Zealand

TRADING BOT

Algorithmic Trading: Harnessing the Power of AI for Enhanced Performance

DISSERTATION PROJECT REPORT

Name: Kacper Jakub Bytnar

Supervisor: Jens Peter Andersen

School: Zealand - Sjællands Erhvervsakademi

Dissertation start: 27.10.2023

Dissertation end: 08.01.2024

Count: 50 pages,

11 125 words,

67,190 characters without spacebars,

77,941 characters with spacebars

Y

TABLE OF CONTENTS

1. INTRODUCTION

Cryptocurrency markets have witnessed significant growth and volatility in recent years, creating both opportunities and challenges for investors. This context underscores the strategic convergence of algorithmic trading and artificial intelligence (AI), reshaping engagements with the dynamic crypto landscape.

Cryptocurrencies, notably Bitcoin and Ethereum, have transcended technological niches to permeate mainstream financial discourse. As these digital assets gain broader acceptance, the markets they inhabit attract attention from both seasoned and emerging investors. The promise of substantial returns is juxtaposed with challenges arising from the unpredictable nature of cryptocurrency price movements.

This project aims to explore the intersection of algorithmic trading and artificial intelligence (AI) to develop an advanced crypto trading bot with an AI-based strategy. The primary objective is to create a system that can autonomously execute trades based on learned patterns and market conditions. Conventional trading techniques often fail in algorithmic trading due to market shifts and pattern changes. The goal is to transcend conventional trading paradigms by developing a bot, that instead of reacting passively to market shifts, is designed to proactively navigate through them.

OBJECTIVES:

- Develop a robust algorithmic trading framework capable of interfacing with cryptocurrency exchanges.
- Implement an algorithmic trading strategy that leverages various trading techniques to make a profitable trading strategy.
- Implement an AI-based strategy that leverages machine learning techniques to make informed trading decisions.
- Compare the traditional algorithmic trading strategy with an AI-based strategy.
- Assess the trading bot's performance rigorously through comprehensive back-testing and real-time "paper trading" simulations, employing the Binance TestNet.

LITERATURE REVIEW:

The project initiates with a meticulous literature review spanning algorithmic trading, AI applications in financial markets, and prior studies on AI in crypto trading. This review aims to establish a robust foundation by elucidating current trends, challenges, and opportunities in these domains. The examination of algorithmic trading literature delves into methodologies and evolving strategies. The scrutiny of AI applications in financial markets investigates the integration of machine learning and predictive modeling. The review of prior studies on AI in crypto trading distills insights and informs the project's approach. This synthesis guides the subsequent development of an innovative AI-based crypto trading bot.

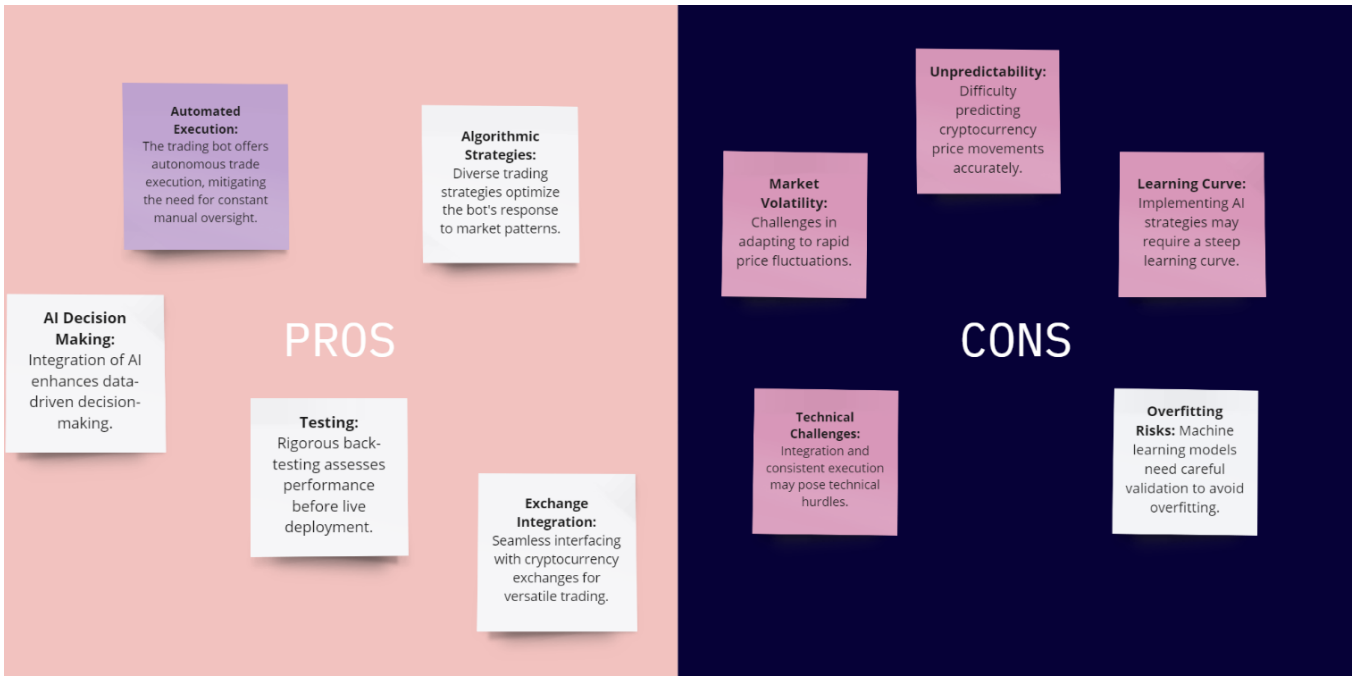
ACADEMIC BACKGROUND

During the 4th semester at Sjællands Erhvervsakademi, I pursued a Machine Learning elective, which became a pivotal point for cultivating my interest in artificial intelligence (AI) and data analysis. My enthusiasm for these fields further solidified during the exploration of a BITCOIN PRICE PREDICTION research project as part of the program's Synopsis. This research involved a comprehensive comparison of various machine learning algorithms and their performance in predicting Bitcoin prices.

Given my active involvement in crypto trading since 2017, I recognized the opportunity to synergize my Python programming, data analysis, machine learning, and trading skills. Motivated by this realization, I embarked on the endeavor to leverage the insights gained from my research. The aim was to construct a versatile bot framework capable of executing trades through diverse trading strategies while facilitating seamless transitions between them. This academic background sets the stage for the development of an innovative AI-powered crypto trading bot.

PROS AND CONS ANALYSIS

While automated trading and diverse strategies enhance efficiency, challenges arise from market unpredictability, complex learning curves for AI, potential technical hurdles, and the risk of overfitting. Additionally, evolving regulations add uncertainties. A low-risk level approach is vital to maximize the profits and mitigate huge losses in algorithmic and AI-based cryptocurrency trading. Pros and cons are listed on the board below.=



ABSTRACT

This project aims to elevate the effectiveness of traditional algorithmic trading by integrating AI models. The objective is to identify the most relevant and predictive features, subsequently advancing to the development of an automated trading bot capable of securing profitable trades on real funds.

Data is sourced from two distinct repositories: historical price candles of Bitcoin and real-time data from the Binance API. This dual-source approach enables the algorithm to make instantaneous decisions on whether to buy, hold, or sell. Key features, including Garman-Klass Volatility, RSI, Bollinger Bands, ATR, MACD, and Bitcoin Volume, will be developed and rigorously compared to determine the most suitable ones for the specified trading strategy.

Insights and Innovations:

Live Decision-Making: Real-time data from the Binance API empowers the algorithm to make dynamic decisions, adapting to live market conditions.

Feature Selection Strategy: The project involves developing and comparing multiple features, selecting the most suitable ones to enhance the algorithm's predictive capabilities.

Simulated Portfolio Tracking: An artificial portfolio with a tracker is employed to monitor the algorithm's performance, offering insights into its strategic effectiveness.

Return on Investment (ROI) Indicator: The primary performance metric, coupled with visualizations, provides a comprehensive assessment of the algorithm's profitability.

TESTING PHASES:

Backtesting: Before implementation, the algorithm has to prove itself during the backtesting phase, simulating trades on historical data to measure and refine its performance.

Simulated Portfolio Live Trading: Live trading is executed on a simulated portfolio on a Test Net, refining the algorithm's adaptability and resilience.

Forward-testing provides insights into the algorithm's performance in current market conditions without risking real capital.

Success Indicators:

Performance metrics, such as log returns and Sharpe ratio, will be a foundation of the algorithm success assesment. These metrics provide quantitative insights into the algorithm's profitability, risk-adjusted returns, and overall performance, guiding ongoing refinements for optimal trading outcomes.

2. PROBLEM FORMULATION

HYPOTHESIS

Building upon the foundation of a prior research endeavor, where an in-depth synopsis was conducted to explore the effectiveness of machine learning models in predicting Bitcoin prices, this journey seamlessly transitions into the realm of automated trading. Drawing from the insights gained during the meticulous analysis of historical data, technical indicators, and sentiment patterns, this project leverages that wealth of experience to develop an AI-powered algorithmic trading bot. The aim is to not only apply theoretical knowledge but to bridge the gap between research and real-world application, ushering in a new chapter in the intersection of machine learning and cryptocurrency trading.

1. Methodological Insights:

The review of past academic papers provided crucial insights into feature engineering strategies, emphasizing the significance of technical indicators and sentiment analysis. The exploration of diverse machine learning models, such as regression and ensemble methods, guided our model selection.

2. Temporal Analysis and Risk Management:

Studies on temporal analysis techniques for time-series data in cryptocurrency markets informed our approach to handling Bitcoin's inherent volatility. Understanding risk management strategies, particularly adaptive trading and portfolio diversification, shaped our project's risk mitigation measures.

3. Ethics and Real-world Applicability:

Identifying gaps in discussions around ethical considerations in algorithmic trading prompted a conscious effort to integrate responsible AI practices into our project. Emphasizing the need for real-world applicability, our project aims to bridge the gap between theoretical models and practical implementation in cryptocurrency trading scenarios.

PROBLEM STATEMENT:

Automated trading has become a pervasive force, but the challenge persists in creating strategies that are not only automated but also possess the finesse to thrive in dynamic market conditions. Traditional algorithmic trading, often bound by static rules, struggles to keep pace with the fluidity of live markets. The goal of this dissertation is to identify whether algorithms using an AI-based strategy are capable of dynamic decision-making, leveraging both historical data and real-time data stream from the Binance API.

RESEARCH QUESTIONS

HARMONIZING AI WITH ALGORITHMIC TRADING:

How can the synergy of AI algorithms and algorithmic trading be orchestrated to facilitate effective decision-making in bitcoin trading to generate profitable results?

IMPACT OF FEATURE SELECTION ON ALGORITHMIC TRADING:

How does the selection of specific features, such as technical indicators (e.g., RSI, Bollinger Bands) and market data, influence the accuracy and performance of AI-powered trading strategies?

REAL-TIME DATA IMPACT:

How does the real-time analysis of the live data elevate the adaptability and responsiveness of the trading algorithm, and what tangible benefits does it introduce?

IS AI EMPOWERED TRADING MORE EFFECTIVE THAN CONVENTIONAL TRADING STRATEGY?

To what extent does the integration of artificial intelligence (AI) in trading strategies contribute to the overall effectiveness and performance of trading systems, and how does it compare to conventional trading strategies in terms of profitability, risk management, and adaptability to dynamic market conditions?

ARE WIN RATIO, LOGARITHMIC RETURNS AND SHARPE RATIO SUITABLE PERFORMANCE METRIC?

In what ways do logarithmic returns, calculated percentage win ratio, Sharpe ratio, and mean returns contribute to a thorough evaluation of the algorithm's performance?

3. METHODOLOGY

The research methodology adopts a systematic process for the development of a sophisticated cryptocurrency trading bot. The initial phase involves leveraging the Binance API to collect historical and real-time market data. This data is meticulously managed and organized using Python, along with Pandas and NumPy for efficient data manipulation. Rigorous data cleaning procedures are applied to address missing or inconsistent data, ensuring a clean dataset for subsequent analysis.

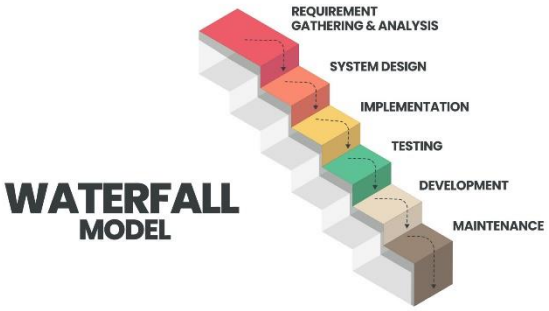
The subsequent step involves a comprehensive Exploratory Data Analysis (EDA) to unveil statistical properties and patterns within the market data. Key metrics are visualized to identify potential correlations and patterns, providing valuable insights for strategy development. The methodology emphasizes the implementation of basic trading strategies, including volume and return-based strategies. This involves the creation of universal backtesting and forward testing classes, laying the foundation for strategy evaluation.

To enhance adaptability to market conditions, machine learning models are integrated into the trading bot. A comparative analysis between AI-based and conventional strategies is conducted, evaluating their effectiveness and robustness in different market scenarios. The live trading bot class is then implemented, connecting to the Binance TestNet for simulated live trading. The utilization of the ThreadedWebsocketManager library ensures a seamless live data stream for strategy execution.

The methodology extends its scope to user interaction through the integration of a graphical user interface (GUI). This interface allows users to select from various strategies and customize bot parameters with ease. Future expansion plans encompass exploring additional strategies, refining the user interface, and considering live trading integration on the primary Binance exchange platform. This comprehensive approach ensures the alignment of research objectives with the iterative development of the trading bot, optimizing its performance and adaptability.

WORKFLOW PLANNING IN WATERFALL METHODOLOGY

The Waterfall Approach keeps me on a steady towards path towards building a functional prototype for the bot. Initial implementation focuses on a simple strategy that involves buying and selling the asset whenever appropriate signals are identified. The goal is to achieve a satisfying prototype working in console as a script then create a very simple GUI with buttons such as Start Trading and constantly displayed executed transactions. Last step is developing an AI-based trading strategy to find out whether AI would outperform traditional trading. This has to be carefully planned through the 10 weeks, making sure that all the crucial steps are achieved in my workflow plan bel



4. PLANNING THE WATERFALL WORKFLOW OF THE SYSTEMATIC DEVELOPMENT:

Week	Workflow Plan
43 Project Setup	➢ Configure Jupyter Notebook environment. ➢ Research materials on algorithmic trading and cryptocurrency markets. ➢ Fetch historical data from the Binance API, ensuring the dataset is comprehensive for the analysis and strategy.
44 Research and Strategy Planning:	➢ Read on basic trading strategies suitable for cryptocurrency markets. ➢ Define the scope and objectives of the trading bot project. ➢ Outline the workflow for the development process. ➢ Explore algorithmic trading resources and communities for better topic insights.
45 Data Retrieval Script:	➢ Develop a script in Python to connect to the Binance API and retrieve historical price data. ➢ Clean the dataset and handle all the outliers. ➢ Perform an exploratory data analysis (EDA) to find any market trends. ➢ Find the best practice for handling and storing financial data.
46	➢ Extend the EDA process to uncover additional statistical properties and patterns in the market data.

Exploratory Data Analysis (EDA)	➢ Visualize key metrics to identify potential correlations and trends more comprehensively. ➢ Document findings to inform the subsequent strategy development.
47 Prototype Development - Simple Strategy	➢ Test all of the functionalities in Jupyter Notebook ➢ Create a Class for the initial prototype with a simple trading strategy. ➢ Develop a Back-testing and Forward-Testing framework for the strategy. ➢ Transform conditions for buying and selling into executable strategy code. ➢ Perform first back-testing to validate the functionality of the prototype. ➢ Explore strategies for handling time-series data and creating lag features.
48 Prototype Testing and Refinement:	➢ Iterative testing of the prototype focused on identifying areas for improvement. ➢ Implement the improvements identified from testing feedback. ➢ Document the prototype's functionality and any modifications made.
49 Documentation	➢ Write the Introduction and formulate the research problem, explanation of trading strategies. ➢ Maintain comprehensive documentation of the prototype, including code comments. ➢ Ensure clarity in the documentation for ease of collaboration.
50 GUI connection with Flask	➢ Design a simple user interface using HTML, CSS, and Bootstrap for the front-end part. ➢ Integrate JavaScript functionalities to connect with the script data and real-time updates on transactions performed. ➢ Update the main Trader python script with the Flask backend code running on localhost to establish communication between the GUI and the trading bot. ➢ Develop Flask endpoints to handle requests from the GUI. ➢ Conduct extensive testing to ensure the stability and responsiveness of the integrated GUI and backend.
51 AI Strategy Implementation	➢ Investigate what AI tools or machine learning models would be the most suitable for algorithmic trading strategy. ➢ Prepare the integration plan for enhancing the trading bot with an AI-powered strategy. ➢ Deal with potential challenges and solutions related to AI strategy implementation. ➢ Document and show the idea of the strategy
52 Integration and Final Testing	➢ Integrate the AI-based trading strategy into the existing framework. ➢ Connect the GUI "Switch to AI strategy!" button to the strategy_update function to switch strategies from client. ➢ Implement necessary adjustments to accommodate the AI strategy's requirements for both trading framework and front-end. ➢ Conduct back and forward testing of the new strategy to measure performance of bot working with new AI strategy component.

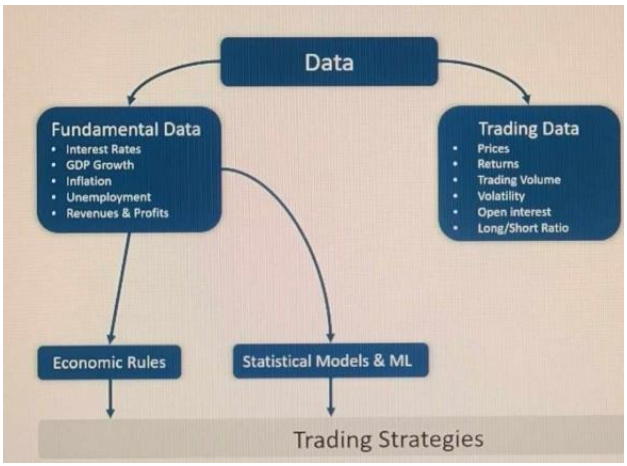
1 Conclusions, Documentation writing	➢ Write conclusions part to write down key findings and insights obtained throughout the entire development process. ➢ Write future plans for the improvement of the performance and adaptability of the trading bot. ➢ Address and answer research questions posed at the beginning of the project. ➢ Finalize the documentation with Reflection and Bibliography

5. INSIGHT INTO TRADING STRATEGIES

Understanding the theory of trading strategies is crucial for developing a robust and profitable trading bot. This section provides insights into various aspects of trading strategies, categorizing them based on data types and highlighting the evolution from conventional techniques to AI-based strategies.

CATEGORIZATION OF TRADING DATA

Trading strategies can be developed based on two primary categories of data:



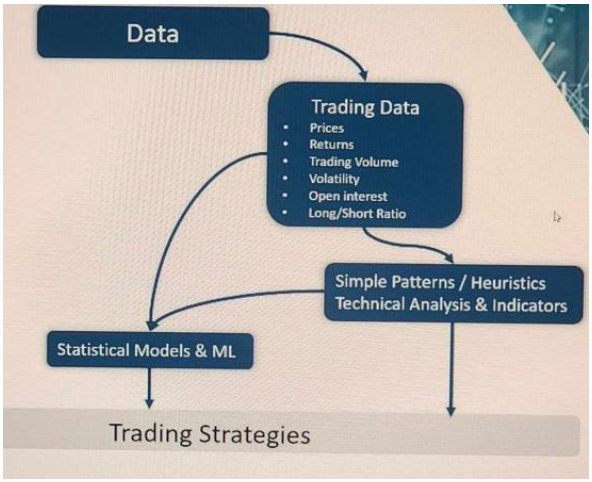
1. Fundamental Data based Strategies

Fundamental data includes economic indicators and asset-specific information, forms the basis for strategies grounded in economic rules and statistical models. Traders often use the fundamental analysis to identify mispriced assets by assessing their underlying value.

2. Trading Data based Strategies

Trading data includes price, returns, and volume which are foundation for the identification of patterns and trends.

Technical analysis, a key component of trading data strategies, involves using historical price and volume patterns to formulate strategies based on market trends, momentum, and reversal points.



CONVENTIONAL TECHNIQUES VS. AI-BASED STRATEGIES

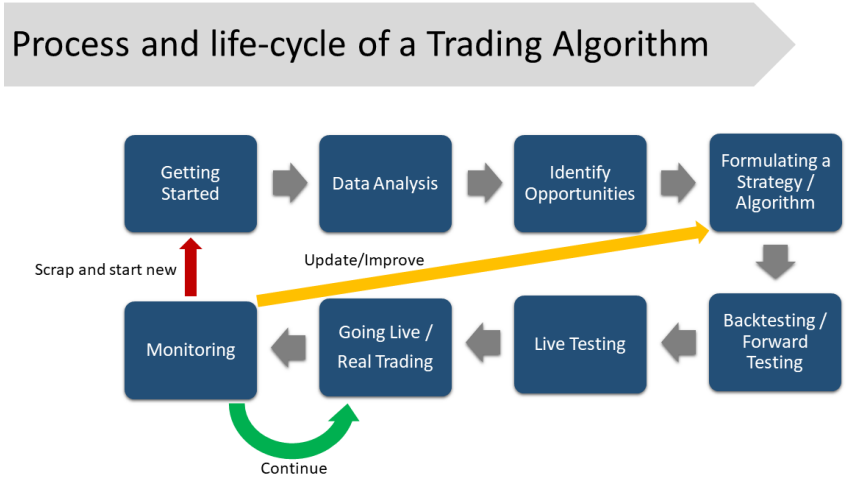
Currently the trading evolves, since AI capabilities were identified by first algorithmic traders the trading strategies are injected with AI-based logic what caused a shift from historical data reliance to dynamic adaptation to real-time market conditions. AI strategies, leveraging advanced algorithms and machine learning, offer adaptability, autonomous learning, and optimization capabilities, providing a significant advantage in navigating complex financial landscapes.

THE LIFECYCLE OF A TRADING STRATEGY

The lifecycle of a trading strategy involves steps such as instrument selection, choosing the right online trading platform with API trading capabilities, and conducting data analysis through exploratory data analysis (EDA). EDA includes data visualization, identifying inconsistencies, cleaning, and formatting data.

Identifying Trading Opportunities and Strategy Testing

Traders can formulate strategies based on rough ideas, defining them with programming languages. The critical step involves testing the strategy's performance through back-testing, considering risk and return metrics. Strategies must prove profitable after accounting for trading costs, which are inevitable as every exchange takes some fee from each transaction.



Forward Testing and Live Trading

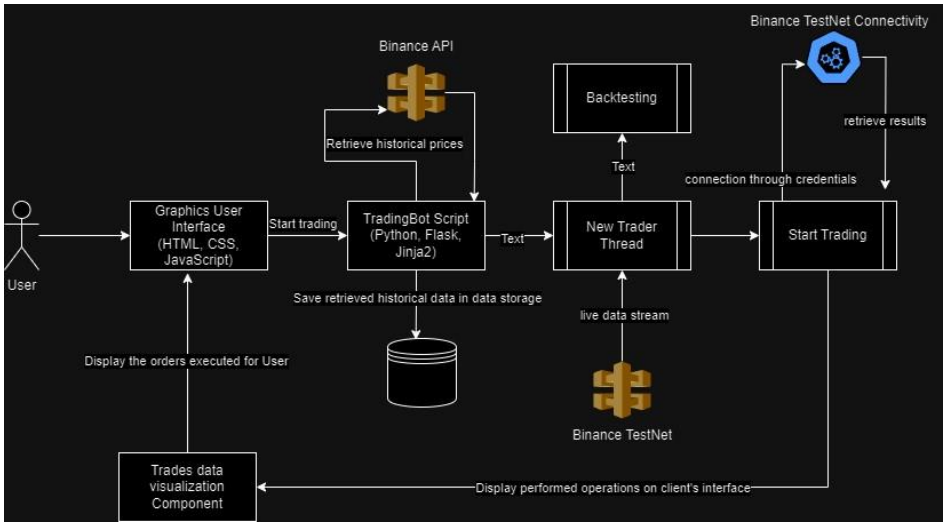
Forward testing, or out-sample back-testing, helps identify overfitting by testing strategies on new data. Once successful, traders move to live testing, implementing algorithms that stream real-time data, convert it into trading signals, and interact with online brokers. Live trading requires a stable technical infrastructure, often deployed on virtual servers for reliability.

Monitoring and Strategy Maintenance

Continuous monitoring is a must for in-production strategies. If live performance aligns with back-testing results, the strategy can continue. Otherwise, updates and improvements are necessary. Unprofitable strategies may need to be discarded, and plans should be updated with the latest data, especially for machine learning algorithms.

6. SYSTEM IMPLEMENTATION

SYSTEM ARCHITECTURE DIAGRAM



The diagram illustrates the basic interactions and components within the cryptocurrency trading bot. The user, through a GUI Client can engage with the trading python Script, the core logic of the system. The TradingBot Script interfaces with the Binance API for historical and live market data, executes trades, and interacts with a database for data storage. The trading operation runs on a new thread that triggers the processing of Binance live data streams, enabling real-time market monitoring and executing the strategy for live trading based on the specified back-tested strategy. The Binance TestNet Connection allows me to perform live trading on virtual assets, it is called "paper trading". The Data Visualization Component allows user to observe and analyze the trading outcomes.

SYSTEM REQUIREMENTS

In order to stay on the steady path to develop a functional system, it is crucial to identify the most important requirements for my trading bot system functionality.

FUNCTIONAL REQUIREMENTS:

- *FR1: Data Retrieval (Must):*

  The system shall retrieve historical and real-time market data from the Binance API, focusing on key candlestick attributes: date, closing price, and volume.

  Historical data should be in the same time interval as the real-time data, which should update dynamically via API live stream subscription.

- *FR2: Data Cleaning and Processing (Must):*

  The system should work on cleaned data, any missing or inconsistent data points are handled effectively.

- *FR3: Exploratory Data Analysis (EDA) (Must):*

  The system performs EDA to find any statistical properties and patterns within the market data.

  Visualizations should facilitate pattern identification and lead to a strategy development.

- *FR4: Return and Volume Calculations (Must):*

  The system accurately calculates returns using natural logarithms of the closing price ratio.

  Performance is mainly measured using the returns and cumulative returns, every trading session ends with summary of performance.

  Volume changes should be logarithmically transformed, categorizing returns into quantiles for insightful analysis.

- *FR5: Trading Strategy Formulation (Must):*

  Base trading strategy (Long-Short strategy), should be formulated based on observed patterns during EDA.

  Strategy specific conditions for buying, selling, and holding must be defined and indicate the signals.

- *FR6: Back-testing Framework (Must):*

  Versatile back-testing framework has to be capable of evaluating multiple trading strategies.

  The framework should calculate cumulative returns based on historical data and provide insights about strategy performance.

- *FR7: Automated Trading (Must):*

  Sn automated trading functionality has to be developed and connected to the Binance TestNet.

  Real-time data stream should trigger the algorithm to execute transactions and provide feedback to the user after each action taken while performing paper trading.

- *FR8: Graphical User Interface (GUI) (Must):*

  Graphical user interface has to be developed, user shall be able to select strategies, visualize trade outcomes and steer the bot with functional buttons like "Start Trading" or "Stop Trading".

- *FR9: Machine Learning Integration (Optional):*

  Identify the best machine learning model to enhance the trading bot's adaptability in the AI-based strategy.

An extensive comparison of the AI-based and conventional trading strategies should be conducted and documented with inferences.

NON-FUNCTIONAL REQUIREMENTS:

- NFR1: Performance (Must):

  The system must handle data retrieval, cleaning, and analysis efficiently to ensure real-time responsiveness. GUI interactions should be within acceptable response time limits for a seamless user experience.

- NFR2: Scalability (Optional):

  The system should be designed to scale its processing capabilities, accommodating a growing volume of historical and real-time data. The flexibility feature is optional but desirable for future multiple strategies integration .

- NFR3: Reliability (Must):

  Data retrieval and processing modules must be reliable, minimizing errors and ensuring accurate results.

  The back-testing and front-testing framework should provide consistent and reliable performance evaluations with logarithmic and percentage returns calculated.

- NFR4: Usability (Must):

  The GUI is simple yet intuitively designed, providing an user-friendly interface for non-technical users to interact with the trading bot.

  User manuals in documentation must be available to guide users on system functionalities.

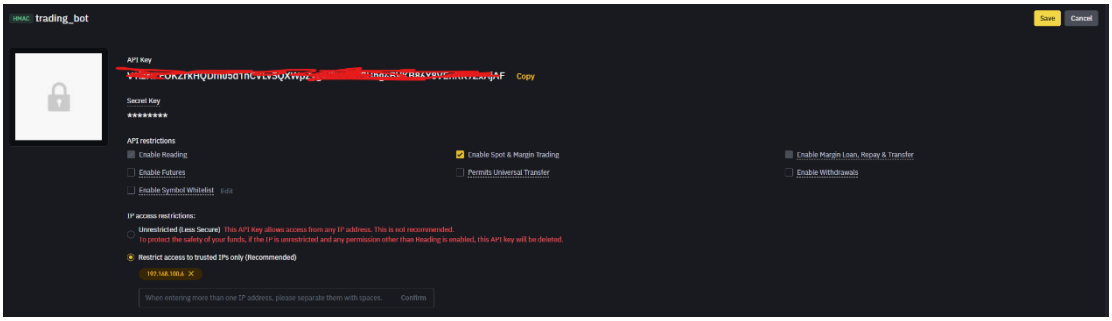
- NFR5: Performance Monitoring (Must):

  Performance monitoring tools (Logarithmic Returns, Percentage Returns, Sharpe Ratio) are tracking system performance over time. This feature aim is to identify areas for optimization.

7. DATA PREPARATION

BINANCE ACCOUNT API CREATION AND CONNECTION VIA RECEIVED CREDENTIALS

In order to gain access to the historical and real-time market data stream from Binance, the process requires to log into my private Binance account, navigate to the API Management section and create the private API which generates an unique API key and secret. It is crucial to specify the allowed private IP address that's eligible to access the exchange through API which is an important security step since these credentials can allow anyone to access my private assets on the Binance.



After obtaining the API key and secret, establishing a connection to Binance through a Python script involves using the Binance API client library. The library provides convenient methods for accessing market data, managing orders, and interacting with Binance services. This step requires installation of the "python-binance" library that provides me with everything I need to establish a successful API connection.

```
(base) C:\Users\matei>pip install python-binance
```

This Python script creates a connection to Binance using the Client that connects to my API key and secret, facilitating subsequent requests for historical price data and trading activities.

```
API_Key = "VKBSmOKljPJF673UkPloF5mZ5X4DruNUvWOOZ6zo8D74rQMtpA0blnPJsEbVTTq2"
Secret_Key = "bdVCIsdAwqSxzIgbRPPuBHr5n0KMgOxNw9JFFr00YUvKmZgGCr9vSdp4nhsYS59O"
```

```
from binance.client import Client
import pandas as pd
```

```
client = Client(api_key = API_Key, api_secret = Secret_Key, tld="com")
```

```
pd.to_datetime(timestamp, unit="ms")
```

```
Timestamp('2017-08-17 04:00:00')
```

```
def get_history(symbol, interval, start, end= None):
    bars = client.get_historical_klines(symbol = symbol,
                                        interval = interval, start_str = start, end_str = end, limit=1000)
    historicalPrices = pd.DataFrame(bars)
    historicalPrices["Date"] = pd.to_datetime(historicalPrices.iloc[:,0], unit= "ms")
    historicalPrices.columns = ["Open Time", "Open", "High", "Low", "Close", "Volume", "Clos Time",
                                "Quote Asset Volume", "Number of Trades", "Taker Buy Base Asset Volume",
                                "Taker Buy Quote Asset Volume","Ignore", "Date"]
    historicalPrices = historicalPrices[["Date", "Open", "High", "Low", "Close", "Volume"]].copy()
    historicalPrices.set_index("Date", inplace=True)
    for column in historicalPrices.columns:
        historicalPrices[column] = pd.to_numeric(historicalPrices[column], errors = "coerce")

    return historicalPrices
```

```
df = get_history(symbol="BTCUSDT", interval="1h", start=timestamp)
```

Executed script successfully provides me data stream that describes a price candle on specified time interval.

```
(base) C:\Users\matei>python "C:\Users\matei\TradingBot_Disseratation\TWM_Script.py"
{'e': '24hrMiniTicker', 'E': 1703906360617, 's': 'BTCUSDT', 'c': '41876.19000000', 'o': '42480.00000000', 'h': '43111.00
000000', 'l': '41300.00000000', 'v': '39293.35152000', 'q': '1662039268.82405200'}
{'e': '24hrMiniTicker', 'E': 1703906361250, 's': 'BTCUSDT', 'c': '41876.19000000', 'o': '42480.00000000', 'h': '43111.00
000000', 'l': '41300.00000000', 'v': '39293.58812000', 'q': '1662049176.72931430'}
{'e': '24hrMiniTicker', 'E': 1703906362574, 's': 'BTCUSDT', 'c': '41876.18000000', 'o': '42477.09000000', 'h': '43111.00
000000', 'l': '41300.00000000', 'v': '39294.29627000', 'q': '1662078781.12494180'}
{'e': '24hrMiniTicker', 'E': 1703906362858, 's': 'BTCUSDT', 'c': '41876.18000000', 'o': '42477.09000000', 'h': '43111.00
000000', 'l': '41300.00000000', 'v': '39294.31905000', 'q': '1662079734.93212200'}
{'e': '24hrMiniTicker', 'E': 1703906364619, 's': 'BTCUSDT', 'c': '41875.01000000', 'o': '42474.25000000', 'h': '43111.00
000000', 'l': '41300.00000000', 'v': '39295.48097000', 'q': '1662128059.34164890'}
```

Data in this form might look messy and contains some features that wouldn't be required for my analysis and strategy, therefore I will analyze the most important price candlestick attributes and extract them for further operations.

```
{
  "e": "kline",     // Event type
  "E": 123456789,   // Event time
  "s": "BNBBTC",    // Symbol
  "k": {
    "t": 123400000, // Kline start time
    "T": 123460000, // Kline close time
    "s": "BNBBTC",  // Symbol
    "i": "1m",      // Interval
    "f": 100,       // First trade ID
    "L": 200,       // Last trade ID
    "o": "0.0010",  // Open price
    "c": "0.0020",  // Close price
    "h": "0.0025",  // High price
    "l": "0.0015",  // Low price
    "v": "1000",    // Base asset volume
    "n": 100,       // Number of trades
    "x": false,     // Is this kline closed?
    "q": "1.0000",  // Quote asset volume
    "V": "500",     // Taker buy base asset volume
    "Q": "0.500",   // Taker buy quote asset volume
    "B": "123456"   // Ignore
  }
}
```

CANDLE DATAFRAME STRUCTURE

Obtaining the historical price data in the form of candlestick charts is crucial for conducting technical analysis and developing trading strategies. A candlestick chart represents price movements over a specific time interval, commonly referred to as a candle. Each candlestick provides information about the opening, closing, high, and low prices during that interval.

Candlestick Components

- Open Price (o): The price at which the asset started trading during the given time interval.
- High Price (h): The highest price reached by the asset during the time interval.
- Low Price (l): The lowest price reached by the asset during the time interval.
- Close Price (c): The price at which the asset concluded trading during the time interval.
- Volume(v): The total trading volume (number of assets traded) during the time interval.

That knowledge allows me to feed the algorithm with these attributes so it can react the same way as traditional trader performing a technical analysis on the candlestick graph.

```
df
```

Date	Open	High	Low	Close	Volume
2017-08-17 04:00:00	4261.48	4313.62	4261.32	4308.83	47.181009
2017-08-17 05:00:00	4308.83	4328.69	4291.37	4315.32	23.234916
2017-08-17 06:00:00	4330.29	4345.45	4309.37	4324.35	7.229691
2017-08-17 07:00:00	4316.62	4349.99	4287.41	4349.99	4.443249
2017-08-17 08:00:00	4333.32	4377.85	4333.32	4360.69	0.972807
...	...	...	...	...	...
2023-12-30 18:00:00	42489.00	42515.26	42329.62	42411.11	790.866750
2023-12-30 19:00:00	42411.11	42455.46	42339.24	42361.13	741.062640
2023-12-30 20:00:00	42361.13	42399.25	42199.98	42283.95	797.193820
2023-12-30 21:00:00	42283.95	42329.77	42223.99	42305.08	754.613100
2023-12-30 22:00:00	42305.08	42308.69	42218.87	42223.31	395.453460

55715 rows × 5 columns

```
df.to_csv('btc_data.csv', columns=['Close', 'Volume'])
```

```
(base) C:\Users\matei>python "C:\Users\matei\TradingBot_Disseratation\TWM_Script.py"
Time: 2023-12-30 03:26:58.045000  || Price: 41860.00000000
Time: 2023-12-30 03:26:59.233000  || Price: 41860.00000000
Time: 2023-12-30 03:27:00.060000  || Price: 41860.01000000
Time: 2023-12-30 03:27:00.930000  || Price: 41860.01000000
Time: 2023-12-30 03:27:02.340000  || Price: 41860.00000000
Time: 2023-12-30 03:27:02.826000  || Price: 41860.01000000
```

Example of simple Date and Price stream. 1

8. FEATURES ENGINEERING

PERFORMANCE METRICS:

The fundamental features are the performance metrics, which are crucial to asses the effectiveness of algorithmic trading strategies. Utilizing Logarithmic Returns, Cumulative Log Returns, Sharpe Ratio, and Mean Returns, the evaluation aims to provide a nuanced understanding of risk-adjusted returns from the transactions executed.

LOGARITHMIC RETURNS, the first performance metric are calculated by taking the natural logarithm of the ratio of the closing price of an asset today to its closing price on the previous day.

```
data["returns"] = np.log(data.Close.div(data.Close.shift(1)))
```

```
data.describe()
```

	Close	Volume	returns
count	55715.000000	55715.000000	55714.000000
mean	20898.976866	3060.588844	0.000041
std	15907.106248	4341.220599	0.008538
min	2919.000000	0.000000	-0.201033
25%	7933.500000	962.233038	-0.002640
50%	15465.830000	1682.426251	0.000088
75%	30429.995000	3289.242270	0.002812
max	68633.690000	137207.188600	0.160280

This process transforms simple returns into log returns, providing a mathematical basis for financial analysis. The resulting log returns are commonly used in academic research and financial modeling due to their favorable statistical properties.

The CUMULATIVE RETURNS ("creturns") are computed by taking the cumulative sum of the log returns and applying the exponential function to the result. This process allows for the transformation of log returns back into absolute returns and provides a continuous, normalized representation of the asset's price movement. Normalizing the prices with a base value of 1 simplifies the interpretation of the cumulative returns, making it easier to compare the performance of different assets over the same time frame.

```
data["creturns"] = data.returns.cumsum().apply(np.exp) # Normalized Prices with Base Value 1
```

DAILY VOLUME CHANGE (VOL_CH) is a very simply calculated feature for the historical prices dataset, it is assessed by using a logarithmic transformation of the ratio between the current day's volume and the preceding day's volume. This approach, capturing percentage changes, offers a nuanced perspective on the dynamics of trading activity, particularly in the context of financial markets.

```
data["vol_ch"] = np.log(data.Volume.div(data.Volume.shift(1)))
```

At this point the dataset it almost ready for the EDA process, although few more features should be engineered at this stage, which will also require further data cleaning.

RETURNS & VOLUME CATEGORIES

The returns metric feature is categorized into ten quantiles using the pd.qcut function, and corresponding labels ranging from -5 to 5 are assigned to each quantile. This process results in the creation of a "ret_cat" column, facilitating the analysis of return distribution in distinct categories. Using this approach allows me to use these categories in form of a scale of how profitable the executed orders were so that the algorithm can use them into consideration to take better actions in the future.

```
pd.qcut(data.returns, q = 10)
```

```
Date
2017-08-17 04:00:00                      NaN
2017-08-17 05:00:00      (0.000998, 0.00209]
2017-08-17 06:00:00       (0.00209, 0.00377]
2017-08-17 07:00:00       (0.00377, 0.00705]
2017-08-17 08:00:00       (0.00209, 0.00377]
                                ...
2023-12-30 18:00:00    (-0.00192, -0.000847]
2023-12-30 19:00:00    (-0.00192, -0.000847]
2023-12-30 20:00:00    (-0.00192, -0.000847]
2023-12-30 21:00:00     (8.76e-05, 0.000998]
2023-12-30 22:00:00     (-0.00352, -0.00192]
```

```
data["ret_cat"] = pd.qcut(data.returns, q = 10, labels = [-5, -4, -3, -2, -1, 1, 2, 3, 4, 5])
```

The same approach is applied to Volume category feature. Volume is one of the most important basic candle attribute therefore it's high impact on the price will be also categorized in a scale from -5 to 5.

Date	Close	Volume	returns	creturns	vol_ch	ret_cat	vol_cat
2017-08-17 04:00:00	4308.83	47.181009	NaN	NaN	NaN	NaN	NaN
2017-08-17 05:00:00	4315.32	23.234916	0.001505	1.001506	-0.708335	2	-5
2017-08-17 06:00:00	4324.35	7.229691	0.002090	1.003602	-1.167460	3	-5
2017-08-17 07:00:00	4349.99	4.443249	0.005912	1.009552	-0.486810	4	-4
2017-08-17 08:00:00	4360.69	0.972807	0.002457	1.012036	-1.518955	3	-5
...	...	...	...	...	...	...	...
2023-12-30 18:00:00	42411.11	790.866750	-0.001835	9.842837	-0.833230	-2	-5
2023-12-30 19:00:00	42361.13	741.062640	-0.001179	9.831237	-0.065044	-2	-1
2023-12-30 20:00:00	42283.95	797.193820	-0.001824	9.813325	0.073013	-2	2
2023-12-30 21:00:00	42305.08	754.613100	0.000500	9.818229	-0.054893	1	-1
2023-12-30 22:00:00	42223.31	395.453460	-0.001935	9.799252	-0.646172	-3	-5

RETURNS MEAN assesses the average profitability of a trading strategy. It measures the strategy's ability to generate consistent returns over a specified period. A positive Returns Mean signifies an overall profit, while a negative value suggests an average loss.

mean return and risk

```
18]: mu = data.returns.mean()
     mu

18]: 4.096467732821058e-05

19]: std = data.returns.std()
     std

19]: 0.008538432132621808
```

annualized mean return and risk

```
number_of_periods = 24 * 365.25
number_of_periods
```

```
8766.0
```

```
ann_mean = mu * number_of_periods
ann_mean
```

```
0.3590963614590939
```

```
ann_std = std * np.sqrt(number_of_periods)
ann_std
```

```
0.7994270954265839
```

The SHARPE RATIO evaluates the risk-adjusted performance of a trading strategy. It quantifies the excess return per unit of risk, considering both the strategy's returns and the inherent volatility.

A higher Sharpe Ratio indicates a more favorable risk-return profile, signifying superior performance in generating returns relative to the associated risk.

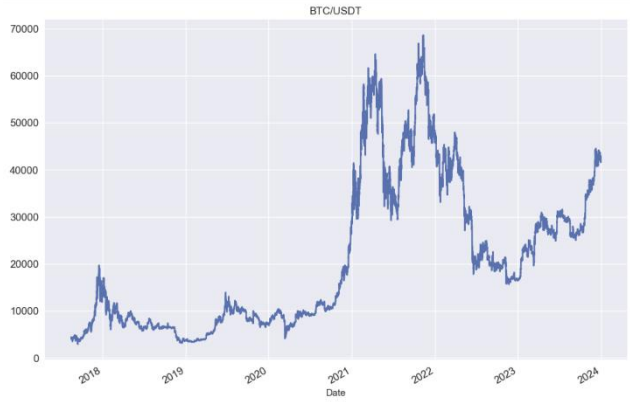
Risk-adjusted Return ("Sharpe Ratio")

```
24]: ann_mean / ann_std

24]: 0.449192132107401

25]: cagr / ann_std
```

9. EXPLORATORY DATA ANALYSIS (EDA)

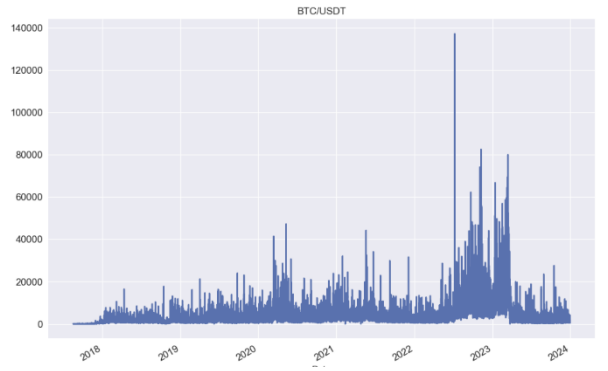
This process serves as a foundational step in understanding the underlying patterns and characteristics of the market data. Through visualizations and statistical examinations, this section's aim is to uncover any potential insights from the currently obtained data that will serve as a foundation for the development and refinement of trading strategies.

```
data.Close.plot(figsize = (12, 8), title = "BTC/USDT", fontsize = 12)
plt.show()
```



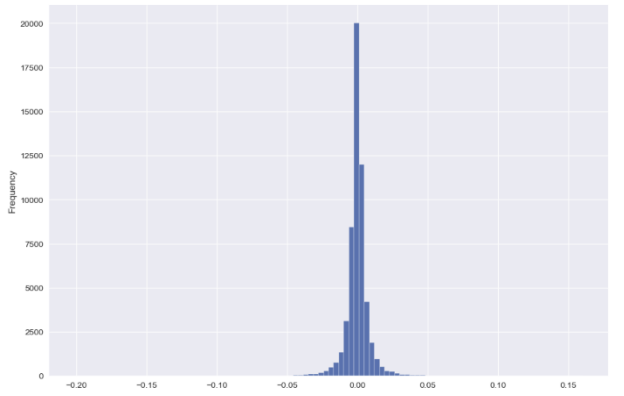
DATA CHART provides an overview of Bitcoin prices from August, 2017, to the present day. It offers a visual representation of the cryptocurrency's historical performance, allowing for the identification of trends, fluctuations, and potential patterns.

```
data.Volume.plot(figsize = (12, 8), title = "BTC/USDT", fontsize = 12)
plt.show()
```



VOLUME plot visualizes the trading volume associated with Bitcoin during previously specified time frame. Analyzing volume trends assists in understanding market activity and identifying periods of heightened or reduced trading interest.

```
data.returns.plot(kind = "hist", bins = 100, figsize = (12,8))
plt.show()
```



RETURNS PLOT illustrates the logarithmic returns of Bitcoin prices over time. Logarithmic returns offer a standardized measure, facilitating the assessment of the cryptocurrency's performance in a consistent manner.

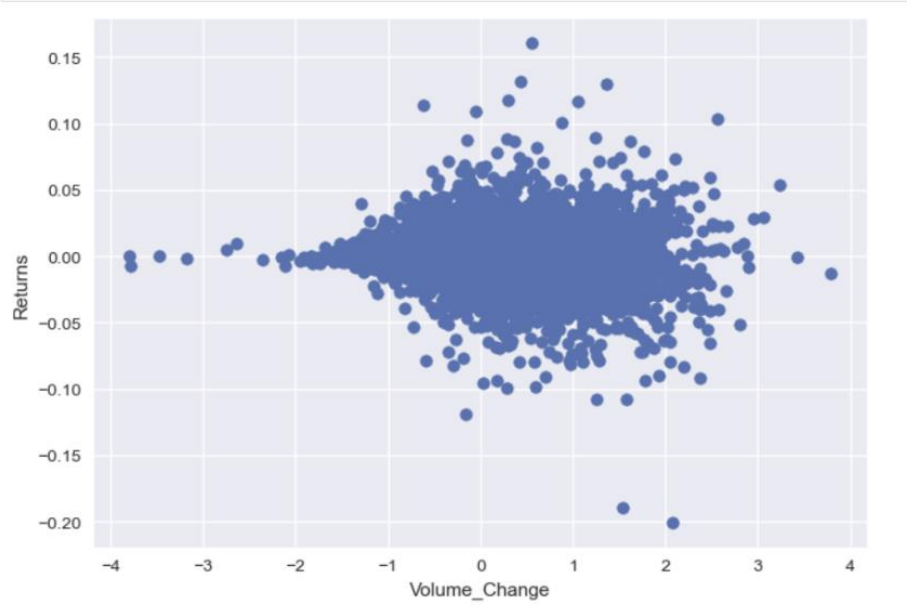
VOLUME CHANGES TO RETURNS PLOT

This plot combines volume changes and returns, highlighting the relationship between trading volume fluctuations and corresponding returns. The observed correlation between high volume changes and significant returns provides valuable insights for the formulation of trading strategies, as outlined in the conclusions.

During the analysis it's clearly visible that high volume changes go hand to hand with fixed returns.

Inferences from the plot could state that high volume changes result in either huge positive or negative returns.

This pattern will be used in my strategy to identify transaction signals.



During EDA process, the presence of multiple outliers in volume changes has been identified. To address this, the four largest and smallest volume change values were removed.

This approach leaves me with another issue of handling the resulting missing values. In this case a forward-filling technique is applied, ensuring a continuous and coherent dataset.

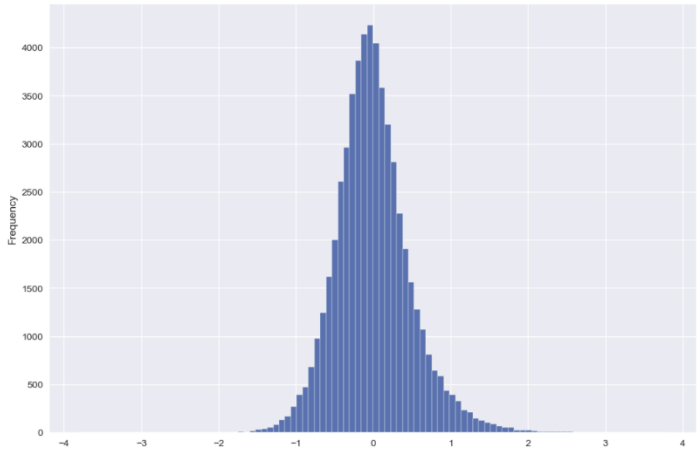
```
data.vol_ch.nlargest(20)
```

Date	
2017-09-06 23:00:00	inf
2019-06-07 22:00:00	inf
2021-02-11 05:00:00	inf
2023-03-24 14:00:00	inf
2018-01-04 05:00:00	5.256246
2021-04-25 08:00:00	5.051831
2017-08-20 00:00:00	3.794985
2017-08-26 05:00:00	3.428566
2023-08-29 14:00:00	3.240614
2023-10-01 22:00:00	3.058111
2022-03-27 20:00:00	2.949728

```
data.vol_ch.nsmallest(20)
```

Date	
2017-09-06 16:00:00	-inf
2019-06-07 21:00:00	-inf
2021-02-11 03:00:00	-inf
2023-03-24 12:00:00	-inf
2021-04-25 04:00:00	-5.644090
2018-01-04 03:00:00	-5.428025
2019-06-07 20:00:00	-4.780619
2017-08-19 23:00:00	-3.801014
2017-08-20 09:00:00	-3.782857
2017-08-26 04:00:00	-3.470297
2017-12-04 06:00:00	-3.178488

```
data.vol_ch.plot(kind = "hist", bins = 100, figsize = (12,8))
plt.show()
```



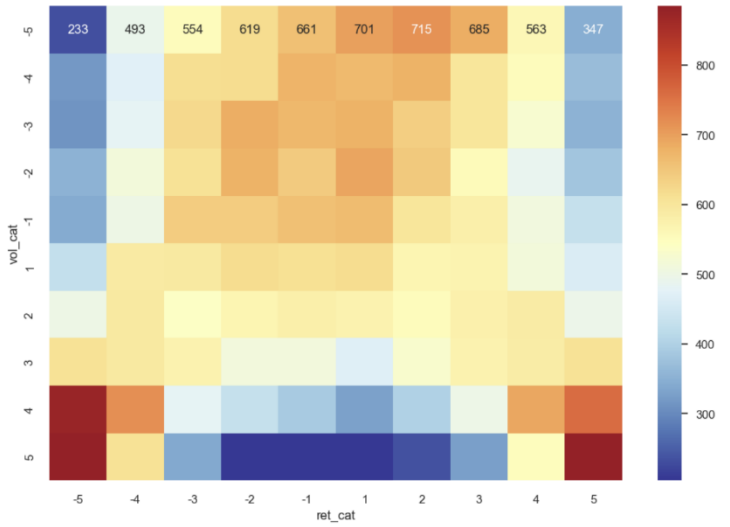
The VOLUME CHANGE HISTOGRAM serves as a visual representation of the cleaned volume changes distribution.

This histogram captures the frequency of various percentage changes in trading volume. Examining the histogram shows that volume change distribution graph is normally distributed and the outliers were successfully handled.

HEATMAP OF RETURN AND VOLUME CATEGORIES.

A heatmap of return and volume categories visually represents the relationship between trading volume and stock returns columns. The intensity of colors in the heatmap corresponds to the magnitude of returns and changes in trading volume. Darker colors signify higher intensity, indicating stronger relationships between specific volume and return levels. Patterns, trends, or anomalies can be identified by analyzing the distribution of colors across the heatmap, giving insights into how changes in trading volume correlate with various levels of stock returns.

```
plt.figure(figsize=(12, 8))
sns.set(font_scale=1)
sns.heatmap(matrix, cmap = "RdYlBu_r", annot = True, robust = True, fmt = ".0f")
plt.show()
```

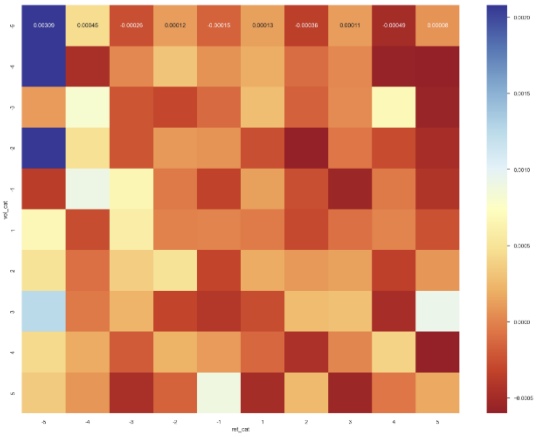


Concusions

-> Extreme increases in Trading Volume trigger extreme (positive and negative) returns.
-> Low to moderate changes in the Trading Volume occur frequently with low to moderate returns.

AVERAGE RETURNS HEATMAP

Analyzing the Average Returns Heatmap involves interpreting the relationship between different combinations of returns and volume categories. The identified contrarian signals provide valuable insights into potential market trends and reversals based on historical patterns. In this case we can clearly observe two market trends:



- Extremely High (positive) returns and Decreases in Volume is a Contrarian (mean-reverting) signal -> prices will fall.
- Extremely Low (negative) returns and Decreases in Volume is a Contrarian (mean-reverting) signal -> prices will rise.

10. FORMULATING A LONG-ONLY PRICE/VOLUME TRADING STRATEGY

The Long-Only Price/Volume Trading Strategy is designed to capitalize on the insights gained during the Exploratory Data Analysis (EDA), specifically focusing on the cluster characterized by "Extremely High (positive) returns and Decreases in Volume." This distinctive cluster, marked by falling prices, serves as a key signal for strategic decision-making.

IMPLEMENTATION OVERVIEW

The core implementation involves strategic actions in the BTC/USDT Spot market:

Initiate Long Position:

- Action: Buy BTC/USDT in the Spot market.
- Objective: Hold the position until a signal to sell is triggered.

Signal to Sell:

- Action: Sell the held position and transition to a neutral stance.
- Objective: Mitigate exposure to potential significant price drops.

Signal Absence:

- Action: Re-enter the market by buying back BTC/USDT.
- Objective: Resume the long position when the signal to sell has dissipated.

POSITION MANAGEMENT

The strategy requires a new column for "position". This column dynamically captures the current state of the trading position, indicating whether the strategy is in a long position, a neutral stance, or transitioning between the two.

```
data["position"] = 1 # Trading position -> long(1) for all bars: Buy-and-Hold
```

Sell and go Neutral (position = 0) *if* returns are very high (cond1) *and* vol_ch is negative (cond2)

"Very high returns" have to be defined by return threshold.

```
# getting returns threshold for very high returns (>= 90th percentile)
return_thresh = np.percentile(data.returns.dropna(), 90)
return_thresh
```

```
0.007045856596361843
```

Condition 1 - 10% of returns are greater than 0.7%, therefore if returns are higher than 0,7% then we call i t "very high return" what gives me the first condition for the strategy.

```
cond1 = data.returns >= return_thresh
cond1
```

Volume change thresholds for (moderate) Volume Decreases (between 5th and 20th percentile)

```
volume_thresh = np.percentile(data.vol_ch.dropna(), [5, 20])
volume_thresh
```

```
array([-0.72802012, -0.37337145])
```

Condition 2 - Volume change has to be between -73% and -37%

If both conditions are met, the position value is changed to 0

```
data.loc[cond1 & cond2, "position"] = 0
```

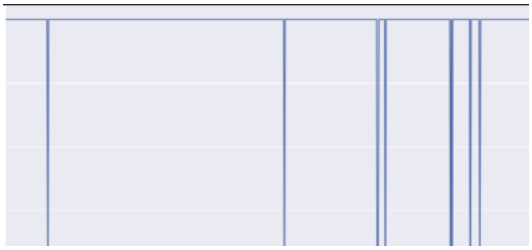
The historical dataset contains 541 cases when the strategy would go "neutral" and sell the asset.

```
data.position.value_counts()
```

```
position
1    55174
0      541
Name: count, dtype: int64
```

VISUALIZATION OF THE LONG HOLD STRATEGY FOR JUNE 2019

The line on the top represents the algorithm staying in the "Long" position and every vertical line means a "SELL" and position switch during the period.



THE LONG-ONLY STRATEGY BACKTESTING

The "Buy and hold" strategy is very simple – whenever the position is 1 the algorithm holds the asset and the return is very same as the asset price increase. If both conditions are met and the position reaches 0 the asset is sold and the position becomes "neutral".

Back-testing on the historical data will asses the efficiency of the volume shift based strategy. This phase employs a straightforward "Buy and Hold" benchmark, providing a baseline for comparison.

Performance metrics such as cumulative returns, risk assessments and Share Ratio will be my indicator for strategy's potential success in history.

In order to calculate return for the strategy the returns value is shifted, since the position for the next bar has to be predicted.

```
data["strategy"] = data.position.shift(1) * data["returns"]
data
```

Date	Close	Volume	returns	creturns	vol_ch	ret_cat	vol_cat	position	strategy
2017-08-17 04:00:00	4308.83	47.181009	NaN	NaN	NaN	NaN	NaN	1	NaN
2017-08-17 05:00:00	4315.32	23.234916	0.001505	1.001506	-0.708335	2	-5	1	0.001505
2017-08-17 06:00:00	4324.35	7.229691	0.002090	1.003602	-1.167460	3	-5	1	0.002090
2017-08-17 07:00:00	4349.99	4.443249	0.005912	1.009552	-0.486810	4	-4	1	0.005912
2017-08-17 08:00:00	4360.69	0.972807	0.002457	1.012036	-1.518955	3	-5	1	0.002457

For example on 2017-08-19 there is a neutral position and the strategy for next hour candle has 0 return since the asset was sold. This metric successfully calculates the log return for this simple strategy.

2017-08-19 07:00:00	4033.47	31.429222	0.011621	0.936094	-0.492978	5	-4	0	0.011621
2017-08-19 08:00:00	3999.00	18.006405	-0.008583	0.928094	-0.557011	-5	-4	1	-0.000000

In order to measure the strategy performance the cumulative return is calculated in a new column "cstrategy"

```
# normalized price with base = 1 for strategy
data["cstrategy"] = data["strategy"].cumsum().apply(np.exp)
```

```
data
```

Date	Close	Volume	returns	creturns	vol_ch	ret_cat	vol_cat	position	strategy	cstrategy
2017-08-17 04:00:00	4308.83	47.181009	NaN	NaN	NaN	NaN	NaN	1	NaN	NaN
2017-08-17 05:00:00	4315.32	23.234916	0.001505	1.001506	-0.708335	2	-5	1	0.001505	1.001506
2017-08-17 06:00:00	4324.35	7.229691	0.002090	1.003602	-1.167460	3	-5	1	0.002090	1.003602
2017-08-17 07:00:00	4349.99	4.443249	0.005912	1.009552	-0.486810	4	-4	1	0.005912	1.009552
2017-08-17 08:00:00	4360.69	0.972807	0.002457	1.012036	-1.518955	3	-5	1	0.002457	1.012036
...	...	...	...	...	...	...	...	...	...	...
2023-12-30 18:00:00	42411.11	790.866750	-0.001835	9.842837	-0.833230	-2	-5	1	-0.001835	25.142267
2023-12-30 19:00:00	42361.13	741.062640	-0.001179	9.831237	-0.065044	-2	-1	1	-0.001179	25.112637
2023-12-30 20:00:00	42283.95	797.193820	-0.001824	9.813325	0.073013	-2	2	1	-0.001824	25.066883
2023-12-30 21:00:00	42305.08	754.613100	0.000500	9.818229	-0.054893	1	-1	1	0.000500	25.079410
2023-12-30 22:00:00	42223.31	395.453460	-0.001935	9.799252	-0.646172	-3	-5	1	-0.001935	25.030934

BACK-TESTING RESULTS 1

As we can see on the chart, the strategy is significantly more profitable than a baseline "Buy and Hold" strategy both starting trading with 1 dollar.

For comparison the "Buy and Hold" strategy has generated 36% yearly return while the "Long-only Price/Volume" strategy generated 50% annual return.

```
ann_mean = data[["returns", "strategy"]].mean() * tp_year # annualized returns
ann_mean
```

```
returns     0.359096
strategy    0.506650
dtype: float64
```

TRADING COSTS

In the last section I have back-tested the strategy with impressive results. However there is one factor that cause a huge bias in profit measurements – I have ignored the trading fees.

Every time an order is executed, there is a transaction fee that goes to the exchange in form of a "tax". This approach is not very realistic therefore trading fee has to be calculated and taken into account.

Binance has 0.075% trading fees for user possessing some BNB, therefore let's assume that as the trading fee.

Level	30-Day Trade Volume (USD*)	and/or	BNB Balance	Maker / Taker	Maker / Taker BNB 25% off
Regular User	< 1,000,000 USD	or	≥ 0 BNB	0.1000% / 0.1000%	0.0750% / 0.0750%

The proportion of trading costs is calculated as a negative log return

```
# total proportional trading costs (negative log return)
ptc = np.log(1 - commissions) + np.log(1 - other)
ptc
```

```
-0.0008502863910375247
```

Using this data, I can calculate the net return by simply subtracting the proportional trading costs for each trade performed by the strategy then calculate the cumulated net strategy returns.

```
data["strategy_net"] = data.strategy + data.trades * ptc # strategy returns net of costs
```

```
data["cstrategy_net"] = data.strategy_net.cumsum().apply(np.exp)
```

BACK-TESTING RESULTS 2 of the strategy profits after taking the trading fees into consideration doesn't look as powerful and profitable as before, however still slightly better results are achieved compared to "Buy and Hold" strategy.

That can illustrate how much of an impact trading fees have on building a strategy that is significantly more profitable than simplest passive approach to just buy and hold.

It becomes evident that crafting a highly profitable strategy requires consideration of transaction costs. The strategy must consistently generate returns that exceed the associated fees, emphasizing the imperative need for effective trade execution to ensure sustained profitability.

11. STRATEGY OPTIMIZATION

In this section, the aim is to enhance the performance of the LongOnly trading strategy by optimizing the combination of strategy parameters. This involves developing a versatile back-testing framework, implemented through a "Backtester Class", which can be applied not only to our current strategy but also to various other trading strategies in the future.

To optimize the strategy I am working on a new dataset containing candle data on 1 hour interval with "returns" column which is crucial for any strategy.

```
data["returns"] = np.log(data.Close / data.Close.shift(1))
data
```

Date	Open	High	Low	Close	Volume	returns
2017-08-17 04:00:00	4261.48	4313.62	4261.32	4308.83	47.181009	NaN
2017-08-17 05:00:00	4308.83	4328.69	4291.37	4315.32	23.234916	0.001505
2017-08-17 06:00:00	4330.29	4345.45	4309.37	4324.35	7.229691	0.002090
2017-08-17 07:00:00	4316.62	4349.99	4287.41	4349.99	4.443249	0.005912
2017-08-17 08:00:00	4333.32	4377.85	4333.32	4360.69	0.972807	0.002457

All back-testing steps were gathered together into one universal back-testing function.

The back-testing function prepares relevant features, including log-transformed volume changes, and defines trading positions based on specified return and volume thresholds. It calculates a trading strategy, considering transaction costs (tc), and evaluates the cumulative returns of both the market and the strategy. The function returns the final strategy multiple, providing a concise tool for assessing the performance of my further developed strategies.

```
def backtest(data, parameters, tc):

    # prepare features
    data = data[["Close", "Volume", "returns"]].copy()
    data["vol_ch"] = np.log(data.Volume.div(data.Volume.shift(1)))
    data.loc[data.vol_ch > 4, "vol_ch"] = np.nan
    data.loc[data.vol_ch < -4, "vol_ch"] = np.nan

    # define trading positions
    return_thresh = np.percentile(data.returns.dropna(), parameters[0])
    cond1 = data.returns >= return_thresh
    volume_thresh = np.percentile(data.vol_ch.dropna(), [parameters[1], parameters[2]])
    cond2 = data.vol_ch.between(volume_thresh[0], volume_thresh[1])

    data["position"] = 1
    data.loc[cond1 & cond2, "position"] = 0

    # backtest
    data["strategy"] = data.position.shift(1) * data["returns"]
    data["trades"] = data.position.diff().fillna(0).abs()
    data.strategy = data.strategy + data.trades * tc
    data["creturns"] = data["returns"].cumsum().apply(np.exp)
    data["cstrategy"] = data["strategy"].cumsum().apply(np.exp)

    # return strategy multiple
    return data.cstrategy[-1]
```

The back-testing function run on previous parameters results with 10.16 usd of returns, which is still a satisfying result considering the trading has started with 1 usd.

```
backtest(data = data, parameters = (90, 5, 20), tc = -0.00085)
```

```
10.166646777111792
```

PARAMETER OPTIMIZATION USING THE BACKTESTER CLASS

The Back-tester class can be reused multiple times, hence a comprehensive backtest is conducted across all conceivable parameter combinations. Previous strategy parameters were pretty much just intuitive values, therefore it is crucial to find more optimal set of parameters with this technique. Python's "product" library from itertools facilitates this exhaustive exploration.

The subsequent step involves the creation of a dedicated dataset, meticulously filtered based on performance metrics. This parameters evaluation process ends with the triumphant set of parameters, the combination (95, 11, 27) stands out as the optimal choice, delivering markedly superior returns. This refinement underscores the importance of systematic parameter tuning in enhancing the strategy's efficacy during the back-testing period.

```
many_results.nlargest(20, "performance")
```

	returns	vol_low	vol_high	performance
2842	95	11	27	21.613189
2310	93	11	27	21.543026
2035	92	11	18	21.363159
2843	95	11	28	21.064513
2309	93	11	26	21.034945

To identify the best parameters, a new parameters dataset was created. This approach allows me to filter the data by performance column and retrieve the most optimal parameters set.

Results state that (95, 11, 27) are the best set of parameters for this strategy resulting with significantly better outperforming the baseline strategy by 113% in net profits. Additionally, the Sharpe Ratio of 0.74 is indicative of its risk-adjusted performance. This metric assesses the return generated per unit of risk undertaken. A Sharpe Ratio above 0 suggests a positive risk-adjusted return, and the obtained value signals a favorable balance between returns and volatility. Furthermore, the Annualized Mean Return of 0.48 represents the strategy's ability to generate consistent returns annually. It indicates a steady performance, aligning with the strategy's objective of sustainable and reliable returns.

```
tester.test_strategy(percentiles = (95, 11, 27))
```

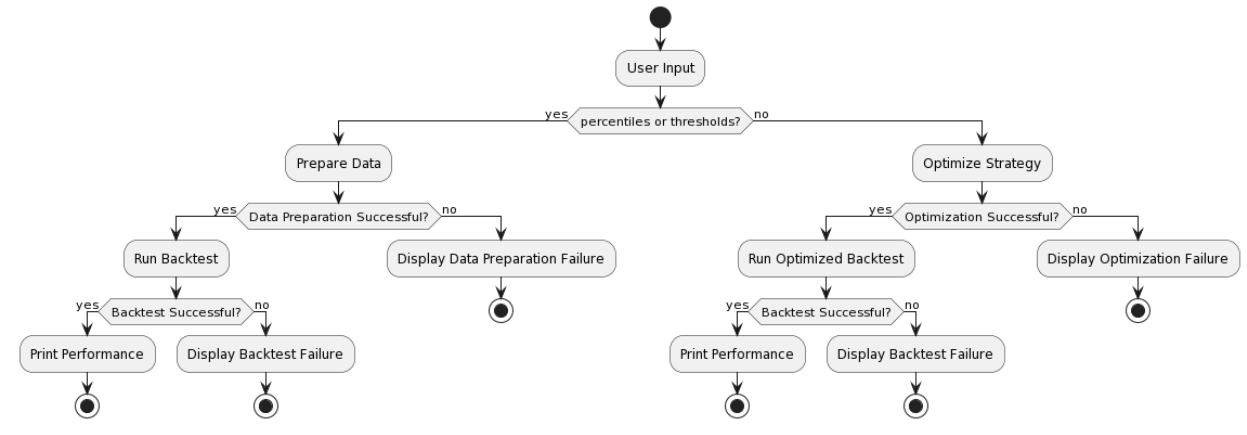
```
=============================================================================================
SIMPLE PRICE & VOLUME STRATEGY | INSTRUMENT = BTCUSDT | THRESHOLDS = 0.01132, [-0.5375  -0.27852]
---------------------------------------------------------------------------------------------
PERFORMANCE MEASURES:


Multiple (Strategy):         20.854728
Multiple (Buy-and-Hold):     9.799252
--------------------------------------
Out-/Underperformance:       11.055476


CAGR:                        0.611216
Annualized Mean:             0.476998
Annualized Std:              0.789805
Sharpe Ratio:                0.773882
=============================================================================================
```

12. BACKTESTER CLASS

The "Long_Only_Backtester" class is a comprehensive framework for the vectorized back-testing of trading strategies. In this case it is adjusted for the Long-Only strategy back-testing, however it can serve as a foundation for back-testing any strategy. It is a main tool to evaluate the performance of a specific trading strategy based on specified parameters and taking into consideration the transaction fees. Key

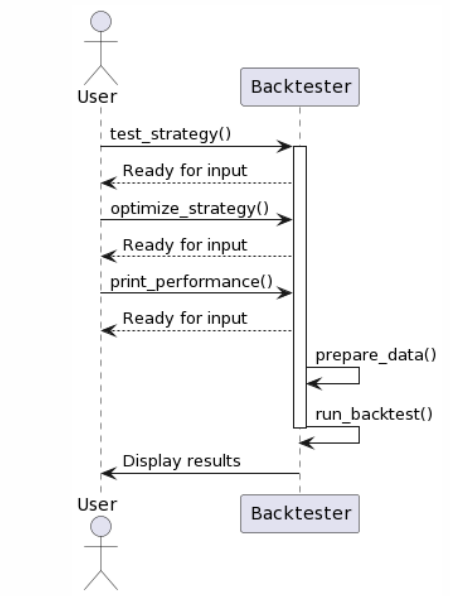
attributes include the file path of the dataset, the symbol (instrument) to be back-tested, the start and end dates to specify the testing period and proportional trading costs per trade.

"BACKTESTER" CLASS ACTIVITY DIAGRAM

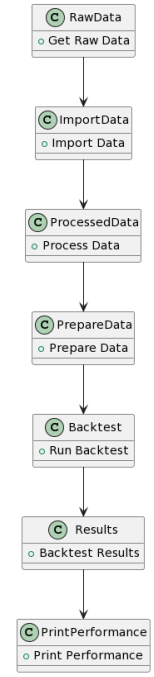


The Activity Diagram visually represents the dynamic user interactions within the "Long_Only_Backtester" framework. User starts the event with an input in form of strategy parameters, choosing between percentiles or thresholds. Depending on the user's decision, the data is prepared accordingly, leading to either a standard back-test of the strategy working on the passed parameters or an strategy optimization process to identify the most optimal set of parameters. The diagram showcases the potential outcomes, including successful data preparation, back-testing results, and optimization, as well as informing about any failure scenarios. This detailed visualization enhances the understanding of the user's decision-making process and the subsequent flow of activities within the system.

"BACKTESTER" CLASS SEQUENCE DIAGRAM



Following Sequence Diagram showcases the dynamic interactions between the user and the "Backtester" framework. An interaction initiated by the user's choice whether to test, optimize, or print performance, the "Backtester" performs desired actions.



"BACKTESTER" DATA FLOW DIAGRAM

The Data Flow Diagram helps visualizing the flow of data within the "Backtester" system. It begins by acquiring raw data then importing and processing it. The processed data from chosen time period is prepared for back-testing, and the actual test is executed. The results are obtained and the performance is printed for analysis. Each step in this diagram represents a crucial stage in the data flow.

The "Backtester" framework functionality is based on various methods that allow user to conduct the strategy testing process. The essential methods are:

- **get_data()**: Imports and processes the data for back-testing.
- **test_strategy()**: Prepares the data and conducts the strategy back-test, including reporting.
- **prepare_data()**: Cleans and organizes the data for back-testing based on specified percentiles or thresholds.
- **run_backtest()**: Executes the strategy testing phase considering transaction costs.
- **plot_results()**: Plots the cumulative performance of the trading strategy compared to a buy-and-hold strategy.
- **optimize_strategy()**: Conducts strategy back-tests for all various sets of parameters, allowing optimization and reporting.

FORWARD TESTING (TO AVOID/REMOVE DATA SNOOPING AND THE LOOKAHEAD BIAS)

Great back-testing results for simple volume fluctuations based strategy - Too good to be true?

Achieving 2000% percent net profit returns from the back-testing phase often prompts skepticism about real-world effectiveness, even in the constantly growing cryptocurrencies market. This is well reasoned, primarily due to the pitfalls of data snooping and look-ahead bias. Results are more than enough to make inferences that this strategy would generate high profits, however there still lies another risk. Would the optimized parameters translate to success with new, unseen volatile bitcoin prices and patterns?

In order to answer this question and address these challenges, my approach involves dividing the historical data into distinct back and forward testing datasets. The back-testing main goal is to optimize the strategy, the next stage, the forward-test is a real strategy's benchmark performed on unseen, real-time

data. This data partitioning is my defense mechanism against over-optimization and look-ahead bias, assuring the credibility of the trading strategy evaluation.

BACKTESTING & PARAMETERS TUNING

To prevent overfitting, the dataset is divided into Back-testing (August 17, 2017, to December 31, 2020) and Forward-testing sets.

```
filepath = "btc_data_1h.csv"
symbol = "BTCUSDT"
start = "2017-08-17"
end = "2020-12-31"
tc = -0.00085
```

```
tester = Long_Only_Backtester(filepath = filepath, symbol = symbol, start = start, end = end, tc = tc)
tester
```

```
Long_Only_Backtester(symbol = BTCUSDT, start = 2017-08-17, end = 2020-12-31)
```

```
tester.optimize_strategy((85, 98, 1), (2, 16, 1), (16, 35, 1))
```

```
Return_Perc: 94 | Volume_Perc: [12, 28] | Multiple: 17.68849
============================================================================================================
SIMPLE PRICE & VOLUME STRATEGY | INSTRUMENT = BTCUSDT | THRESHOLDS = 0.01107, [-0.52736 -0.27342]
------------------------------------------------------------------------------------------------------------
PERFORMANCE MEASURES:


Multiple (Strategy):         17.688488
Multiple (Buy-and-Hold):     6.712641
--------------------------------------
Out-/Underperformance:       10.975847


CAGR:                        1.343699
Annualized Mean:             0.851759
Annualized Std:              0.887773
Sharpe Ratio:                1.513561
------------------------------------------------------------------------------------------------------------
```

Back-testing conducted first provides insights into historical efficacy, with the backtester class optimizing parameters for adaptability. These parameters, derived from simulated trades on historical data, are applied in the subsequent Forward Testing phase.

```
return_thresh = tester.return_thresh
return_thresh
```

```
0.011069826844302733
```

```
volume_thresh = tester.volume_thresh
volume_thresh
```

```
array([-0.55039207, -0.27342444])
```

Return and volume thresholds from back-testing phase are passed to the forward testing phase.

DISCREPANCIES BETWEEN BACKTESTING AND FORWARD TESTING ANALYSIS

Despite promising back-testing outcomes, the forward testing phase revealed less favorable results, with the "Buy-and-Hold" approach outperforming the strategy on unseen data. This disparity poses a challenge in adapting the strategy to the unpredictable nature of volatile market prices.

There are 2 main factors influencing the performance difference between these 2 testing stages:

- **Data Snooping / Over-Optimization (Partly):** The strategy's historical performance may have been influenced by specific dataset characteristics, leading to over-optimization. This can result in reduced adaptability to new market conditions.
- **Look-Ahead Bias (Partly):** Inherent biases in the back-testing process, where the model inadvertently incorporates future information, contribute to disparities between expected and realized outcomes in forward testing.

```
filepath = "btc_data_1h.csv"
symbol = "BTCUSDT"
start = "2021-01-01"
end = "2023-12-31"
tc = -0.00085
```

```
tester = Long_Only_Backtester(filepath = filepath, symbol = symbol, start = start, end = end, tc = tc)
tester
```

```
Long_Only_Backtester(symbol = BTCUSDT, start = 2021-01-01, end = 2023-12-31)
```

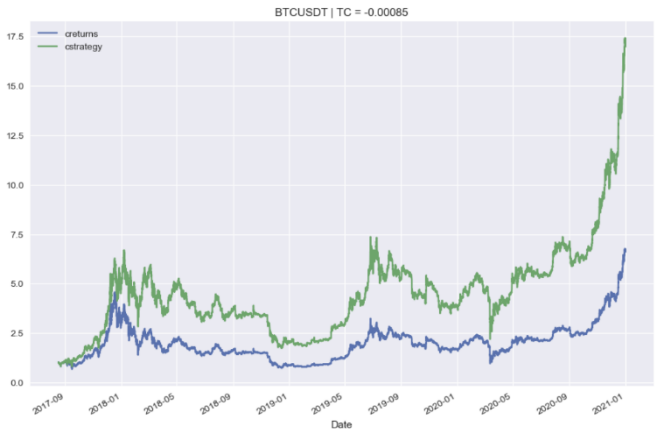
```
tester.test_strategy(thresh = (return_thresh, volume_thresh[0], volume_thresh[1]))
```

```
============================================================================================================
SIMPLE PRICE & VOLUME STRATEGY | INSTRUMENT = BTCUSDT | THRESHOLDS = 0.01107, [-0.55039 -0.27342]
------------------------------------------------------------------------------------------------------------
PERFORMANCE MEASURES:


Multiple (Strategy):         1.151449
Multiple (Buy-and-Hold):     1.456221
--------------------------------------
Out-/Underperformance:       -0.304772


CAGR:                        0.048253
Annualized Mean:             0.047127
Annualized Std:              0.660663
Sharpe Ratio:                0.073038
============================================================================================================
```

The observed performance gaps highlight the need for refined strategies capable of navigating unforeseen market dynamics. Strategies that are excessively tailored to historical data may lack adaptability, making them susceptible to underperformance in real-world scenarios.

BACKTESTING VS FORWARDTESTING RESULTS PLOT

Back-testing results plot



Forward-testing results plot

THE LONG-ONLY STRATEGY ASSESMENT

Long Only strategy is based on the simple assumptions and patterns found during EDA process. It was assumed that extremely high positive returns and moderate to high decreases in the trading volume is a contrarian mean reverting signal which was true in the back-testing period, but this strategy wouldn't perform that well on the unseen data. Patterns change over time and it seems that in the forward-testing period this patterns isn't repetitive enough to achieve "bank-breaking" results.

Automated trading is the most challenging part of the process. It can be done in infinite amount of ways and there is no best strategy that can be hard-coded. Therefore I focused on creating a framework for automated trading that can be used for many strategies so it would be possible to change or adjust them for highest efficiency and depending on the volatile market conditions. For dissertation purposes, This process is done on the Binance TestNet network with access via GitHub.

13. LIVE DATA STREAM WITH PYTHON-BINANCE

There are two alterantives to stream live data with python-binance:

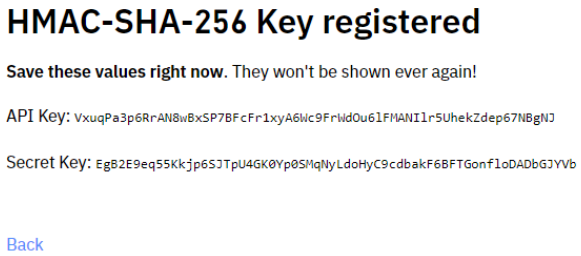
- using **ThreadedWebsocketManager** -or-
- using **BinanceSocketManager**

Streaming live data is fundamental for the trading script that works based watching the price candles and executing trades based on the patterns identified by the strategy.

In order to access the live prices stream from the Binance via either ThreadedWebsocketManager or using BinanceSocketManager, first it's crucial to establish a connection with the Testnet. This process requires creating a client's instance connected to my personal API secret keys and specifying the Testnet connection with testnet=True attribute.

```
client = Client(api_key = api_key, api_secret = secret_key, tld = "com", testnet = True) # Testnet!!!
```

In order to establish connection with the Binance Spot Test Network, It was necessary to sign up via GitHub account and receive private credentials that can be used in the paper-trading process to check strategy's performance live.



CONNECTION IS ESTABLISHED VIA THREADEDWEBSOCKETMANAGER, WHICH IS ONLY EXECUTABLE VIA SCRIPT.

```
twm = ThreadedWebsocketManager()
twm.start()
```

ThreadedWebsocketManager (TWM) connection is initiated by creating an instance of the manager (twm) and subsequently invoking the start() method, facilitating the concurrent management of WebSocket connections and enabling asynchronous communication for data streaming and event processing.

Callback function

```
def stream_candles(msg):
    ''' define how to process incoming WebSocket messages '''

    # extract the required items from msg
    event_time = pd.to_datetime(msg["E"], unit = "ms")
    start_time = pd.to_datetime(msg["k"]["t"], unit = "ms")
    first   = float(msg["k"]["o"])
    high    = float(msg["k"]["h"])
    low     = float(msg["k"]["l"])
    close   = float(msg["k"]["c"])
    volume  = float(msg["k"]["v"])
    complete=       msg["k"]["x"]

    # print out
    print("Time: {} | Price: {}".format(event_time, close))

    # feed df (add new bar / update latest bar)
    df.loc[start_time] = [first, high, low, close, volume, complete]
```

valid intervals - 1m, 3m, 5m, 15m, 30m, 1h, 2h, 4h, 6h, 8h, 12h, 1d, 3d, 1w, 1M

The stream_candles callback function is central to processing incoming WebSocket messages in real-time financial data streaming. It extracts essential candlestick details and updates a DataFrame, facilitating dynamic market information representation crucial for live data analysis and algorithmic trading strategies.

DATA STREAM VIA BINANCESOCKETMANAGER

connection via the BinanceSocketManager is established by creating an asynchronous client (client) and initializing the BinanceSocketManager (bm). The Kline socket (ts) is then configured for the "BTCUSDT" symbol with a 1-minute interval. The asynchronous context manager is utilized to receive WebSocket messages, calling the stream_candles callback function to process and update a DataFrame (df). The connection is closed after processing a specified number of messages, exemplifying the integration of live financial data streaming from Binance into a structured DataFrame for subsequent analysis and strategy development.

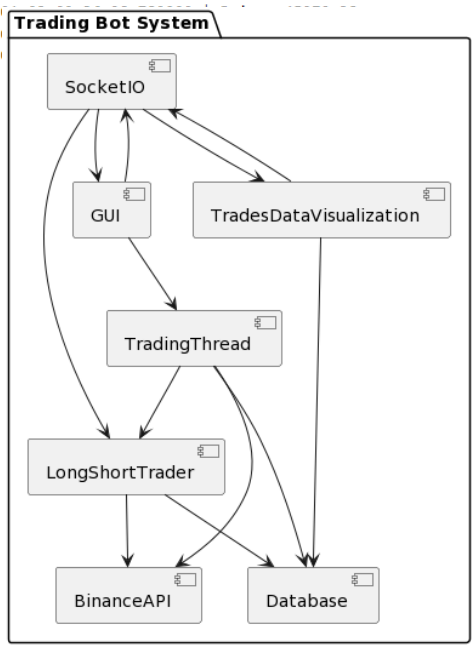
```
async def main():
    client = await AsyncClient.create()
    bm = BinanceSocketManager(client)
    ts = bm.kline_socket(symbol = "BTCUSDT", interval = "1m")

    async with ts as tscm:
        for _ in range(120):  # This is just an example to limi
            res = await tscm.recv()
            stream_candles(res)

    await client.close_connection()
await main()
```

```
Time: 2024-01-03 02:36:17.951000 | Price: 45288.83
Time: 2024-01-03 02:36:20.082000 | Price: 45286.57
Time: 2024-01-03 02:36:22.137000 | Price: 45286.56
Time: 2024-01-03 02:36:24.269000 | Price: 45286.48
Time: 2024-01-03 02:36:26.661000 | Price: 45286.49
Time: 2024-
Time: 2024-
Time: 2024-
```

13. LONGONLYTRADER CLASS – THE CORE FRAMEWORK FOR TRADES

TRADING CLASS ACTIVITY DIAGRAM

The diagram represents the dynamic behavior and interactions within the LongShortTrader class, encapsulating the essential activities of the bot's trading framework. This diagram showcases the sequential flow of actions, decision points, and exchanges of information as the trading bot executes its strategies, processes market data, and manages trading positions. It can provide an overview of the class's operational logic and encapsulates the key functionalities behind the trading activities in response to real-time market events.



The LongOnlyTrader class is a core trading framework designed for seamless live data streaming and trade execution. Key attributes include the financial instrument symbol (symbol), selected candlestick interval (bar_length), and a dynamic DataFrame (data) for real-time data representation.

Methods:

- start_trading(self): Initiates trading by subscribing to a live data stream using ThreadedWebsocketManager.
  If the specified bar_length aligns with permissible intervals, the class connects to the Kline WebSocket for real-time data processing.

- stream_candles(self, msg): A callback function processing WebSocket messages, updating the DataFrame (data) with essential OHLCV information for live market monitoring.

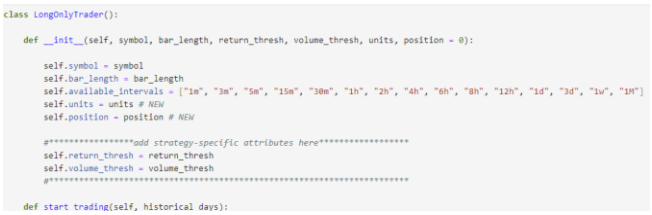
```
class LongOnlyTrader():

    def __init__(self, symbol, bar_length):
        self.symbol = symbol
        self.bar_length = bar_length
        self.data = pd.DataFrame(columns = ["Open", "High", "Low", "Close", "Volume", "Complete"])
        self.available_intervals = ["1m", "3m", "5m", "15m", "30m", "1h", "2h", "4h", "6h", "8h", "12h", "1d", "3d", "1w", "1M"]

    def start_trading(self):

        self.twm = ThreadedWebsocketManager()
        self.twm.start()

        if self.bar_length in self.available_intervals:
            self.twm.start_kline_socket(callback = self.stream_candles,
                                        symbol = self.symbol, interval = self.bar_length)

    def stream_candles(self, msg):

        # extract the required items from msg
        event_time = pd.to_datetime(msg["E"], unit = "ms")
        start_time = pd.to_datetime(msg["k"]["t"], unit = "ms")
        first   = float(msg["k"]["o"])
        high    = float(msg["k"]["h"])
        low     = float(msg["k"]["l"])
        close   = float(msg["k"]["c"])
        volume  = float(msg["k"]["v"])
        complete=       msg["k"]["x"]

        # print out
        print("Time: {} | Price: {}".format(event_time, close))

        # feed df (add new bar / update latest bar)
        self.data.loc[start_time] = [first, high, low, close, volume, complete]
```

This class provides a streamlined interface for long-only trading strategies, facilitating efficient live data integration and trade handling.

Script executed in Windows Console:

```
(base) C:\Users\matei>python "C:\Users\matei\TradingBot_Disseratation\Part2\Trader_LongOnly.py"
Time: 2024-01-04 21:14:48.901000 | Price: 44323.88
Time: 2024-01-04 21:14:51.251000 | Price: 44327.01
Time: 2024-01-04 21:14:53.335000 | Price: 44315.43
Time: 2024-01-04 21:14:55.825000 | Price: 44315.43
Time: 2024-01-04 21:14:58.272000 | Price: 44315.43
Time: 2024-01-04 21:15:00.039000 | Price: 44302.0
Time: 2024-01-04 21:15:02.095000 | Price: 44283.16
Time: 2024-01-04 21:15:04.135000 | Price: 44263.93
Time: 2024-01-04 21:15:06.183000 | Price: 44246.01
Time: 2024-01-04 21:15:08.336000 | Price: 44246.01
Time: 2024-01-04 21:15:10.620000 | Price: 44245.75
```

STRATEGY PARAMETERS



```
class LongOnlyTrader():

    def __init__(self, symbol, bar_length, return_thresh, volume_thresh, units, position = 0):

        self.symbol = symbol
        self.bar_length = bar_length
        self.available_intervals = ["1m", "3m", "5m", "15m", "30m", "1h", "2h", "4h", "6h", "8h", "12h", "1d", "3d", "1w", "1M"]
        self.units = units # NEW
        self.position = position # NEW

        #*****************add strategy-specific attributes here******************
        self.return_thresh = return_thresh
        self.volume_thresh = volume_thresh
        #************************************************************************

    def start_trading(self, historical_days):
```

The **__init__** method initializes an instance of the class with attributes specific to the trading strategy. It takes parameters such as the financial instrument symbol, selected bar_length for candlestick data, and strategy-specific thresholds for returns (return_thresh) and volume changes (volume_thresh). The units attribute specifies amount of bitcoin in the wallet.

DEFINE STRATEGY METHOD

The define_strategy method formulates a trading strategy tailored to the provided data in the trading framework class. It involves extracting relevant features, such as log returns and log volume changes, and subsequently applies strategy-specific conditions to derive a position column. The resulting prepared data, stored in self.prepared_data, serves as the foundation for subsequent trade execution within the framework.

The output is a prepared dataset with the live trading positions so that the process of data preparation is automated.

Every time a bar is "Completed", the strategy is re-defined so the bot can work on live data and react live whenever a candlestick bar closes.

```
def define_strategy(self): # "strategy-specific"

    df = self.data.copy()

    #******************** define the strategy here ************************
    df = df[["Close", "Volume"]].copy()
    df["returns"] = np.log(df.Close / df.Close.shift())
    df["vol_ch"] = np.log(df.Volume.div(df.Volume.shift(1)))
    df.loc[df.vol_ch > 4, "vol_ch"] = np.nan
    df.loc[df.vol_ch < -4, "vol_ch"] = np.nan

    cond1 = df.returns >= self.return_thresh
    cond2 = df.vol_ch.between(self.volume_thresh[0], self.volume_thresh[1])

    df["position"] = 1
    df.loc[cond1 & cond2, "position"] = 0
    #**********************************************************************

    self.prepared_data = df.copy()
```

```
# prepare features and define strategy/trading positions whenever the latest bar is complete
if complete == True:
    self.define_strategy()
    self.execute_trades() # NEW!!!
```

PLACING AND EXECUTING TRADES

The bot is now fed with historical data together with live prices update and the strategy is defined, everything is prepared to start trading.

The execute_trades method works based on the "position". There are 2 possible states:

- Position = 1 means "Long" trade position so in case of current strategy, signal to hold if already on the Long position or Buy if the position is 0 (neutral).
- Position = 0 means neutral, so if the latest signal is 0 then the algorithm stays neutral, and in case the position was on Long before, that means a SELL signal.

```
def execute_trades(self):
    if self.prepared_data["position"].iloc[-1] == 1: # if position is long -> go/stay long
        if self.position == 0:
            order = client.create_order(symbol = self.symbol, side = "BUY", type = "MARKET", quantity = self.units)
            print("GOING LONG")
        self.position = 1
    elif self.prepared_data["position"].iloc[-1] == 0: # if position is neutral -> go/stay neutral
        if self.position == 1:
            order = client.create_order(symbol = self.symbol, side = "SELL", type = "MARKET", quantity = self.units)
            print("GOING NEUTRAL")
        self.position = 0
```

TRADE MONITORING AND REPORTING

The report_trade method is pivotal in the trading framework, offering comprehensive insights into executed transactions. It receives two parameters, order for order instance details and going as a position switch indicator.

```
def report_trade(self, order, going):  # NEW

    # extract data from order object
    side = order["side"]
    time = pd.to_datetime(order["transactTime"], unit = "ms")
    base_units = float(order["executedQty"])
    quote_units = float(order["cummulativeQuoteQty"])
    price = round(quote_units / base_units, 5)

    # calculate trading profits
    self.trades += 1
    if side == "BUY":
        self.trade_values.append(-quote_units)
    elif side == "SELL":
        self.trade_values.append(quote_units)

    if self.trades % 2 == 0:
        real_profit = round(np.sum(self.trade_values[-2:]), 3)
        cum_profits = round(np.sum(self.trade_values), 3)
    else:
        real_profit = 0
        cum_profits = round(np.sum(self.trade_values[:-1]), 3)

    # print trade report
    print(2 * "\n" + 100* "-")
    print("{} | {}".format(time, going))
    print("{} | Base_Units = {} | Quote_Units = {} | Price = {} ".format(time, base_units, quote_units, price))
    print("{} | Profit = {} | CumProfits = {} ".format(time, real_profit, cum_profits))
    print(100 * "-" + "\n")
```

Extracting critical information such as transaction side, time, base, and quote units, along with the corresponding price, the method calculates real and cumulative profits.

Real profits denote immediate gains or losses, while cumulative profits represent the total over the entire trading period. The method produces a detailed trade report, featuring transaction time, position status, base and quote units, transaction price, real profit, and cumulative profits.

This reporting mechanism informs users of each transaction and facilitates continuous monitoring of key indicators for strategic decision-making and performance evaluation.

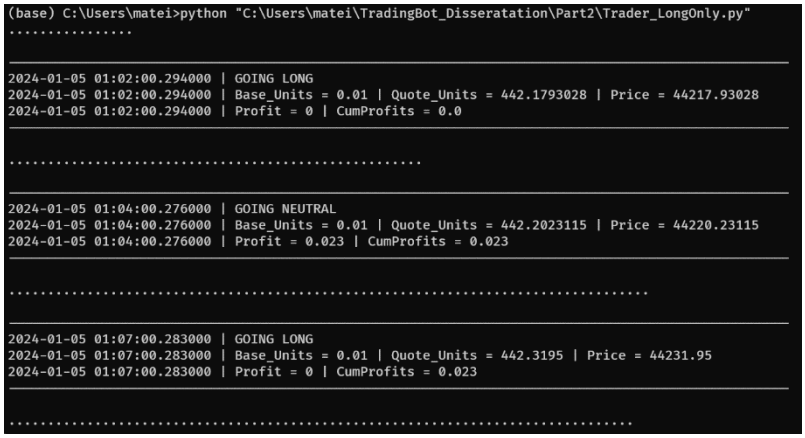
```
def execute_trades(self):
    if self.prepared_data["position"].iloc[-1] == 1: # if position is long -> go/stay long
        if self.position == 0:
            order = client.create_order(symbol = self.symbol, side = "BUY", type = "MARKET", quantity = self.units)
            self.report_trade(order, "GOING LONG")
        self.position = 1
    elif self.prepared_data["position"].iloc[-1] == 0: # if position is neutral -> go/stay neutral
        if self.position == 1:
            order = client.create_order(symbol = self.symbol, side = "SELL", type = "MARKET", quantity = self.units)
            self.report_trade(order, "GOING NEUTRAL")
        self.position = 0
```

In order to get live information, the report_trade function is injected inside the execute_trade, resulting in clean feedback after each transaction.

```
api_key = "VxuqPa3póRrAN8wBxSP7BFcFr1xyA6Wc9FrWdOu6lFMANIlr5UhekZdepó7NBgNJ"
secret_key = "EgB2E9eq55KkjpóSJTpU4GK0YpOSMqNyLdoHyC9cdbakFóBFTGonfloDADbGJYVb"

client = Client(api_key = api_key, api_secret = secret_key, tld = "com", testnet = True)

symbol = "BTCUSDT"
bar_length = "1m"
return_thresh = 0
volume_thresh = [-3, 3]
units = 0.01
position = 0

trader = LongOnlyTrader(symbol = symbol, bar_length = bar_length, return_thresh = return_thresh,
                        volume_thresh = volume_thresh, units = units, position = position)

trader.start_trading(historical_days = 1/24)
```

LONG-ONLY STRATEGY TRADING RUN

Long-Only Strategy trial trading run on 1 minute trading interval with -3, 3 volume threshold as condition resulted with slightly positive cumulative profits.



14. GRAPHICAL USER INTERFACE (GUI) DEVELOPMENT

COLOR SCHEME AND AESTHETICS:

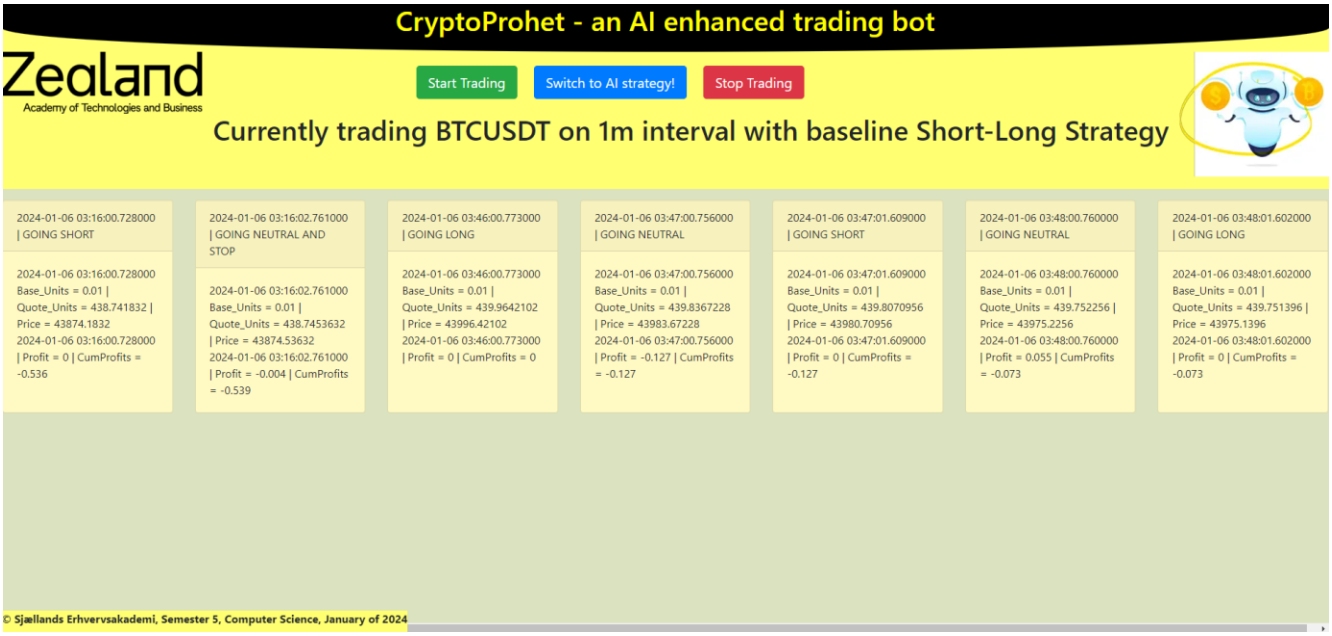
The chosen color scheme uses a lot of bright and contrasting colors to enhance visual appeal. The use of yellow and black in the banner reflects a dynamic and attention-grabbing theme, it is also in style of Zealand colors. Beige cards provide a subtle and professional contrast, ensuring a clean and organized appearance.

GUI FEATURES:

- **Start Trading Button:** Connects to the back-end script and executes the script to start trading the BTCUSDT pair based on the specified interval. Users is provided with a seamless and straightforward way to activate the trading process.
- **Stop Trading Button:** On click, stops all ongoing trades and closes positions. It is a mandatory feature that allows user to feel control over the trading assets.
- **Switch to AI Strategy Button:** Dynamically switches the algorithm's trading strategy to the AI-based strategy, as discussed in the subsequent section. Enables users to explore different trading approaches effortlessly.

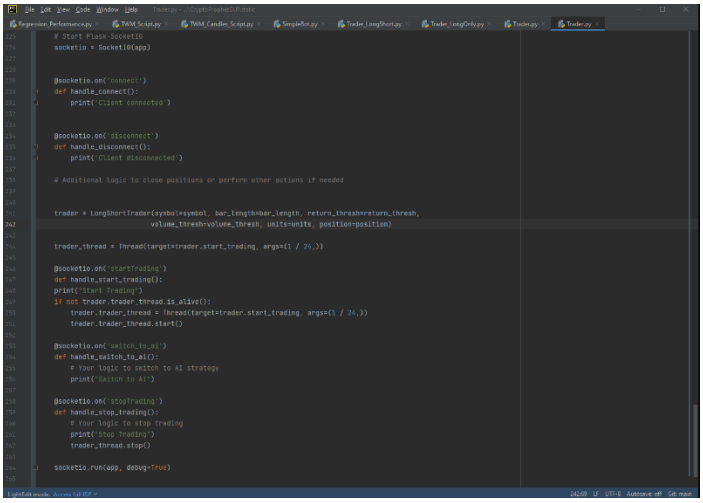
Transaction Feedback:

During any executed transaction, users receive instant feedback in the form of a card. This card contains vital information about the trade, including the date, order type, traded units, transaction price, profit, and cumulative profits. This concise and informative display ensures users stay well-informed about each transaction's details, fostering transparency and facilitating a better understanding of the algorithm's performance.



BACKEND DEVELOPMENT

For the simple back-end , the script leverages Flask, a web framework in Python. This allows me to simply establish a backend server streaming data between the graphical user interface. Flask serves as the foundation for the web application, managing route configurations and rendering HTML templates. Furthermore, the SocketIO extension enhances real-time communication via WebSockets, enabling responsive interactions between the server and clients. SocketIO event handlers respond to predefined actions initiated by the frontend.

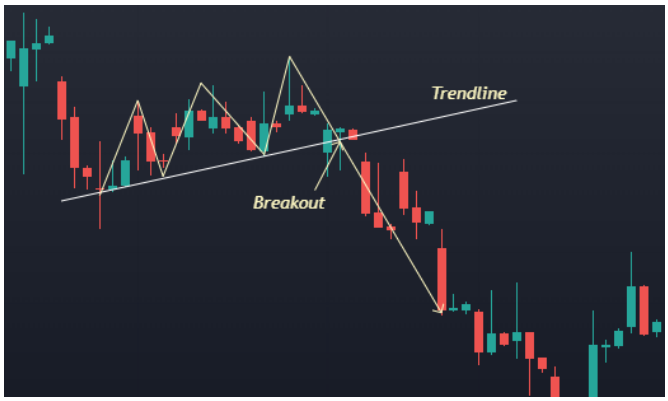


The asynchronous nature of Flask facilitates concurrent task execution, contributing to the creation of an interactive and dynamic GUI for the trading bot.

15. AI ENCHANCED STRATEGY

This section focuses on developing an AI strategy that utilizes machine learning to enhance a traditional trend line following strategy by filtering out the false breakouts. A trend line indicates whether the market is bullish or bearish. Drawing the trend lines on the highest and lowest candle prices can lead to identifying patterns formed by the candlesticks. Each pattern can lead to different signal that can serve as indicator for the bot.

The trend lines function finds the line that has the minimum distance to the input prices while also being above or below every price. My strategy will be focusing on one traditional approach to look for trendline breakouts.



In trading, a breakout occurs when the price of a financial instrument moves beyond a certain predefined resistance level indicated by the trend line. Breakouts are powerful indicating the end of a consolidation phase and the beginning of a new trend. Strategy based on identifying breakout can find potential opportunities for entering or exiting positions.

STRATEGY IMPLEMENTATION

The trendline_breakout function dynamically calculates the support and resistance trendlines for each candle in the closing price array, providing valuable insights into potential breakout signals for trading decisions.

The function iterates through each closing price in the array. For each iteration, it uses the fit_trendlines_single function on a subset of the dataset (window) to obtain two sets of coefficients representing the support and resistance trendlines.

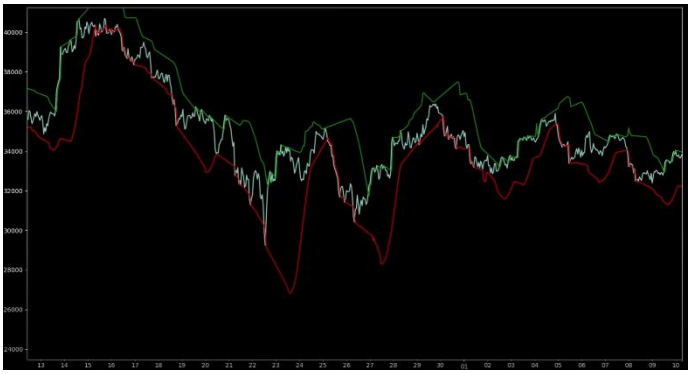
These coefficients are used to calculate the values of the support (s_tl) and resistance (r_tl) trendlines for the current candle.

Trading signals (sig) are generated based on the relationship between the closing price and the trendlines.

The fit_trendlines_single function utilizes linear regression to find the slope and intercept of trendlines.

It takes a window of closing prices as input and returns two sets of coefficients representing the support and resistance trendlines.

```
    s_tl[:] = np.nan

    r_tl = np.zeros(len(close))
    r_tl[:] = np.nan

    sig = np.zeros(len(close))

    for i in range(lookback, len(close)):
        # NOTE window does NOT include the current candle
        window = close[i - lookback: i]

        s_coefs, r_coefs = fit_trendlines_single(window)

        # Find current value of line, projected forward to current bar
        s_val = s_coefs[1] + lookback * s_coefs[0]
        r_val = r_coefs[1] + lookback * r_coefs[0]

        s_tl[i] = s_val
        r_tl[i] = r_val

        if close[i] > r_val:
            sig[i] = 1.0
        elif close[i] < s_val:
            sig[i] = -1.0
        else:
            sig[i] = sig[i - 1]

    return s_tl, r_tl, sig
```



These two outputs together with the Price are shown on the plot. The light blue line is the closing price, red and green lines are the current values of the support and resistance trend lines respectively.

This allows me to formulate a TREND FOLLOWING RULE:

Once the closing price is above the green band then take the Long position and hold it until the price closes below the Red band then reverse to a short position. The short position is held until the price again closes below the green band.

```
if close[i] > r_val:
    sig[i] = 1.0
elif close[i] < s_val:
    sig[i] = -1.0
else:
    sig[i] = sig[i - 1]
```

Position is identified based on the strategy condition. When the price is above the current resistance value the signal is set to 1, in second case the value is set to -1, if there was no breakout copy the signal from the previous candle.

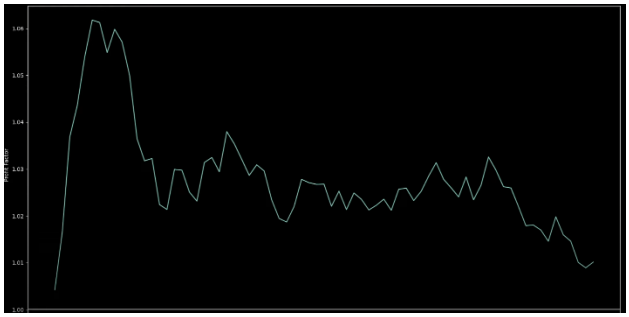
BREAKOUT STRATEGY BACK-TESTING

The strategy has been back tested on the historical data using slightly modified backtester framework. Cumulative log return plot shows very satisfactory results with a profit factor of 1.02 and Win Rate 50%.


```
Breakout based strategy
--------------------------------------------------
PERFORMANCE MEASURES:


Annualized Mean:            12.78208592467801
Annualized Std:             4.990775138739193
Sharpe Ratio:               0.92
Profit Factor:              22.430925625570683
```

This strategy shows excellent performance, evident in an Annualized Mean Return of 12.78%, a Sharpe Ratio of 0.92 indicating favorable risk-adjusted returns, and a Profit Factor of 22.43, emphasizing positive risk-reward characteristics. Additionally, the Cumulative Log Return of 1.021 reflects overall positive growth in strategy returns. This combination of metrics suggests a successful and efficient trading strategy.

However the plot of profit factors once the strategy is run on wide range of lookbacks. We can observe huge profits spike at the beginning then a huge drop occurs. This simple breakout identification strategy seems to be already efficient and a perfect candidate to use some machine learning for enchancement.



STRATEGY UPDATE WITH AI IMPLEMENTATION

In this section the strategy will be enhanced by the machine learning metal labeling approach to filter the false breakouts and measure performance to compare results with baseline breakout strategy. However before implementing that model, the strategy needs to be defined into a more specific type of trade.

Example visualization of breakout trade that will be the aim of the strategy. The price entry happens once the price closes above the upper resistance trend line. In case of the hold period exceeding 12 candles, the algorithm exits the position.

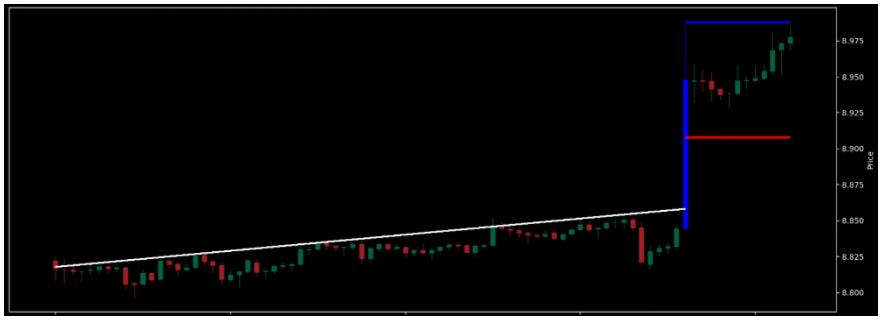
FEATURES SELECTION

In order to access whether specific breakout scenario is optimal entry position, it is crucial to identify some features. These features need to identify various patterns and signals on the same time interval.

One of my key features is the trend line slope that allows the algorithm to know whether the breakout has happened in a "downtrend" or an "uptrend". The assumption here is that returns tend to be positive when the slope is positive and vice versa.

Another important feature is the normalized Volume which can vary on breakouts.

Another feature is Resistance Trendline Average Distance which allows the bot to estimate when potential breakdown could happen. Assumption here is that the closer the prices "hug" to the trend lines the more likely the trade is going to be successful.



For features engineering the trend line break data set function. Whenever an entry is identified, the slope indicator is computed as the raw resistance slope divided by the average true range.

```
# Trendline features
# Resist slope
trades.loc[trade_i, 'resist_s'] = r_coefs[0] / atr_arr[i]

# Resist erorr
line_vals = (r_coefs[1] + np.arange(lookback) * r_coefs[0])
err = np.sum(line_vals - window) / lookback
err /= atr_arr[i]
trades.loc[trade_i, 'tl_err'] = err

# Max distance from resist
diff = line_vals - window
trades.loc[trade_i, 'max_dist'] = diff.max() / atr_arr[i]

# Volume on breakout
trades.loc[trade_i, 'vol'] = vol_arr[i]

# ADX
trades.loc[trade_i, 'adx'] = adx_arr[i]
```

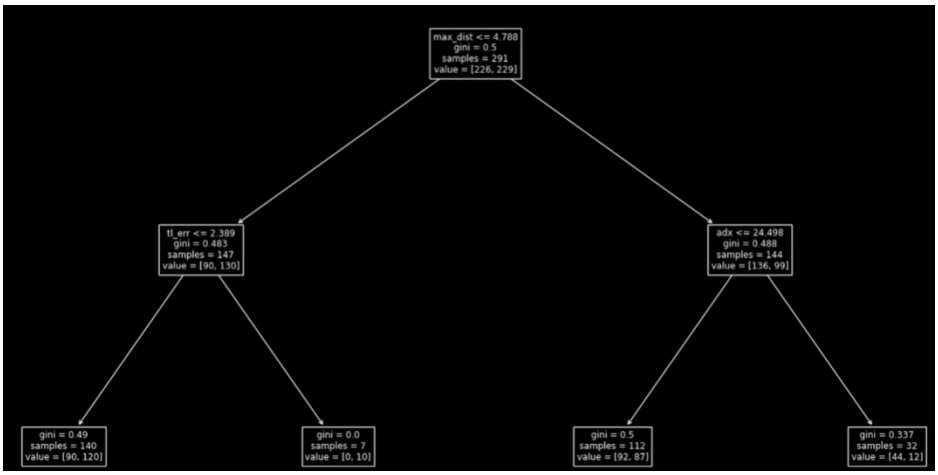
The next indicator is the resistance error. In order to calculate this metric, the value of the resistance line for each value in the window is computed and prices are subtracted from it. Next the error is summed up and averaged by the lookback window. Last step is to divide this value by the average true range to normalize it to volatility. This metric will measure how close the prices are on average to the resistance line.

Another potentially impactful feature could be the maximum distance from the resistance line. The assumption here is that whenever there is a price in the window that is very far from the breakout, the trades tend to be less successful.

Next feature is the volume on the candle that triggers the breakout.

Lastly predictive attribute will be the ADX indicator that measures the strength of the trend.

RANDOM FOREST will be the machine learning model to predict and prevent false breakouts. Random forest model is built on decision trees, which are performing very well at learning hierarchical relationships that divide the dataset into the data groups based on the thresholds in the features.



BREAKOUT STRATEGY WITH RANDOM FOREST PREDICTIONS IMPLEMENTATION

The walkforward_model function utilizes a Random Forest model to adaptively make trading decisions based on historical data. It creates two output signals: 'signal' stores binary values representing selected trades, and 'prob_signal' contains the predictive probability of a profitable trade from the classifier.

IMPLEMENTATION STEPS:

- The 'next_train' variable denotes the index at which the model will be trained, initialized as 'train_size'.
- The function iterates through each candle in the dataset, evaluating optimal times for potential trade entries.
- When a favorable trade entry point is identified, the Random Forest model is created with a depth of 3, 1000 epochs, and a random state of 69420.
- The model is then trained on the test data, adapting dynamically to changing market conditions.

```
def walkforward_model(
        close: np.array, trades:pd.DataFrame,
        data_x: pd.DataFrame, data_y: pd.Series,
        train_size: int, step_size: int
):

    signal = np.zeros(len(close))
    prob_signal = np.zeros(len(close))

    next_train = train_size
    trade_i = 0

    in_trade = False
    tp_price = None
    sl_price = None
    hp_i = None


    model = None
    for i in range(_len(close)):
        if i == next_train:
            start_i = i - train_size

            train_indices = trades[(trades['entry_i'] > start_i) & (trades['exit_i'] < i)].index

            x_train = data_x.loc[train_indices]
            y_train = data_y.loc[train_indices]
            print('training', i, 'N cases', len(train_indices))
            model = RandomForestClassifier(n_estimators=1000, max_depth=3, random_state=69420)
            model.fit(x_train.to_numpy(), y_train.to_numpy())

            next_train += step_size

        if in_trade:
            if close[i] >= tp_price or close[i] <= sl_price or i >= hp_i:
                signal[i] = 0
                prob_signal[i] = 0
```

```
    if in_trade:
        if close[i] >= tp_price or close[i] <= sl_price or i >= hp_i:
            signal[i] = 0
            prob_signal[i] = 0
            in_trade = False
        else:
            signal[i] = signal[i - 1]
            prob_signal[i] = prob_signal[i - 1]


    if  trade_i < len(trades) and i == trades['entry_i'].iloc[trade_i]:

        if model is not None:
            prob = model.predict_proba(data_x.iloc[trade_i].to_numpy().reshape(1, -1))[0][1]
            prob_signal[i] = prob

            trades.loc[trade_i, 'model_prob'] = prob

            if prob > 0.5:  # greater than 50%, take trade
                signal[i] = 1

            in_trade = True
            trade = trades.iloc[trade_i]
            tp_price = trade['tp']
            sl_price = trade['sl']
            hp_i = trade['hp_i']

        trade_i += 1

return signal, prob_signal
```

The if statement manages the trade exit and Signal output if the algorithm is currently in the trade. The closing price is checked against the take profit and stop loss values as well as time limit. If the time to exit trade is signalized, the signal value is set to 0 to prevent copying signal from previous candle.

Furthermore, the function handles trade entries by checking if the current index matches the entry index from the trade DataFrame. If the model is not None (indicating it has been trained), the 'predict_proba' method is used to predict the probability of a successful trade. This probability is saved to 'prob_signal', and the predicted probability is added to the trades DataFrame. If the probability exceeds 50%, the 'signal' is set to 1, indicating a potential trade.

AI ENCHANCED BREAKOUT STRATEGY EVALUATION WITH COMPARISON

The evaluation of the AI-enhanced breakout strategy, leveraging a Random Forest model for trade filtering, demonstrates significant improvements over the baseline strategy. Utilizing a meticulous metal labeling model, this strategy proves to be robust, delivering promising and potentially profitable trading outcomes.

Baseline Breakout Strategy Metrics:

- Profit Factor: 1.05
- Average Trade Return: 0.13%
- Win Rate: 51%
- Time Holding Positions: 29.4%
- Sharpe Ratio: 0.92

AI-Enhanced Breakout Strategy Metrics:

- Profit Factor: 1.14
- Average Trade Return: 0.33%
- Win Rate: 54.7%
- Time Holding Positions: 19.2%
- Sharpe Ratio: 1.32

The comparison of the baseline strategy with the AI-enhanced predictions reveals a substantial outperformance of the raw conventional strategy. The AI filtering mechanism contributes to superior results, enhancing key performance indicators such as profit factor, average trade return, win rate, sharpe ratio and efficient time spent in the market. This underscores the potential of integrating machine learning techniques for refining and optimizing trading strategies, presenting a notable advancement in strategy development and performance.

16. REFLECTION AND CONCLUSION

Taking a look back on this project, the initial problem formulation aimed to develop and evaluate an automated trading strategy, specifically focusing on a Long-Only strategy as the first conventional strategy initially and later developing breakout based strategy and enhancing it with Radom Forest Model for prediction and better trade entries. The journey involved meticulous steps, from backtesting to the integration of live data streaming, GUI development, and finally, the implementation of an AI-enhanced breakout strategy.

SOLVING THE STATED PROBLEMS:

The documented process reveals a conscious effort to address the challenges posed in the problem formulation. By introducing a two-phase testing approach and incorporating a machine learning model for filtering false breakouts, the project demonstrated a commitment to developing strategies that can withstand real-world market dynamics.

REFLECTION ON PROJECT EXECUTION:

The report acknowledges the potential pitfalls, such as technology expectations and methodological choices, enhances its credibility. These challenges faced during the project indicates a reflective and honest approach. It aligns with the notion that addressing problems head-on is more beneficial than attempting to conceal or ignore them.

In conclusion, the project successfully navigated the complexities of automated trading strategy development. The LongOnlyTrader class, live data streaming, and AI-enhanced breakout strategy collectively represent a well-rounded solution. The results and insights gained from this project contribute not only to the specific domain of automated trading but also provide a framework for future projects seeking adaptive and resilient strategies in dynamic environments. The journey undertaken, along with the valuable reflections, encapsulates an earnest retrospective view of the project, demonstrating a clear connection to the original problem formulation.

ADRESSING THE RESEARCH QUESTIONS

HARMONIZING AI WITH ALGORITHMIC TRADING:

During the dissertation project implementation, the AI algorithms and algorithmic trading are combined, as demonstrated by the integration of machine learning into the Breakout identification strategy. The predictive features indicate a good entry price given the market conditions which proves to be effective decision-making in bitcoin trading.

IMPACT OF FEATURE SELECTION ON ALGORITHMIC TRADING:

The selection of technical indicators and market data significantly influences the accuracy and performance of AI-powered trading strategies. Key features, including trend line slope, volume normalization, and trendline distance, played a critical role in enhancing the strategy's effectiveness.

REAL-TIME DATA IMPACT:

The inclusion of real-time data stream from the Binance API enhances the adaptability and responsiveness of the trading algorithm. Leveraging ThreadedWebsocketManager and BinanceSocketManager facilitates live data analysis, enabling prompt decision-making based on the latest market information.

IS AI EMPOWERED TRADING MORE EFFECTIVE THAN CONVENTIONAL TRADING STRATEGY?

The comparative analysis indicates that using AI-enhanced trading algorithms usually outperforms conventional strategies. In fact the AI predictions are like a "cherry on top of a cake" in this case just increasing the strategy capabilities instead of competing with it. The AI strategy enhancement demonstrates superior profitability, risk management, and adaptability to dynamic market conditions.

ARE WIN RATIO, LOGARITHMIC RETURNS, AND SHARPE RATIO SUITABLE PERFORMANCE METRICS?

Various performance metrics such as win ratio, logarithmic returns, and Sharpe ratio are used for the comparison of baseline and AI-enhanced strategy. Utilizing these metrics, validates the efficacy of the AI-enhanced strategy in generating profits while managing risk efficiently.

This partial research and product development affirms successful harmonization of AI with algorithmic trading, emphasizes the impact of feature selection on strategy performance, highlights the significance of real-time data in enhancing adaptability, showcases the superiority of AI-empowered trading, and validates the suitability of win ratio, logarithmic returns, and Sharpe ratio as performance metrics. This research contributes valuable insights to the field of automated trading strategies, particularly in cryptocurrency markets.

16. BIBLIOGRAPHY

Research about machine learning model implementation with trading algorithms:

"Machine Learning for Algorithmic Trading: Predictive models to extract signals from market and alternative data for systematic trading strategies with Python" by Stefan Jansen

Find the source code on my GitHub:

https://github.com/KacperBytnar/CryptoProphet_TradingBot

Binance API documentation – WebSocket Stream

https://binance-docs.github.io/apidocs/spot/en/#websocket-market-streams

Spot Trade Binance Api Documentation for binance trade integration:
https://binance-docs.github.io/apidocs/spot/en/#change-log

My price prediction algorithms developed for Synopsis Research:

https://github.com/KacperBytnar/Bitcoin_Price_Prediction/blob/main/Kacper_Bytnar_Synopsis_Bitcoin_Price_Prediction.ipynb

Basics of algorithmic trading:

https://www.investopedia.com/articles/active-trading/101014/basics-algorithmic-trading-concepts-and-examples.asp

Crypto Trading Algorithms: Complete Overview

https://www.coinbureau.com/education/crypto-trading-algorithms/

Stream live cryptocurrency data from Binance API:

https://medium.com/mlearning-ai/how-to-get-live-crypto-data-from-binance-7a5e5ec10de4

Insights into conventional trading strategies:

https://www.elluminatiinc.com/crypto-trading-strategies/

Flask basics to create a local server for communication with GUI

https://flask.palletsprojects.com/en/3.0.x/

17. LIST OF APPENDICES

Waterfall Strategy diagram:

https://www.vecteezy.com/vector-art/7388691-the-waterfall-model-infographic-vector-is-used-in-software-engineering-or-software-development-processes-the-illustration-has-6-steps-like-agile-methodology-or-design-thinking-for-application-system

Look-Ahead Bias diagram:

https://cdn.corporatefinanceinstitute.com/assets/look-ahead-bias.png

Trading bot logo on 1st page.

https://www.freepik.com/free-photos-vectors/robot-trading

Trendlines chart:

https://assets.cmcmarkets.com/images/Trendline-channels_extra.webp

Breakout example figure picture:

https://blogger.googleusercontent.com/img/a/AVvXsEjqlFrfeaQ0KLkeIDKnkfOPmZx1hUrZNWko6tWZw4KdKA3yYPLc2hKtnZqhijHf8gaeDMKvEpWkbKYPu9AiwpPrph44YECIucaeaLTWrK1NsIbC96reln o68v8eUsmS9zgNkUuSE_ex5Gg3gHTNbUABo737rw8f6ckquDDaZ-kk097EJFsLRV3RwA