency. From this standardized format, common risk and return metrics can be calculated (`df_results_metrics`). The process depicted here is also part of figure 2, column 3 on page 18. The user can decide to run another backtest with adjusted settings after inspecting the results. The user should be aware of the risk of overfitting when running many scenarios and repeating this process. Own visualization created with Google Slides.

8.2.2 Trading Simulation, `execute_order()`.

8.2.2.1 General Order Simulation The order simulation logic is at the heart of `quantbacktest`. The function `execute_order()` handles this step. All strategies use the `execute_order()` function as an interface for simulated order routing. `execute_order()` may or may not execute the requests of a strategy, depending on the cash balance, exposure constraints, and other checks. It also performs a sanity check whether the request is consistent in itself by comparing the signal type (buy or sell) with the order amount (positive or negative). Orders can also be filled partially

if the settings allow for partial order filling. As `execute_order()` contains extensive logic and tests, it aims to mitigate and raise errors that occur in the strategy definition and the settings. The extensive logic within the order execution simulation aims to resemble real-world order execution systems as closely as possible and to reduce the required amount of logic that needs to be included in the strategy functions.

The `execute_order()` function takes the strategy signals as an input and returns the corresponding transaction details, the new state of the portfolio, and account details. These details are then added to `df_trading_journal` after the `execute_order()` function stops. The function is strictly called sequentially, never in parallel, and receives only a single signal at a time. The arguments of the function are:

```
boolean_buy,
index,
date,
crypto_key,
number_to_be_bought,
strategy_id,
df_prices,
df_trading_journal,
margin_loan_rate,
fees,
float_budget_in_usd,
price,
display_options,
constraints,
general_settings,
boolean_allow_partially_filled_orders
```

Code Snippet 3: Arguments of `execute_order()`, code from `components/_2_strategy_execution.py`.

One can see that this function takes many arguments. The function is thus further divided into `inner functions` that handle the most complex sub-tasks.

The function returns a single dictionary. This dictionary is appended to `df_trading_journal` and contains the same fields as `df_trading_journal`. For a detailed overview of what `execute_order()` returns, please refer to section 8.1.4

`df_trading_journal`. `df_trading_journal` is simply a table that lists all returns from `execute_order()`:

```
order
```

Code Snippet 4: Return of `execute_order()`, code from `components/_2_strategy_execution.py`.

8.2.2.2 Detailed Explanation of the Handling of Transaction Costs The most common types of transaction costs are brokerage and stock exchange commissions, spread, and slippage. They can all be simulated with `quantbacktest`.

Slippage[25] is the time- and the volume-induced difference between the initially observed price and the real execution price and is the result of high-volume trades, slow internet speeds on the client's side, delayed order routing by the broker, and other factors (Chan, 2009, p. 23). I define market impact to be one subset of slippage (volume-induced slippage). Slippage is taken care of by the tool; users specify slippage as a setting. There is no volume-dependent slippage model yet.

For simplicity, slippage also accounts for the bid-ask spread[26], so it is highly recommended to specify a slippage higher than 0%. The spread is the difference between the bid price and the ask price and reflects the loss that one would make when buying and immediately selling an asset at an individual exchange.

Commissions are fees that brokers and exchanges charge for their services. They may be flat (per order) or relative to the nominal amount traded, both of which options can be simulated with this tool.

Transaction costs can be minimized by trading on cheap (low commission) and liquid (low spread) exchanges, trading highly liquid cryptocurrencies and using smart order routing[27]. As a guide, one usually aims to keep orders smaller than 1%

[25] Slippage is defined in more detail in Frino and Oetomo, 2005 for the special case of futures markets.

[26] While slippage and bid-ask spread are different concepts, they are both commonly expressed in percent and are both based on nominal transaction volume in this backtesting. Therefore, the two concepts are captured under the same setting.

[27] Smart order routing distributes volume over several exchanges and over time in order to achieve the best possible average execution price.

of the total daily trading volume of any asset in equity trading (Chan, 2009, pp. 87–88).

8.2.3 How the Trading Book is Processed Into Return Series.

8.2.3.1 Calculate Returns (Rolling/Single Journal Pair), `calculate_returns_single()`

This function calculates the mark-to-market returns for one trade to the next at a given frequency.

```
previous_trading_journal_row,
current_trading_journal_row,
df_prices,
strategy_hyperparameters,
display_options,
general_settings,
constraints
```

Code Snippet 5: Arguments of `calculate_returns_single()`, code from `components/_3_performance_evaluation.py`.

The calculate returns function takes one adjacent pair of trading journal entries and calculates all returns (at a given frequency) that fall between these two trades. For example, suppose that `df_trading_journal` contains a trade on day one and another trade on day four. The `calculate_returns_single()` function will get the row from day one (row one) and the row from day four (row two) as an input. It will calculate the portfolio value on day one, day two, day three, and day four. It finally returns a list of dictionaries:

```
list_of_dict_returns
```

Code Snippet 6: Return of `calculate_returns_single()`, code from `components/_3_performance_evaluation.py`.

The `dictionaries` within this list not only contain returns, but also other helpful fields:

```
datetime
```

```
portfolio_value portfolio_return
relative_portfolio_return
dict_of_assets_in_portfolio
benchmark_portfolio_value
benchmark_portfolio_value_normalized
portfolio_value_normalized
benchmark_return
benchmark_relative_return
benchmark_dict_of_assets_in_portfolio
```

Code Snippet 7: Fields of the `dictionaries` from `calculate_returns_single()`.

The function returns a list that contains returns at a specified frequency for the timeframe between two given trades with the size of the list given by:

$$\text{Length of output list} = \frac{t_{\text{second row}} - t_{\text{first row}}}{\text{frequency}} - 1.$$

Returns in themselves are not a performance metric; they are just a means for making calculations of performance metrics possible. For completeness and as the daily returns are so central, the method of generating daily returns from the trading journal is outlined here along with the metrics. Expressing the calculation of the daily return calculation in purely mathematical terms is not possible.

As a first step, the method calculates the portfolio value for each day, given the cash holdings and asset holdings. If there is no portfolio information in the trading journal for a given day, it is assumed that there were no cash or asset changes since the last trade. This is not a problematic assumption. For example, if at day one, there were $100 and 1 Bitcoin in the portfolio and one wanted to calculate the portfolio value of day two without having a trading journal entry for that day, one would assume that the cash value is still $100[28] and that the number of Bitcoin is still 1. Bitcoin will be re-evaluated with current market prices on day two, and the result will be added to the cash value to yield the portfolio value for day two.

Here is the technical documentation of this function:

```
"""Returns a list of dicts with portfolio return data for a given frequency.

```

[28] Daily costs for margin positions (interest payments) will be included as soon as margin trading is supported by `quantbacktest`.

```
Dict fields: 'timestamp' (datetime.datetime), 'portfolio_value' (float),
             'return' (float), 'relative_return' (float),
             'dict_of_assets_in_portfolio' (dict with itins as keys and
             integer with the number of pieces held of this asset as as
             values)

The reasoning behind the return calculation and the frequency handling is
described using an exemplary time series. The frequency is minutes here, so
frequency_in_seconds=60. The first column represents the executions
(previous_trading_journal_row and current_trading_journal_row), the second
column represents the frequency.

Trades                      Frequency increments

-No trade-                  Minute 1
Trade 1                     Minute 2
-No trade-                  Minute 3
-No trade-                  Minute 4
Trade 2                     Minute 5

In the example above, the return calculation would work as follows: The
function would receive DataFrame rows for Trade 1 and Trade 2. It would run
the return calculation loop for Minute 2-3, Minute 3-4, and Minute 4-5.
Thus, the function would return a list of three dicts. The returns for
Minute 1-2 were already calculated in an earlier function call (Trade 0-1).
It is crucial that the datetime.datetime objects that are contained in
Trade 1 and Trade 2 have the same frequency as the frequency that is passed
as an argument. Otherwise, there can be missing entries in the aggregated
return DataFrame.

datetime.datetime objects assume 'there are exactly 3600*24 seconds in every
day'. https://docs.python.org/2/library/datetime.html#datetime-objects
"""
```

Code Snippet 8: Docstring of `calculate_returns_single()`, code from `components/_3_performance_evaluation.py`.

8.2.3.2 Calculate Returns (Batch), `calculate_returns_batch()`

Arguments:

```
df_trading_journal,
df_prices,
strategy_hyperparameters,
display_options,
general_settings,
```

```
6 constraints
```

Code Snippet 9: Arguments of `calculate_returns_batch()`, code from `components/_3_performance_evaluation.py`.

Return:

```
1 df_returns
```

Code Snippet 10: Return of `calculate_returns_batch()`, code from `components/_3_performance_evaluation.py`.

This function calculates the mark-to-market returns for a set of trades at a given frequency. It is simply a wrapper for the previously outlined `calculate_returns_single()` function. It iterates over all trades in one `df_trading_journal` and aggregates them in a table (`df_daily_returns`).

8.2.4 Financial Metrics.

8.2.4.1 Jensen's Alpha, `calculate_alpha()`

Arguments:

```
1 annualized_portfolio_return,
2 risk_free_rate,
3 beta_exposure,
4 annualized_market_return
```

Code Snippet 11: Arguments of `calculate_alpha()`, code from `components/_3_individual_metrics.py`.

Return:

```
1 alpha
```

Code Snippet 12: Return of `calculate_alpha()`, code from `components/_3_individual_metrics.py`.

Jensen, 1968 introduced alpha (α) as a risk-adjusted performance measure. Portfolios with a positive alpha manage to generate returns that do not root in

market risk. Large alphas are generally desirable. Jensen's alpha is defined as:

$$\text{Alpha } \alpha = r_i - (r_f + \beta_{i,M} \times (r_M - r_f))$$

8.2.4.2 Equity Beta, `calculate_beta()`

Arguments:

```
df_daily_returns,
df_daily_benchmark_returns
```

Code Snippet 13: Arguments of `calculate_beta()`, code from `components/_3_individual_metrics.py`.

Return:

```
beta
```

Code Snippet 14: Return of `calculate_beta()`, code from `components/_3_individual_metrics.py`.

β (beta) reflects the sensitivity of a portfolio to the systematic risk of the market. The market is defined here as the user-defined benchmark. If β is higher than one, the portfolio is expected to react overproportionately to market movements. If β is smaller than one, the portfolio is expected to react underproportionately to market movements. β can be smaller than zero. Assets and portfolios with betas smaller than zero can serve as a hedge against market risk. β is calculated as

$$\text{Equity beta } \beta = \frac{\text{Covariance between } r_i \text{ and } r_M}{\text{Variance of } r_M}$$

with r_i being the portfolio returns and r_M being the market returns.

8.2.4.3 Maximum Drawdown, `calculate_maximum_drawdown()`

Arguments:

```
df_daily_returns
```

Code Snippet 15: Arguments of `calculate_maximum_drawdown()`, code from `components/_3_individual_metrics.py`.

Return:

```
drawdown,
maximum_drawdown_duration.days,
peak_date,
trough_date
```

Code Snippet 16: Return of `calculate_maximum_drawdown()`, code from `components/_3_individual_metrics.py`.

The maximum drawdown is the worst-case scenario that an investor would have experienced when investing in the portfolio. It assumes the worst possible entry (buying) time and the worst possible exit (selling) time. The maximum drawdown duration is the duration (in days) between the worst entry time and the subsequent worst exit time.

The maximum drawdown does not necessarily take the highest portfolio value as an input for the high point. It may also use an earlier high point. The maximum drawdown represents the maximum relative decrease in portfolio value. If, for example, the portfolio would be at $100 at day 1, $50 at day 2, $200 at day 3, and $120 at day 4, the maximum drawdown would be 50%, the high date would be day 1, and the low date would be day 2. The maximum drawdown duration would be one day.

The maximum drawdown is a risk measure and can give investors an indication of what they should be ready to lose if they invest in an individual portfolio (or follow a specific strategy). Future losses can be even higher than the backtested maximum drawdown.

8.2.4.4 Annualized Return on Investment, `calculate_roi()`

Arguments:

```
df_trading_journal,
float_budget_in_usd,
timeframe_whole_or_first_to_last_trade,
df=None,
df_benchmark=None,
df_price_column_name='token_price_usd',
df_time_column_name='time',
df_benchmark_price_column_name='token_price_usd',
df_benchmark_time_column_name='time',
df_trading_journal_price_column_name='Portfolio value',
df_trading_journal_time_column_name='Date of execution'
```

Code Snippet 17: Arguments of `calculate_roi()`, code from `components/_3_individual_metrics.py`.

Return:

```
return_on_investment
```

Code Snippet 18: Return of `calculate_roi()`, code from `components/_3_individual_metrics.py`.

The annualized return on investment is a result of downscaled (from larger time intervals than one year) or upscaled (from time intervals that are lower than one year) total portfolio returns. The calculation is:

$$r_{\text{annualized}} = \left(\frac{K_n}{K_0}\right)^{\frac{1}{n}} - 1$$

with n being the number of years and K_i being the portfolio value in year i.

Interpolating short time intervals to one year in this way can lead to substantial bias because shorter time intervals usually exhibit significant return variance. Scaling larger time intervals down to one year, however, is usually not problematic.

8.2.4.5 Sharpe Ratio, `calculate_sharpe_ratio()`

Arguments:

```
df_daily_returns,
portfolio_roi_usd,
days=None,
risk_free_rate=None
```

Code Snippet 19: Arguments of `calculate_sharpe_ratio()`, code from `components/_3_individual_metrics.py`.

Return:

```
sharpe_ratio
```

Code Snippet 20: Return of `calculate_sharpe_ratio()`, code from `components/_3_individual_metrics.py`.

$$\text{Sharpe ratio} = \frac{\text{annualized return on investment} - \text{risk-free rate}}{\sigma \text{ annualized volatility}}$$

The risk-free rate should be deducted to make the numerator and the denominator fit, as originally intended by William Sharpe, 1966.[29]

8.2.4.6 Transaction Costs, `calculate_transaction_cost()`

Arguments:

```
df_trading_journal
```

Code Snippet 21: Arguments of `calculate_transaction_cost()`, code from `components/_3_performance_evaluation.py`.

[29] There is some controversy about whether to subtract the risk-free rate from the portfolio return. For that reason, both ways of calculating the Sharpe ratio are implemented here, and it is left to the user to decide which one to use. I recommend using the Sharpe ratio that includes the risk-free rate. Only excess returns should have an associated risk in the denominator of the formula. As a supporting example, if one omits the risk-free rate when calculating the Sharpe ratio, a risk-free asset would have a Sharpe ratio of infinity. Assuming a positive risk-free rate, omitting the risk-free rate would yield inflated results. Chan, 2009, pp. 43–44, one of the most renowned practitioners in algorithmic trading, argues against adjusting for the risk-free rate. He argues that one can earn "a credit interest close to the risk-free rate," giving an excess return of $R + r_f - r_f = R$. Arguing against this, and to the best of my knowledge, I do not think that it is realistic to assume that cash holdings can reliably be invested at the risk-free rate; there is not a single cryptocurrency exchange that offers risk-free interest rates on unused funds today.

Return:

```
total_transaction_cost
```

Code Snippet 22: Return of `calculate_transaction_cost()`, code from `components/_3_performance_evaluation.py`.

The sum of transaction costs is helpful when making trade-offs between transaction costs and rebalancing intervals. It can also help to contribute cost impact assessments for trading with different exchanges and brokers.

8.2.4.7 Volatility, `calculate_volatility()`

Arguments:

```
df_daily_returns,
time_adjustment_in_days=None
```

Code Snippet 23: Arguments of `calculate_volatility()`, code from `components/_3_individual_metrics.py`.

Return:

```
volatility
```

Code Snippet 24: Return of `calculate_volatility()`, code from `components/_3_individual_metrics.py`.

Investors usually have a preference for specific volatility (σ) observation periods. Long-term investors may be familiar with yearly volatility, while investors with shorter holding periods may be more familiar with daily and weekly volatility. Therefore, the backtesting allows users to adjust volatility measures to any time interval. Adjusted volatilities are also used in the Sharpe ratio calculation. The time-adjusted volatility (Chan, 2009, p. 44) is defined as

$$\text{Volatility } \sigma = \sigma_{\text{of all daily returns}} * \frac{\sqrt{\text{adjusted duration}}}{\sqrt{\text{original duration (trading period in days)}}}.$$

The assumption behind this formula is that returns are serially uncorrelated. This approach can be highly problematic as volatility may change significantly over time.

So if short timeframes are extrapolated to longer timeframes, the results could be inaccurate. In summary, historical volatility measures should be interpreted with caution when using them for risk analyses, especially if volatility is scaled from short to long timeframes.

8.3 Additional Remarks on the Usage of the Backtesting

8.3.1 Possible Mechanisms of Trading Strategies. A potential issue that the backtesting needs to be capable of handling is that the strategy and the price data are not fully aligned – for some signals, there can be missing prices on a particular day. The backtesting gives the user full control over how to handle missing data. If there are intermediate gaps between points in time of the buy and the sell signal, approximate prices may be used to calculate portfolio values. The `find_price()` and `find_alternative_date()` functions from the `_helper_functions.py` module provide logic for finding an optimal price approximation in the case of price gaps. This problem was not relevant to the strategies presented here as the Bitcoin price time series does not have missing values.

8.3.2 Technical Implementation of the Backtesting.

8.3.2.1 Code Access, System Requirements, and Setup All code is available in the section 12 *Appendix* from page 58. Nevertheless, I recommend to refer to the code from the GitLab repository or to install the module by typing `pip install quantbacktest` in a shell. After a successful installation, please define the arguments that are presented in the following paragraphs. With all arguments defined, start the backtesting by calling `backtest_visualizer()`:

```
from quantbacktest import backtest_visualizer

# For managing dates
from datetime import datetime


backtest_visualizer(
    file_path_with_price_data='/home/janspoerer/code/janspoerer/
    quantbacktest/quantbacktest/assets/raw_itsa_data/20190717
```

```
    _itsa_tokenbase_top600_wtd302_token_daily.csv',
    # ONLY LEAVE THIS LINE UNCOMMENTED IF YOU WANT TO USE ETH-ADDRESSES AS ASSET IDENTIFIERS!
    # file_path_with_token_data='raw_itsa_data/20190717_itsa_tokenbase_top600_wtd301_token.csv',  # Only for multi-asset strategies.
    name_of_foreign_key_in_price_data_table='token_itin',
    name_of_foreign_key_in_token_metadata_table='token_itin',
    # 1: execute_strategy_white_noise()
    # 2: Not used anymore, can be reassigned
    # 3: execute_strategy_multi_asset() -> Uses strategy table
    # 4: execute_strategy_ma_crossover()
    int_chosen_strategy=4,
    dict_crypto_options={
        'general': {
            'percentage_buying_fees_and_spread': 0.005,  # 0.26% is the taker fee for low-volume clients at kraken.com https://www.kraken.com/features/fee-schedule
            'percentage_selling_fees_and_spread': 0.005,  # 0.26% is the taker fee for low-volume clients at kraken.com https://www.kraken.com/features/fee-schedule
            # Additional fees may apply for depositing money.
            'absolute_fee_buy_order': 0.0,
            'absolute_fee_sell_order': 0.0,
        }
    },
    float_budget_in_usd=1000000.00,
    file_path_with_signal_data=file_path_with_signal_data,
    strategy_hyperparameters=strategy_hyperparameters,
    margin_loan_rate=0.05,
    list_times_of_split_for_robustness_test=[
        [datetime(2014, 1, 1), datetime(2019, 5, 30)]
    ],
    benchmark_data_specifications={
        'name_of_column_with_benchmark_primary_key': 'id',  # Will be id after processing. Columns will be renamed.
        'benchmark_key': 'TP3B-248N-Q',  # Ether: T22F-QJGB-N, Bitcoin: TP3B-248N-Q
        'file_path_with_benchmark_data': '/home/janspoerer/code/janspoerer/quantbacktest/quantbacktest/assets/raw_itsa_data/20190717_itsa_tokenbase_top600_wtd302_token_daily.csv',
        'risk_free_rate': 0.02
    },
    display_options=display_options,
    constraints=constraints,
    general_settings=general_settings,
    comments=comments,
```

```
)
```

Code Snippet 25: Configuring and calling the backtest – Calling `backtest_visualizer()`.

The repository[30] is stored within the *Frankfurt School Blockchain Center* GitLab Group. There is one public repository with the newest version and one private repository that contains history before April 21, 2020. The reason for this split is that the older history contains confidential data that had to be removed before any public release. The repository is structured as follows:

```
quantbacktest/
|
| -- quantbacktest/
|    | -- assets/
|    | -- components/
|         | -- _0_wrappers.py
|         | -- _1_data_preparation.py
|         | -- _2_strategy_execution.py
|         | -- _3_individual_metrics.py
|         | -- _3_performance_evaluation.py
|         | -- _helper_functions.py
|    | -- __init__.py
|
| -- tests/
|    | -- __init__.py
|
| -- LICENSE
| -- MANIFEST.in
| -- README.md
| -- setup.py
```

Code Snippet 26: Folder structure of the `quantbacktest` module.

For using the tool, using a Unix-like operating system such as Apple macOS or a Linux distribution (e.g., Ubuntu 18.04) is recommended as the tool is designed to be accessed from a Unix shell. Alternatively, shells in Windows 10, such as those provided by Anaconda[31] and *git for windows*[32], may be used. The author used Anaconda (Python package manager), zsh (shell), and Ubuntu 18.04 LTS (Linux distribution, operating system).

[30] gitlab.com/fsbc/theses/quantbacktest.
[31] anaconda.com.
[32] gitforwindows.org.

Installing the necessary dependencies should not pose a problem as most libraries are commonly used, and many users probably have them pre-installed. No specialized backtesting libraries or other financial libraries were used. Ubuntu, macOS, and Windows 10 users can use Anaconda to install libraries. Please refer to the `read.me` that you can find on GitLab, it contains additional information that is required to set up the project.

8.3.2.2 Usage Before starting the program, the user defines settings. Possible settings are described in the following paragraphs.

The `display_options dictionary` allows the user to turn diagrams, warnings, and errors on or off. These options do not affect the results of the backtesting. All currently implemented display options are listed here:

```
display_options = {
    'boolean_plot_heatmap': False,
    'boolean_test': False,  # If multi-asset strategy is used, this will
    cause sampling of the signals to speed up the run for testing during
    development.
    'warning_no_price_for_last_day': False,
    'warning_no_price_during_execution': False,
    'warning_no_price_for_intermediate_valuation': True,
    'warning_alternative_date': False,
    'warning_calculate_daily_returns_alternative_date': False,
    'warning_no_price_for_calculate_daily_returns': False,
    'warning_buy_order_could_not_be_filled': True,
    'warning_sell_order_could_not_be_filled': True,
    'errors_on_benchmark_gap': True,
    'boolean_plot_equity_curve': False,
    'boolean_save_equity_curve_to_disk': True,
}
```

Code Snippet 27: Configuring and calling the backtest – `display_options`.

The `general_settings dictionary` contains information about the desired rounding precision that internal backtesting calculations use. Token-specific rounding settings are not yet implemented. The general settings are listed here:

```
general_settings = {
    'rounding_decimal_places': 4,
    'rounding_decimal_places_for_security_quantities': 0,
```

```
}
```

Code Snippet 28: Configuring and calling the backtest – `general_settings`.

The `strategy_hyperparameters dictionary` is reserved for strategy settings and will impact results. It also contains hyperparameter spaces for scenario testing. The strategy hyperparameters are listed here:

```
# For allowing for flexible time differences (frequencies)
from pandas.tseries.offsets import Timedelta

strategy_hyperparameters = {
    'maximum_deviation_in_days': 300,
    'prices_table_id_column_name': 'token_itin',
    'excel_worksheet_name': excel_worksheet_name,  # Set this to None if CSV is used!
    'buy_parameter_space': [9.8],  # [11, 20] # Times 10! Will be divided by 10.
    'sell_parameter_space': [9.7],  # [5, 9] # Times 10! Will be divided by 10.
    'maximum_relative_exposure_per_buy': 0.34,
    'frequency': Timedelta(days=1),
    'moving_average_window_in_days': 14,
    'id': 'TP3B-248N-Q',
    'boolean_allow_partially_filled_orders': True,
}
```

Code Snippet 29: Configuring and calling the backtest – `strategy_hyperparameters`.

The `constraints dictionary` contains instructions for the trading simulation function, `execute_order()`. It primarily contains risk constraints. The backtesting does not offer many constraints yet. These are the fields of the constraint options, only two of which are implemented:

```
constraints = {
    'maximum_individual_asset_exposure_all': 1.0,  # Not yet implemented
    'maximum_individual_asset_exposure_individual': {},  # Not yet implemented
    'maximum_gross_exposure': 1.0,  # Already implemented
    'boolean_allow_shortselling': False,  # Shortselling not yet implemented
    'minimum_cash': 100,
}
```

Code Snippet 30: Configuring and calling the backtest – `constraints`.

The optional `comments dictionary` does not impact the logic of the backtesting at all; it is only a place for users and developers to record useful information. The comments that the user defines are saved in the `df_results_metrics DataFrame` and stored to disk as a CSV file when the backtesting finishes. Developers can add additional information to the comments in their strategies as the program runs. Some exemplary comments are the `display_options` and the `strategy_hyperparameters` and are listed here:

```
comments = {
    'display_options': repr(display_options),
    'strategy_hyperparameters': repr(strategy_hyperparameters),
}
```

Code Snippet 31: Configuring and calling the backtest – `comments`.

The `dict_crypto_options dictionary` contains general and token-specific settings. In particular, it contains commission, slippage, and spread assumptions as default and as token-specific parameters. The user can indicate token-specific options by using the asset identifier as a `dictionary` key. Here is an example without any token-specific options:

```
dict_crypto_options={
    'general': {
        'percentage_buying_fees_and_spread': 0.005,  # 0.26% is the taker fee for low-volume clients at kraken.com https://www.kraken.com/features/fee-schedule
        'percentage_selling_fees_and_spread': 0.005,  # 0.26% is the taker fee for low-volume clients at kraken.com https://www.kraken.com/features/fee-schedule
        # Additional fees may apply for depositing money.
        'absolute_fee_buy_order': 0.0,
        'absolute_fee_sell_order': 0.0,
    }
}
```

Code Snippet 32: Configuring and calling the backtest – `dict_crypto_options`.

The user can specify the benchmark and the file path to the benchmark data within the `benchmark_data_specifications dictionary`. Furthermore, this setting contains the risk-free rate.

```
benchmark_data_specifications={
```

```
    'name_of_column_with_benchmark_primary_key': 'id', # Will be id after processing. Columns will be renamed.
    'benchmark_key': 'TP3B-248N-Q', # Ether: T22F-QJGB-N, Bitcoin: TP3B-248N-Q
    'file_path_with_benchmark_data': 'raw_itsa_data/20190717_itsa_tokenbase_top600_wtd302_token_daily.csv',
    'risk_free_rate': 0.02,
}
```

Code Snippet 33: Configuring and calling the backtest – `benchmark_data_specifications`.

The `file_path_with_price_data string` variable indicates where the price data is located. The backtesting assumes the following column names for price data: Different assets are identified through an `id` column. Prices (in `float`) are identified through a `price` column. Dates/times are identified through a `datetime` column. Signals are identified through a `signal_strength` column.

The `int_chosen_strategy integer` variable contains the identifier of the strategy that the user wants to execute. This variable allows the user to change between strategies quickly. The strategies are currently numbered as 1, 2, 3, and 4, but the list of available strategies can be extended at will.

The `float_budget_in_usd float` variable contains the initial budget (cash) in fiat currency.

The optional `file_path_with_signal_data string` variable contains the path to the black-box signal table. Only strategies that rely on signal tables require this option. If the format of the black-box signal table is in Excel format, the user also needs to specify the worksheet name in `excel_worksheet_name`.

To start the backtest, the user executes the function `backtest_visualizer()`. `backtest_visualizer()` can be imported from `quantbacktest`. The tool outputs a CSV file that provides information about the performance of all strategies. Each row stands for one strategy; the results for the different performance metrics are divided into columns. A graphical user interface (GUI) is not available yet.

8.3.2.3 Technical Considerations and Design Choices The backtesting emphasizes a functional programming style and works without custom classes. Price data that contains information about different assets over time is multi-dimensional. Therefore, it makes sense (computationally and logically) to use a multi-dimensional data representation. `pandas DataFrames` solve the need for multi-dimensional data representation. A record in a table (a `DataFrame` is simply a table) has multiple criteria for unique identification. In this case, one needs a `datetime` object and an asset `id` to uniquely identify the price for a specific asset at a specific point in time. The documentation describes the advantage of using `pandas DataFrame MultiIndex`: "[...] it enables you to store and manipulate data with an arbitrary number of dimensions in lower/dimensional data structures like Series (1d) and DataFrame (2d)"[33].

9 Results

9.1 Dummy Strategy – White Noise Signals

The white noise strategy is based on a simple random number generator. Every day, a random number generator produces a buy or a sell signal with equal probability. In the presence of a buy signal, 34% of the remaining cash is converted into Bitcoin; in the presence of a sell signal, all Bitcoin in the portfolio is sold. This simple strategy tests if the backtesting works as expected.

From a strategy like this, one can expect two results: 1) lower market exposure (as measured by the equity beta) due to the cash-heaviness of the portfolio, and 2) a drag to performance as the frequent rebalancings cause high transaction costs (a typical run will produce approximately 2000 trades).

The two expected properties that I just mentioned can also be observed from the results of the test run. The results are visualized in the equity curve from figure 4 and summarized in the output `results/backtesting_result_metrics.csv`. In

[33] pandas.pydata.org/pandas-docs/stable/user_guide/advanced.html.

this run, the beta is 0.46, indicating that the market returns only moderately influence the portfolio returns. This is consistent with expectation 1). Also, the portfolio value drifts downward over time. This is consistent with the high transaction costs of 726,361.07 USD in total and with expectation 2). The results are a back-of-the-envelope indication that `quantbacktest` works correctly. In addition to tests like this, I made many manual checks on individual transactions that also verified the correctness of the method.

When I developed `quantbacktest`, I also ran other scenarios to get a sense of how closely the results from the backtesting match with prior expectations (sanity checks). One essential test was to keep one unit of a specific asset in the portfolio and benchmark this portfolio against the same asset. For example, I set the beginning cash balance equal to the Bitcoin price on the first day of the backtesting and created a strategy that buys one unit of Bitcoin at the beginning and holds the asset for the whole period. One would expect minor differences between the portfolio and the benchmark in this case. All of these tests were either in line with expectations or revealed inaccuracies that were subsequently fixed.

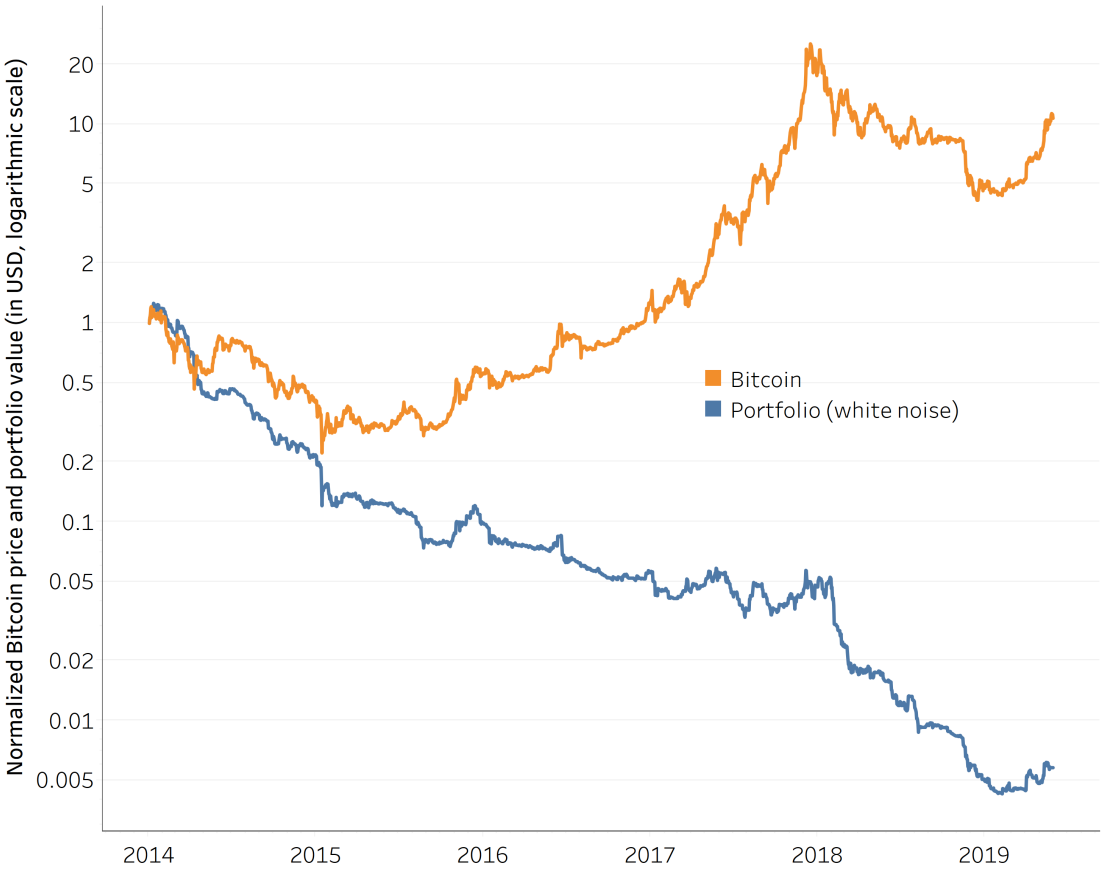

Figure 4. Standardized and benchmarked (portfolio value = 1 and Bitcoin price = 1 at t_0) equity curve for a white noise strategy. This portfolio was created using random signals and is only a proof-of-concept for the backtesting method, and not an economically sensible strategy. Buy and sell signals are generated every day with equal-weighted probability. The minimum cash requirement for this strategy is 100 USD. To avoid rounding problems, I set the budget to 1m USD. Of course, this may lead to a higher market impact than shown here. The percentage spread and slippage are 0.5% in total; no fixed fees were considered. Own visualization created with Tableau Desktop based on `results/df_result_metrics.csv`.

9.2 Momentum – Simple Moving Average

Trend-following strategies are among the most commonly used elements in quantitative trading strategies. This portfolio is based on one of the most straightforward trend-following strategies: simple moving average price crossover. The strategy buys 34% of the remaining cash for every day that the Bitcoin price is above its 14-day moving average and sells all Bitcoin if the Bitcoin price is below its 14-day moving average. Avramov et al., 2018 explain the concept of moving average trading in equity markets, and Hong et al., 2000 provide an intuitive explanation for the existence of momentum anomalies by pointing out that information needs time to

reflect in prices.

This momentum strategy underperforms the market by −9.08% (annualized return). Considering that the strategy performed a high number of rebalancings (930) and incurred high transaction costs (1,929,875 USD), the result is good. The results are in line with findings from Grobys and Sapkota, 2019, indicating that momentum strategies work well for Bitcoin. The equity beta of the strategy is surprisingly low, 0.39, and the maximum drawdown beats the benchmark with 49.68% (Bitcoin: 83.64%).

As Rouwenhorst, 1998 points out, momentum strategies are sensitive to the lookback period used and to the forward returns that the strategy assumes (holding period). Crucial factors like these were not considered here. Upon deciding to craft an even more useful trend-following strategy, an investor should put more care into the selection of the lookback period and of the holding period. Exponentially weighted moving averages may also be useful here. Allowing for short sales and multi-asset long-short portfolios analogous to Berghoff, 2020 would yield more reliable results.

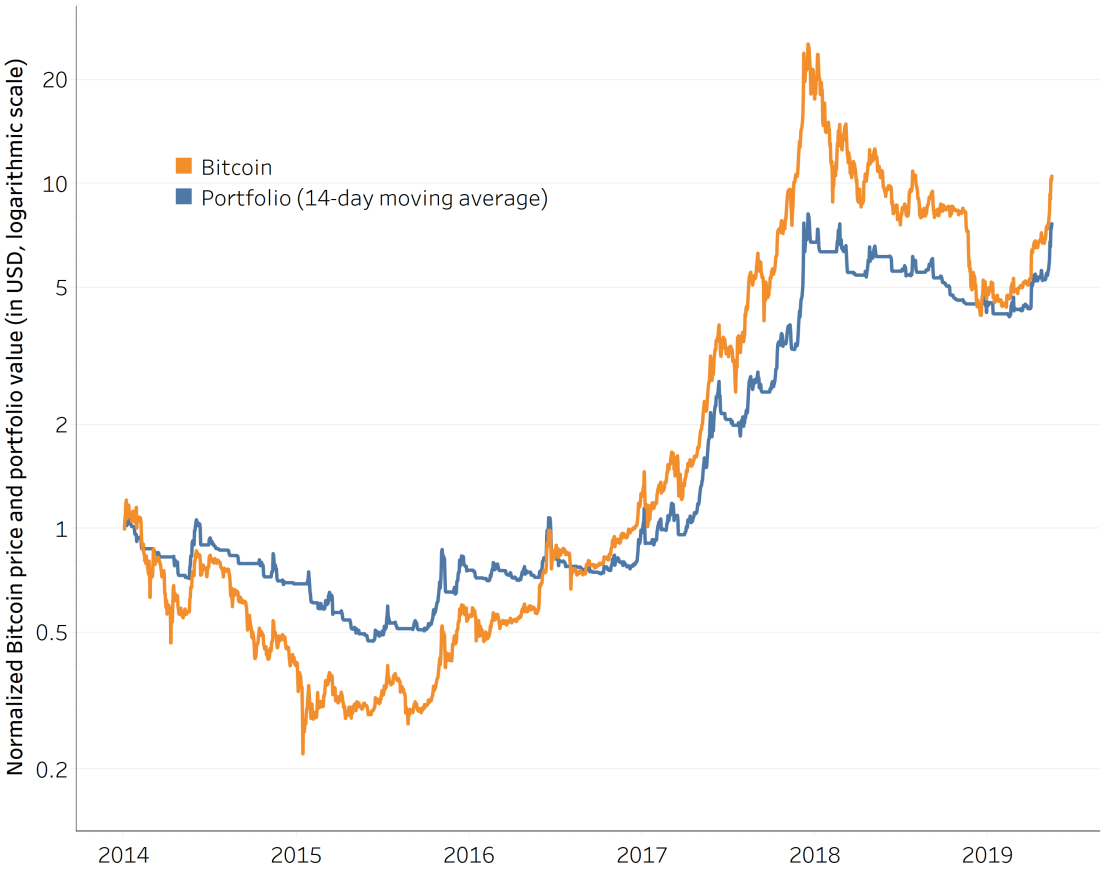

Figure 5. Standardized and benchmarked (portfolio value = 1 and Bitcoin price = 1 at t_0) for a momentum strategy. This portfolio was created using a 14-day moving average strategy. If the Bitcoin price is above its 14-day moving average, 34% of the remaining cash in the portfolio will be spent to enter a long position in Bitcoin. If the Bitcoin price is below its 14-day moving average, all exposure will be liquidated. Rebalancing happens every day. The minimum cash requirement for this strategy is 100 USD. To avoid rounding problems, I set the budget to 1m USD. Of course, this may lead to a higher market impact than shown here. The percentage spread and slippage are in total 0.5%, no fixed fees were considered. Own visualization created with Tableau Desktop based on `results/df_result_metrics.csv`.

9.3 Outside Black-box Signals – Sentiment

The sentiment strategy presented here is used as an example of the usage of a black-box signal table. While the two previous strategies are based on code that is executed at backtest runtime, the signals that underlie this sentiment strategy were generated beforehand. The signals were provided in a simple CSV table. The backtesting reads the table and checks signal-by-signal if the `signal_strength` for the signal exceeds a certain buy threshold or falls below a certain sell threshold. Each buy signal leads to increasing Bitcoin exposure of 34% of the current cash balance. `quantbacktest` executes accordingly and produces the portfolio shown

here. In this case, only Bitcoin is traded, but a signal table can contain signals for any number of assets as long as they are covered by the given price data.

From a sentiment strategy in this configuration, one naturally expects substantial exposure in good times and 100% cash holdings in turbulent times. The more defining question is if the sentiment data from this strategy react faster or slower than the price action. So the question is if this sentiment strategy exhibits predictive power for future price action.

The strategy, at least at first sight, seems to have favorable risk management properties. The diagram shows that the sentiment data helps to stay out of the market before (or at the beginning of) some bad market phases. The performance of the sentiment strategy is only slightly impaired, with an annualized underperformance of -5.52% in comparison to a simple investment in Bitcoin (costs already considered). The maximum drawdown was mitigated from 83.64% (Bitcoin, peak at 2017-12-16, trough at 2018-12-15) to 66.23% (portfolio, peak at 2017-12-16, through at 2018-04-07) because the position was exited before the bottom of the cryptocurrency crash of early 2018.

Despite the attractive risk-return trade-off that this strategy can achieve relative to its benchmark, there is a chance of overfitting in how the black-box signal table was created and how the hyperparameters were chosen that determine the sentiment threshold for entry and exit. The strategy only performed a few trades, and those trades were (possibly luckily) favorable. Also, the strong performance of Bitcoin in the past is the major driver for this long-only, Bitcoin-only strategy. Live-trading or paper trading will show if the risk-mitigating properties of this strategy reflect a true edge.

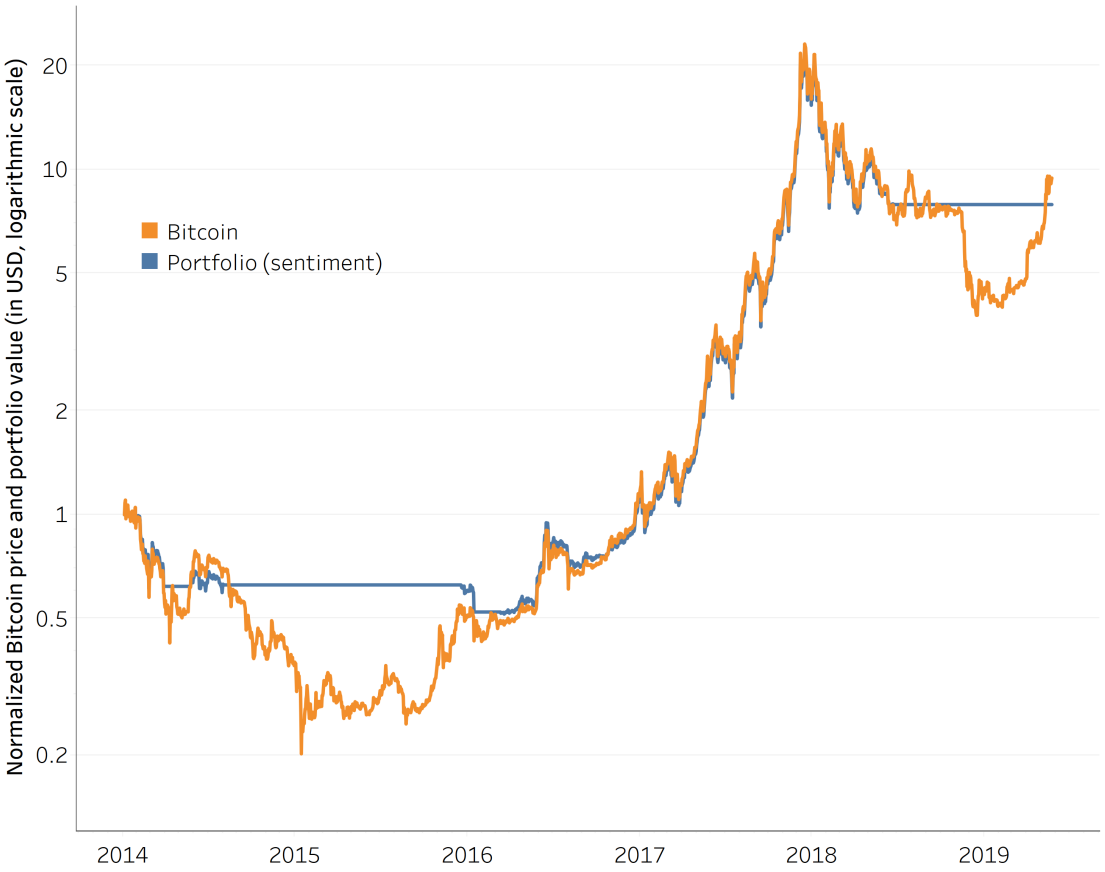

Figure 6. Standardized and benchmarked (portfolio value = 1 and Bitcoin price = 1 at t_0) equity curve for a sentiment strategy. This portfolio was created using sentiment data as buy and sell signals. If the sentiment is above the buying threshold, a Bitcoin position will be increased each day (34% of remaining cash is used each day) until cash is depleted. All Bitcoin is sold if the sentiment value is smaller than the selling threshold. Rebalancing happens very day. The minimum cash requirement for this strategy is 100 USD. To avoid rounding problems, I set the budget to 1m USD. Of course, this may, in turn, lead to a higher market impact than shown here. The percentage spread and slippage are in total 0.5%, no fixed fees were considered. Own visualization created with Tableau Desktop based on `results/df_result_metrics.csv`.

10 Conclusion

10.1 Summary

I have presented a backtesting tool for cryptocurrency markets that, to the best of my knowledge, accounts for real-world conditions more realistically than any other open-source framework while also being easy to implement. In section 6 *Theory and Literature Review*, it became clear that there is still a literature gap around factor modeling in cryptocurrency research, showing the need for powerful research tools that aid academics and investors in search of new factors. Then, in section

7 *Data*, I described the price data that is used in the backtesting and presented trade-offs between higher-frequency and lower-frequency data. Section 8 *Method (Trading Strategy and Backtesting)* contains the technical setup of the backtesting and its functions. I also presented assumptions and made internal calculations of the tool transparent. Lastly, in 9 *Results*, I tested some commonly known and easy-to-understand strategies with the backtesting and analyzed if the results meet expectations. All three strategies, 1) white noise, 2) momentum, and 3) sentiment yielded the expected results. From these findings, I concluded that the backtesting works well for realistic applications.

10.2 Implications for Practice

The tool presented allows investors to build and test algorithmic trading strategies for all major cryptocurrencies. Thanks to the ITSA TOKENBASE price data integration, investors can access price data of approximately 600 different cryptocurrencies and build strategies around these cryptocurrencies without the need for advanced technical expertise. Consequently, the tool paves the way for broader investor sophistication in the cryptocurrency space and facilitates sound investment decisions.

There are strategy parameters that are hard to test without a certain level of backtesting sophistication. Transaction costs are a major concern in algorithmic trading, and they have a significant impact on the required certainty that a trader needs to have about the magnitude of market mispricing (Chan, 2009, p. 22). This is because low gross returns may be dominated by transaction costs. Only a backtesting tool that is capable of realistically simulating transaction costs can aid in setting the minimum expected gross return threshold accurately.

10.3 Implications for Research

This tool makes complex backtesting tasks easier for researchers and can serve to calculate portfolio returns and risk-relevant metrics realistically. Backtests can

reflect trading costs, can simulate portfolio constraints, and can be separated into training and test periods.

Tensor-based backtesting will still be performed using tensor libraries as they are computationally superior to this tool. Fees can also be considered in a simplified form when using tensor-based strategies. Jegadeesh and Titman, 2001 use a popular method for long-short portfolio simulation that can consider transaction costs.

I see most use of this backtesting for later steps in the quantitative trading process and less in the academic idea generation phase. Initial research and hypothesis testing can also be performed without a realistic backtesting engine. The tool is most relevant for researchers that have to take into account fees and risk management constraints as realistically as possible.

10.4 Limitations

10.4.1 Forks and Other Distorting Events. As mentioned in section 6 *Theory and Literature Review* on page 10, cryptocurrencies are exposed to events that are comparable in their price impact to stock splits and dividends. Notably, cryptocurrency forks are a phenomenon that a comprehensive cryptocurrency backtesting framework should be able to handle. The backtesting does not yet have such a function.

10.4.2 Margin Trading. The core trading execution function in the backtesting (`execute_order()`) cannot yet handle margin trading. `df_trading_journal` already contains fields for margin trading and the respective settings are already there, but there is no logic in `execute_order()` that can handle negative cash balances.

10.4.3 Arbitrage Trading. The backtesting can handle two-dimensional price data (see sections 7 *Data* and 8.1.3 `df_prices`) with asset identifiers and time identifiers. Two-dimensional price data is not enough for arbitrage trading, which would require an additional identification dimension, i.e., the exchange. In

practice, each cryptocurrency is usually quoted in different pairs by each exchange. For Bitcoin, there are usually quotes such as Bitcoin/USD, Bitcoin/EUR, and Bitcoin/Ether. The backtesting cannot yet handle the added complexity that arbitrage trading and multiple currency pairs bring. Even though the incompatibility with arbitrage trading is a major weakness of this backtesting, I do not plan to include this functionality. There are two reasons for this reluctance.

First, inefficiencies in cryptocurrency markets are often already covered by specialized traders. Also, some seemingly high arbitrage opportunities are not a result of mispricing; they reflect real risk. For example, certain exchanges are less safe or offer more restrictive withdrawal policies than other exchanges; price differences are usually based on exchange quality or market accessibility.

Second, there are already market participants that use sophisticated, low-latency software (possibly even hardware, in the future[34]) to address arbitrage. I believe that a backtesting tool for arbitrage trading should be managed in a separate, dedicated arbitrage backtesting project.

10.4.4 Different Data Sources in Sensitivity Analyses. Users can already test the parameter and the timeframe sensitivity of their models. These tests help to validate robustness. However, price differences between exchanges can also make differences in backtesting results. Therefore, the backtesting should allow users to specify multiple data sources (which were obtained from multiple exchanges). Each backtesting scenario would then run with all data sources. This function would show whether performance is based on exchange-specific anomalies.

10.4.5 Computational Constraints. The backtesting is slower than tensor-based operations in `NumPy` and similar software packages. As the simulation needs to take into account trades in strictly sequential order, parallel processing is not possible for a single scenario run. The reason for this lack of parallel computing

[34] In traditional finance, high-frequency traders use hardware-accelerated network stacks, network processing units, field-programmable gate arrays, and relocation to gain an edge. These technological improvements lead to situations where it is not enough anymore to compete with software; one also needs to employ sophisticated hardware to stay competitive.

potential is that all steps in one scenario run of the backtesting depend on each other. For example, the cash balance after the first trade is dependent on the full simulation of the first trade. The second trade cannot be simulated without knowing the cash balance after the first trade, so the backtest waits for the first trade to finish before simulating the second trade. It is, however, possible to use multiple CPU cores by running several scenarios in parallel.

10.4.6 Improved Benchmarks. The results from this thesis all stem from Bitcoin-only trading strategies with Bitcoin as a benchmark. The benchmark influences relative performance metrics, such as market risk exposure (beta). Even though Bitcoin has a consistently dominant market position in terms of market capitalization, a more fine-grained benchmarking system, similar to those benchmarks used in equity markets, would add value. Sophisticated benchmarking is an especially pressing issue for multi-asset strategies that also trade altcoins. The backtesting needs to provide a way to calculate a benchmark from a given price series. A custom benchmark could be a market capitalization-weighted benchmark or another approach. Incorporating the index calculation method proposed by Trimborn and Härdle, 2018 would be an example of how this backtesting could be further improved. Alternatively, one could use already available indices such as the market capitalization-weighted Bloomberg Galaxy Crypto Index[35].

10.4.7 Limited Timeframe and Pending Market Maturity. Bitcoin's white paper (Nakamoto, 2008) was published only a little more than ten years ago[36]. Market volume has since increased sharply. As a result, there is only little data available to test the aforenamed strategies in times of financial market turmoil, quickly changing interest rates, fleeing into safe havens and liquidity shortages among investors and banks[37]. Financial crises (and hidden systematic risks) could change the patterns one can observe in cryptocurrency backtests.

[35] bloomberg.com/professional/product/indices/bloomberg-galaxy-crypto-index.

[36] The date of publication is not stated on the white paper, so here is a trusted source that confirms the date of publication: "Liechtenstein Blockchain Act," 2018, pp. 9–10.

[37] The corona crisis of 2020 happened at the same time that this thesis was written and could not be considered in the presented results.

The backtested performance may not hold in a more mature and in a more regulated cryptocurrency market. Retail investors may experience more regulatory scrutiny (e.g., tighter know your customer regulations for cryptocurrency exchanges), limiting market access. Already in 2014, the U.S. Federal Reserve showed interest in Bitcoin (Badev & Chen, 2014), and the Liechtenstein regulation for distributed ledger technology sets an example for actual regulatory action ("Liechtenstein Blockchain Act," 2018, pp. 11, 28).

Also, active institutional investors may increase their cryptocurrency exposure while exchange-traded cryptocurrency funds may emerge. All of these factors will have an impact on the correlations between future price movements and the predictive metrics that were used in this thesis and could eliminate the usefulness of the strategies presented or even limit the algorithmic trading potential in general. The development toward more investor sophistication in equity and bond markets over the last decades can be seen as an example here.

The potential of blockchain, and in particular the potential for Bitcoin, is not fully used today, and many of the applications described by Iansiti and Lakhani, 2017, p. 7 and "Liechtenstein Blockchain Act," 2018, p. 18 are still under development. Changes in the purpose of Bitcoin may change price movement patterns in the future. Bitcoin could also lose importance as other DLT concepts such as Ethereum, Hyperledger Fabric, and Corda offer a wide field of applications that is not available for Bitcoin (Valenta & Sandner, 2017).

Analogies from traditional finance show that inefficiencies in asset pricing tend to vanish as the understanding of an asset's characteristics increases (McLean & Pontiff, 2016). Brauneis and Mestel, 2018 and Khuntia and Pattanayak, 2018 observe a similar trend of diminishing inefficiencies in cryptocurrency markets. Another effect is that previously well-established risk factors may lose relevance. Fama and French, 2019 recently described how even one of their cornerstone findings in finance, the study of the value factor, seems to lose relevance as a risk factor over time. Fama and French do not provide an intuitive explanation for this phenomenon, but one can

suggest that the rise of technology companies and a general shift in how economic value is created (value of expensed intangibles) plays a role here. In the future, similar market shifts are thinkable in cryptocurrency markets.

Time will help to solve the presented maturity-related issues. Expanding the use of the tool to intraday data could help to mitigate the importance of systematic events and make strategies more robust despite the limited timeframe of available data. Users of this tool need to be aware of the constant change that is present in cryptocurrency markets and consider this when re-training and re-configuring their models.

10.4.8 Data-induced Survivorship Bias. Cryptocurrencies have been markedly volatile since their emergence. Not only have prices fluctuated heavily, but there has also been a large number of new cryptocurrencies and a large number of left-behind cryptocurrencies that have no relevance anymore today. These fundamental shifts may lead to problems; the data may not be survivorship bias-free. The data may show a tendency toward containing too many successful cryptocurrencies. As the backtesting can flexibly handle different datasets, one can also use another dataset that is known to be free of survivorship bias.

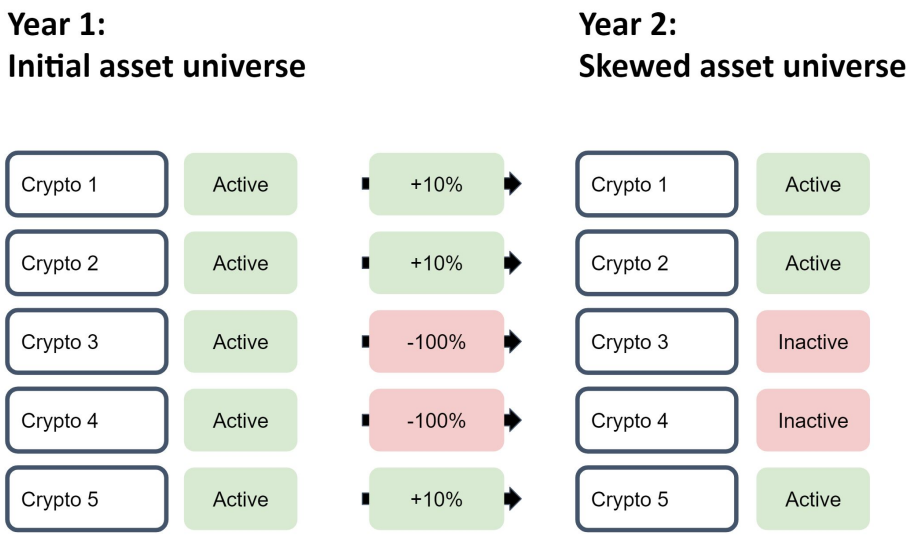

Figure 7. A visual example of selection bias. The figure shows that failed cryptocurrencies may fall through the cracks today, leading to inflated backtesting results. In this example, the hindsight performance of those cryptocurrencies that are still on the market is 10% on average. But the overall performance of all five cryptocurrencies is −34%. The early days of cryptocurrencies may not be adequately reflected in the composition of today's cryptocurrency price databases. If this is true, backtests will often lead to biased results. Own visualization created with Google Slides.

10.5 Future Research

10.5.1 Reconstruction of Backtests From Popular Research Articles. As mentioned before, the tensor-based method used by Jegadeesh and Titman, 2001 is commonly used in financial market research. It would be interesting to see how significant the difference between the two methods of backtesting is. To achieve this, one could use `quantbacktest` to reconstruct the methodologies from respected papers such as Daniel and Moskowitz, 2016. The results could give further insights into whether more complex backtesting methods are worth the additional effort.

10.5.2 Flexible Slippage Model. The simple slippage model presented here (manual input of relative slippage) could be extended. If the daily trading volume of the respective asset is considered in the slippage model, one could make more accurate slippage predictions by comparing total daily volume with the order size or by using historical order book depth.

10.5.3 Risk Constraints. More risk constraints could be added to the execution simulation. For example, executions could be dependent on minimum levels of liquidity and volume in the market. Furthermore, position sizes could be limited to specific fractions of total portfolio value. Another useful constraint could be to allow users to define maximum leverage boundaries. Even more advanced would be a constraint that checks if a trade would exceed certain risk factor exposure limits, and that would reduce the traded amount or cancel the trade.

10.5.4 Recording Risk Metrics Over Time. In addition to considering constraints in the simulation, it would also be helpful to record risk management metrics over time, not just as a summary at the end. In particular, it would be helpful if the market risk exposure for the current portfolio would be calculated and recorded at all times so that the user could investigate (undesired) intermediary spikes in market risk factor exposure. This feature may be further extended to more risk factors such as momentum and size.

10.5.5 Factor Identification Methods That Simplify the Research Process. Research of Feng et al., 2017 suggests that there are robust methods for finding risk factors for stock markets. Adopting a quantitative approach like theirs directly in the backtesting would ease the process of finding novel cryptocurrency factors by further streamlining the research process toward idea generation. Any additional features that can prevent the factor mining/overfitting problems described by Harvey and Liu, 2019 will be useful additions to the study presented here.

10.5.6 Usability, Access, and Graphical User Interface. From a non-technical user's perspective, a graphical user interface would further improve accessibility to `quantbacktest`. Executing a Python script and changing settings as text can lead to errors and confusion, especially for people that are not familiar with coding. A good solution would be a hosted web server that users can control via a simple web page.

11 References

Ang, A., Hodrick, R., Xing, Y., & Zhang, X. (2009). High idiosyncratic volatility and low returns: International and further U.S. evidence. *Journal of Financial Economics*, *91*(1), 1–23.

Asness, C. S., Frazzini, A., Israel, R., Moskowitz, T. J., & Pedersen, L. H. (2018). Size matters, if you control your junk. *Journal of Financial Economics*, *129*(3), 479–509.

Asness, C. S., Moskowitz, T. J., & Pedersen, L. H. (2013). Value and momentum everywhere. *The Journal of Finance*, *68*(3), 929–985.

Athey, S., Parashkevov, I., Sarukkai, V., & Xia, J. (2016). Bitcoin pricing, adoption, and usage: Theory and evidence. *Stanford University Graduate School of Business Research Paper*.

Avramov, D., Kaplanski, G., & Subrahmanyam, A. (2018). The predictability of equity returns from past returns: A new moving average-based perspective. *SSRN*.

Badev, A. I., & Chen, M. (2014). Bitcoin: Technical background and data analysis. *Federal Reserve Board working paper*.

Banz, R. W. (1981). The relationship between return and market value of common stocks. *Journal of Financial Economics*, *9*(1), 3–18.

Basu, S. (1977). Investment performance of common stocks in relation to their price-earnings ratios: A test of the efficient market hypothesis. *The Journal of Finance*, *32*(3), 663–682.

Berghoff, L. (2020). *Applying asset pricing factors in quantitative cryptoasset trading strategies* (Bachelor's Thesis). Frankfurt School of Finance and Management.

Bhandari, L. C. (1988). Debt/equity ratio and expected common stock returns: Empirical evidence. *The Journal of Finance*, *43*(2), 507–528.

Brauneis, A., & Mestel, R. (2018). Price discovery of cryptocurrencies: Bitcoin and beyond. *Economics Letters*, *165*, 58–61.

Carhart, M. M. (1997). On persistence in mutual fund performance. *The Journal of Finance*, *52*(1), 57–82.

Chan, E. (2009). *Quantitative trading: How to build your own algorithmic trading business*. John Wiley & Sons, Inc.

Daniel, K., & Moskowitz, T. J. (2016). Momentum crashes. *Journal of Financial Economics*, *122*(2), 221–247.

Desai, H., & Jain, P. C. (1997). Long-run common stock returns following stock splits and reverse splits. *The Journal of Business*, *70*(3), 409–433.

Fama, E. F., & French, K. R. (1992). The cross-section of expected stock returns. *The Journal of Finance*, *47*(2), 427–465.

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics*, *33*(1), 3–56.

Fama, E. F., & French, K. R. (1996). Multifactor explanations of asset pricing anomalies. *The Journal of Finance*, *51*(1), 55–84.

Fama, E. F., & French, K. R. (1998). Value versus growth: The international evidence. *The Journal of Finance*, *53*(6), 1975–1999.

Fama, E. F., & French, K. R. (2015). A five-factor asset pricing model. *Journal of Financial Economics*, *116*(1), 1–22.

Fama, E. F., & French, K. R. (2017). International tests of a five-factor asset pricing model. *Journal of Financial Economics*, *123*(3), 441–463.

Fama, E. F., & French, K. R. (2019). The value premium. *Fama-Miller Working Paper*, (20-01).

Feng, G., Giglio, S., & Xiu, D. (2017). Taming the factor zoo. *Chicago Booth research paper*, (17-04).

Frino, A., & Oetomo, T. (2005). Slippage in futures markets: Evidence from the Sydney Futures Exchange. *Journal of Futures Markets: Futures, Options, and Other Derivative Products*, *25*(12), 1129–1146.

Fürst, J. C. F. (2019). *Crypto markets: A quantitative empirical analysis of Bitcoin and Ethereum* (Bachelor's Thesis).

Grobys, K., & Sapkota, N. (2019). Cryptocurrencies and momentum. *Economics Letters*, *180*, 6–10.

Harvey, C. R., & Liu, Y. (2019). A census of the factor zoo. *SSRN*.

Hayes, A. (2015). A cost of production model for Bitcoin. *SSRN*.

Hong, H., Lim, T., & Stein, J. C. (2000). Bad news travels slowly: Size, analyst coverage, and the profitability of momentum strategies. *The Journal of Finance*, *55*(1), 265–295.

Hubrich, S. (2017). "Know When to Hodl 'Em, Know When to Fodl 'Em": An investigation of factor based investing in the cryptocurrency space. *SSRN*.

Iansiti, M., & Lakhani, K. R. (2017). The truth about blockchain. *Harvard Business Review*.

Jegadeesh, N., & Titman, S. (1993). Returns to buying winners and selling losers: Implications for stock market efficiency. *The Journal of Finance*, *48*(1), 65–91.

Jegadeesh, N., & Titman, S. (2001). Profitability of momentum strategies: An evaluation of alternative explanations. *The Journal of Finance*, *56*(2), 699–720.

Jensen, M. C. (1968). The performance of mutual funds in the period 1945–1964. *The Journal of Finance*, *23*(2), 389–416.

Jiang, F., Lee, J., Martin, X., & Zhou, G. (2019). Manager sentiment and stock returns. *Journal of Financial Economics*, *132*(1), 126–149.

Kakushadze, Z. (2018). Cryptoasset factor models. *Algorithmic Finance*, *7*(3–4), 87–104.

Khuntia, S., & Pattanayak, J. (2018). Adaptive market hypothesis and evolving predictability of Bitcoin. *Economics Letters*, *167*, 26–28.

Li, J., & Yi, G. (2019). Toward a factor structure in crypto asset returns. *The Journal of Alternative Investments*, *21*(4), 56–66.

Liechtenstein Blockchain Act [Full title: Unofficial translation of the government consultation report and the draft-law on transaction systems based on trustworthy technologies (Blockchain Act)]. (2018).

Lintner, J. (1965a). Security prices, risk, and maximal gains from diversification. *The Journal of Finance*, *20*(4), 587–615.

Lintner, J. (1965b). The valuation of risk assets and the selection of risky investments in stock portfolios and capital budgets. *The Review of Economics and Statistics*, *47*(1), 13–37.

Markowitz, H. (1952). Portfolio selection. *The Journal of Finance*, *7*(1), 77–91.

McGlone, M. (2020). Bitcoin maturity leap. *Bloomberg Crypto Outlook*.

McLean, R. D., & Pontiff, J. (2016). Does academic research destroy stock return predictability? *The Journal of Finance*, *71*(1), 5–32.

Moskowitz, T. J., & Grinblatt, M. (1999). Do industries explain momentum? *The Journal of Finance*, *54*(4), 1249–1290.

Mossin, J. (1966). Equilibrium in a capital asset market. *Econometrica: Journal of the Econometric Society*, 768–783.

Nakamoto, S. (2008). Bitcoin: A peer-to-peer electronic cash system.

Publication manual of the American Psychological Association. (2001). American Psychological Association.

Ross, S. A. (1976). The arbitrage theory of capital asset pricing. *Journal of Economic Theory*, *13*(3), 341–360.

Rouwenhorst, K. G. (1998). International momentum strategies. *The Journal of Finance*, *53*(1), 267–284.

Sharpe, W. F. (1964). Capital asset prices: A theory of market equilibrium under conditions of risk. *The Journal of Finance*, *19*(3), 425–442.

Sharpe, W. F. (1966). Mutual fund performance. *The Journal of Business*, *39*(1), 119–138.

Sharpe, W. F. (1994). The Sharpe ratio. *Journal of Portfolio Management*, *21*(1), 49–58.

Shen, D., Urquhart, A., & Wang, P. (2019). A three-factor pricing model for cryptocurrencies. *Finance Research Letters*.

Sovbetov, Y. (2018). Factors influencing cryptocurrency prices: Evidence from Bitcoin, Ethereum, Dash, Litcoin, and Monero. *Journal of Economics and Financial Analysis*, *2*(2), 1–27.

Thaler, R. H. (1987). Anomalies: The January effect. *Journal of Economic Perspectives*, *1*(1), 197–201.

Trimborn, S., & Härdle, W. K. (2018). CRIX an index for cryptocurrencies. *Journal of Empirical Finance*, *49*, 107–122.

Valenta, M., & Sandner, P. (2017). Comparison of Ethereum, Hyperledger Fabric and Corda. *Frankfurt School Blockchain Center Working Paper*.

Woo, W. (2017). Is Bitcoin in a bubble? Check the NVT ratio. *Forbes*. https://www.forbes.com/sites/wwoo/2017/09/29/is-bitcoin-in-a-bubble-check-the-nvt-ratio/

12 Appendix

12.1 Additional Visualizations

The three diagrams in this section give additional insights into the outputs that `quantbacktest` provides. As an overview, figure 8 shows the built-in equity curve plot. It is created without the need for additional software and is automatically saved as a PNG file. Figure 9 shows a visual analysis of the results from the `results/df_result_metrics.csv` output and aims to portrait `quantbacktest`'s ability to handle multiple periods in a single run of the backtesting. Figure 10 uses the same data source (`results/df_result_metrics.csv`), but uses two sets of strategy hyperparameters that were analyzed in a single run of the backtest. The strategy that was used in the displayed backtest will not be discussed in detail here as the strategy is not the primary objective of showing these diagrams.

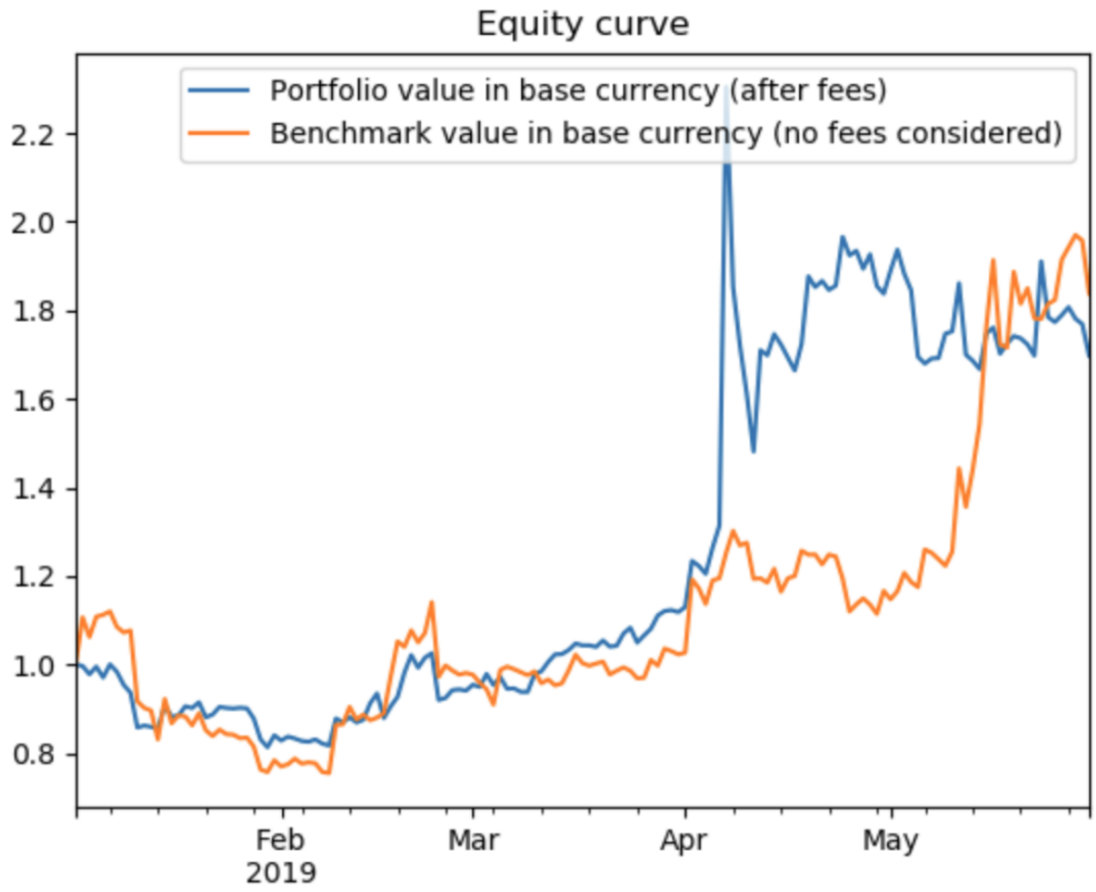

Figure 8. Standardized and benchmarked (portfolio value = 1 and Bitcoin price = 1 at t_0) equity curve. This visualization shows an automatically generated output from one multi-asset run (scenario) of `quantbacktest`. While the visual appeal of this diagram is lower than those from Tableau or Google Data Studio, it provides the user with a zero-effort way of analyzing results without using additional software. The user can choose the benchmark. The visualization was created directly by `quantbacktest` using the open-source `matplotlib` visualization library.

Duration of the tested interval1	Begin date of tested interval: 2018						
	Q1		Q2			Q3	
	January	March	April	May	June	July	August
365	80%	40%	60%	60%	40%	30%	-20%
730	70%	50%	90%	30%	90%	90%	10%
1095	160%	50%	0%	30%	80%	20%	50%
1460	50%	30%	20%	50%	30%	-10%	80%

Figure 9. Heatmap for visual robustness comparisons (1/2): Time sensitivity. `quantbacktest` also automatically creates a heatmap (using `matplotlib`), but I decided to show the results of a Tableau visualization. The heatmap from `matplotlib` does not look nearly as good as this one. Own visualization created using data from the `results/df_result_metrics.csv` file and Tableau.

Figure 10. Heatmap for visual robustness comparisons (2/2): Parameter sensitivity. The heatmap shows the profitability (annualized return on investment) in percent (and highlighted by color) for different thresholds of two arbitrary dimensions that could be used to execute cryptocurrency trading strategies. Unlike the other heatmap (figure 9), this heatmap shows robustness for different parameter settings, not time intervals. This visualization is not directly available from `quantbacktest` at the moment but needs to be created with another software tool. Own visualization created using data from the `results/df_result_metrics.csv` file and the commercial Tableau visualization software.

12.2 Source Code

12.2.1 Remarks About the Code. Setup instructions are provided in section 8.3.2.1 *Code Access, System Requirements, and Setup* on page 31. The code that is shown below simply serves for completeness of the thesis. The reader is advised to follow the instructions and install the module (`pip install quantbacktest`) to be able to test the tool. Also, the code base is continuously improved upon and the code shown below may be outdated at the time of reading this.

12.2.2 Calling the Backtest – `tests/__init__.py`.

```
from quantbacktest import backtest_visualizer

# Importing modules from this repository
import sys

# For managing dates
from datetime import datetime

# For allowing for flexible time differences (frequencies)
from pandas.tseries.offsets import Timedelta


display_options = {
    'boolean_plot_heatmap': False,
    'boolean_test': False,  # If multi-asset strategy is used, this will cause sampling of the signals to speed up the run for testing during development.
    'warning_no_price_for_last_day': False,
    'warning_no_price_during_execution': False,
    'warning_no_price_for_intermediate_valuation': True,
    'warning_alternative_date': False,
    'warning_calculate_daily_returns_alternative_date': False,
    'warning_no_price_for_calculate_daily_returns': False,
    'warning_buy_order_could_not_be_filled': True,
    'warning_sell_order_could_not_be_filled': True,
    'errors_on_benchmark_gap': True,
    'boolean_plot_equity_curve': False,
    'boolean_save_equity_curve_to_disk': True,
    'string_results_directory': '/home/janspoerer/code/janspoerer/tmp/results'
}

```

```
general_settings = {
    'rounding_decimal_places': 4,
    'rounding_decimal_places_for_security_quantities': 0,
}

file_path_with_signal_data = '/home/janspoerer/code/janspoerer/quantbacktest
    /quantbacktest/assets/strategy_tables/test.csv'
excel_worksheet_name = 'weights'

strategy_hyperparameters = {
    'maximum_deviation_in_days': 300,
    'prices_table_id_column_name': 'token_itin',
    'excel_worksheet_name': excel_worksheet_name,  # Set this to None if CSV
     is used!
    # For OpenMetrics: 9.8
    'buy_parameter_space': [9.8],  # [11, 20] # Times 10! Will be divided by
     10.
    # For OpenMetrics: 9.7
    'sell_parameter_space': [9.7],  # [5, 9] # Times 10! Will be divided by
    10.
    'maximum_relative_exposure_per_buy': 0.34,
    'frequency': Timedelta(days=1),
    'moving_average_window_in_days': 14,
    'id': 'TP3B-248N-Q',
    'boolean_allow_partially_filled_orders': True,
}

constraints = {
    'maximum_individual_asset_exposure_all': 1.0,  # Not yet implemented
    'maximum_individual_asset_exposure_individual': {},  # Not yet
    implemented
    'maximum_gross_exposure': 1.0,  # Already implemented
    'boolean_allow_shortselling': False,  # Shortselling not yet implemented
    'minimum_cash': 100,
}

comments = {
    'display_options': repr(display_options),
    'strategy_hyperparameters': repr(strategy_hyperparameters)
}

backtest_visualizer(
    file_path_with_price_data='/home/janspoerer/code/janspoerer/
    quantbacktest/quantbacktest/assets/raw_itsa_data/20190717
    _itsa_tokenbase_top600_wtd302_token_daily.csv',
    # ONLY LEAVE THIS LINE UNCOMMENTED IF YOU WANT TO USE ETH-ADDRESSES AS
    ASSET IDENTIFIERS!
    # file_path_with_token_data='raw_itsa_data/20190717
    _itsa_tokenbase_top600_wtd301_token.csv',  # Only for multi-asset
```

```
    strategies.
    name_of_foreign_key_in_price_data_table='token_itin',
    name_of_foreign_key_in_token_metadata_table='token_itin',
    # 1: execute_strategy_white_noise()
    # 2: Not used anymore, can be reassigned
    # 3: execute_strategy_multi_asset() -> Uses strategy table
    # 4: execute_strategy_ma_crossover()
    int_chosen_strategy=4,
    dict_crypto_options={
        'general': {
            'percentage_buying_fees_and_spread': 0.005,  # 0.26% is the taker fee for low-volume clients at kraken.com https://www.kraken.com/features/fee-schedule
            'percentage_selling_fees_and_spread': 0.005,  # 0.26% is the taker fee for low-volume clients at kraken.com https://www.kraken.com/features/fee-schedule
            # Additional fees may apply for depositing money.
            'absolute_fee_buy_order': 0.0,
            'absolute_fee_sell_order': 0.0,
        }
    },
    float_budget_in_usd=1000000.00,
    file_path_with_signal_data=file_path_with_signal_data,
    strategy_hyperparameters=strategy_hyperparameters,
    margin_loan_rate=0.05,
    list_times_of_split_for_robustness_test=[
        [datetime(2014, 1, 1), datetime(2019, 5, 30)]
    ],
    benchmark_data_specifications={
        'name_of_column_with_benchmark_primary_key': 'id',  # Will be id after processing. Columns will be renamed.
        'benchmark_key': 'TP3B-248N-Q',  # Ether: T22F-QJGB-N, Bitcoin: TP3B-248N-Q
        'file_path_with_benchmark_data': '/home/janspoerer/code/janspoerer/quantbacktest/quantbacktest/assets/raw_itsa_data/20190717_itsa_tokenbase_top600_wtd302_token_daily.csv',
        'risk_free_rate': 0.02
    },
    display_options=display_options,
    constraints=constraints,
    general_settings=general_settings,
    comments=comments,
)
```

Code Snippet 34: This example triggers a backtest and is used to change the settings – `tests/__init__.py`.

12.2.3 Main Components – `/components/*.py`.

```
"""Serves as a high-level caller."""

# For counting the number of result files in the "results" folder to allow
    for
# consistent file versioning when saving the results as a csv file.
import os
import os.path

# For managing dates
from datetime import datetime

# For vector operations
from numpy import array

# For plots (especially the heatmap)
import matplotlib.pyplot as plt

from _1_data_preparation import prepare_data, save_dataframe_to_csv
from _2_strategy_execution import test_strategies


def backtest(
        file_path_with_price_data,
        int_chosen_strategy,
        dict_crypto_options,
        float_budget_in_usd,
        margin_loan_rate,
        display_options,
        general_settings,
        constraints,
        start_time,
        comments,
        file_path_with_signal_data=None,
        file_path_with_token_data=None,# Only for multi-asset strategies.
        name_of_foreign_key_in_price_data_table=None,
        name_of_foreign_key_in_token_metadata_table=None,
        boolean_allow_shorting=False,
        list_trading_execution_delay_after_signal_in_hours={
            'delay_before_buying': 0,
            'delay_before_selling': 0
        },
        minimum_expected_mispricing_trigger_in_percent={
            'mispricing_when_buying': 0.0,
            'mispricing_when_selling': 0.0
        },
        strategy_hyperparameters=None,
```

```
        sell_at_the_end=True,
        list_times_of_split_for_robustness_test=None,
        benchmark_data_specifications={# Bitcoin as default benchmark
            'name_of_column_with_benchmark_primary_key': 'id',
            'benchmark_key': 'TP3B-248N-Q',
            'file_path_with_benchmark_data': 'raw_itsa_data/20190717
    _itsa_tokenbase_top600_wtd302_token_daily.csv'
        }
    ):
    """Backtests a user-defined strategy.

    With user-defined price data and possibly many other parameters.
    """
    # assert percentage_selling_fees_and_spread >= 0, "Please choose
    positive selling slippage."
    # assert percentage_buying_fees_and_spread >= 0, "Please choose positive
     buying slippage."
    # assert percentage_selling_fees_and_spread == 0, "Please choose a
    selling slippage greater than zero."
    # assert percentage_buying_fees_and_spread == 0, "Please choose a buying
     slippage greater than zero."
    assert float_budget_in_usd > 0, "Please choose positive budget."
    # assert list_trading_execution_delay_after_signal_in_hours >= 0, "
    Please choose an execution delay greater than zero."
    # assert list_trading_execution_delay_after_signal_in_hours == 0, "
    Please choose an exectution delay greater than zero."

    ## Check if exposure is > 0 and <= 100
    ## Check if margin is between 0.01 and 1
    ## Check if fees are >= 0 and warn if fees = 0

    df_prices = prepare_data(
        file_path_with_price_data=file_path_with_price_data,
        file_path_with_token_data=file_path_with_token_data,
        strategy_hyperparameters=strategy_hyperparameters,
        name_of_foreign_key_in_price_data_table=
    name_of_foreign_key_in_price_data_table,
        name_of_foreign_key_in_token_metadata_table=
    name_of_foreign_key_in_token_metadata_table,
    )

    if list_times_of_split_for_robustness_test is None:
        dfs_results = test_strategies(
            df_prices=df_prices,
            int_chosen_strategy=int_chosen_strategy,
            float_budget_in_usd=float_budget_in_usd,
            file_path_with_signal_data=file_path_with_signal_data,
            margin_loan_rate=margin_loan_rate,
            boolean_allow_shorting=boolean_allow_shorting,
```

```
86              list_trading_execution_delay_after_signal_in_hours=
    list_trading_execution_delay_after_signal_in_hours,
87              dict_crypto_options=dict_crypto_options,
88              minimum_expected_mispricing_trigger_in_percent=
    minimum_expected_mispricing_trigger_in_percent,
89              strategy_hyperparameters=strategy_hyperparameters,
90              sell_at_the_end=sell_at_the_end,
91              benchmark_data_specifications=benchmark_data_specifications,
92              display_options=display_options,
93              general_settings=general_settings,
94              start_time=start_time,
95              comments=comments
96          )
97
98      else:
99          for sell_parameter in strategy_hyperparameters['sell_parameter_space
    ']:
100             for buy_parameter in strategy_hyperparameters['
    buy_parameter_space']:
101                 strategy_hyperparameters['sell_parameter'] = sell_parameter
    /10
102                 strategy_hyperparameters['buy_parameter'] = buy_parameter/10
103
104                 for index, dates in enumerate(
    list_times_of_split_for_robustness_test):
105                     strategy_hyperparameters['start_time'] = dates[0]
106                     strategy_hyperparameters['end_time'] = dates[1]
107
108                     dfs_intermediate_results = test_strategies(
109                         df_prices,
110                         int_chosen_strategy=int_chosen_strategy,
111                         float_budget_in_usd=float_budget_in_usd,
112                         file_path_with_signal_data=
    file_path_with_signal_data,
113                         margin_loan_rate=margin_loan_rate,
114                         boolean_allow_shorting=boolean_allow_shorting,
115                         list_trading_execution_delay_after_signal_in_hours=
    list_trading_execution_delay_after_signal_in_hours,
116                         dict_crypto_options=dict_crypto_options,
117                         minimum_expected_mispricing_trigger_in_percent=
    minimum_expected_mispricing_trigger_in_percent,
118                         strategy_hyperparameters=strategy_hyperparameters,
119                         sell_at_the_end=sell_at_the_end,
120                         benchmark_data_specifications=
    benchmark_data_specifications,
121                         display_options=display_options,
122                         constraints=constraints,
123                         general_settings=general_settings,
124                         start_time=start_time,
```

```
                        comments=comments
                    )

                    try:
                        dfs_results[0] = dfs_results[0].append(
                            dfs_intermediate_results[0]
                        )
                        dfs_results[1] = dfs_results[1].append(
                            dfs_intermediate_results[1]
                        )
                    except:
                        dfs_results = dfs_intermediate_results

    save_dataframe_to_csv(
        dfs_results[0],
        string_name='backtesting_result_metrics',
        string_directory=display_options['string_results_directory'],
    )

    comments = {
        **{
            'file_path_with_price_data': file_path_with_price_data,
            'file_path_with_token_data': file_path_with_token_data,
            'name_of_foreign_key_in_price_dat_table':
    name_of_foreign_key_in_price_data_table,
            'name_of_foreign_key_in_token_metadata_table':
    name_of_foreign_key_in_token_metadata_table,
            'float_budget_in_usd': float_budget_in_usd,
            'margin_loan_rate': margin_loan_rate,
            'boolean_allow_shorting': boolean_allow_shorting,
            'list_trading_execution_delay_after_signal_in_hours':
    list_trading_execution_delay_after_signal_in_hours,
            'dict_crypto_options': dict_crypto_options,
            'minimum_expected_mispricing_trigger_in_percent':
    minimum_expected_mispricing_trigger_in_percent,
            'strategy_hyperparameters': strategy_hyperparameters,
            'sell_at_the_end': sell_at_the_end,
        },
        **comments
    }


    return {
        'df_performance': dfs_results[0],
        'df_trading_journal': dfs_results[1],
        'comments': comments
    }

def plot_robustness_heatmap(
```

```
170         df_performance,
171         display_options,
172         boolean_save=True,
173         boolean_show=False
174     ):
175     """Plots each row that is contained in the performance table.
176 
177     With the attributes 'start date' on the x-axis and 'duration' on the y-
        axis.
178     """
179 
180     dict_heatmap = {
181         'Begin time of tested interval': [],
182         'Duration of the tested interval': [],
183         'USD annualized ROI (from first to last trade)': []
184     }
185 
186     for index, row in df_performance.iterrows():
187         begin_date = row['Begin time of tested interval']
188         strategy_duration_in_days = int(
189             row['Duration of the tested interval'].days
190         )
191         roi = row['USD annualized ROI (from first to last trade)']
192         dict_heatmap['Begin time of tested interval'].append(begin_date)
193         dict_heatmap['Duration of the tested interval'].append(
        strategy_duration_in_days)
194         dict_heatmap['USD annualized ROI (from first to last trade)'].append
        (roi)
195 
196     # x_axis = array(dict_heatmap['Begin time of tested interval'], dtype=
        datetime)
197     x_axis = [i.to_pydatetime() for i in dict_heatmap['Begin time of tested
        interval']]
198     y_axis = dict_heatmap['Duration of the tested interval']
199     z_values = dict_heatmap['USD annualized ROI (from first to last trade)']
200 
201     figure, axis = plt.subplots()
202     plt.ylabel('Duration (in days)')
203     axis.set_xlabel('Begin time of tested interval')
204     plt.xticks(rotation=70)
205     plt.title('ROI heatmap (in percent)')
206     points = axis.scatter(
207         x_axis,
208         y_axis,
209         c=z_values,
210         s=1300,# Size of scatters
211         cmap='RdYlGn',# Taken from here: https://matplotlib.org/users/
        colormaps.html
212         marker="s"
```

```
    )
    figure.colorbar(points)

    string_directory = display_options['string_results_directory']
    result_no = len(
        [name for name in os.listdir(string_directory) if os.path.isfile(
            os.path.join(
                string_directory,
                name
            )
        )]
    ) / 2

    number_of_result_files_plus_1 = 1 + int(result_no)

    if boolean_save:
        plt.savefig(string_directory + '/robustness_heatmap_' + str(
    number_of_result_files_plus_1) + '.png')

    if boolean_show:
        plt.show()

def backtest_visualizer(
        file_path_with_price_data,
        int_chosen_strategy,
        dict_crypto_options,
        benchmark_data_specifications,
        display_options,
        strategy_hyperparameters,
        constraints,
        general_settings,
        margin_loan_rate,
        file_path_with_token_data=None, # Only for multi-asset strategies.
        name_of_foreign_key_in_price_data_table=None,
        name_of_foreign_key_in_token_metadata_table=None,
        float_budget_in_usd=10000,
        boolean_allow_shorting=False,
        list_trading_execution_delay_after_signal_in_hours={
            'delay_before_buying': 24,
            'delay_before_selling': 24
        },
        minimum_expected_mispricing_trigger_in_percent={
            'mispricing_when_buying': 0.0,
            'mispricing_when_selling': 0.0
        },
        sell_at_the_end=True,
        list_times_of_split_for_robustness_test=None,
        file_path_with_signal_data=None,
        comments={}
```

```
    ):
    """Prints and plots results from the performance table."""

    start_time = datetime.now()
    dict_backtesting = backtest(
        file_path_with_price_data=file_path_with_price_data,
        file_path_with_token_data=file_path_with_token_data,
        name_of_foreign_key_in_price_data_table=
name_of_foreign_key_in_price_data_table,
        name_of_foreign_key_in_token_metadata_table=
name_of_foreign_key_in_token_metadata_table,
        float_budget_in_usd=float_budget_in_usd,
        margin_loan_rate=margin_loan_rate,
        boolean_allow_shorting=boolean_allow_shorting,
        list_trading_execution_delay_after_signal_in_hours=
list_trading_execution_delay_after_signal_in_hours,
        dict_crypto_options=dict_crypto_options,
        minimum_expected_mispricing_trigger_in_percent=
minimum_expected_mispricing_trigger_in_percent,
        strategy_hyperparameters=strategy_hyperparameters,
        sell_at_the_end=sell_at_the_end,
        list_times_of_split_for_robustness_test=
list_times_of_split_for_robustness_test,
        benchmark_data_specifications=benchmark_data_specifications,
        int_chosen_strategy=int_chosen_strategy,
        file_path_with_signal_data=file_path_with_signal_data,
        display_options=display_options,
        constraints=constraints,
        general_settings=general_settings,
        start_time=start_time,
        comments=comments
    )
    end_time = datetime.now()
    elapsed_time = end_time - start_time
    asset_name = list(dict_crypto_options.keys())[0]
    print("\n")
    print("Execution started at: " + str(start_time) + ", finished at: " +
str(end_time) + ", elapsed time:", str(elapsed_time.total_seconds()) + "s
")
    print("****  Performance overview ->", asset_name, "<- ****")
    print("Key metrics")
    print("*USD annualized ROI (from first to last trade):", "{0:.2%}".
format(dict_backtesting['df_performance']['USD annualized ROI (from first
 to last trade)'].iloc[-1]))
    if type(dict_backtesting['df_performance']['Cryptocurrency annualized
ROI delta (from first to last trade)'].iloc[-1]) is str:
        print("*Cryptocurrency annualized ROI delta (from first to last
trade):", dict_backtesting['df_performance']['Cryptocurrency annualized
ROI delta (from first to last trade)'].iloc[-1])
```

```
    else:
        print("*Cryptocurrency annualized ROI delta (from first to last trade):", "{0:.2%}".format(dict_backtesting['df_performance']['Cryptocurrency annualized ROI delta (from first to last trade)'].iloc[-1]))
    print("Number of trades:", len(dict_backtesting['df_trading_journal']))
    print("Other metrics")
    print("**** Assumptions ****")
    print("*Budget:", dict_backtesting['comments']['float_budget_in_usd'])
    print("*Check arguments and parameter defaults for a full list of assumptions.*")

    plot_robustness_heatmap(
        dict_backtesting['df_performance'],
        display_options=display_options,
        boolean_show=display_options['boolean_plot_heatmap']
    )

    return dict_backtesting
```

Code Snippet 35: `components/_0_wrappers.py` – The functions from this module manage the tool's functionality and aggregate batch results.

```
"""This module contains functions that load, save, and manipulate data."""

# For counting the number of result files in the "results" folder to allow for
# consistent file versioning when saving the results as a csv file.
from os import listdir, path

# For managing tables properly
from pandas import read_csv, merge
import pandas


def load_data(
        file_path_with_price_data,
        strategy_hyperparameters,
        file_path_with_token_data=None,
        name_of_foreign_key_in_price_data_table=None,
        name_of_foreign_key_in_token_metadata_table=None
    ):
    """Loads price data and metadata.

    This function loads prices and token metadata (token data). It also uses the
    date as an index.
    """

```

```
    # Handles different csv formats and handles program calls from root and from
    # /backtesting
    try:
        df_prices = read_csv(
            file_path_with_price_data,
            sep=',',
            usecols=['datetime', 'token_itin', 'price'],
            parse_dates=['datetime'],
            infer_datetime_format=True
        )
    except (pandas.errors.ParserError, FileNotFoundError, ValueError):
        df_prices = read_csv(
            file_path_with_price_data,
            sep=';',
            usecols=['datetime', 'token_itin', 'price'],
            parse_dates=['datetime'],
            infer_datetime_format=True
        )
    # Enrich data with metadata
    if file_path_with_token_data is not None:
        try:
            df_token_metadata = read_csv(
                file_path_with_token_data,
                sep=';',
                usecols=['token_itin', 'token_address_eth']
            )
        except ValueError:
            df_token_metadata = read_csv(
                file_path_with_token_data,
                sep=',',
                usecols=['token_itin', 'token_address_eth']
            )

        df_prices = merge_data(
            df_prices=df_prices,
            df_token_metadata=df_token_metadata,
            name_of_foreign_key_in_price_data_table=name_of_foreign_key_in_price_data_table,
            name_of_foreign_key_in_token_metadata_table=name_of_foreign_key_in_token_metadata_table
        )

    df_prices.rename(
        columns={
            strategy_hyperparameters['prices_table_id_column_name']: 'id'
        },
        inplace=True
    )
```

```
71
72      df_prices.set_index(['datetime', 'id'], inplace=True)
73
74      df_prices.sort_index(level=['datetime', 'id'], ascending=[1, 1], inplace
        =True)
75
76      if not df_prices.index.is_lexsorted():
77          raise ValueError('df_prices is not lexsorted.')
78
79      return df_prices
80
81  def merge_data(
82          df_prices,
83          df_token_metadata,
84          name_of_foreign_key_in_price_data_table,
85          name_of_foreign_key_in_token_metadata_table
86      ):
87      """Enriches price data with metadata about the tokens."""
88
89      df_prices_enriched_with_metadata = merge(
90          df_prices,
91          df_token_metadata,
92          left_on=name_of_foreign_key_in_price_data_table,
93          right_on=name_of_foreign_key_in_token_metadata_table
94      )
95
96      return df_prices_enriched_with_metadata
97
98  def prepare_data(
99          file_path_with_price_data,
100         strategy_hyperparameters,
101         file_path_with_token_data=None,
102         name_of_foreign_key_in_price_data_table=None,
103         name_of_foreign_key_in_token_metadata_table=None
104     ):
105     """Loads prices and token data and joins those into a pandas DataFrame.
        """
106
107     df_prices = load_data(
108         file_path_with_price_data=file_path_with_price_data,
109         file_path_with_token_data=file_path_with_token_data,
110         strategy_hyperparameters=strategy_hyperparameters,
111         name_of_foreign_key_in_price_data_table=
        name_of_foreign_key_in_price_data_table,
112         name_of_foreign_key_in_token_metadata_table=
        name_of_foreign_key_in_token_metadata_table
113     )
114
115     return df_prices
```

```
116
117 def save_dataframe_to_csv(
118         df_prices,
119         string_name,
120         string_directory,
121     ):
122     """Saves a pandas Dataframe to csv without overwriting existing files.
    """
123     result_no = len(
124         [name for name in listdir(string_directory) if path.isfile(
125             path.join(
126                 string_directory,
127                 name
128             )
129         )]
130     ) / 2
131
132
133     number_of_result_files_plus_1 = 1 + int(result_no)
134     df_prices.to_csv(
135         string_directory + '/' + string_name + '_' + str(
    number_of_result_files_plus_1) + '.csv'
136     )
```

Code Snippet 36: `components/_1_data_preparation.py` – The functions from this module load the price data, merge the price data with additional resources (if specified), and load the signals (if specified).

```
1 """Contains strategies and order execution functionality."""
2
3 # For deep-copied dictionaries
4 from copy import deepcopy
5
6 # For managing dates
7 from datetime import datetime, timedelta
8
9 # For managing tables properly
10 from numpy import where
11 from pandas import DataFrame, read_csv, read_excel, date_range, Timedelta,
    Timestamp
12
13 # For mathematical operations
14 from math import isnan
15
16 # For nice-looking progress bars
17 from tqdm import tqdm
18
19 # For white noise strategy
20 from random import choice
```

```
21
22 from _helper_functions import find_dataframe_value_with_keywords, \
23     find_price, calculate_portfolio_value, calculate_relative_gross_exposure
24 from _1_data_preparation import save_dataframe_to_csv
25 from _3_performance_evaluation import evaluate_performance
26
27
28 def initialize_trading_journal():
29     """Initializes a pandas DataFrame that serves as a trading journal.
30
31     Initialization is important for determining the column order.
32     """
33     df_trading_journal = DataFrame(columns=[
34         'datetime',
35         'Cash',
36         'Cash before',
37         'Asset',
38         'Buy or sell',
39         'Number bought',
40         'Price (quote without any fees)',
41         'Value bought',
42         'Portfolio value',
43         'Dict of assets in portfolio',
44         'Absolute fees (as absolute)',
45         'Current equity margin',
46         'Exposure (in currency)',
47         'Exposure (number)',
48         'Gross exposure',
49         'Interest paid',
50         'Money spent',
51         'Relative fees (as absolute)',
52         'Relative fees (as relative)',
53         'Strategy ID',
54         'Total exposure',
55         'Total fees (as absolute)',
56         'Total fees (as relative)'
57     ])
58
59     return df_trading_journal
60
61 def execute_order(
62         boolean_buy,
63         index,
64         date,
65         crypto_key,
66         number_to_be_bought,
67         strategy_id,
68         df_prices,
69         df_trading_journal,
```

```
70          margin_loan_rate,
71          fees,
72          float_budget_in_usd,
73          price,
74          display_options,
75          constraints,
76          general_settings,
77          boolean_allow_partially_filled_orders
78      ):
79      """Executes all kinds of orders.
80
81      Can handle more than one order per point in time by subsequently calling
82      this function.
83      """
84      if boolean_buy and (number_to_be_bought < 0) or not boolean_buy and (
        number_to_be_bought > 0):
85          raise ValueError(f'boolean_buy: {boolean_buy} and
        number_to_be_bought: {number_to_be_bought} is contradictory.')
86
87      order = {
88          'datetime': date,
89          'Strategy ID': strategy_id,
90          'Asset': crypto_key,
91          'Buy or sell': boolean_buy
92      }
93
94      if len(df_trading_journal) > 0:
95          available_funds = df_trading_journal['Cash'].iloc[-1] - constraints[
        'minimum_cash']
96          order['Dict of assets in portfolio'] = deepcopy(df_trading_journal['
        Dict of assets in portfolio'].iloc[-1])
97          order['Cash before'] = df_trading_journal['Cash'].iloc[-1]
98      else:
99          available_funds = float_budget_in_usd
100         order['Dict of assets in portfolio'] = {crypto_key: 0}
101         order['Cash before'] = float_budget_in_usd
102
103     def reduce_quantity_until_max_gross_exposure_is_met(
104             number_to_be_bought,
105             df_prices,
106             dict_of_assets_in_portfolio,
107             time,
108             display_options,
109             constraints,
110             rounding_accuracy,
111             cash_value,
112             general_settings,
113             crypto_key
114         ):
```

```
115         # Checks if the maximum_gross_exposure constraint is violated and
116         # reduces the order volumne step-by-step in case of a violation
117         # until the exposure falls within the constraint.
118         initial_number_to_be_bought = number_to_be_bought
119         dict_of_assets_in_portfolio[crypto_key] =
    dict_of_assets_in_portfolio[crypto_key] + number_to_be_bought
120         for reduction_step in range(1, 100):
121             dict_of_assets_in_portfolio = deepcopy(
    dict_of_assets_in_portfolio)
122             relative_gross_exposure = calculate_relative_gross_exposure(
123                 df_prices=df_prices,
124                 dict_of_assets_in_portfolio=dict_of_assets_in_portfolio,
125                 time=time,
126                 display_options=display_options,
127                 constraints=constraints,
128                 rounding_accuracy=rounding_accuracy,
129                 cash_value=cash_value,
130                 general_settings=general_settings
131             )
132
133             # Stop reduction as soon as constraint is met, reduce if
134             # constraint is violated.
135             if relative_gross_exposure <= constraints['
    maximum_gross_exposure']:
136                 break
137             else:
138                 dict_of_assets_in_portfolio[crypto_key] = ((100 -
    reduction_step) / 100) * initial_number_to_be_bought
139
140         return initial_number_to_be_bought
141
142     def quantity_that_can_be_bought_given_budget(order,
    dict_of_assets_in_portfolio, boolean_allow_partially_filled_orders,
    number_to_be_bought, available_funds, general_settings, display_options,
    date):
143         initial_number_to_be_bought = number_to_be_bought
144
145         number_to_be_bought = min(
146             max(round(available_funds / price, general_settings['
    rounding_decimal_places_for_security_quantities']), 0),
147             number_to_be_bought
148         )
149
150         # Todo: Individual asset constraints.
151         # constraints['maximum_individual_asset_exposure_all']
152         dict_of_assets_in_portfolio = deepcopy(dict_of_assets_in_portfolio)
153         number_to_be_bought =
    reduce_quantity_until_max_gross_exposure_is_met(
154             number_to_be_bought=number_to_be_bought,
```

```
155             df_prices=df_prices,
156             dict_of_assets_in_portfolio=dict_of_assets_in_portfolio,
157             time=date,
158             display_options=display_options,
159             constraints=constraints,
160             rounding_accuracy=general_settings['rounding_decimal_places'],
161             cash_value=order['Cash before'] - (number_to_be_bought * price)
    - fees['absolute_fee_buy_order'] - round(
162                 fees['percentage_buying_fees_and_spread'] * price *
    number_to_be_bought,
163                 general_settings['rounding_decimal_places']
164             ),
165             general_settings=general_settings,
166             crypto_key=crypto_key
167         )
168 
169         if boolean_allow_partially_filled_orders:
170             if number_to_be_bought == 0 and initial_number_to_be_bought !=
    0:
171                 if display_options['warning_buy_order_could_not_be_filled']:
172                     print(f'Order execution warning: Buy order for {
    initial_number_to_be_bought} units of {crypto_key} could not be filled.')
173             elif number_to_be_bought < initial_number_to_be_bought:
174                 if display_options['warning_buy_order_could_not_be_filled']:
175                     print(f'Order execution warning: Buy order for {
    initial_number_to_be_bought} units of {crypto_key} could only partially
    be filled: {number_to_be_bought} units bought.')
176             elif number_to_be_bought > initial_number_to_be_bought:
177                 raise ValueError(f'It is not possible that the signal
    quantity {initial_number_to_be_bought} is lower than the final quantity {
    number_to_be_bought} for {order["Asset"]}.')
178             return number_to_be_bought
179         else:
180             if initial_number_to_be_bought != number_to_be_bought:
181                 raise InputError('Quantity {number_to_be_bought} for {
    crypto_key} cannot be covered with the given budget or constraints.
    Maximum {max_possible_quantity} units can be bought.')
182             else:
183                 return number_to_be_bought
184 
185     def quantity_that_can_be_sold_given_portfolio(order,
    dict_of_assets_in_portfolio, boolean_allow_partially_filled_orders,
    number_to_be_bought, df_trading_journal, general_settings,
    display_options, date):
186         initial_number_to_be_bought = number_to_be_bought
187 
188         # Todo: Individual asset constraints.
189         # constraints['maximum_individual_asset_exposure_all']
190 
```

```
        positive_quantity = (-1) * number_to_be_bought
        try:
            number_to_be_bought = (-1) * min(
                positive_quantity,
                df_trading_journal['Dict of assets in portfolio'].iloc[-1][
crypto_key]
            )
        except IndexError:
            number_to_be_bought = 0

        dict_of_assets_in_portfolio = deepcopy(order['Dict of assets in
portfolio'])
        number_to_be_bought =
reduce_quantity_until_max_gross_exposure_is_met(
            number_to_be_bought=number_to_be_bought,
            df_prices=df_prices,
            dict_of_assets_in_portfolio=dict_of_assets_in_portfolio,
            time=date,
            display_options=display_options,
            constraints=constraints,
            rounding_accuracy=general_settings['rounding_decimal_places'],
            cash_value=order['Cash before'] - (number_to_be_bought * price)
- fees['absolute_fee_buy_order'] - round(
                fees['percentage_buying_fees_and_spread'] * price *
number_to_be_bought,
                general_settings['rounding_decimal_places']
            ),
            general_settings=general_settings,
            crypto_key=crypto_key
        )

        if boolean_allow_partially_filled_orders:
            if number_to_be_bought == 0 and initial_number_to_be_bought !=
0:
                if display_options['warning_buy_order_could_not_be_filled']:
                    print(f'Order execution warning: Buy order for {
number_to_be_bought} units of {crypto_key} could not be filled.')
            elif number_to_be_bought > initial_number_to_be_bought:
                if display_options['warning_buy_order_could_not_be_filled']:
                    print(f'Order execution warning: Buy order for {
number_to_be_bought} units of {crypto_key} could not be filled.')
            elif number_to_be_bought < initial_number_to_be_bought:
                raise ValueError('It is not possible that the signal
quantity {initial_number_to_be_bought} is higher (less assets sold) than
the final quantity {number_to_be_bought} for {order["Asset"]}.')
            return number_to_be_bought
        else:
            if number_to_be_bought != initial_number_to_be_bought:
                raise InputError('Quantity {number_to_be_bought} for {
```

```
    crypto_key} cannot be sold with the given assets or constraints. Maximum
    {number_to_be_bought} units can be sold.')
230             else:
231                 return number_to_be_bought
232 
233     if boolean_buy:
234         number_to_be_bought = quantity_that_can_be_bought_given_budget(
235             order=order,
236             dict_of_assets_in_portfolio=order['Dict of assets in portfolio'
    ],
237             boolean_allow_partially_filled_orders=
    boolean_allow_partially_filled_orders,
238             number_to_be_bought=number_to_be_bought,
239             available_funds=available_funds,
240             general_settings=general_settings,
241             display_options=display_options,
242             date=date
243         )
244     else:
245         number_to_be_bought = quantity_that_can_be_sold_given_portfolio(
246             order=order,
247             dict_of_assets_in_portfolio=order['Dict of assets in portfolio'
    ],
248             boolean_allow_partially_filled_orders=
    boolean_allow_partially_filled_orders,
249             number_to_be_bought=number_to_be_bought,
250             df_trading_journal=df_trading_journal,
251             general_settings=general_settings,
252             display_options=display_options,
253             date=date
254         )
255 
256     if isnan(price):
257         number_to_be_bought = 0
258         price = 0
259 
260     order['Dict of assets in portfolio'][crypto_key] = order['Dict of assets
     in portfolio'][crypto_key] + number_to_be_bought
261 
262     if number_to_be_bought != 0:
263         if boolean_buy:
264             order['Absolute fees (as absolute)'] = fees['
    absolute_fee_buy_order']
265             order['Relative fees (as absolute)'] = round(
266                 fees['percentage_buying_fees_and_spread'] * price *
    number_to_be_bought,
267                 general_settings['rounding_decimal_places']
268             )
269             order['Relative fees (as relative)'] = fees['
```

```
    percentage_buying_fees_and_spread']
270             order['Total fees (as absolute)'] = fees['absolute_fee_buy_order
    '] + order['Relative fees (as absolute)']
271             order['Total fees (as relative)'] = round(
272                 order['Total fees (as absolute)'] / (number_to_be_bought *
    price),
273                 general_settings['rounding_decimal_places']
274             )
275         else:
276             order['Absolute fees (as absolute)'] = fees['
    absolute_fee_sell_order']
277             order['Relative fees (as absolute)'] = round(
278                 fees['percentage_selling_fees_and_spread'] * price *
    number_to_be_bought * (-1),
279                 general_settings['rounding_decimal_places']
280             )
281             order['Relative fees (as relative)'] = fees['
    percentage_selling_fees_and_spread']
282             order['Total fees (as absolute)'] = fees['
    absolute_fee_sell_order'] + order['Relative fees (as absolute)']
283             order['Total fees (as relative)'] = round(
284                 order['Total fees (as absolute)'] / (number_to_be_bought *
    price) * (-1),
285                 general_settings['rounding_decimal_places']
286             )
287     else:
288         order['Absolute fees (as absolute)'] = 0
289         order['Relative fees (as absolute)'] = 0
290         order['Relative fees (as relative)'] = 0
291         order['Total fees (as absolute)'] = 0
292         order['Total fees (as relative)'] = 0
293 
294     if len(df_trading_journal) > 0:
295         # Date conversion from string to date format; datetime is in
    microsecond format
296         previous_time = df_trading_journal['datetime'].iloc[-1]
297         days_since_last_order = date - previous_time
298 
299         order['Number bought'] = number_to_be_bought
300         order['Value bought'] = round(price * number_to_be_bought,
    general_settings['rounding_decimal_places'])
301         if days_since_last_order == timedelta(seconds=0):
302             order['Interest paid'] = 0
303         else:
304             order['Interest paid'] = round(
305                 (
306                     (1 - df_trading_journal['Current equity margin'].iloc
    [-1])
307                     * (
```

```
                        df_trading_journal['Portfolio value'].iloc[-1]
                        * margin_loan_rate
                    )
                ) ** (
                    (
                        days_since_last_order.total_seconds() / 86400
                    ) / 365
                ),
                general_settings['rounding_decimal_places']
            ) # "/ 86400" because one day has 86400 seconds

        order['Money spent'] = round(
            number_to_be_bought * (
                price
            ) + (
                order['Total fees (as absolute)'] + order['Interest paid']
            ), general_settings['rounding_decimal_places']
        ) # Todo But everything that can be bought minus fees and other
    costs

        order['Cash'] = round(
            df_trading_journal['Cash'].iloc[-1] - order['Money spent'],
            general_settings['rounding_decimal_places']
        )

    else:
        # For initial row
        order['Number bought'] = number_to_be_bought
        order['Value bought'] = price * number_to_be_bought
        order['Interest paid'] = 0.0
        order['Money spent'] = number_to_be_bought * (
            price
        ) + (
            + order['Total fees (as absolute)']
            + order['Interest paid']
        ) # Todo But everything that can be bought minus fees and other
    costs

        order['Cash'] = round(
            float_budget_in_usd - order['Money spent'],
            general_settings['rounding_decimal_places']
        )

    order['Price (quote without any fees)'] = price

    # Todo: For now just 1
    order['Current equity margin'] = 1

    order['Exposure (number)'] = order['Dict of assets in portfolio'][order[
```

```
    'Asset']]

    order['Exposure (in currency)'] = round(
        price * order['Exposure (number)'],
        general_settings['rounding_decimal_places']
    )

    order['Portfolio value'] = calculate_portfolio_value(
        df_prices=df_prices,
        dict_of_assets_in_portfolio=order['Dict of assets in portfolio'],
        time=order['datetime'],
        display_options=display_options,
        constraints=constraints,
        cash_value=order['Cash'],
        rounding_accuracy=general_settings['rounding_decimal_places']
    )

    order['Total exposure'] = round(
        order['Portfolio value'] - order['Cash'],
        general_settings['rounding_decimal_places']
    )

    assert round(order['Total exposure'], 2) == round(
    calculate_portfolio_value(
        df_prices=df_prices,
        dict_of_assets_in_portfolio=order['Dict of assets in portfolio'],
        time=order['datetime'],
        display_options=display_options,
        constraints=constraints,
        rounding_accuracy=general_settings['rounding_decimal_places']
    ), 2)

    order['Gross exposure'] = calculate_relative_gross_exposure(
        df_prices=df_prices,
        dict_of_assets_in_portfolio=order['Dict of assets in portfolio'],
        time=order['datetime'],
        display_options=display_options,
        constraints=constraints,
        rounding_accuracy=general_settings['rounding_decimal_places'],
        cash_value=order['Cash'],
        general_settings=general_settings
    )

    return order

def prepare_signal_list_ii(
        file_path_with_signal_data,
        strategy_hyperparameters
    ):
```

```
    try:
        try:
            df_signals = read_csv(
                file_path_with_signal_data,
                sep=',',
                parse_dates=True,
                infer_datetime_format=True,
                index_col=['datetime', 'id']
            )
        except UnicodeDecodeError:
            df_signals = read_excel(
                file_path_with_signal_data,
                sep=',',
                parse_dates=True,
                infer_datetime_format=True,
                index_col=['datetime', 'id']
            )
    except:
        try:
            df_signals = read_csv(
                'backtesting/' + file_path_with_signal_data,
                sep=',',
                parse_dates=True,
                infer_datetime_format=True,
                index_col=['datetime', 'id']
            )
        except:
            raise ValueError("Please setup a signal table. A signal table needs the following columns: 'datetime', 'id' (hexadecimal ERC20 token identifier), 'signal_strength' (numeric that is used to infer buy or sell orders)")

    # Only use top 50 tokens
    print(f'\nNumber of signals before dropping non-top-50 tokens: {len(df_signals)}')
    comments['Number of signals before dropping non-top-50 tokens'] = len(df_signals)
    print(f'Number of unique eth IDs before: {len(df_signals["id"].unique())}')
    comments['Number of unique eth IDs before'] = len(df_signals['id'].unique())
    df_top50 = read_csv('strategy_tables/Top50Tokens.csv', sep=',')
    list_top50 = df_top50['a.ID'].values.tolist()
    df_signals = df_signals[df_signals['id'].isin(list_top50)]
    print(f'Number of signals after dropping non-top-50 tokens: {len(df_signals)}')
    comments['Number of signals after dropping non-top-50 tokens'] = len(df_signals)
    print(f'Number of unique eth IDs after: {len(df_signals["id"].unique())}
```

```
    ')
442 comments['Number of unique eth IDs after'] = len(df_signals['id'].unique
    ())
443
444 # Drop weak signals
445 print(f'\nNumber of signals before dropping weak signals: {len(
    df_signals)}')
446 comments['Number of signals before dropping weak signals'] = len(
    df_signals)
447 df_signals['signal_type'] = where(df_signals["rawSignal"]<=
    strategy_hyperparameters['sell_parameter'], "SELL", where(df_signals["
    rawSignal"]>=strategy_hyperparameters['buy_parameter'], "BUY", "HODL"))
448 #indexNames = df_signals[(df_signals['rawSignal'] >=
    strategy_hyperparameters['sell_parameter']) & (df_signals['rawSignal'] <=
     strategy_hyperparameters['buy_parameter']) ].index # FOr testing
    purposes: 0.03 and 12.0
449 #df_signals.drop(indexNames, inplace=True)
450 df_signals = df_signals[~df_signals['signal_type'].isin(["HODL"])]
451 print(f'Number of signals after dropping weak signals: {len(df_signals)}
    ')
452 comments['Number of signals after dropping weak signals'] = len(
    df_signals)
453
454 # Drop signals that are not covered by ITSA
455 print(f'\nNumber of signals before dropping unavailable data: {len(
    df_signals)}')
456 comments['Number of signals before dropping unavailable data'] = len(
    df_signals)
457 df_signals = df_signals[
458     df_signals['id'].isin(df_prices['token_address_eth'].unique())
459 ]
460 print(f'Number of signals after dropping unavailable data: {len(
    df_signals)}')
461 comments['Number of signals after dropping unavailable data'] = len(
    df_signals)
462
463 # Drop data that is not needed
464 print(f'\nNumber of data points before dropping unnecessary data: {len(
    df_prices)}')
465 comments['Number of data points before dropping unnecessary data'] = len
    (df_prices)
466 df_prices = df_prices[
467     df_prices['token_address_eth'].isin(df_signals['id'].unique())
468 ]
469 print(f'Number of data points after dropping unnecessary data: {len(
    df_prices)}')
470 comments['Number of data points after dropping unnecessary data'] = len(
    df_prices)
471
```

```
472     # Drop signals that have assets that have no prices for the last day.
        This
473     # is necessary because for the final portfolio valuation, there has to
        be a
474     # price for the last day.
475     print(f'\nNumber of signals before dropping unavailable prices: {len(
        df_signals)}')
476     comments['Number of signals before dropping unavailable prices'] = len(
        df_signals)
477     print(f'Number of unique eth IDs before: {len(df_signals["id"].unique())
        }')
478     comments['Number of unique eth IDs before'] = len(df_signals['id'].
        unique())
479 
480     last_signal_date = df_signals['datetime'].iloc[-1]
481     price = None
482 
483     list_of_assets_dropped_due_to_price_lag_at_last_day = []
484 
485     for asset_in_signal_table in df_prices['token_address_eth'].unique():
486         price = find_price(
487             df_prices,
488             desired_index=index,
489             boolean_allow_older_prices=False,
490             boolean_allow_newer_prices=False,
491             boolean_warnings=display_options['warning_no_price_for_last_day'
        ],
492             boolean_errors=False
493         )
494 
495         if price is None:
496             df_signals = df_signals[df_signals.ID != asset_in_signal_table]
497             if display_options['warning_no_price_for_last_day']:
498                 print(f'{asset_in_signal_table} was dropped because there is
         no price for the last day. {price}')
499             list_of_assets_dropped_due_to_price_lag_at_last_day.append(
500                 asset_in_signal_table
501             )
502         elif isnan(price):
503             df_signals = df_signals[df_signals.ID != asset_in_signal_table]
504             if display_options['warning_no_price_for_last_day']:
505                 print(f'{asset_in_signal_table} was dropped because there is
         only a NaN price for the last day. {price}')
506             list_of_assets_dropped_due_to_price_lag_at_last_day.append(
507                 asset_in_signal_table
508             )
509         elif price == 0:
510             df_signals = df_signals[df_signals.ID != asset_in_signal_table]
511             if display_options['warning_no_price_for_last_day']:
```

```
512                 print(f'asset_in_signal_table was dropped because there is
    only a 0 price for the last day. {price}')
513             list_of_assets_dropped_due_to_price_lag_at_last_day.append(
514                 asset_in_signal_table
515             )
516
517     df_signals = df_signals[~df_signals['id'].isin(
    list_of_assets_dropped_due_to_price_lag_at_last_day)]
518
519     print(f'Number of assets that do not have a price on the last day: {len(
    list_of_assets_dropped_due_to_price_lag_at_last_day)}')
520     comments['Number of assets that do not have a price on the last day'] =
    len(
521         list_of_assets_dropped_due_to_price_lag_at_last_day
522     )
523
524     print(f'Number of signals after dropping unavailable prices: {len(
    df_signals)}')
525     comments['Number of signals after dropping unavailable prices'] = len(
    df_signals)
526
527     print(f'Number of unique eth IDs after: {len(df_signals["id"].unique())}
    ')
528     comments['Number of unique eth IDs after'] = len(df_signals['id'].unique
    ())
529
530     id_column_name = 'token_address_eth'
531
532     for asset_with_possible_later_price in
    list_of_assets_dropped_due_to_price_lag_at_last_day:
533         price = find_price(
534             df_prices,
535             asset_with_possible_later_price,
536             time=datetime(2019, 7, 1),
537             boolean_allow_older_prices=False,
538             boolean_allow_newer_prices=True,
539             boolean_warnings=display_options['warning_no_price_for_last_day'
    ]
540         )
541         if price is None:
542             list_of_assets_dropped_due_to_price_lag_at_last_day.remove(
543                 asset_with_possible_later_price
544             )
545
546     print(f'Number of assets that do not have a price on the last day, but
    on a later day: {len(list_of_assets_dropped_due_to_price_lag_at_last_day)
    }')
547     comments['Number of assets that do not have a price on the last day, but
     on a later day'] = len(
```

```
    list_of_assets_dropped_due_to_price_lag_at_last_day)
548
549     return df_signals, id_column_name
550
551 def prepare_signal_list_san(
552         file_path_with_signal_data,
553         strategy_hyperparameters
554     ):
555     try:
556         df_signals = read_csv(
557             file_path_with_signal_data,
558             sep=',',
559             parse_dates=True,
560             infer_datetime_format=True,
561             index_col=['datetime', 'id']
562         )
563     except:
564         raise ValueError("Please set up a signal table. A signal table needs
     the following columns: 'date' (yyyy-mm-dd), 'signal_strength' (numeric
    that is used to infer buy or sell orders), 'id' (hexadecimal ERC20 token
    identifier or ITIN). You can change the column names in the Excel/CSV
    file to fit this convention or in main.py so that the program adjusts to
    the naming in the table.")
565
566     df_signals['signal_type'] = where(df_signals['signal_strength']<=
    strategy_hyperparameters['sell_parameter'],'SELL',where(df_signals['
    signal_strength']>=strategy_hyperparameters['buy_parameter'],'BUY','HODL'
    ))
567     df_signals = df_signals[~df_signals['signal_type'].isin(['HODL'])]
568
569     return df_signals
570
571 def execute_strategy_multi_asset(
572         df_prices,
573         int_chosen_strategy,
574         float_budget_in_usd,
575         margin_loan_rate,
576         boolean_allow_shorting,
577         list_trading_execution_delay_after_signal_in_hours,
578         dict_crypto_options,
579         minimum_expected_mispricing_trigger_in_percent,
580         strategy_hyperparameters,
581         sell_at_the_end,
582         file_path_with_signal_data,
583         display_options,
584         constraints,
585         general_settings,
586         comments
587     ):
```

```
588     """Filters signals and executes all remaining signals."""
589 
590     df_signals = prepare_signal_list_san(
591         file_path_with_signal_data,
592         strategy_hyperparameters
593     )
594 
595     df_signals = df_signals.loc[strategy_hyperparameters['start_time']:
    strategy_hyperparameters['end_time'], ]
596 
597     if display_options['boolean_test']:
598         # Drop non-test signals
599         print('Warning: Test run!')
600         print(f'\nNumber of signals before dropping non-test signals: {len(
    df_signals)}')
601         comments['Number of signals before dropping non-test signals'] = len
    (df_signals)
602         list_indexes_to_be_dropped = []
603         df_signals = df_signals.sample(frac=0.05, random_state=0)
604         df_signals.sort_index(inplace=True)
605         print(f'Number of signals after dropping non-test signals: {len(
    df_signals)}')
606         comments['Number of signals after dropping non-test signals'] = len(
    df_signals)
607 
608     df_trading_journal = initialize_trading_journal()
609 
610     for index, row in tqdm(df_signals.iterrows(), desc='Going through
    signals', unit='signal'):
611         boolean_buy = None
612 
613         crypto_key = index[1]
614 
615         price = find_price(
616             df_prices,
617             desired_index=index,
618             boolean_allow_older_prices=False,
619             boolean_allow_newer_prices=False,
620             boolean_warnings=display_options['
    warning_no_price_during_execution']
621         )
622 
623         if price is not None and price > 0:
624             if row['signal_type'] == "BUY":
625                 boolean_buy = True
626 
627                 # Check if minimum is already crossed
628                 if len(df_trading_journal) > 0:
629                     if df_trading_journal['Cash'].iloc[-1] <= constraints['
```

```
    minimum_cash']:
630                         allow_buy_orders = False
631
632                     else:
633                         allow_buy_orders = True
634
635                 else:
636                     allow_buy_orders = True
637
638                 if allow_buy_orders:
639                     try:
640                         float_budget_in_usd = df_trading_journal['Cash'].
    iloc[-1]
641                     except:
642                         pass
643
644                     number_to_be_bought = round(
645                         (
646                             float_budget_in_usd
647                             - constraints['minimum_cash']
648                         ) / price,
649                         general_settings['
    rounding_decimal_places_for_security_quantities']
650                     )
651
652                 else:
653                     number_to_be_bought = None
654
655                 if number_to_be_bought is not None:
656                     number_to_be_bought = number_to_be_bought *
    strategy_hyperparameters['maximum_relative_exposure_per_buy']
657
658             elif row['signal_type'] == "SELL":
659                 boolean_buy = False
660                 # Check if exposure for this asset is zero and skip order if
     so
661
662                 if not (
663                     (
664                         boolean_buy == False
665                     ) and (
666                         find_dataframe_value_with_keywords(
667                             df_trading_journal,
668                             search_term_1=crypto_key,
669                             search_column_name_1='Asset'
670                         )
671                     ) is None
672                 ):
673                     number_to_be_bought = (-1) * (
```

```
674                         find_dataframe_value_with_keywords(
675                             df_trading_journal,
676                             search_term_1=crypto_key,
677                             search_column_name_1='Asset',
678                             output_column_name='Exposure (number)',
679                             first_last_or_all_elements='Last'
680                         )
681                     )
682 
683                 else:
684                     number_to_be_bought = None
685 
686             else:
687                 raise ValueError('Ambiguous signal:', row['signal_type'])
688 
689             try:
690                 amount = df_trading_journal['Cash'].iloc[-1]
691             except:
692                 amount = float_budget_in_usd
693 
694             if number_to_be_bought is None:
695                 number_to_be_bought = 0
696 
697             number_to_be_bought = round(number_to_be_bought,
    general_settings['rounding_decimal_places_for_security_quantities'])
698 
699             dataseries_trading_journal = execute_order(
700                 boolean_buy=boolean_buy,
701                 index=index,
702                 date=index[0],
703                 strategy_id=int_chosen_strategy,
704                 crypto_key=crypto_key,
705                 number_to_be_bought=number_to_be_bought,
706                 df_prices=df_prices,
707                 df_trading_journal=df_trading_journal,
708                 margin_loan_rate=margin_loan_rate,
709                 float_budget_in_usd=float_budget_in_usd,
710                 price=price,
711                 fees={
712                     'absolute_fee_buy_order':dict_crypto_options['general'][
    'absolute_fee_buy_order'],
713                     'absolute_fee_sell_order':dict_crypto_options['general'
    ]['absolute_fee_sell_order'],
714                     'percentage_buying_fees_and_spread':dict_crypto_options[
    'general']['percentage_buying_fees_and_spread'],
715                     'percentage_selling_fees_and_spread':dict_crypto_options
    ['general']['percentage_selling_fees_and_spread']
716                 },
717                 display_options=display_options,
```

```
718                 constraints=constraints,
719                 general_settings=general_settings,
720                 boolean_allow_partially_filled_orders=
    strategy_hyperparameters['boolean_allow_partially_filled_orders']
721             )
722 
723             df_trading_journal = df_trading_journal.append(
724                 dataseries_trading_journal,
725                 ignore_index=True
726             )
727 
728     comments['constraints'] = constraints
729     comments['general_settings'] = general_settings
730 
731     dict_return = {
732         'df_trading_journal': df_trading_journal,
733         'Strategy ID': '3',
734         'Strategy label': 'X',
735         'strategy_hyperparameters': strategy_hyperparameters,
736         'comments': comments
737     }
738 
739     save_dataframe_to_csv(
740         df_trading_journal,
741         'trading_journal',
742         string_directory=display_options['string_results_directory'],
743     )
744 
745     return dict_return
746 
747 def execute_strategy_white_noise(
748         df_prices,
749         int_chosen_strategy,
750         float_budget_in_usd,
751         margin_loan_rate,
752         list_trading_execution_delay_after_signal_in_hours,
753         dict_crypto_options,
754         minimum_expected_mispricing_trigger_in_percent,
755         strategy_hyperparameters,
756         display_options,
757         constraints,
758         general_settings,
759         sell_at_the_end,
760         comments,
761         boolean_allow_shorting=False,
762     ):
763     """Executes buy and sell orders in alternating order.
764 
765     Has positive average exposure and is therefore not expected to yield a
```

```
    gross
    return of zero.
    """
    df_prices = df_prices.loc[(slice(strategy_hyperparameters['start_time'],
     strategy_hyperparameters['end_time']), strategy_hyperparameters['id']),
    :]

    df_trading_journal = initialize_trading_journal()

    usd_safety_buffer = 100

    # Single-asset only
    crypto_key = strategy_hyperparameters['id']

    for index, row in df_prices.iterrows():
        try:
            amount = df_trading_journal['Cash'].iloc[-1]
        except:
            amount = float_budget_in_usd

        if choice(['buy', 'sell']) == 'buy':
            # Determine the number of assets to be bought or sold
            try:
                float_budget_in_usd = df_trading_journal['Cash'].iloc[-1]
            except:
                pass

            number_to_be_bought = round(
                (
                    float_budget_in_usd - usd_safety_buffer
                ) / row['price'],
                general_settings['
rounding_decimal_places_for_security_quantities']
            )

            if number_to_be_bought < 0:
                number_to_be_bought = 0

            boolean_buy = True

        else:
            # Determine the number of assets to be bought or sold
            try:
                number_to_be_bought = (-1) * df_trading_journal[
                    'Exposure (number)'
                ].iloc[-1]
            except:
                number_to_be_bought = 0

```

```
811             boolean_buy = False
812
813         dataseries_trading_journal = execute_order(
814             df_prices=df_prices,
815             df_trading_journal=df_trading_journal,
816             boolean_buy=boolean_buy,
817             index=index,
818             date=index[0],
819             strategy_id=int_chosen_strategy,
820             crypto_key=crypto_key,
821             number_to_be_bought=number_to_be_bought,
822             margin_loan_rate=margin_loan_rate,
823             float_budget_in_usd=float_budget_in_usd,
824             price=row['price'],
825             fees={
826                 'absolute_fee_buy_order':dict_crypto_options['general']['
    absolute_fee_buy_order'],
827                 'absolute_fee_sell_order':dict_crypto_options['general']['
    absolute_fee_sell_order'],
828                 'percentage_buying_fees_and_spread':dict_crypto_options['
    general']['percentage_buying_fees_and_spread'],
829                 'percentage_selling_fees_and_spread':dict_crypto_options['
    general']['percentage_selling_fees_and_spread']
830             },
831             display_options=display_options,
832             constraints=constraints,
833             general_settings=general_settings,
834             boolean_allow_partially_filled_orders=strategy_hyperparameters['
    boolean_allow_partially_filled_orders']
835         )
836
837         df_trading_journal = df_trading_journal.append(
838             dataseries_trading_journal,
839             ignore_index=True
840         )
841
842     dict_return = {
843         'df_trading_journal': df_trading_journal,
844         'Strategy ID': int_chosen_strategy,
845         'Strategy label': 'White Noise',
846         'strategy_hyperparameters': strategy_hyperparameters,
847         'comments': ''
848     }
849
850     save_dataframe_to_csv(
851         df_trading_journal,
852         'trading_journal',
853         string_directory=display_options['string_results_directory'],
854     )
```

```
855
856     return dict_return
857
858 def execute_strategy_ma_crossover(
859         df_prices,
860         int_chosen_strategy,
861         float_budget_in_usd,
862         margin_loan_rate,
863         boolean_allow_shorting,
864         list_trading_execution_delay_after_signal_in_hours,
865         dict_crypto_options,
866         general_settings,
867         minimum_expected_mispricing_trigger_in_percent,
868         strategy_hyperparameters,
869         sell_at_the_end,
870         file_path_with_signal_data,
871         display_options,
872         constraints,
873         comments
874     ):
875     """Filters signals and executes all remaining signals."""
876     df_prices['moving_average'] = df_prices.groupby(
877         level='id'
878     )['price'].transform(
879         lambda x: round(
880             x.rolling(
881                 window=strategy_hyperparameters['
    moving_average_window_in_days'],
882                 # on='datetime'
883             ).mean(),
884             general_settings['rounding_decimal_places']
885         )
886     )
887
888     df_trading_journal = initialize_trading_journal()
889
890     price = None
891
892     current_time = strategy_hyperparameters['start_time']
893     previous_time = Timestamp(current_time) - strategy_hyperparameters['
    frequency']
894
895     times_to_loop_over = date_range(start=strategy_hyperparameters['
    start_time'], end=strategy_hyperparameters['end_time'], freq=
    strategy_hyperparameters['frequency']).to_series()
896     for current_time in times_to_loop_over:
897     #for time_elapsed in tqdm(range(strategy_hyperparameters['frequency'],
    ((strategy_hyperparameters['end_time'] - strategy_hyperparameters['
    start_time']) + strategy_hyperparameters['frequency'])), desc='Going
```

```
    through signals', unit='signal'):
898
899         boolean_buy = None
900         number_to_be_bought = 0
901
902         # CONTINUE HERE
903         previous_time = current_time - Timedelta(strategy_hyperparameters['
    frequency'])
904         # This intermediate step is used to erase the frequency from the
    Timestamp
905         previous_time = Timestamp(str(previous_time))
906
907         old_row = df_prices.loc[(previous_time, strategy_hyperparameters['id
    ']), :]
908         previous_ma = old_row['moving_average']
909         old_price = old_row['price']
910
911         new_ma = df_prices.loc[(current_time, strategy_hyperparameters['id'
    ]), 'moving_average']
912
913         price = find_price(
914             df_prices,
915             desired_index=(current_time, strategy_hyperparameters['id']),
916             boolean_allow_older_prices=False,
917             boolean_allow_newer_prices=False,
918             boolean_warnings=display_options['
    warning_no_price_during_execution']
919         )
920
921         if (previous_ma < old_price) and (new_ma > price):
922             moving_average_crossover = 'Upside breach'
923         elif (previous_ma > old_price) and (new_ma < price):
924             moving_average_crossover = 'Downside breach'
925         elif new_ma < price:
926             moving_average_crossover = 'Above'
927         elif new_ma > price:
928             moving_average_crossover = 'Below'
929         else:
930             moving_average_crossover = None
931
932         if moving_average_crossover == 'Upside breach' or
    moving_average_crossover == 'Above':
933             boolean_buy = True
934         elif moving_average_crossover == 'Downside breach' or
    moving_average_crossover == 'Below':
935             boolean_buy = False
936         else:
937             boolean_buy = None
938
```

```
939         if price is not None and price > 0:
940             if boolean_buy:
941                 try:
942                     float_budget_in_usd = df_trading_journal['Cash'].iloc
    [-1]
943                 except:
944                     pass
945 
946                 number_to_be_bought = round(
947                     (
948                         float_budget_in_usd
949                         - constraints['minimum_cash']
950                     ) / price,
951                     general_settings['
    rounding_decimal_places_for_security_quantities']
952                 )
953 
954                 if number_to_be_bought is not None:
955                     number_to_be_bought = round(
956                         number_to_be_bought * strategy_hyperparameters['
    maximum_relative_exposure_per_buy'],
957                         general_settings['
    rounding_decimal_places_for_security_quantities']
958                     )
959 
960             elif boolean_buy == False:
961                 if len(df_trading_journal) > 0:
962                     number_to_be_bought = (-1) * round(
963                         df_trading_journal.iloc[-1]['Dict of assets in
    portfolio'][strategy_hyperparameters['id']],
964                         general_settings['
    rounding_decimal_places_for_security_quantities']
965                     )
966                 else:
967                     number_to_be_bought = 0
968             else:
969                 number_to_be_bought = 0
970 
971             if number_to_be_bought != 0:
972                 dataseries_trading_journal = execute_order(
973                     boolean_buy=boolean_buy,
974                     index=current_time,
975                     date=current_time,
976                     strategy_id=int_chosen_strategy,
977                     crypto_key=strategy_hyperparameters['id'],
978                     number_to_be_bought=number_to_be_bought,
979                     df_prices=df_prices,
980                     df_trading_journal=df_trading_journal,
981                     margin_loan_rate=margin_loan_rate,
```

```
                    float_budget_in_usd=float_budget_in_usd,
                    price=price,
                    fees={
                        'absolute_fee_buy_order': dict_crypto_options['general']['absolute_fee_buy_order'],
                        'absolute_fee_sell_order': dict_crypto_options['general']['absolute_fee_sell_order'],
                        'percentage_buying_fees_and_spread': dict_crypto_options['general']['percentage_buying_fees_and_spread'],
                        'percentage_selling_fees_and_spread': dict_crypto_options['general']['percentage_selling_fees_and_spread']
                    },
                    display_options=display_options,
                    constraints=constraints,
                    general_settings=general_settings,
                    boolean_allow_partially_filled_orders=strategy_hyperparameters['boolean_allow_partially_filled_orders']
                )

                df_trading_journal = df_trading_journal.append(
                    dataseries_trading_journal,
                    ignore_index=True
                )

    dict_return = {
        'df_trading_journal': df_trading_journal,
        'Strategy ID': '4',
        'Strategy label': 'Moving Average Crossover',
        'strategy_hyperparameters': strategy_hyperparameters,
        'comments': comments
    }

    save_dataframe_to_csv(
        df_trading_journal,
        'trading_journal',
        string_directory=display_options['string_results_directory'],
    )

    return dict_return

def test_strategies(
        df_prices,
        int_chosen_strategy,
        float_budget_in_usd,
        margin_loan_rate,
        boolean_allow_shorting,
        list_trading_execution_delay_after_signal_in_hours,
        dict_crypto_options,
        minimum_expected_mispricing_trigger_in_percent,
```

```
1026        strategy_hyperparameters,
1027        sell_at_the_end,
1028        benchmark_data_specifications,
1029        display_options,
1030        constraints,
1031        general_settings,
1032        start_time,
1033        file_path_with_signal_data=None,
1034        comments={}
1035    ):
1036    """Calls user-defined strategy.
1037
1038    Chooses the correct strategy (as per user input) and returns
1039    the performance metrics and the trading journal of that strategy.
1040    """
1041
1042    if int_chosen_strategy == 1:
1043        dict_execution_results = execute_strategy_white_noise(
1044            df_prices=df_prices,
1045            float_budget_in_usd=float_budget_in_usd,
1046            margin_loan_rate=margin_loan_rate,
1047            boolean_allow_shorting=boolean_allow_shorting,
1048            list_trading_execution_delay_after_signal_in_hours=
    list_trading_execution_delay_after_signal_in_hours,
1049            int_chosen_strategy=int_chosen_strategy,
1050            dict_crypto_options=dict_crypto_options,
1051            minimum_expected_mispricing_trigger_in_percent=
    minimum_expected_mispricing_trigger_in_percent,
1052            strategy_hyperparameters=strategy_hyperparameters,
1053            display_options=display_options,
1054            constraints=constraints,
1055            general_settings=general_settings,
1056            sell_at_the_end=sell_at_the_end,
1057            comments=comments
1058        )
1059    elif int_chosen_strategy == 2:
1060        raise NotImplementedError('Strategy 2 is not implemented.')
1061    elif int_chosen_strategy == 3:
1062        dict_execution_results = execute_strategy_multi_asset(
1063            df_prices=df_prices,
1064            file_path_with_signal_data=file_path_with_signal_data,
1065            float_budget_in_usd=float_budget_in_usd,
1066            int_chosen_strategy=int_chosen_strategy,
1067            margin_loan_rate=margin_loan_rate,
1068            boolean_allow_shorting=boolean_allow_shorting,
1069            list_trading_execution_delay_after_signal_in_hours=
    list_trading_execution_delay_after_signal_in_hours,
1070            dict_crypto_options=dict_crypto_options,
1071            minimum_expected_mispricing_trigger_in_percent=
```

```
    minimum_expected_mispricing_trigger_in_percent,
1072            strategy_hyperparameters=strategy_hyperparameters,
1073            sell_at_the_end=sell_at_the_end,
1074            display_options=display_options,
1075            constraints=constraints,
1076            general_settings=general_settings,
1077            comments=comments
1078        )
1079    elif int_chosen_strategy == 4:
1080        dict_execution_results = execute_strategy_ma_crossover(
1081            df_prices=df_prices,
1082            file_path_with_signal_data=file_path_with_signal_data,
1083            int_chosen_strategy=int_chosen_strategy,
1084            float_budget_in_usd=float_budget_in_usd,
1085            margin_loan_rate=margin_loan_rate,
1086            boolean_allow_shorting=boolean_allow_shorting,
1087            list_trading_execution_delay_after_signal_in_hours=
    list_trading_execution_delay_after_signal_in_hours,
1088            dict_crypto_options=dict_crypto_options,
1089            minimum_expected_mispricing_trigger_in_percent=
    minimum_expected_mispricing_trigger_in_percent,
1090            strategy_hyperparameters=strategy_hyperparameters,
1091            sell_at_the_end=sell_at_the_end,
1092            display_options=display_options,
1093            constraints=constraints,
1094            general_settings=general_settings,
1095            comments=comments
1096        )
1097
1098    save_dataframe_to_csv(
1099        dict_execution_results['df_trading_journal'],
1100        string_name='trading_journal',
1101        string_directory=display_options['string_results_directory'],
1102    )
1103
1104    df_performance = evaluate_performance(
1105        df_prices=df_prices,
1106        dict_execution_results=dict_execution_results,
1107        float_budget_in_usd=float_budget_in_usd,
1108        benchmark_data_specifications=benchmark_data_specifications,
1109        strategy_hyperparameters=strategy_hyperparameters,
1110        display_options=display_options,
1111        constraints=constraints,
1112        general_settings=general_settings,
1113        start_time=start_time,
1114    )
1115
```

```
1116     return [df_performance, dict_execution_results['df_trading_journal']]
```

Code Snippet 37: `components/_2_strategy_execution.py` – The functions from this module create `df_trading_journal`, i.e., they simulate trades.

```
1  """Provides individual metrics calculations.
2
3  This module is more low-level than _3_performance_evaluation."""
4
5  # For mathematical operations
6  from math import sqrt, exp, log
7
8  # For tensor calculations
9  from numpy import cov
10
11 # For managing tables properly
12 from pandas import to_datetime
13
14 from _helper_functions import find_dataframe_value_with_keywords, \
15     datetime_to_string
16
17
18 def calculate_alpha(
19         annualized_portfolio_return,
20         risk_free_rate,
21         beta_exposure,
22         annualized_market_return
23     ):
24     """Calculates the Jensen's alpha of a portfolio against a benchmark."""
25     market_risk_premium = annualized_market_return - risk_free_rate
26
27     alpha = annualized_portfolio_return - (
28         risk_free_rate + beta_exposure * market_risk_premium
29     )
30
31     return alpha
32
33 def calculate_beta(df_daily_returns, df_daily_benchmark_returns):
34     """Calculates the beta of a portfolio against a benchmark."""
35     portfolio_returns = df_daily_returns['relative_portfolio_return'].
       to_numpy()
36     benchmark_returns = df_daily_benchmark_returns['
       benchmark_relative_return'].to_numpy()
37
38     covariance_matrix = cov(portfolio_returns, benchmark_returns)
39
40     benchmark_portfolio_covariance = covariance_matrix[0][1]
41     benchmark_variance = covariance_matrix[1][1]
42
```

```
43      return benchmark_portfolio_covariance / benchmark_variance
44
45  def calculate_maximum_drawdown(
46          df_daily_returns,
47          column_with_portfolio_values
48      ):
49      """Calculates the maximum_drawdown of a portfolio using portfolio_value
        from df_daily_returns.
50
51      Outputs the maximum_drawdown ratio.
52      """
53      dict_peak_tracking = {
54          'first_peak': {
55              'peak': {
56                  column_with_portfolio_values: df_daily_returns[
        column_with_portfolio_values].iloc[0],
57                  'datetime': df_daily_returns.index.values[0]
58              },
59              'trough': {
60                  column_with_portfolio_values: df_daily_returns[
        column_with_portfolio_values].iloc[0],
61                  'datetime': df_daily_returns.index.values[0]
62              }
63          },
64          'second_peak': {
65              'peak': {
66                  column_with_portfolio_values: df_daily_returns[
        column_with_portfolio_values].iloc[0],
67                  'datetime': df_daily_returns.index.values[0]
68              },
69              'trough': {
70                  column_with_portfolio_values: df_daily_returns[
        column_with_portfolio_values].iloc[0],
71                  'datetime': df_daily_returns.index.values[0]
72              }
73          }
74      }
75
76      for datetime, row in df_daily_returns.iterrows():
77          if row[column_with_portfolio_values] > dict_peak_tracking['
        second_peak']['peak'][column_with_portfolio_values]:
78              dict_peak_tracking['second_peak']['peak'][
        column_with_portfolio_values] = row[column_with_portfolio_values]
79              dict_peak_tracking['second_peak']['peak']['datetime'] = datetime
80              dict_peak_tracking['second_peak']['trough'][
        column_with_portfolio_values] = row[column_with_portfolio_values]
81              dict_peak_tracking['second_peak']['trough']['datetime'] =
        datetime
82          if row[column_with_portfolio_values] < dict_peak_tracking['
```

```
    second_peak']['trough'][column_with_portfolio_values]:
83              dict_peak_tracking['second_peak']['trough'][
    column_with_portfolio_values] = row[column_with_portfolio_values]
84              dict_peak_tracking['second_peak']['trough']['datetime'] =
    datetime
85          if row[column_with_portfolio_values] < dict_peak_tracking['
    first_peak']['trough'][column_with_portfolio_values]:
86              dict_peak_tracking['first_peak']['trough'][
    column_with_portfolio_values] = row[column_with_portfolio_values]
87              dict_peak_tracking['first_peak']['trough']['datetime'] =
    datetime
88
89          drawdown_second = (
90              dict_peak_tracking['second_peak']['peak'][
    column_with_portfolio_values] - dict_peak_tracking['second_peak']['trough
    ']['column_with_portfolio_values]
91          ) / dict_peak_tracking['second_peak']['peak'][
    column_with_portfolio_values]
92          drawdown_first = (
93              dict_peak_tracking['first_peak']['peak'][
    column_with_portfolio_values] - dict_peak_tracking['first_peak']['trough'
    ]['column_with_portfolio_values]
94          ) / dict_peak_tracking['first_peak']['peak'][
    column_with_portfolio_values]
95          if drawdown_second > drawdown_first:
96              dict_peak_tracking['first_peak'] = dict_peak_tracking['
    second_peak']
97
98      maximum_drawdown_duration = dict_peak_tracking['first_peak']['trough']['
    datetime'] - dict_peak_tracking['first_peak']['peak']['datetime']
99      peak_date = dict_peak_tracking['first_peak']['peak']['datetime']
100     trough_date = dict_peak_tracking['first_peak']['trough']['datetime']
101
102     return drawdown_first, maximum_drawdown_duration, peak_date, trough_date
103
104 def calculate_roi(
105         df_trading_journal,
106         float_budget_in_usd,
107         df_benchmark=None,
108         df_price_column_name='price',
109         df_time_column_name='datetime',
110         df_benchmark_price_column_name='price',
111         df_benchmark_time_column_name='datetime',
112         df_trading_journal_price_column_name='Portfolio value',
113         df_trading_journal_time_column_name='datetime'
114     ):
115     """Calculates annualized returns (on equity, not total assets).
116
117     It can calculate standalone returns without benchmarking (in USD/fiat)
```

```
    or
118     returns in relation to a benchmark. It can also use different start and
    end
119     points for calculating the time frame, i.e., the duration between the
    first
120     and the last trade OR the duration between the first and the last data
    point
121     of the price data.
122     """
123     start_time = df_trading_journal[
124         df_trading_journal_time_column_name
125     ].iloc[0]
126     end_time = df_trading_journal[
127         df_trading_journal_time_column_name
128     ].iloc[-1]
129 
130     portfolio_roi = calculate_roi_math(
131         end_value=df_trading_journal[
132             df_trading_journal_price_column_name
133     ].iloc[-1],
134         start_value=float_budget_in_usd,
135         end_time=end_time,
136         start_time=start_time
137     )
138 
139     roi_delta_compared_to_benchmark = None
140     benchmark_roi = None
141 
142     if df_benchmark is not None:
143         end_value_benchmark = find_dataframe_value_with_keywords(
144             df_benchmark,
145             search_term_1=end_time,
146             search_column_name_1=df_benchmark_time_column_name,
147             search_term_2=None,
148             search_column_name_2=None,
149             output_column_name=df_benchmark_price_column_name,
150             first_last_or_all_elements='First'
151         )
152 
153         begin_value_benchmark = find_dataframe_value_with_keywords(
154             df_benchmark,
155             search_term_1=start_time,
156             search_column_name_1=df_benchmark_time_column_name,
157             search_term_2=None,
158             search_column_name_2=None,
159             output_column_name=df_benchmark_price_column_name,
160             first_last_or_all_elements='First'
161         )
162 
```

```
163         benchmark_roi = calculate_roi_math(
164             end_value=end_value_benchmark,
165             start_value=begin_value_benchmark,
166             end_time=end_time,
167             start_time=start_time
168         )
169 
170         roi_delta_compared_to_benchmark = portfolio_roi - benchmark_roi
171 
172     return {
173         'portfolio_roi': portfolio_roi,
174         'benchmark_roi': benchmark_roi,
175         'roi_delta_compared_to_benchmark': roi_delta_compared_to_benchmark
176     }
177 
178 def calculate_roi_math(
179         end_value,
180         start_value,
181         end_time,
182         start_time
183     ):
184     unadjusted_return_factor = end_value / start_value
185     time_duration = (end_time - start_time).total_seconds() / 86400
186 
187     roi = exp(
188         log(
189             unadjusted_return_factor
190         ) * 365 / (time_duration)
191     ) - 1
192 
193     return roi
194 
195 def calculate_sharpe_ratio(
196         df_daily_returns,
197         portfolio_roi_usd,
198         days=None,
199         risk_free_rate=None
200     ):
201     """Calculates the Sharpe ratio of a given set of returns."""
202     volatility = calculate_volatility(df_daily_returns, days)
203 
204     if risk_free_rate is None:
205         sharpe_ratio = portfolio_roi_usd / volatility
206     else:
207         sharpe_ratio = (portfolio_roi_usd - risk_free_rate) / volatility
208     return sharpe_ratio
209 
210 
211 def calculate_transaction_cost(df_trading_journal):
```

```
    return round(
        sum(df_trading_journal["Total fees (as absolute)"]),
        2
    )

def calculate_volatility(df_daily_returns, time_adjustment_in_days=None):
    """Calculates the volatility of a portfolio using daily portfolio returns.

    Outputs the volatility over the given time series by default; can also be
    adjusted for any arbitrary time using the time_adjustment_in_days parameter.
    """
    if time_adjustment_in_days is None:
        time_adjustment_in_days = len(df_daily_returns)

    volatility = df_daily_returns['relative_portfolio_return'].std() * sqrt(
        time_adjustment_in_days
    ) / sqrt(len(df_daily_returns))

    if round(volatility, 4) == 0:
        raise ValueError('Volatility cannot be zero or close to zero. Please check if daily returns are correctly calculated and if any trades were made. \n{df_daily_returns}')

    return volatility
```

Code Snippet 38: `components/_3_individual_metrics.py` – The functions from this module supply `_3_performance_evaluation.py` with individual financial calculations.

```
"""Provides functions for performance evaluations of the trading journal.

This module is more high-level than _3_individual_metrics."""

# For deep-copied dictionaries
from copy import deepcopy

# For managing tables properly
from pandas import DataFrame

# For plotting return curve
import matplotlib.pyplot as plt
from matplotlib.dates import MonthLocator
from matplotlib.dates import DateFormatter
import matplotlib.ticker as tickercalculate_alpha

# For managing dates
```

```
18 from datetime import datetime, timedelta
19
20 # For counting the number of result files in the "results" folder to allow
       for
21 # consistent file versioning when saving the results as a CSV file.
22 import os
23
24 from _1_data_preparation import save_dataframe_to_csv
25 from _3_individual_metrics import calculate_volatility, calculate_roi, \
26     calculate_sharpe_ratio, calculate_maximum_drawdown, \
27     calculate_transaction_cost, calculate_beta, calculate_alpha
28 from _helper_functions import find_dataframe_value_with_keywords, find_price
       , \
29     alternative_date_finder, calculate_portfolio_value, datetime_to_string,
       \
30     add_time_column_to_dataframe_from_string
31
32
33 def calculate_returns_single(
34         previous_trading_journal_row,
35         current_trading_journal_row,
36         df_prices,
37         strategy_hyperparameters,
38         display_options,
39         general_settings,
40         constraints
41     ):
42     """Returns a list of dicts with portfolio return data for a given
      frequency.
43
44     Dict fields: 'timestamp' (datetime.datetime), 'portfolio_value' (float),
45                  'return' (float), 'relative_return' (float),
46                  'dict_of_assets_in_portfolio' (dict with itins as keys and
47                  integer with the number of pieces held of this asset as as
48                  values)
49
50     The reasoning behind the return calculation and the frequency handling
       is
51     described using an exemplary time series. The frequency is minutes here,
        so
52     frequency_in_seconds=60. The first column represents the executions
53     (previous_trading_journal_row and current_trading_journal_row), the
       second
54     column represents the frequency.
55
56     Trades                          Frequency increments
57
58     -No trade-                      Minute 1
59     Trade 1                         Minute 2
```

```
    -No trade-                  Minute 3
    -No trade-                  Minute 4
    Trade 2                     Minute 5

    In the example above, the return calculation would work as follows: The
    function would receive DataFrame rows for Trade 1 and Trade 2. It would
    run
    the return calculation loop for Minute 2-3, Minute 3-4, and Minute 4-5.
    Thus, the function would return a list of three dicts. The returns for
    Minute 1-2 were already calculated in an earlier function call (Trade
    0-1).
    It is crucial that the datetime.datetime objects that are contained in
    Trade 1 and Trade 2 have the same frequency as the frequency that is
    passed
    as an argument. Otherwise, there can be missing entries in the
    aggregated
    return DataFrame.

    datetime.datetime objects assume 'there are exactly 3600*24 seconds in
    every
    day'. https://docs.python.org/2/library/datetime.html#datetime-objects
    """
    list_of_dict_returns = []

    dict_of_assets_in_portfolio = previous_trading_journal_row['Dict of
    assets in portfolio']
    copy_dict_of_assets_in_portfolio = deepcopy(dict_of_assets_in_portfolio)

    cash = previous_trading_journal_row['Cash']

    datetime_counter = previous_trading_journal_row['datetime'].
    to_pydatetime()

    while (
            datetime_counter + strategy_hyperparameters['frequency']
    ) <= (
            current_trading_journal_row['datetime'].to_pydatetime()
    ):
        returns = {'dict_of_assets_in_portfolio':
    copy_dict_of_assets_in_portfolio}

        # The counter is purposefully incremented at the beginning
        datetime_counter += strategy_hyperparameters['frequency']

        returns['datetime'] = datetime_counter

        previous_portfolio_value = calculate_portfolio_value(
            df_prices=df_prices,
            dict_of_assets_in_portfolio=dict_of_assets_in_portfolio,
```

```
            time=datetime_counter - strategy_hyperparameters['frequency'],
            cash_value=cash,
            display_options=display_options,
            constraints=constraints,
            rounding_accuracy=general_settings['rounding_decimal_places']
        )

        returns['portfolio_value'] = calculate_portfolio_value(
            df_prices=df_prices,
            dict_of_assets_in_portfolio=dict_of_assets_in_portfolio,
            time=datetime_counter,
            cash_value=cash,
            display_options=display_options,
            constraints=constraints,
            rounding_accuracy=general_settings['rounding_decimal_places']
        )

        returns['dict_of_assets_in_portfolio'] = deepcopy(
    copy_dict_of_assets_in_portfolio)

        try:
            returns['portfolio_return'] = round(
                returns['portfolio_value'] - previous_portfolio_value,
                general_settings['rounding_decimal_places']
            )
            try:
                returns['relative_portfolio_return'] = round(
                    returns['portfolio_return'] / previous_portfolio_value,
                    general_settings['rounding_decimal_places']
                )
            except ZeroDivisionError:
                raise ZeroDivisionError(
                    """
                        This is a rather unlikely scenario, therefore an
    error
                        was raised. The previous portfolio value is zero.
    This
                        cannot be.
                    """ +
                    '\nportfolio_value: ' + str(
                        returns['portfolio_value']
                    ) +
                    '\nportfolio_return: ' + str(
                        returns['portfolio_return']
                    )
                )
        except IndexError:
            returns['portfolio_return'] = round(
                returns['portfolio_value'] - df_trading_journal['Cash before
```

```
    '].iloc[0],
147                 general_settings['rounding_decimal_places']
148             )
149             returns['relative_portfolio_return'] = round(
150                 returns['portfolio_return'] / df_trading_journal['Cash
    before'].iloc[0],
151                 general_settings['rounding_decimal_places']
152             )
153 
154         returns = deepcopy(returns)
155 
156         list_of_dict_returns.append(returns)
157 
158     return list_of_dict_returns
159 
160 def calculate_returns_batch(
161         df_trading_journal,
162         df_prices,
163         strategy_hyperparameters,
164         display_options,
165         general_settings,
166         constraints
167     ):
168     """Calculates returns of a portfolio from a trading journal.
169 
170     Any frequency between miliseconds and infinity can be used (hourly,
    daily
171     weekly, etc.).
172     """
173 
174     if len(df_trading_journal) < 1:
175         raise ValueError(f'There were no trades. The provided trading
    journal has {str(len(df_trading_journal))} trades.')
176 
177     df_returns = DataFrame(
178         columns=[
179             'datetime',
180             'portfolio_value',
181             'portfolio_return',
182             'relative_portfolio_return',
183             'dict_of_assets_in_portfolio'
184         ]
185     )
186 
187     df_returns.set_index(
188         keys=['datetime'],
189         inplace=True
190     )
191 
```

```
    first_date = df_trading_journal['datetime'].iloc[0]
    last_date = df_trading_journal['datetime'].iloc[-1]

    if df_trading_journal['Cash before'].iloc[0] is None:
        raise ValueError('initial_budget should not be None.')

    current_date = first_date
    for index, row in df_trading_journal.iterrows():
        if index > 0:
            previous_trading_journal_row = df_trading_journal.loc[index - 1]
            current_trading_journal_row = df_trading_journal.loc[index]

            list_of_dict_returns = calculate_returns_single(
                previous_trading_journal_row=previous_trading_journal_row,
                current_trading_journal_row=current_trading_journal_row,
                df_prices=df_prices,
                strategy_hyperparameters=strategy_hyperparameters,
                display_options=display_options,
                general_settings=general_settings,
                constraints=constraints
            )

            for dict_returns in list_of_dict_returns:
                df_returns.loc[dict_returns['datetime']] = {
                    'portfolio_value': dict_returns['portfolio_value'],
                    'portfolio_return': dict_returns['portfolio_return'],
                    'relative_portfolio_return': dict_returns['
    relative_portfolio_return'],
                    'dict_of_assets_in_portfolio': dict_returns['
    dict_of_assets_in_portfolio']
                }
        else:
            pass

    assert len(df_returns) > 0

    return df_returns

def calculate_daily_returns_from_benchmark(
        first_date,
        last_date,
        df_prices,
        benchmark_id,
        display_options,
        strategy_hyperparameters,
        general_settings,
        constraints
    ):
    """Calculates daily returns of a benchmark."""
```

```
    df_daily_returns = DataFrame(
        columns=['benchmark_datetime', 'benchmark_dict_of_assets_in_portfolio', 'benchmark_portfolio_value', 'benchmark_return', 'benchmark_relative_return']
    )

    df_daily_returns.set_index(['benchmark_datetime'], inplace=True)

    initial_budget = find_price(
        df_prices=df_prices,
        desired_index=(first_date, slice(None)),
        boolean_allow_older_prices=False,
        boolean_allow_newer_prices=False,
        boolean_warnings=True,
        boolean_errors=True
    )

    if initial_budget is None:
        raise ValueError(f'initial_budget should not be None. Benchmark ID: {benchmark_id}. No price found for {first_date} in \n{df_prices}.')

    current_date = first_date
    for days_elapsed in range(1, ((last_date - first_date).days + 1)):
        current_date = first_date + timedelta(days=days_elapsed)
        # previous_date = current_date - timedelta(days=1)

        dict_of_assets_in_portfolio = {
            str(benchmark_id): 1.0
        }

        portfolio_value = calculate_portfolio_value(
            df_prices=df_prices,
            dict_of_assets_in_portfolio=dict_of_assets_in_portfolio,
            time=current_date,
            cash_value=0,
            rounding_accuracy=general_settings['rounding_decimal_places'],
            display_options=display_options,
            constraints=constraints
        )

        try:
            portfolio_return = round(
                portfolio_value - df_daily_returns['benchmark_portfolio_value'].iloc[-1],
                2
            )
            try:
                relative_portfolio_return = round(
                    portfolio_return / df_daily_returns['
```

```
    benchmark_portfolio_value'].iloc[-1],
284                     2
285                 )
286             except ZeroDivisionError:
287                 raise ZeroDivisionError(
288                     """
289                         This is a rather unlikely scenario, therefore an
    error
290                         was raised. The previous portfolio value is zero.
    This
291                         cannot be.
292                     """ +
293                     '\nprevious portfolio value: ' + str(
294                         df_daily_returns['benchmark_portfolio_value'].iloc
    [-1]
295                     ) +
296                     '\nportfolio_value: ' + str(
297                         portfolio_value
298                     ) +
299                     '\nportfolio_return: ' + str(
300                         portfolio_return
301                     )
302                 )
303         except IndexError:
304             portfolio_return = round(
305                 portfolio_value - initial_budget,
306                 2
307             )
308             relative_portfolio_return = round(
309                 portfolio_return / initial_budget,
310                 2
311             )
312
313         copy_dict_of_assets_in_portfolio = deepcopy(
    dict_of_assets_in_portfolio)
314
315         df_daily_returns.loc[current_date] = {
316             'benchmark_portfolio_value': portfolio_value,
317             'benchmark_return': portfolio_return,
318             'benchmark_relative_return': relative_portfolio_return,
319             'benchmark_dict_of_assets_in_portfolio':
    copy_dict_of_assets_in_portfolio
320         }
321
322     assert len(df_daily_returns) > 0
323
324     return df_daily_returns
325
326 def initialize_performance_overview():
```

```
327     """Initializes a pandas DataFrame that serves as a performance overview.
328 
329     Initialization is important for determining the column order.
330     """
331     df_performance = DataFrame(columns=[
332         'Strategy metadata ---->',
333         'Strategy ID',
334         'Strategy label',
335         'Trading info ---->',
336         'Begin time of tested interval',
337         'End time of tested interval',
338         'Duration of the tested interval',
339         'Duration of the tested interval (in days)',
340         'Average cash',
341         'Average ticket size',
342         'Number of trades',
343         'Number of unique assets traded',
344         'Total transaction cost',
345         'Return metrics ---->',
346         'USD annualized ROI (from first to last trade)',
347         'Cryptocurrency annualized ROI delta (from first to last trade)',
348         'Ending benchmark value (first to last trade)',
349         'Initial budget',
350         'Ending portfolio value',
351         'Risk metrics ---->',
352         'Holding period volatility',
353         'Annual volatility',
354         'Monthly volatility',
355         'Weekly volatility',
356         'Beta relative to benchmark',
357         'Maximum drawdown',
358         'Maximum drawdown duration',
359         'Maximum drawdown peak date',
360         'Maximum drawdown trough date',
361         'Other metrics ---->',
362         'Alpha',
363         'Sharpe ratio (holding period)',
364         'Sharpe ratio (yearly)',
365         'Beginning benchmark value (first to last trade)',
366         'Other info ---->',
367         'Start time',
368         'End time',
369         'Parameter 1',
370         'Parameter 2',
371         'Comments',
372         'Benchmark return metrics ---->',
373         'Benchmark USD annualized ROI (from first to last trade)',
374         'Benchmark cryptocurrency annualized ROI delta (from first to last
    trade)',
```

```
375         'Benchmark ending benchmark value (first to last trade)',
376         'Benchmark initial budget',
377         'Benchmark ending portfolio value',
378         'Benchmark risk metrics --->',
379         'Benchmark holding period volatility',
380         'Benchmark annual volatility',
381         'Benchmark monthly volatility',
382         'Benchmark weekly volatility',
383         'Benchmark Beta relative to benchmark',
384         'Benchmark maximum drawdown',
385         'Benchmark maximum drawdown duration',
386         'Benchmark maximum drawdown peak date',
387         'Benchmark maximum drawdown trough date',
388         'Benchmark other metrics --->',
389         'Benchmark Sharpe ratio (holding period)',
390         'Benchmark Sharpe ratio (yearly)',
391     ])
392 
393     return df_performance
394 
395 def evaluate_performance(
396         df_prices,
397         dict_execution_results,
398         float_budget_in_usd,
399         benchmark_data_specifications,
400         strategy_hyperparameters,
401         display_options,
402         general_settings,
403         constraints,
404         start_time
405     ):
406     """Evaluates the performance of a trading journal.
407 
408     Common metrics to evaluate trading strategies are used.
409     """
410 
411     df_performance = initialize_performance_overview()
412 
413     df_trading_journal = dict_execution_results['df_trading_journal']
414 
415     try:
416         df_daily_returns = calculate_returns_batch(
417             df_trading_journal=df_trading_journal,
418             df_prices=df_prices,
419             strategy_hyperparameters=strategy_hyperparameters,
420             display_options=display_options,
421             general_settings=general_settings,
422             constraints=constraints
423         )
```

```
424     except ValueError:
425         print(f'Warning: No trades were executed with the given paremters: {
    strategy_hyperparameters}')
426         return None
427 
428     if len(df_trading_journal) < 1:
429         df_performance = df_performance.append(
430             {
431                 'Strategy ID': None
432             },
433             ignore_index=True
434         )
435 
436     else:
437         begin_time_of_tested_interval = df_trading_journal[
438             'datetime'
439         ].iloc[0]
440         end_time_of_tested_interval = dict_execution_results[
441             'df_trading_journal'
442         ]['datetime'].iloc[-1]
443 
444         # Ethereum does not have an eth_address and needs to be filtered
    usint the ITIN.
445         try:
446             df_benchmark = df_prices[df_prices['token_itin'] ==
    benchmark_data_specifications['benchmark_key']]
447         except:
448             df_benchmark = df_prices.loc[(slice(None),
    benchmark_data_specifications['benchmark_key']), : ]
449 
450         df_daily_benchmark_returns = calculate_daily_returns_from_benchmark(
451             first_date=begin_time_of_tested_interval,
452             last_date=end_time_of_tested_interval,
453             df_prices=df_benchmark,
454             benchmark_id=benchmark_data_specifications['benchmark_key'],
455             display_options=display_options,
456             general_settings=general_settings,
457             strategy_hyperparameters=strategy_hyperparameters,
458             constraints=constraints
459         )
460 
461         df_daily_returns['benchmark_portfolio_value'] =
    df_daily_benchmark_returns['benchmark_portfolio_value']
462         df_daily_returns['benchmark_portfolio_value_normalized'] =
    df_daily_returns['benchmark_portfolio_value'] / df_daily_returns['
    benchmark_portfolio_value'][0]
463         df_daily_returns['portfolio_value_normalized'] = df_daily_returns['
    portfolio_value'] / df_daily_returns['portfolio_value'][0]
464 
```

```
465         df_daily_returns['benchmark_portfolio_value'] =
    df_daily_benchmark_returns['benchmark_portfolio_value']
466         df_daily_returns['benchmark_return'] = df_daily_benchmark_returns['
    benchmark_return']
467         df_daily_returns['benchmark_relative_return'] =
    df_daily_benchmark_returns['benchmark_relative_return']
468         df_daily_returns['benchmark_dict_of_assets_in_portfolio'] =
    df_daily_benchmark_returns['benchmark_dict_of_assets_in_portfolio']
469
470         save_dataframe_to_csv(
471             df_daily_returns,
472             string_name='df_daily_returns',
473             string_directory=display_options['string_results_directory'],
474         )
475
476         beginning_benchmark_value = find_dataframe_value_with_keywords(
477             df_benchmark,
478             search_term_1=df_trading_journal['datetime'].iloc[0],
479             search_column_name_1='datetime',
480             search_term_2=None,
481             search_column_name_2=None,
482             output_column_name='price',
483             first_last_or_all_elements='First'
484         )
485
486         ending_benchmark_value = find_dataframe_value_with_keywords(
487             df_benchmark,
488             search_term_1=df_trading_journal['datetime'].iloc[-1],
489             search_column_name_1='datetime',
490             search_term_2=None,
491             search_column_name_2=None,
492             output_column_name='price',
493             first_last_or_all_elements='First'
494         )
495
496         dict_roi_results = calculate_roi(
497             df_trading_journal=df_trading_journal,
498             float_budget_in_usd=float_budget_in_usd,
499             df_benchmark=df_benchmark
500         )
501
502         # Average ticket size
503         dict_execution_results['df_trading_journal']['Value bought absolute'
    ] = dict_execution_results['df_trading_journal']['Value bought'].abs()
504         average_ticket_size_absolute_value = round(
505             dict_execution_results['df_trading_journal']['Value bought
    absolute'].mean(),
506             2
507         )
```

```
508
509         maximum_drawdown, maximum_drawdown_duration,
    maximum_drawdown_peak_date, maximum_drawdown_trough_date =
    calculate_maximum_drawdown(
510             df_daily_returns=df_daily_returns,
511             column_with_portfolio_values='portfolio_value'
512         )
513
514         benchmark_maximum_drawdown, benchmark_maximum_drawdown_duration,
    benchmark_maximum_drawdown_peak_date,
    benchmark_maximum_drawdown_trough_date = calculate_maximum_drawdown(
515             df_daily_returns=df_daily_returns,
516             column_with_portfolio_values='benchmark_portfolio_value'
517         )
518
519         beta = calculate_beta(
520             df_daily_returns,
521             df_daily_benchmark_returns
522         )
523
524         df_performance = df_performance.append(
525             {
526                 'Strategy ID': dict_execution_results['Strategy ID'],
527                 'Strategy label': dict_execution_results['Strategy label'],
528                 'Trading info ---->': 'Trading info ---->',
529                 'Begin time of tested interval':
    begin_time_of_tested_interval,
530                 'End time of tested interval': end_time_of_tested_interval,
531                 'Duration of the tested interval':
    end_time_of_tested_interval - begin_time_of_tested_interval,
532                 'Duration of the tested interval (in days)': (
    end_time_of_tested_interval - begin_time_of_tested_interval).days,
533                 'USD annualized ROI (from first to last trade)':
    dict_roi_results['portfolio_roi'],
534                 'Cryptocurrency annualized ROI delta (from first to last
    trade)': dict_roi_results['roi_delta_compared_to_benchmark'],
535                 'Beginning benchmark value (first to last trade)':
    find_price(
536                     df_benchmark,
537                     desired_index=(begin_time_of_tested_interval,
    benchmark_data_specifications['benchmark_key']),
538                     boolean_allow_older_prices=False,
539                     boolean_allow_newer_prices=False,
540                     boolean_warnings=True,
541                     boolean_errors=display_options['errors_on_benchmark_gap'
    ]
542                 ),
543                 'Ending benchmark value (first to last trade)': find_price(
544                     df_benchmark,
```

```
545                     desired_index=(end_time_of_tested_interval,
    benchmark_data_specifications['benchmark_key']),
546                     boolean_allow_older_prices=False,
547                     boolean_allow_newer_prices=False,
548                     boolean_warnings=True,
549                     boolean_errors=display_options['errors_on_benchmark_gap'
    ]
550                 ),
551                 'Initial budget': float_budget_in_usd,
552                 'Ending portfolio value': df_trading_journal['Portfolio
    value'].iloc[-1],
553                 'Holding period volatility': calculate_volatility(
    df_daily_returns, len(df_daily_returns)),
554                 'Annual volatility': calculate_volatility(df_daily_returns,
    365),
555                 'Monthly volatility': calculate_volatility(df_daily_returns,
     30),
556                 'Weekly volatility': calculate_volatility(df_daily_returns,
    7),
557                 'Alpha': calculate_alpha(
558                     annualized_portfolio_return=dict_roi_results['
    portfolio_roi'],
559                     risk_free_rate=benchmark_data_specifications[
560                         'risk_free_rate'
561                     ],
562                     beta_exposure=beta,
563                     annualized_market_return=dict_roi_results['benchmark_roi
    ']
564                 ),
565                 'Sharpe ratio (holding period)': calculate_sharpe_ratio(
566                     df_daily_returns,
567                     portfolio_roi_usd=dict_roi_results['portfolio_roi'],
568                     risk_free_rate=benchmark_data_specifications[
569                         'risk_free_rate'
570                     ]
571                 ),
572                 'Sharpe ratio (yearly)': calculate_sharpe_ratio(
573                     df_daily_returns,
574                     portfolio_roi_usd=dict_roi_results['portfolio_roi'],
575                     risk_free_rate=benchmark_data_specifications[
576                         'risk_free_rate'
577                     ],
578                     days=365
579                 ),
580                 'Beta relative to benchmark': beta,
581                 'Maximum drawdown': maximum_drawdown,
582                 'Maximum drawdown duration': maximum_drawdown_duration,
583                 'Maximum drawdown peak date': maximum_drawdown_peak_date,
584                 'Maximum drawdown trough date': maximum_drawdown_trough_date
```

```
    ,
585             'Other metrics ---->': 'Other metrics ---->',
586             'Total transaction cost': calculate_transaction_cost(
    df_trading_journal),
587             'Number of trades': len(
588                 dict_execution_results['df_trading_journal']
589             ),
590             'Number of unique assets traded': len(df_trading_journal['
    Asset'].unique()),
591             'Average ticket size': average_ticket_size_absolute_value,
592             'Average cash': round(
593                 dict_execution_results['df_trading_journal']['Cash'].
    mean(),
594                 2
595             ),
596             'Other info ---->': 'Other info ---->',
597             'Start time': start_time,
598             'End time': datetime.now(),
599             'Parameter 1': strategy_hyperparameters['sell_parameter'],
600             'Parameter 2': strategy_hyperparameters['buy_parameter'],
601             'Comments': dict_execution_results['comments'],
602             'Benchmark return metrics ---->': 'Benchmark return metrics
    ---->',
603             'Benchmark USD annualized ROI (from first to last trade)':
    dict_roi_results['benchmark_roi'],
604             'Benchmark risk metrics ---->': 'Benchmark risk metrics ---->'
    ,
605             'Benchmark holding period volatility': 'NOT IMPLEMENTED',
606             'Benchmark annual volatility': 'NOT IMPLEMENTED',
607             'Benchmark monthly volatility': 'NOT IMPLEMENTED',
608             'Benchmark weekly volatility': 'NOT IMPLEMENTED',
609             'Benchmark Beta relative to benchmark': 'NOT IMPLEMENTED',
610             'Benchmark maximum drawdown': benchmark_maximum_drawdown,
611             'Benchmark maximum drawdown duration':
    benchmark_maximum_drawdown_duration,
612             'Benchmark maximum drawdown peak date':
    benchmark_maximum_drawdown_peak_date,
613             'Benchmark maximum drawdown trough date':
    benchmark_maximum_drawdown_trough_date,
614             'Benchmark other metrics ---->': 'NOT IMPLEMENTED',
615             'Benchmark Sharpe ratio (holding period)': 'NOT IMPLEMENTED'
    ,
616             'Benchmark Sharpe ratio (yearly)': 'NOT IMPLEMENTED',
617         },
618         ignore_index=True
619     )
620
621     plot_equity_curve(
622         df_daily_returns,
```

```
623             df_daily_benchmark_returns,
624             display_options=display_options,
625             boolean_plot=display_options['boolean_plot_equity_curve'],
626             boolean_save_to_disk=display_options['
    boolean_save_equity_curve_to_disk'],
627         )
628 
629     return df_performance
630 
631 def plot_equity_curve(
632         df_daily_returns,
633         df_daily_benchmark_returns,
634         display_options,
635         boolean_plot=False,
636         boolean_relative=False,
637         boolean_save_to_disk=True,
638     ):
639     """Creates an equity based on daily returns."""
640     string_directory = display_options['string_results_directory']
641     result_no = len(
642         [name for name in os.listdir(string_directory) if os.path.isfile(
643             os.path.join(
644                 string_directory,
645                 name
646             )
647         )]
648     ) / 2
649 
650     number_of_result_files_plus_1 = 1 + int(result_no)
651 
652     fig, ax = plt.subplots(figsize=(12, 8))
653     ax.xaxis.set_major_locator(
654         MonthLocator(
655             bymonthday=1,
656             interval=3,
657             tz=None
658         )
659     )
660     ax.xaxis.set_major_formatter(DateFormatter("%y-%m-%d"))
661 
662     plt.legend()
663 
664     df_daily_returns.plot(
665         y=['portfolio_value_normalized', '
    benchmark_portfolio_value_normalized'],
666         label=['Portfolio value in base currency (after fees)', 'Benchmark
    value in base currency (no fees considered)'],
667         title='Equity curve'
668     )
```

```
669
670     if boolean_save_to_disk:
671         plt.savefig(string_directory + '/equity_curve_' + str(
        number_of_result_files_plus_1) + '.png')
672
673     if boolean_plot:
674         plt.show()
```

Code Snippet 39: `components/_3_performance_evaluation.py` – The functions from this module calculate financial metrics from `df_trading_journal` after standardizing `df_trading_journal` into a fixed-frequency format.