TÉCNICO LISBOA

Using the Order Book and Machine Learning for Cryptocurrency Trading

João Guilherme Esteves de Andrade

Thesis to obtain the Master of Science Degree in

Telecommunications and Informatics Engineering

Supervisor: Prof. Dr. Rui Fuentecilla Maia Ferreira Neves

Examination Committee

Chairperson: Prof. Dr. Ricardo Jorge Fernandes Chaves
Supervisor: Prof. Dr. Rui Fuentecilla Maia Ferreira Neves
Member of the Committee: Prof. Dr. Pável Pereira Calado

May 2019

Abstract

This thesis presents a new computational approach for profit optimization on cryptocurrency trading, using trade and order book data from a major digital asset trading platform of four digital currencies (Bitcoin, Ethereum, Litecoin and Bitcoin Cash) in the United States Dollar (USD) and Euro (EUR) markets. An end-to-end solution was designed, starting with the database generation, information extraction, trading algorithm, simulation and ending with a dynamic report of the results. All the parts of the system are designed in a microservice architecture to ensure that they are scalable, strongly encapsulated and tightly scoped components. Three use cases where considered for the trading system where the first two use order book volume variation to assess suitable trading periods whilst the last one uses an ARIMA model to forecast price fluctuation and consequently serve as a verification mechanism to the first two test cases during a simulation. Furthermore, a new type of trailing stop called Percentage Stop Order is introduced, allowing profit improvement. Monthly test periods were used alongside different types of trading positions to all test scenarios to find the most performing strategy according to the market's situation. The results obtained managed to surpass the gains from the Buy & Hold strategy up to 22,72% and reducing losses up to 33,24%.

Keywords: Cryptocurrency, Order Book, ARIMA, Microservices.

Resumo

Esta tese apresenta uma nova abordagem computacional para otimização de lucro em transações com criptomoedas, utilizando dados de negociação e de ordens de uma grande plataforma de negociação de ativos digitais de quatro moedas digitais (Bitcoin, Ethereum, Litecoin e Bitcoin Cash). Uma solução completa foi projetada, começando com a geração da base de dados, extração de informações, algoritmo de negociação, simulação e terminando com um relatório dinâmico dos resultados. Todas as partes do sistema estão inseridas numa arquitetura de microsserviços para garantir que sejam componentes escaláveis, encapsulados e com um âmbito bem definido. Três casos de uso são considerados para o sistema de negociação onde os dois primeiros usam a variação do volume do livro de ordens para avaliar períodos de negociação adequados, enquanto o último usa um modelo ARIMA para prever a tendência dos preços e, consequentemente, servir como um mecanismo de verificação para os dois primeiros casos de teste durante uma simulação. Além disso, um novo tipo de controlo de posições chamado Percentage Stop Order é introduzido, permitindo a melhoria do lucro. Períodos de teste mensais foram usados juntamente com diferentes tipos de posições de negociação para todos os cenários de teste, para encontrar a estratégia com melhor desempenho de acordo com a situação do mercado. Os resultados obtidos conseguiram superar os ganhos da estratégia Buy & Hold até 22,72% e reduzir as perdas em 33,24%.

Palavras-chave: Criptomoeda, Livro de Ordens, ARIMA, Microsserviços.

Contents

List of Figures

List of Tables

List of Acronyms

ACF Autocorrelation Function

ADF Augmented Dicky-Fuller

AR Autoregressive

API Application Programming Interface

ARIMA Autoregressive Integrated Moving Average

ARMA Autoregressive Moving Average

BCH Bitcoin Cash

BTC Bitcoin

CSV Comma-Separated Values

ENET Elastic-Net

ETH Ethereum

EUR Euro

EWMA Exponential Weighted Moving Average

GARCH Generalized Autoregressive Conditional Heteroskedasticity

GP Gaussian Process

GBT Gradient Boosted Tree

HTML Hypertext Markup Language

JSON JavaScript Object Notation

LTC Litecoin

MA Moving Average

MLP Multilayer Perceptron

MSE Mean Squared Error

NARX Nonlinear Autoregressive Exogenous

NoSQL Not Only Structured Query Language

PACF Partial Autocorrelation Function

PDF Portable Document Format

PSO Percentage Stop Order

RNN Recurrent Neural Network

ROI Return On Investment

SHA Secure Hash Algorithm

SQL Structured Query Language

SSH Secure Shell

USD United States Dollar

Chapter 1

Introduction

The cryptocurrency trading realm is progressively attracting attention from communities linked to several domains of finance and computational intelligence. One of the main reasons associated with this phenomenon is the high volatility associated with the market of digital currencies. This means that finding the optimal timings to enter and leave the market is the main objective of researchers since it leads to higher profits on a short period of time. However, this task is really challenging since the cryptocurrency market is non-linear, non-stationary and heavily influenced by speculation.

1.1 Background

In recent years, cryptocurrencies are becoming increasingly popular, getting a significant amount of media attention worldwide. Despite only being on the spotlight for a short time, the concept of cryptocurrencies and blockchain technology was introduced in 2009 with the introduction of Bitcoin [1]. The main idea was to create a decentralized shared public ledger (blockchain) that validates and saves immutably all financial transactions by using a computer network as a timestamp server, maintaining the correct order of negotiations.

There were 1474 types of digital coins at end-January 2018 and their total market capitalization was 830 billion USD [2], a consequence of an increasing public acceptance of cryptocurrencies. Nevertheless, volatility is a crucial aspect of digital markets and, as a measure of price fluctuations, has an important impact on trade planning and investment decisions [3]. This component can be partially handled since exchange offices store “buy” and “sell” orders on an order book, providing insights into the liquidity and trading purposes [4]. Thus, the first step is to define how will this information be used and which temporal model must be chosen to adaptively learn volatility using the gathered data.

1.2 Proposed Solution

In this thesis a new approach for profit optimization on cryptocurrency trading is presented, using trade and order book data from a major digital asset trading platform (Coinbase Pro, formerly known as GDAX)

of the four main digital currencies, Bitcoin (BTC), Ethereum (ETH), Litecoin (LTC) and Bitcoin Cash (BCH), in USD and EUR, to create a trading system that uses order book volume variation and machine learning to forecast price fluctuation and consequently present trading advice.

1.3 Thesis Contribution

The main contributions of the system introduced in this report are:

1. A database containing order books and trading information of the four major digital currencies built entirely from scratch (more than 70 Gigabytes of data obtained since March 2018);
2. An algorithm that pre-processes and aggregates information from order books, correlating it with confirmed trades on a monthly basis, to define the optimal start positions for long and short trading;
3. Machine Learning forecasting applied to the previous algorithm, using ARIMA;
4. An innovative type of trailing stop to optimize profits.

1.4 Outline

This document describes the research and work developed and it is organized as follows:

- **Chapter 1** presents the motivation, background and proposed solution.
- **Chapter 2** describes the previous work in the field.
- **Chapter 3** describes the system architecture and its implementation.
- **Chapter 4** describes the evaluation tests performed and the corresponding results.
- **Chapter 5** summarizes the work developed and future work.

Chapter 2

Related Work

The main idea of Bitcoin explained in the Introduction only grasps a minuscule part of all the mechanisms needed to create a cryptocurrency. Despite being a relatively new area of study, we can already find several scientific articles about this subject where those that contributed the most for this assignment will be considered in this section. Before handling those papers, we will clarify the fundamental components behind the creation of a digital currency.

2.1 Blockchain Background

Blockchain is essentially a public ledger of all transactions or digital events that have been executed and shared among participating parties [5], but where does it all start? To initiate the explanation, we will use a straightforward scenario, let's assume a transaction between Alice and Bob, where Alice sends 100 units of currency to Bob. When using a traditional currency like the EUR or the USD, it's mandatory to use a third trusted party (such as a bank or PayPal) so it validates both users, guarantees that Alice has the minimum balance needed to perform the transaction, safeguards the whole process and preserves an historic record of the operation. Naturally, this process requires the payment of a fee to the company providing these payment services which results in considerable transaction costs. Blockchain does not need this third party since it is a shared public ledger, which means that there is no central administrator or centralized data storage, but how does it all work?

2.1.1 The Decentralized Ledger

Following the example of Alice and Bob, now dealing with digital currency where there is no central authority, the negotiation still needs to be verified before being recorded in the public ledger. Therefore, the element of trust changed, cryptographic proof, detailed in depth in section 2.1.3 of this article, is used instead of relying on a financial institution to execute the transaction. The transaction will be broadcasted to every node on the cryptocurrencies' network and, after authentication, is recorded in a public ledger. Two problems arise immediately, how do we verify if it is really Alice that wants to send 100 units of the digital coin to Bob and how can we check if she has enough cryptocurrency to fulfill the order? To handle

these issues, a simple solution was established, the usage of digital signatures to offer protection and reliability.

2.1.2 Hashing and Digital Signatures

Digital signatures, like real signatures, are a mechanism to ensure someone's identity with the difference that we rely on mathematics or cryptography, turning the signature less prone to forgery. Before further explaining this notion, we will address the other significant concept that underpins the blockchain technology: hashing.

Hashing is the approach of selecting a random amount of input data, executing an algorithm to it, and generating a fixed-size output data called the hash or digest. There is a variety of algorithms to choose from (Message Digest Algorithm, Secure Hash Algorithm, etc.) but they all take an input of bits, employ some calculations to them, and output a limited number of bits, for instance Bitcoin uses digests with 256 bits. We generally use hashes to check if a file wasn't tampered with, although the blockchain also uses them to symbolize its current state by taking as input all the transactions that have taken place so far and generating a digest.

The resulting hash allows all parties in the world to agree on the current state of the blockchain but how are they calculated? For the first block, also called Genesis block, we simply calculate the digest using all the transactions inside that block. Afterwards, for every new block, we use the previous block's transactions and hash along the new transactions as input. This process forms a chain of blocks, thus the name blockchain, with each new block hash pointing to the one that came immediately before it. Therefore, no transaction can be tampered with because any change will modify the hash of the block it belongs to, and, as a result, all the digests of the following blocks. Comparing hashes to catch any fraudulent attempt and only needing to agree on 256 bits to represent the blockchain (as of April 2018, Bitcoin's blockchain size surpassed 160 000 Mb) are the most relevant advantages of using hashing on cryptocurrencies.

As explained on the first paragraph of this sub-section, digital signatures are used to authenticate a specific person, avoiding identity theft. The most transparent way to understand all the steps needed to create and use a digital signature is by analyzing an example, so let's assume a user named Bob wants to sign a document. Initially, Bob must generate two keys, the signing key, a private key and the verification key, a public key. These two keys will be produced at the same time and they will have a mathematical relationship between them, but it should be hard to obtain the private key with the public key to offer security and reliability to the user. To sign the document, all Bob needs to do is to apply a cryptographic hash function to the document, to simplify the process since we will stop dealing with an arbitrary length input and sign the resulting digest with his signing key. The corresponding output is called the signature. Digital signature schemes are typically designed so that only the person who possesses the private key can generate this type of result.

If Bob sent the document he digitally signed to Alice, how does she check its integrity and authenticity? To be able to verify the document, Alice must receive three different inputs, the message that needs

to be authenticated, the corresponding signature and the public key of the sender, in this case Bob's. Afterwards, Alice will apply a mathematical transformation to these inputs to ascertain that the signature she received from Bob corresponding to the document is the only one that would have been generated by Bob's signing key. The most important aspect here is that anyone can carry out the verification process having just the public key of the sender. Basically, this method will output a simple 'yes' or 'no', telling Alice if she should accept or reject the signature. Figure 2.1 portrays the illustrated situation.

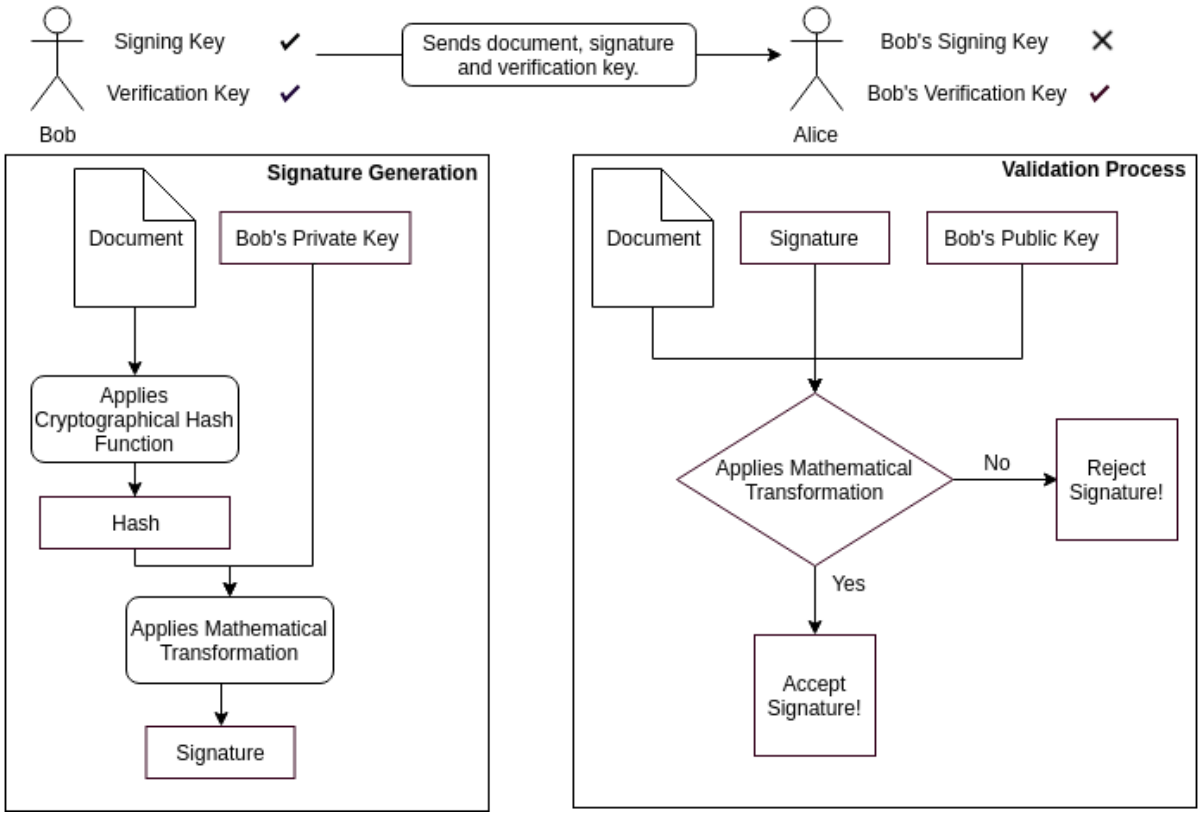

Figure 2.1: Digital Signature Generation and Verification

Since the original document is one of the inputs needed to produce the actual digital signature, if a hacker modifies the contents of the document before it reaches Alice, she'll obtain a different signature on the verification phase, which ensures that she will reject the document. This property of digital signatures guarantees the integrity of the document or message sent by Bob to Alice. Another critical aspect to digital signatures is that the used hash functions must be collision resistant, which means that it must be impossible to generate identical hashes to two different messages, because this would lead to identical signatures on distant messages, allowing falsification. Finally, to avoid replay attacks, where a hacker re-sends a signed message, we use a unique id or nonce, an arbitrary number used only once in a cryptographic communication that is time dependent or generated with enough random bits to ensure statistical protection against a brute force attack.

2.1.3 Proof of Work

A proof of work in blockchain is a cryptographic challenge used to guarantee that someone has performed a certain amount of work. It has two main properties, to ensure that the party providing the proof of work has done a predefined amount of labor to produce the evidence and secondly, that the evidence is quickly verifiable. Generally, finding an answer to a proof of work puzzle is a probabilistic process with a success rate that varies with the chosen difficulty. To illustrate this notion let's assume there are two users, John and Mary, communicating and let Mary require John to execute a certain amount of computational work for each message he sends her. Towards achieving this, Mary may ask John to provide a string whose one-way hash respects a predefined structure. The probability of finding such a string determines how much work John must put on average to find a valid solution.

In Bitcoin, the hashing algorithm used is double Secure Hash Algorithm (SHA) 256, commonly represented as SHA256^2, and the predefined structure is a digest inferior or equal to a target value T, that represents the current difficulty. In this algorithm, the success probability of finding a nonce n or a certain message m, such that the digest, given by

$$Digest = SHA256^2(m||n) \tag{2.1}$$

is less or equal to the target T is

$$Probability\ (Digest \leq T) = \frac{T}{2^{256}} \tag{2.2}$$

Which means that, following the example of John and Mary, John would have to perform, on average, 2^{256} divided by the target value T computations. Lastly, Mary can easily check that the nonce sent along with the message is indeed a valid proof of work by simply assessing

$$Digest \leq T \tag{2.3}$$

In cryptocurrencies, this proof of work process is called mining and the parties participating in this mechanism are called miners. The process in Bitcoin consists on finding a valid block to be linked to the blockchain and, to do so, miners perform the following actions in an endless loop:

- Save all received transactions and check whether they comply the miner's policy. Generally, a transaction incorporates a fee that works as an incentive for the miner to include it in the block. Nonetheless, if there is no fee involved, the miner chooses whether or not to add it.
- Authenticate all transactions that will be included in the block.
- Select the most recent block on the longest path, the one that required most accumulated computation effort, in the blockchain and include the digest of the chosen block header into the new block.
- Resolve the proof of work already described and broadcast the solution. If another party solves the proof of work problem before, then we proceed by verifying the block by checking the proposed

solution and all the transactions included in the block. On a successful confirmation, the loop is repeated. It is important to mention that any transactions that were not included in this new block are saved and incorporated in the next cycle.

2.1.4 Blockchain

Blockchain is a log of all transactions that have occurred in a cryptocurrency system and is distributed by every party in it. BTC structures the blockchain to guarantee chronological order of transactions and to prevent double-spending attacks, where the attacker spends the same Bitcoins repeatedly and only one vendor actually gets the money [6], although it takes approximately 10 minutes to verify the legitimacy of a transaction [7]. Every block in the blockchain contains the hash of the previous block, maintaining the chronological order of the blocks. The components of a generic Bitcoin block are exemplified in Figure 2.2.

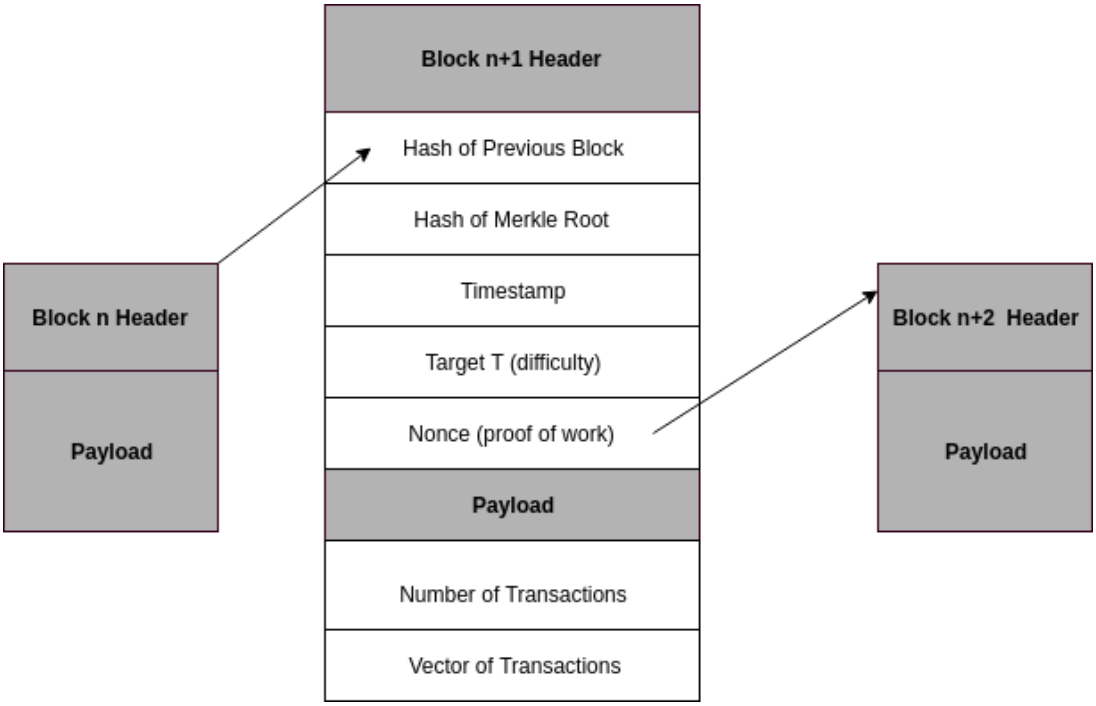

Figure 2.2: Bitcoin Block

Additionally, a proof of work obtained by solving a challenge with a certain difficulty must be included in each block. The data processing power needed to solve the proof of work problem for each block is utilized as a voting scheme to allow all parties in the network to agree on the current version of the blockchain. Specifically, nodes choose the blockchain that involved the highest accumulated computational work to be generated. In this manner, modifying a block in the chain would require an attacker to recompute evidence of equal or greater data processing power than the ones that are chained up to the newest block. To accomplish this, the attacker would need to have a better computational performance than half of the network combined, which is considered absurd.

It is possible that two different blocks are mined simultaneously, which means that two parties solved

the proof of work and obtained a reward, resulting in a chain bifurcation. When this situation occurs, nodes will accept the first received block and continue building the blockchain upon it. If another block is discovered, then the branch that was used will become the main blockchain, re-adding all correct transactions on the shorter chain to the pool of queued transactions. Figure 2.3 exemplifies the described blockchain structure.

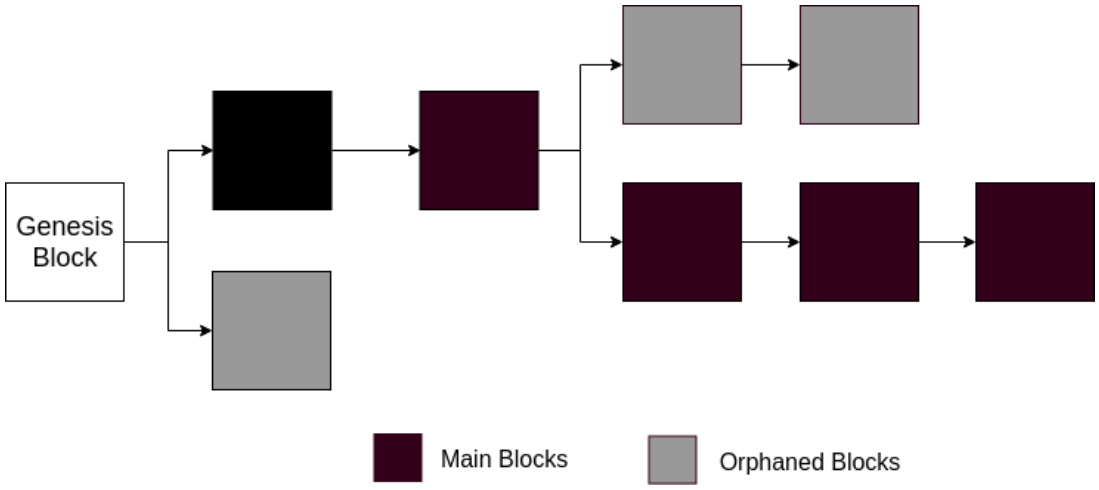

Figure 2.3: Blockchain

2.1.5 Merkle Tree

Merkle trees, term invented after their creator Ralph Merkle, are binary hash trees used for efficient verification of data integrity [8] and in the blockchain they are used to swiftly check transactions. In a Merkle tree, leaves are generated directly by hashing data blocks, like cryptocurrency transactions, while nodes higher up the tree are computed by chaining and hashing their respective children as shown in Figure 2.4.

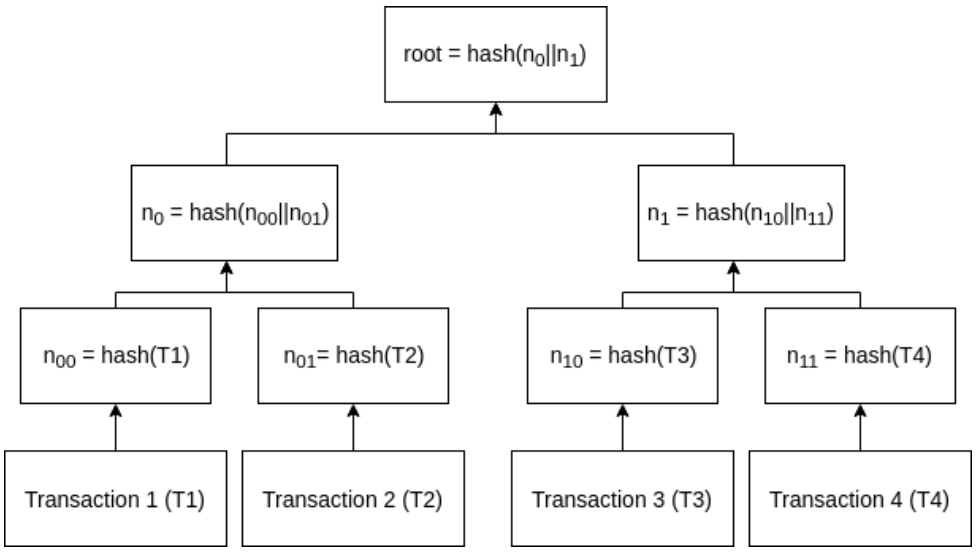

Figure 2.4: Merkle Tree Example

One fundamental benefit of using Merkle trees on cryptocurrency blocks is that when one transaction is altered, there is no need to compute a hash over all transactions, as opposed to naive hashing. Let's assume transaction T1 from Figure 2.4 is modified, so n_{00} must be re-calculated along with all the nodes further up the branch until the root node. For this reason, the number of required digest computations scales logarithmically with the number of transactions. This process is efficient since both hashes and transactions have small sizes. A situation that may happen is to have an odd number of nodes. The answer to this problem is easy, when forming a row in the tree (except the root), whenever the number of nodes is not even, just duplicate the last node. Using this solution will ensure that each intermediary row has an even number of nodes thence each node, excluding the leaves, will precisely have two children.

2.2 Machine Learning

Data mining is described as the extraction of previously unidentified and implicit information from data, which can result in advantageous input. Machine learning specifies the technical base for data mining [9]. The collection of observations that contain one or more variables, known as attributes, is defined as a dataset. We can divide machine into two categories, supervised learning and unsupervised learning. Supervised learning includes the modelling of datasets with labelled instances that can be represented as *x* and *y*, *x* being a set of predictor attributes and *y* the dependent target attribute. A machine learning problem follows one of two methodologies: regression, when the target attribute is continuous; classification, when the target variable is discrete [10]. There are some hybrid methodologies which allow for a classification and regression at the same time, e.g. neural networks. Unsupervised learning includes modeling datasets with no established outcome or result. Since this project has a task with a known objective, predicting cryptocurrencies price fluctuation, it is a supervised machine learning task.

2.2.1 Supervised Learning

Supervised learning is the most common methodology in machine learning. As previously explained, supervised learning is achieved when we input variables *x* and an output variable *Y* alongside an algorithm to learn the mapping method from the input to the output. This type of learning can be algebraically exemplified by

$$Y = f(x) \tag{2.4}$$

The main goal of supervised learning is to obtain a mapping function so that when we have new input data, we can predict the output variables for that observation with a defined target. Supervised learning generates the mapping function based on example input-output pairs. Knowing the current answers, the algorithm makes forecasts on the training set iteratively and is rectified accordingly. When the algorithm achieves the desired level of performance, the learning process is finished. We can divide supervised learning challenges into two categories, classification and regression problems.

Classification algorithms are used when the aimed output is a discrete label, which means that we

should choose them to handle problems where the objective falls under a finite set of possible results. In many use cases, such as defining whether it will rain or a gender, there are only two possible results. This is called binary classification. Situations where there are more than two classes are handled by multi-label classification, where each sample is mapped to a set of target labels, for example a book can be about art, a person and an era simultaneously. In order to build a classification model, we need to address the following steps: start the classifier (the algorithm that maps the input data to a specific class), train it, predict the target and evaluate the classifier model. Naïve Bayes [11], Decision Trees [12] and Support Vector Machines [13] are examples of well-known classification models.

Regression models generally present reliable results in forecasting outputs that are continuous. In other words, regression models are used when the solution to our problem can be expressed by a quantity that can be determined based on the inputs of the model instead of being restricted to a set of possible classes. Regression issues with time-ordered inputs are named time-series forecasting problems and will be thoroughly explained with the next sub-section. Some of the most popular regression algorithms are Linear Regressions and the ARIMA model [14].

Supervised learning algorithms also explored several types of neural networks including the Multilayer Perceptron (MLP), the basis for other neural network models [15]. A general example of an MLP model is presented in Figure 2.5.

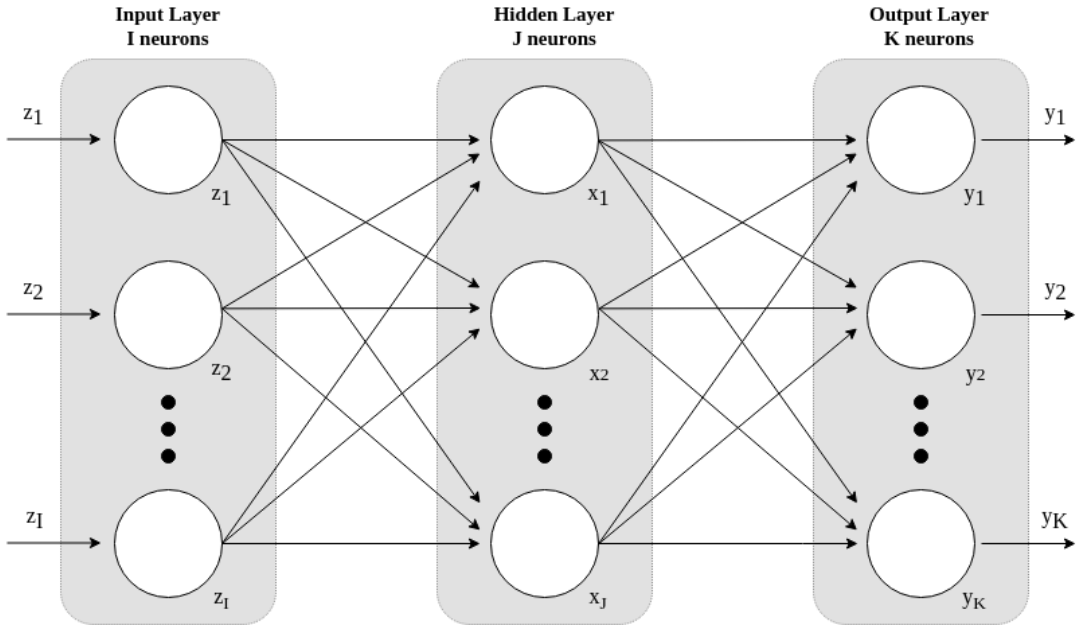

Figure 2.5: Multilayer Perceptron

A Multilayer Perceptron model contains an input and output layer with hidden layers in between. Every single one of the outputs of any of these layers is a component which can be compared to a neuron in the human brain. The connections between these units are known as a weight which is analogous to a brain synapse. When training a model, these weights determine the model's function as they are periodically adjusted. Another important limitation of the MLP is that its signals always move forward in the network in a static nature. This model does not recognize the chronological element of a time series problem efficiently as its memory is considered stopped in time but a simple approach would

be to transform the challenge into a time independent one. A simpler explanation would be to think that MLP treats all input as an unordered bag of objects. The Recurrent Neural Network (RNN), introduced by Elman [16] is organized similarly to the MLP but allows signals to flow forward and backwards in an iterative manner by adding a new layer called context layer.

2.2.2 Time Series

Time series are collections of data points chronologically organized. Their purpose is to understand patterns evolving over time and apply these patterns to forecast behavior in several fields (stock prices, meteorology and heart arrhythmias are some examples). Since this field can be applied in many areas science, studies on time series have become increasingly popular on the last few decades.

Most of the scientific studies about time series tackle time series modelling, which is the process of understanding and defining (mathematically) the inherent structure of the time series [17]. The usefulness of time series modelling ranges from forecasting the future [18] to describing the feature responsible for the time series but in this thesis, we will focus on the first part since our goal is to simulate the generation of data to an unforeseen future by studying the model of a time series information. When making predictions using time series, we can classify the time periods into three different categories, short-term, medium-term and long-term. While long-term refers to periods significantly distant to the last observed data, short-term denotes spans that are near the observations. An important fact to mention is the ambiguity associated with how the boundaries between each class are chosen since each application defines them in a distinguishing way using their data and objectives.

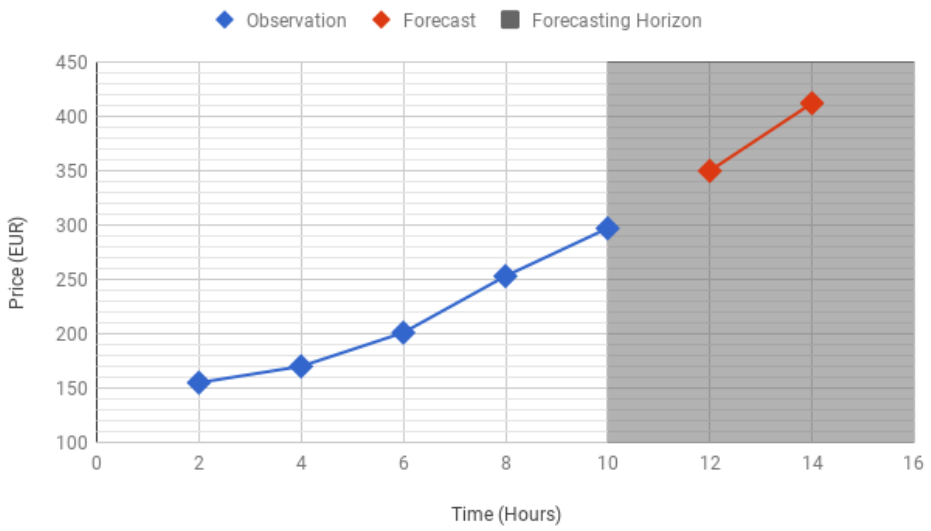

Figure 2.6: Time Series Forecasting

Long-term prediction is a complex challenge to work out because of the lack of information, error accumulation and the growing uncertainties [18]. These complications only intensify their negative effect as the forecasting region grows. Consequently, to elaborate a reliable time series prediction technique it is mandatory to forecast the future precisely while avoiding error accumulation as the forecasting region

expands. A general example of a time series forecasting is shown in Figure 2.6 where the shaded region represents the forecasting horizon, the zone containing points in the future that we want to predict.

To correctly study time series, it is necessary to model the underlying stochastic process, a time sequence representing the evolution of a predefined system expressed by a variable whose change is subject to a random variation, that creates time series. We can use a joint probability distribution between observations to obtain a complete description of a time series, but the required calculations might be too complex, so a valid alternative would be to compute the mean and autocovariance. The mean is simply the one measure of the central tendency of observations while autocovariance calculates the statistical dependence between two observations in a time series, computing the covariance between itself at different periods of time.

A stationary time series has the property that the mean, variance and autocovariance structure do not change over time, a very desirable property on a time series. Consequently, proper estimation of the enumerated properties should be equal for all points in time, allowing forecasting for all time points in the future. Regrettably, there are two commonly found time series components that obliterate the stationary property, season and trend. Cryptocurrencies are affected by both elements [19, 20, 21].

ARIMA

Autoregressive Integrated Moving Average is one of the most used time series prediction models, having a more complicated approach than the simplest model, a linear regression. ARIMA arises from the Autoregressive Moving Average (ARMA) model while the latter itself is a mixture of an Autoregressive (AR) model with a Moving Average (MA) model. The notion of both MA and AR is fundamental not only to the ARIMA model but for several time series models.

The AR model considers that the actual value of a time series is a linear combination of its prior values while the MA model defines the current value of a time series as a linear combination of several past errors. Usually the number of previous values is symbolized by *p* and named as the order of the autoregressive. Each one of the models uses a constant *c* and represents white noise as ϵ.

Combining the two explained models we obtain the ARMA(p,q) model, given by

$$y_t = c + \epsilon_t + \sum_{i=1}^{p} \phi_i y_{t-i} + \sum_{j=1}^{q} \theta_j \epsilon_{t-j} \tag{2.5}$$

where *p*, ϕ and y_{t-i} represent respectively the order, current parameters and previous value of the autoregressive model while *q*, θ and ϵ_{t-j} respectively symbolize the order, current parameters and past errors of the moving average model.

For the ARMA model to function, it assumes that the associated time series is stationary. Remembering that the objective of this project is to handle data from cryptocurrency markets which are affected by season and trend so who can we transform a non-stationary time series into a stationary time series? The mechanism to resolve this issue is called differencing and can be applied to each one of the two components. Differencing calculates the variance between consecutive observations, if a lower order differencing doesn't achieve the intended de-trending or de-seasonalizing we are obliged to compute a

greater order differencing.

Applying a *d*th order of differencing to an ARMA model results in an ARIMA model, often defined as ARIMA(*p*,*d*,*q*) where *p* represents the order of the AR, d is the order of differencing and q is the order of the MA. Despite being studied and used for a considerable amount of time, the ARIMA technique still supports several applications that feature forecasting, ranging from medicine [22] to network engineering [23]. This model has numeral software implementations, fact that strongly contributed to the widespread utilization of ARIMA. Although the predominance of this model, its prediction power reduces as the forecasting horizon augments due to the forecast converging to the mean of the observations as the predicting region grows [24]. With this information we should be aware that this model is only adequate to short-term prediction which is the forecasting objective of this thesis.

NARX

Despite remaining at the forefront of academic and applied research, simple linear time series models generally ignore certain aspects of financial data since these systems suffer both behavioral and structural modifications. This section introduces Nonlinear Autoregressive Exogenous (NARX) models, the answer for modeling nonlinear behavior in economic time series information. Being exogenous means this model uses external inputs generated by an externally defined series that affects the main series of the model. A NARX model correlates the present value of a time series to the current and past values of the exogenous series and the previous values of the same series, taking into consideration an error term since knowledge of the other variables won't be sufficient to make an exact forecast of the current value of the time series. Such a model can be algebraically exemplified as

$$y_t = F(y_{t-1}, y_{t-2}, ..., u_{t-1}, u_{t-2}, ...) + \epsilon_t \tag{2.6}$$

where *y* is the variable of interest and *u* represents the external variable while represents the error term. The F function is any nonlinear function such as a neural network or a polynomial. This type of models was used in various fields such as predicting daily direct solar radiation for a solar boat [25] or forecasting financial time series [26].

2.2.3 Methodology Examples

The idea of extracting features from order book data over time and use them to create probabilistic models that simultaneously apply volatility and feature series to predict Bitcoin price fluctuations was introduced in a recent study from Tian Guo and Nino Antulov-Fantulin [27]. In each order book snapshot, the price spread, weighted spread, ask/bid volume, depth and slope and their difference were gathered. Spread is the variance between the lowest price that a seller is willing to accept (ask) and the highest price that a buyer is willing to pay (bid). Depth is defined by the number of orders on the ask or bid side while volume is the number of Bitcoins on the ask or bid side. Weighted spread is the difference between cumulative price over 10% of ask depth and the cumulative price over 10% of bid depth. Slope is estimated as the volume until α price offset from the actual traded price, where α is estimated by the

bid price at the order that has at least 10% of orders with the greater price. After getting this data, we need to choose a methodology from one of two categories, statistics or machine learning.

Statistics Examples

This category of baseline is only trained with order book data or volatility, not both.

- Exponential Weighted Moving Average (EWMA) models predict volatility by executing moving average over historical values [28].
- Generalized Autoregressive Conditional Heteroskedasticity (GARCH) model is a widely used method to forecast volatility of returns and prices [29]. A simple explanation of these methods is to model the variance of the error term of a time series as a function of the preceding error terms.
- ARIMA models [14], as explained previously, can be used to include non-stationary time series data either to better understand the underlined data or to forecast future points in the series.

Machine Learning Examples

This second category learns with volatility and order book simultaneously.

- Gradient Boosted Tree (GBT) is the application of boosting methods to regression trees [30], training a sequence of simple regression trees and then summing the prediction of singular trees to deliver a final estimate.
- Elastic-Net (ENET) is a regularized regression method linearly combining the L1 and L2 penalties of the lasso and ridge methods [31].
- Gaussian Process (GP) based regression [32] is a supervised learning technique that provides a Bayesian non parametric method to smoothing and interpolation.

The authors of the referred scientific paper created a generative temporal mixture model of the volatility and trade order book data, which is able to outperform the current state-of-the-art machine learning and time series statistical models by using a gate weighting function that detects regimes when the features of buy and sell orders significantly affects the future high volatility periods [27]. The only problem with this approach was to only consider one cryptocurrency, Bitcoin, which limits the application scope. Since Bitcoin is the cryptocurrency with most volume traded daily, most of the published work in the digital currencies field applies their findings to this coin. Although being limited to one form of digital currency, these studies provide a solid knowledge base than can be applied to a larger spectrum of cryptocurrencies. Price fluctuations were studied from several perspectives such as using order book data and found that a lack of buyers generated panic amongst sellers [33], which culminated in the price crash of Bitcoin on the 10th of April 2013, or using cryptocurrency web communities data to obtain sentiment and forecast value fluctuations [34]. Price prediction studies apply some techniques used in traditional financial market like autoregression, as used by Garcia et. al. who identified word of mouth

and new Bitcoin adopters as feedback loops that drive to price bubbles [35] or simply employ historical time series price information to develop an accurate trading strategy with price prediction [36].

2.3 Trading Orders

While investing in the financial market, may it be cryptocurrencies or regular stocks, we have several categories of orders that enable us to be more precise about how our trades are filled. The market order is straightforward since it is defined as an order made at the current price of sell or buy. Contrarily, limit and stop orders, two of the main types of instructions, enable investors to reject the current price of the asset, allowing them to place an order that will be carried out when the price moves in a particular direction. In this section, we will consider these type of orders alongside stop-limit orders, where we combine both categories to mitigate risk.

2.3.1 Limit Orders

A limit order defines that we want to trade only with a certain price. As a simple example, let us consider a trader named Bob who intends to buy BTC that is selling at $1000. Bob thinks that the current value is overpriced and only wants to purchase at $950 so he places a limit order that won't be executed until the price of BTC meets his stipulated value. On the other hand, if Bob made a mistake while setting his limit order and set a plain limit superior to the current price, the broker would reject his action because a better valuation is already applicable. Likewise, if Bob instead wants to sell 1 BTC at $1050 he may put a limit order at that price that will only be filled at that rate or better. As explained previously for a purchasing case, it is also not possible to place a sell limit order inferior to the current market price.

Traders might want to bypass limit orders rules since they can't establish an order to buy or sell once a future value has been matched. A common case is wanting to sell before the value of their digital currency plummets. To handle this sort of issues, a stop order can be employed.

2.3.2 Stop Orders

There are several variations of a stop order, but they all have one or more specified requirements depending on a future price that is different from the current one when the order is placed. This stop order to sell or buy will become active only if the defined future value is reached. Stop orders operate in the inverse direction as limit orders since a buy stop order is placed higher than the market value, and a sell stop order is put below the current value. When the stop price is attained, the order is immediately transformed into a market or limit order, depending on the designated style of order. Figure 2.7 illustrates a generic order book with the potential positions of buy and sell stop orders.

Stop orders are commonly used to define a risk limit for a transaction, this type of order is called a stop-loss. Stop-loss orders are applied to establish a price inferior to the trade entry value which the investor is not willing to risk any more money, providing insurance.

Figure 2.7: Stop Orders Placement Example

The trailing stop is another function of a stop order which is dynamic, following price variations so that we can achieve higher profits. An investor can define the degree of the trailing stop as either an amount of currency or a percentage, deciding the range between the trailing stop level and current price. A broad trailing stop ensures more flexibility whereas a tighter one is more sensible to volatility. Figure 2.8 exemplifies a situation where an investor uses a 10% trailing stop.

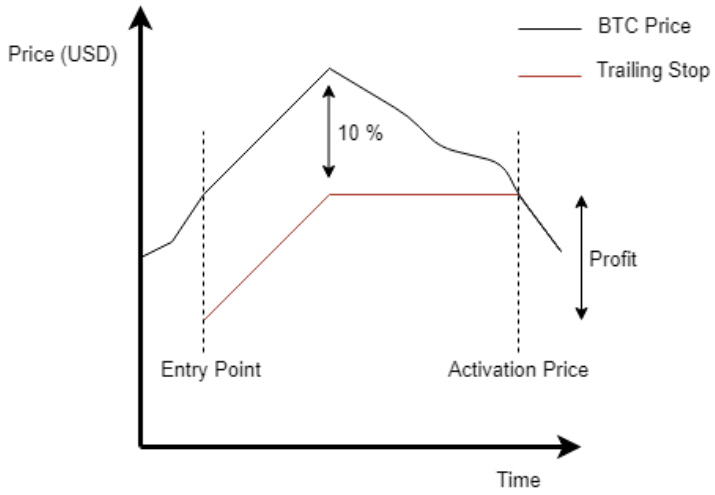

Figure 2.8: Trailing Stop Order Example

The secret for optimizing a trailing stop successfully lies on finding a balance. If we set it too tight, the trade getting might close before it has a chance to work. Conversely, setting it too wide may result in losing an important part of unrealized profit.

2.3.3 Stop-Limit Orders

Both types of orders introduced have their own strengths and weaknesses so one might question if there is a way to combine them. The answer is called a stop-limit order, a conditional transaction that fuses a stop and a limit order in order to reduce risk. But how can we use it?

To be able to utilize a stop-limit order, an investor must establish two price positions. The first one, the stop, will dictate the beginning of the action while the second, the limit, serves as the ceiling target price. This implies that we will only buy below the limit value and sell over it. Another important aspect

of this order category is time frame definition since the investor must set the interval while the stop-limit order is deemed executable.

For example, assume that ETH is trading at $100 and an investor named John wants to buy this digital currency when it starts going upwards. To handle this intention, John sets a stop-limit to buy ETH with the stop price at $110 and the limit value at $115. The limit order will only be triggered if the price of ETH moves above the stop value. If that happens, the trade will be filled as long as it can be under $115. Otherwise, the trade will not occur.

Chapter 3

Architecture

The objective of this thesis is to produce a trading system that uses temporal models and machine learning, taking into consideration several features such as order book volume or current value, to forecast price fluctuation and consequently invest and improve profits for the end user. To achieve this, a decision to use micro-services was made since this project will rely on several constituents and a micro-service can be defined as a scalable, strongly encapsulated, tightly scoped and independently deployable application component. This chapter will describe all the elements developed and how they were combined.

3.1 System's Architecture

Before going into further detail about the application components, an overall overview of the system's architecture is presented in Figure 3.1.

This system was designed according to a multi-layered micro-services architecture built on three layers: the data layer, the application layer and the presentation layer. Separating the project in different layers provides the ability to modify each one of them without crashing the others, reducing maintenance and enabling flexibility for application expansion. The flow of the system described can be explained through the following steps:

1. Market order books and market trades are extracted from a digital asset exchange, order books every five seconds and trades as they happen;

2. Two components, one for trades, another for order books, are in charge of Pre-Processing the information provided by the cryptocurrency exchange. They gather all the obtained information, filter it into the standard representations of the system and populate the database accordingly;

3. The Data Aggregation element will read all the data stored on the database, aggregate it into a monthly time frame so several features (for example, the maximum monthly volume on each side of the order book) can be extracted.

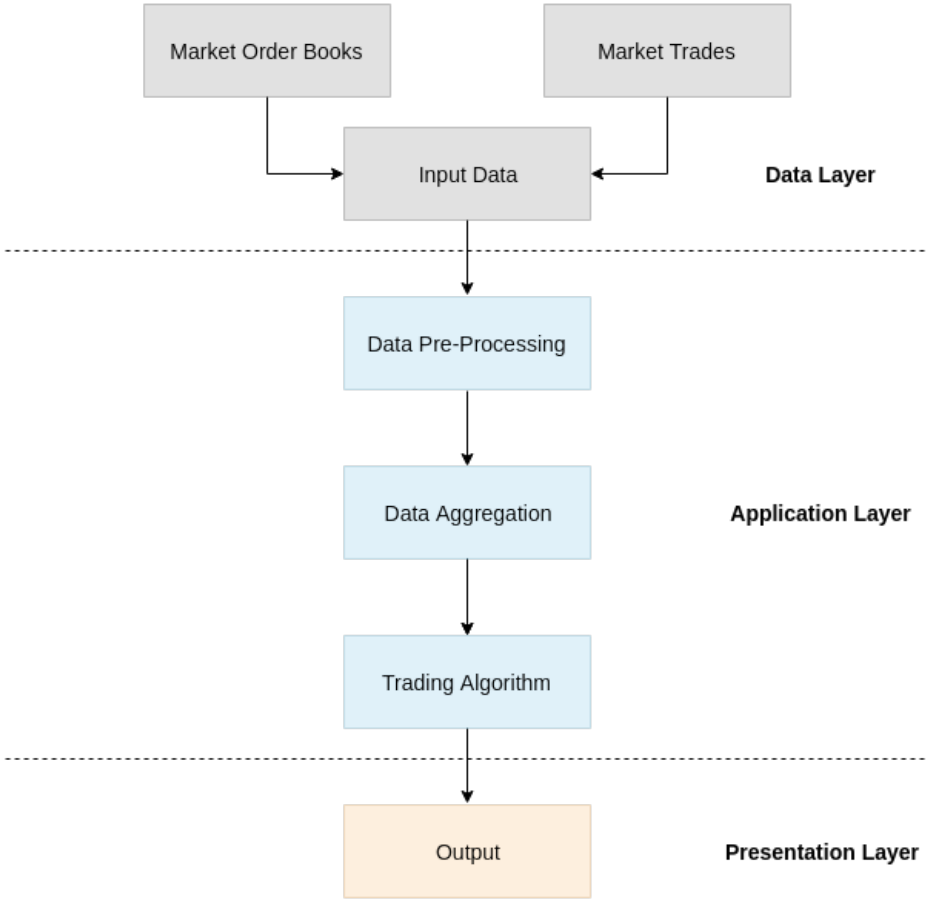

Figure 3.1: System's Architecture

4. The Trading Algorithm component will receive a period of time as an argument and all the data regarding it, apply one of the three variants of the algorithm created for this thesis and generate a report with the execution results. The first alternative analyzes the best periods based on the lowest and highest volume values within the time frame to start short or long positions. On the other hand, the second relies on inversions (over a defined threshold) of total percentage in the bid or ask side of the order book to define start positions. Lastly, the third variant uses with the ARIMA model to forecast prices and determine when to invest accordingly. All methods use the percentage stop order developed for this project to improve profitability.

5. Finally, the results obtained from the Trading Algorithm running through the simulation window will be presented to the user for analysis.

The following subsections will thoroughly detail the mentioned architectural components, starting with the implementation options that were taken for each one of them.

3.2 Implementation Options

After choosing to use micro-services we had to select a platform that would ensure the features we pretended: flexibility, safety and encapsulation. Docker [37], an open platform for programming, shipping

and running applications, was chosen to create the micro-services since it is the world's leading software containerization platform. The Docker platform grants the ability to package and run an application in an almost isolated environment called a container. Since the security and isolation are guaranteed by Docker, it is possible to run several containers simultaneously on a given host. Containers are lightweight because they don't need the extra load of a virtual machine monitor despite running directly in the host machine's kernel, as illustrated on Figure 3.2.

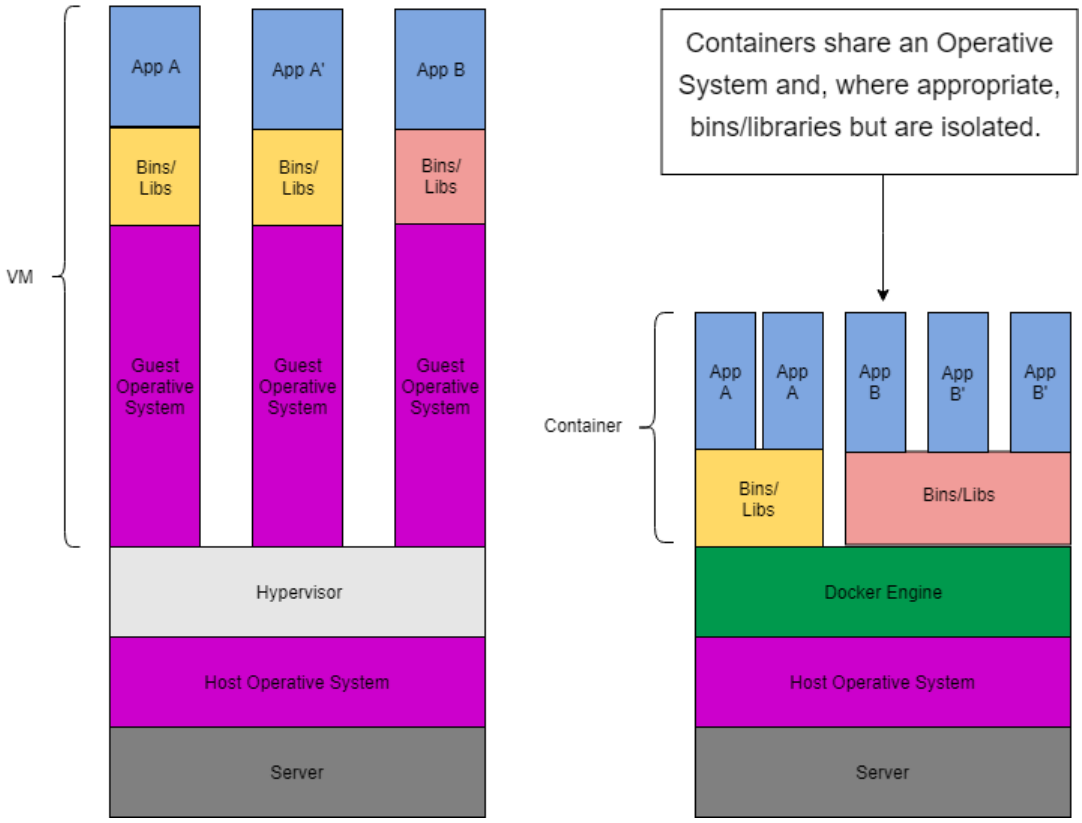

Figure 3.2: Virtual Machines vs Containers

All in all, Docker provides a platform and tools to allow developers to have a fast and consistent delivery of their applications, responsive deployment and scaling (they can run in any machine, requiring just the installation of Docker's software) and the ability to run more workloads on the same hardware.

The project's structure represents the micro-services that were implemented to generate the database (Node.js, Python and MongoDB containers) alongside the code that was developed in the following months referring to the Trading Algorithm and User Interface components. One important aspect worth mentioning is Docker's ability to create a network that connects containers while they don't even are aware that they are deployed on Docker or even to peers that are not Docker containers. In this project, the default network driver named bridge is used, providing a network for the applications that run in standalone containers. This allows every component to easily communicate if desired. Initially, the main goal was to create a database containing information about order books and transactions of the four cryptocurrencies with the most significant daily volume (BTC, ETH, LTC, BCH). This was the first priority of the practical aspect of the thesis since this type of data wasn't available for free online. Docker

offers containers with a database software but a decision between using the current market leaders of Not Only Structured Query Language (NoSQL), MongoDB, and Structured Query Language (SQL), MySQL, solutions had to be made. Since the scalability and efficiency of the chosen database software are the most important aspects for this project, MongoDB was selected since it delivers better results in both basic and complex queries [38]. The developed database will be thoroughly explained in the next subsection.

Python was the selected programming language to develop this thesis since it is easy to use, powerful, and versatile due to a vast range of libraries. The most important Python tool used in this thesis' development was Pandas, a data analysis library that receives data (for example a Comma-Separated Values (CSV) file or a SQL database) and creates an object with rows and columns called *DataFrame* that is akin to a table in statistical software such as Microsoft's Excel or Apache's OpenOffice Calc. Additionally, it can be used alongside Jupyter Notebook to produce instant outputs and to have markdown documentation. The Jupyter Notebook is an open-source web application that allows a user to create and share documents that contain live code, mathematical equations, visualizations and narrative text. Some of the main uses include data cleaning and transformation, numerical simulation, statistical modeling, data visualization and machine learning.

All programming was made in a Linux operative system (Ubuntu) but Docker and Jupyter Notebook allow to easily use any part of the work in any other operative system. Two computers were used to develop this project, one provided by my supervisor that is permanently on and is responsible for hosting and populating the database and my personal one where all the code was written and tested. Table 3.1 shows the hardware specifications of both computers.

Table 3.1: Hardware Specifications

	Provided Computer	**Personal Computer**
Motherboard	Z170-P	MS-7A69
Processor	3.40GHz (4 CPUs)	3.20GHz (4 CPUs)
RAM	8 GB	8 GB
Display Memory	2 GB	2 GB

3.3 Database

Since the world of cryptocurrency trading is decades younger than stock trading, we don't have the same amount of free information available on the internet. Therefore, one of the first challenges of this thesis was to establish how would the trading data be obtained. After searching for free databases containing the desired information, one of two problems always emerged. The database didn't had order book data or information was aggregated on a daily basis with maximum and minimum values. These complications alongside the scarcity of free databases lead to a different approach regarding the acquisition of trading data: scrapping information from a digital asset exchange.

There are several ways to scrape data from web trading platforms but exchanges try to simplify that process by providing free and well documented Application Programming Interface (API)s which enable

their clients to easily gather knowledge. Additionally, those APIs are available in various programming languages, although their features may vary slightly between them. Two information categories were needed for this project, the buys and sells and the state of the order book within a determined time period. Evidently, we didn't gather information about all the available digital currencies because it was not in the scope of this work so we decided to pick the top four cryptocurrencies according to daily volume: BTC, ETH, LTC and BCH. We also thought that it would be interesting to compare the differences between USD and EUR digital markets so each cryptocurrency trading information was extracted on both currencies. Because of the quantity of data we defined separate documents (the SQL table equivalent in MongoDB) for each digital currency's order books. For confirmed trades, a single document was used and the digital currency associated with it was one of the fields. To make a distinction between the USD and EUR digital markets, each document has two collections, one of each mentioned currency. Likewise, in the trades document there is a 'Buy' collection and a 'Sell' collection. The database architecture used is depicted in Figure 3.3.

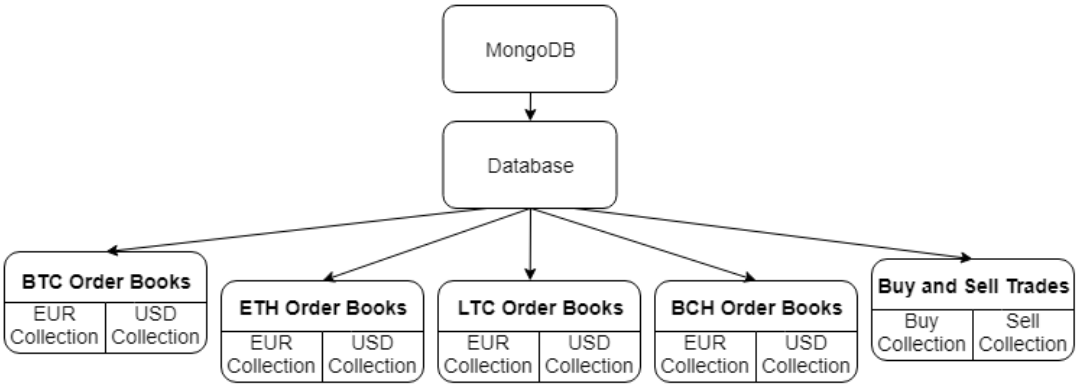

Figure 3.3: Database Architecture

Unfortunately, since the database was populated and stored on a remote computer, some problems emerged. First of all, there were several time periods where the machine was powered off so all the trading information that took place between that period of time was lost. While occasional restarts or shutdowns that took less than five minutes didn't have a noticeable impact on data usability, there was one major period that had to be discarded from this work, from the forth of August 2018 to the sixth of September 2018. This period is related to the end of the academic year since the machine is placed at our university Lisbon campus. Table 3.2 illustrates the top three most significant time gaps due to complications regarding the remote computer. Another dilemma was how to access the stored information and create backups, issues that will be addressed on the following subsections.

Table 3.2: Most Significant Database Time Gaps

Last Entry Date	Next Entry Date	Time Difference
2018-03-15 02:26:56.266	2018-03-16 20:23:08.260	1 Day, 17 hours, 56 minutes and 12 seconds
2018-06-15 16:46:51.345	2018-06-18 16:55:57.125	3 Days, 9 minutes and 6 seconds
2018-08-04 12:41:34.811	2018-09-06 10:01:28.556	32 Days, 21 hours, 19 minutes and 54 seconds

3.4 Trading Data

After a thorough study of digital asset platforms with APIs, Coinbase Pro, formerly known as GDAX, was chosen for this master's thesis. Coinbase Pro provided a API that was easy to use and accomplished the information goals defined previously. Nonetheless, we wanted to save every confirmed traded and the best option was to use the API's web socket client but it was not available with Python so we developed a Node.js scrapper to extract trades and a Python one to gather order books. In this subsection, both programs and the information contained in every trade and order book will be clarified.

Firstly, what kind of information was retrieved from a single trade and an order book? For trades, we used the API's web socket client, starting by establishing a connection to the MongoDB database. Afterwards, the web socket is actively listening for all messages sent by Coinbase Pro and filters them for confirmed trades. These trades are documented on a message containing the 'match' type. Every time a 'match' message is caught, we will save the trade in the database according to its side (buy or sell) with the following fields: currency, trade identifier, amount, date, rate and total. On the other hand, order books were requested every five seconds since high frequency trading is not in the scope of this thesis. After configuring the connection to the MongoDB database, every five seconds the order books of the four digital currencies on both USD and EUR were saved as a pair [*date,dictionary*], where *date* refers to the time of extraction and *dictionary* contains all the ask and bid amounts and prices. Figure 3.4 illustrates how order books and trades were saved for each digital currency in both USD and EUR.

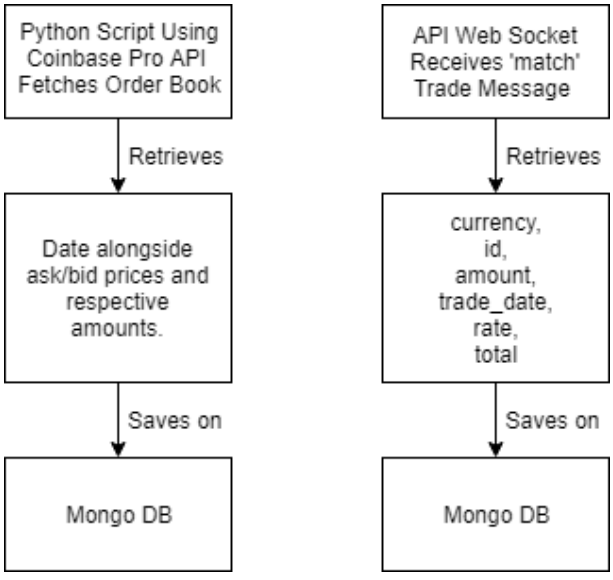

Figure 3.4: Order Book and Trade Extraction

Relevant to mention that the three components used to extract trading information (trades scrapper, order book scrapper and database) are individual Docker containers. The start-up configuration file to define aspects such as the links between them or automatic restarts on failures is called docker-compose. The configuration file used for composing the mentioned containers is displayed on Appendix A.

3.5 Data Flow and Data Processing

This section will describe the data flow and processing, after extracting trade and order book information, until sending the treated data as input to the trading algorithm. We can break down this procedure in three main steps:

1. Database Communication, that clarifies how does our solution handles the connection to a remote database and retrieves necessary information;
2. Saving Monthly Data, where we define how to gather, by parts, monthly raw data of buy or sell trades or order books;
3. Monthly Data Manipulation, composed by receiving all files generated from the previous step and using aggregation and, in some cases, resampling to create *DataFrames* that will be used as input to the trading algorithm.

All the aforementioned stages will be utterly explained in the following subsections.

3.5.1 Database Communication

Since we were using a remote computer to store the database containing all the information obtained from the digital asset exchange, how would we access that data? There are several solutions available, since manually creating backups of the database to an external hard drive (restoring it afterwards in the machine where the trading algorithm will be executed) to establishing a permanent connection between computers that would periodically send new data from the remote database to the local one. Our decision relied on manually defining when to gather monthly data, via an Secure Shell (SSH) tunnel, by parts, to avoid memory shortage and the overhead associated with maintaining a permanent SSH connection. But what is an SSH tunnel?

Firstly, the SSH protocol is a mechanism for secure remote login from one computer to another, providing various alternative options for strong authentication while protecting the integrity and security of communications with strong encryption. SSH tunneling is a method of transporting any kind of networking data over an encrypted SSH connection. There are several Python libraries that enable us to efficiently implement an SSH tunnel which was one of the main reasons to use this approach. Figure 3.5 depicts the project's usage of an SSH tunnel.

3.5.2 Saving Monthly Data

After establishing a communication with the remote computer, we need to obtain data about digital currency trades and order books. To assist the progress of this thesis, we defined that we should obtain monthly information because it would only be gathered once and we could easily divide it into parts (by week for example) to avoid memory shortage problems. The only obvious downside of this approach is that we can only extract months prior to the current one. Since trade and order book information

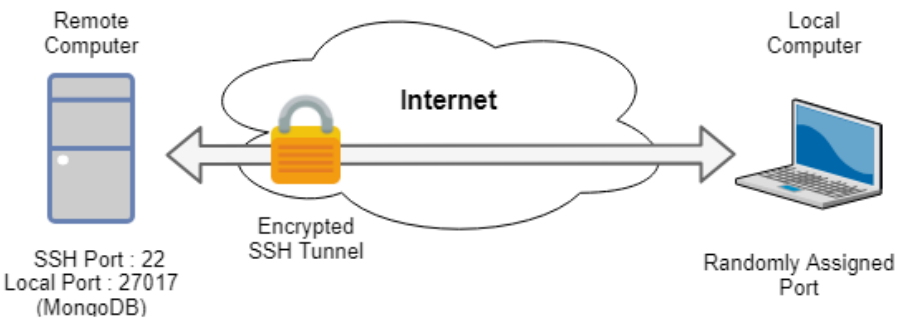

Figure 3.5: SSH Tunnel

is provided with different specifications, despite both being in the lightweight data-interchange format called JavaScript Object Notation (JSON), we will address each case separately.

Trades

To get a month's worth of trades, the specifications of which cryptocurrency is desired, the time intervals for each data part, to avoid memory shortage, and the type of trades (buy or sell) must be provided. The extraction algorithm will return a number of *DataFrames* equivalent to the number of time intervals defined by the user. Finally, the concatenation of those *DataFrames* will be saved to a CSV file, thus generating a record with all the buy or sell trades of that month. An example of the first five lines of one sell trades file is illustrated in Table 3.3.

Table 3.3: First Five Lines of Monthly Sell Trades CSV File

trade_date	amount	currency	rate	total
2018-11-07T00:00:00.223000Z	0.185479	BTC-USD	6448.50	1196.059719
2018-11-07T00:00:01.256000Z	0.258158	BTC-USD	6448.50	1664.730702
2018-11-07T00:00:01.256000Z	0.037040	BTC-USD	6448.50	238.852440
2018-11-07T00:00:07.024000Z	0.001000	BTC-USD	6448.73	6.448730
2018-11-07T00:00:07.025000Z	0.006871	BTC-USD	6448.78	44.309632

Obviously we always pick up both types of trades and concatenate them, but that manipulation will be explained in the next subsection.

Order Books

For order books, there is the demand of defining from which currency market we want the order books but there is no need for time intervals since we can gather the whole month without facing memory issues. The algorithm creates a new temporary output folder with the name "orderbooks_[Month][Year]" where each order book has the top fifty entries and will be saved to a CSV file having as title the date of the order book. An example of the first five lines of one order book file is represented in Table 3.4.

For memory management reasons, after aggregating the information present in all the order book files, the temporary folder will be deleted.

Table 3.4: First Five Lines of Monthly Order Book CSV File (BTC-USD)

price_bid	amount_bid	price_ask	amount_ask
6449.95	0.057044	6450.05	0.123041
6449.90	0.023020	6450.10	0.055580
6449.85	0.153013	6450.15	0.243619
6449.80	0.201010	6450.20	0.470035
6449.75	0.061298	6450.25	0.619753

3.5.3 Monthly Data Manipulation

After gathering monthly data about trades and order books, we must aggregate it since we are dealing with millions of records and the useful information can be extracted (or even created) with this method. But what does aggregation mean? In this particular case we are referencing data aggregation, that can be defined as a process in which information is obtained and expressed in a summary form. The main intent for this summary in this work is statistical analysis.

Like in the previous subsection, since trades and order books must be handled in distinct ways, we will detail those mechanisms independently.

Trades

The manipulation of raw monthly trade data can be broke down into three steps, aggregation, resampling and concatenation. Our aggregation function gets as input the period (using thirty seconds as default) that we intend to summarize and the path to the CSV file containing the buy or sell trades of the month we are handling. The generated output will be a *DataFrame* with the date corresponding to the start of the period, the number of trades, the total volume of money (USD,EUR) and amount of digital currency associated with trades. Additionally, the maximum, mean and minimum rates are also presented. Table 3.5 shows the first five lines of an example *DataFrame* output of the aggregation.

Table 3.5: First Five Lines of Aggregated Monthly Trades CSV File (BTC-USD)

date_end	num_trades	total	amount	rate_max	rate_min	rate_mean
2018-11-01 00:00:41.275	8	7975.47	1.27	6304.17	6304.17	6304.17
2018-11-01 00:01:11.275	4	1454.42	0.23	6304.17	6304.17	6304.17
2018-11-01 00:01:41.275	1	14.56	0.00	6304.17	6304.17	6304.17
2018-11-01 00:02:11.275	2	704.40	0.11	6304.17	6304.17	6304.17

Afterwards, we use resampling to reduce the quantity of information to be processed by the trading algorithm. The default resampling time is five minutes but, as the period in the aggregation function, this value is passed as an argument to the method so it can be easily modified. Finally, after doing this process to both buy and sell trades, we concatenate the resulting *DataFrames*.

Order Books

The manipulation of raw monthly trade data can be broke down into two steps, aggregation and information extraction. In this subsection, we will only address the former because the latter is closely related to the trading algorithm, which will be presented in the next section. Our aggregation function starts

by receiving the path to the folder containing all the CSV's with the monthly order book data. Before iterating through each file, our method creates a *DataFrame* where each row will have aggregated information obtained from one order book. There are three types of data summarized from each order book, total volume (in USD), total asks and bids percentage and four levels of each type of order volume's percentage. Each level takes into account approximately twelve entries of the order book, where the associated percentage represents the portion of the total order type percentage (bids or asks). Level one is the closest to the market value. Table 3.6 depicts an example (April 2018) of the first five lines of an aggregated monthly CSV.

An overview of all the components responsible for data processing is illustrated in Figure 3.6.

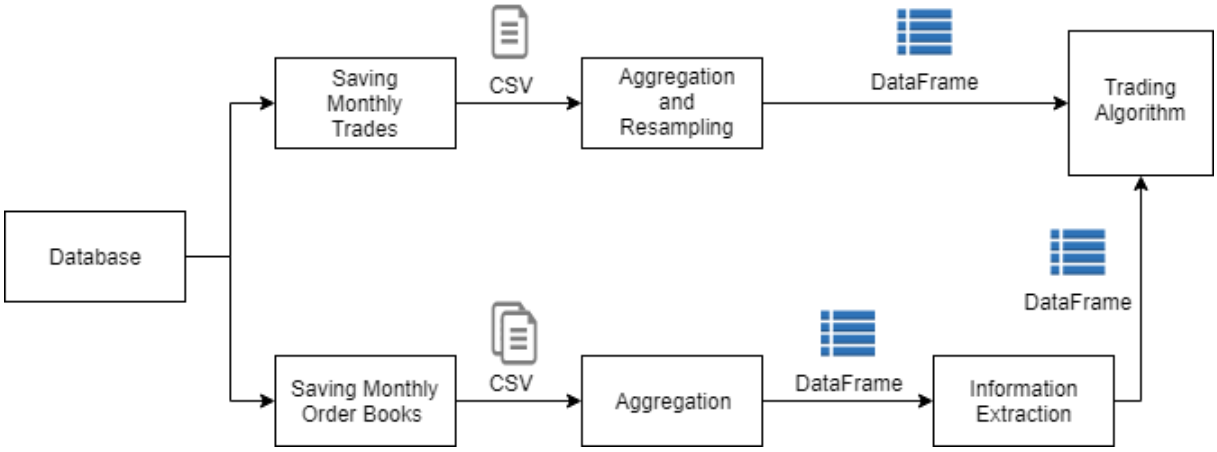

Figure 3.6: Data Processing

3.6 Trading Algorithm

The trading algorithm is one of the main elements of this thesis, since it allows to test several use cases and techniques and evaluate if they improve profitability. This section will describe all the parts related to the trading algorithm, cut down into four subsections:

1. Extracting Information From Order Book Data, where we detail the type of analysis made to raw data and how we infer usable information from this examination;
2. Forecasting With The ARIMA API, subsection explaining how forecasting is used and how do we determine the parameters of the ARIMA model;
3. Percentage Stop Order, subsection detailing the stop order type created to improve trade profitability;
4. Output, where the methodology and tools used for the simulation report will be explained.

3.6.1 Extracting Information From Order Book Data

After aggregating the monthly order book information, we had the challenge of defining what kind of data we would want to gather from the four levels of total asks and bids percentage and total volume. Firstly we defined that our trading algorithm should take advantage of the digital currency market volatility. With this aspect in mind, we established the first two use cases for our algorithm, using a percentage of the preeminent monthly volume of the order book to find periods with high demand for sales or buys and identify total volume inversions. The second use case can be defined by a shift of the total volume percentage (over a defined threshold) from the asks side to the bids side or vice versa.

To be able to test the mentioned use cases, first we had to extract the monthly maximum volume values from all the levels of the ask and bid side of the order book. An example of these values is depicted in table 3.6.

Table 3.6: Example of Monthly Max Volumes in the Order Book(BTC-USD)

side	lvl1_Max	lvl2_Max	lvl3_Max	lvl4_Max	total_max
bids	2103445.32	2412686.87	1134828.17	1265354.70	3104766.46
asks	2289236.78	2452814.35	1507181.41	1589485.41	2665952.61

After defining our first use case and gathering the maximum monthly volumes, we had to specify the threshold that retrieved at least fifteen periods, number delineated with my supervisor's guidance. The equation used for this situation was

$$Volume \geq (MaxMonthlyVolume * Threshold) \tag{3.1}$$

where our objective was to find order books whose volume, from the ask or bid side, is higher or equal to the maximum monthly volume from the same order book part multiplied by the chosen threshold. The difference between the two mentioned use cases is that the first one is represented by the maximum monthly volume amount in the four levels whilst the second one only considers the top monthly volume in the front level.

To provide a better understanding of both test cases, Figures 3.7 and 3.8 illustrate simple examples of how their mechanism is used to find suitable investment periods. The first plot depicts a typical situation where the conditions for selection are met. Let us start by considering the yellow line as the total order book volume in the bids side, the blue line as the total bids orders maximum volume in the testing month and the red line as the value from where we should start a position. Point A represents the first period that would be retrieved since it follows the condition presented by the 3.1 equation. From this point until point C, every stage will be seen as a possible investment situation. Two additional points are worth being mentioned, the first being the fact that all volumes below the red line will not be selected and that the periods must have a distance between them of five minutes, to avoid redundancy. This will also be done for the volume on the first level of the order book, on both asks and bids side.

On the other hand, Figure 3.8 exemplifies the second use case. Now we are considering the total order book percentage on both sides, the red line represents the percentage of asks while the green line represents the percentage of bids. Furthermore let us consider a threshold of 20 %. Point A shows

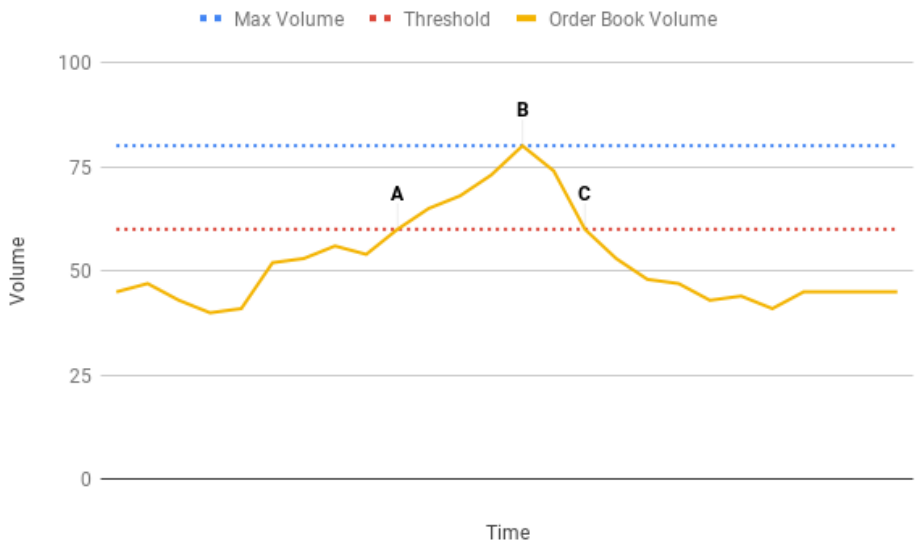

Figure 3.7: First Use Case Example

the moment where the inversion takes place since total bids percentage will be more significant than the percentage of asks orders. When the next period (B) is reached, we achieved an inversion that is equal to our threshold, so this period will be returned from the use case as a possible moment for starting a long position (since the inversion was from the asks to the bids side). The following periods in the figure will not be considered since no inversion occured.

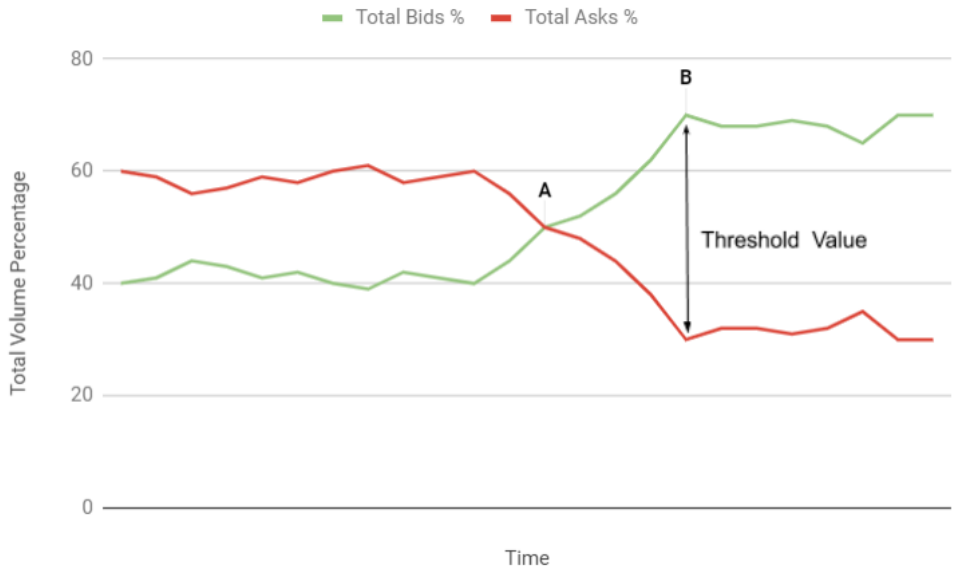

Figure 3.8: Second Use Case Example

Finally we needed to save this information to be able to use it in our simulation while avoiding repeating this information extraction. Nonetheless, while checking the resulting periods, we verified that the majority were consecutive which did not give us useful information so we established that we would only save periods with at least five minutes of interval between them. An example of the successive periods

is presented in table 3.7. Afterwards, we saved the *DataFrames* containing the order books with the respective timestamps, that respected our temporal condition, in CSV files.

Table 3.7: Example of Order Book Periods Extracted (BTC-USD)

Timestamp	Time Difference Previous Entry	Period
2018-07-06 18:56:22.832	0 days 00:08:30.204000	1
2018-07-06 18:56:28.817	0 days 00:00:05.985000	0
2018-07-06 18:56:31.470	0 days 00:00:02.653000	0
2018-07-06 18:56:37.516	0 days 00:00:06.046000	0
2018-07-06 18:56:41.499	0 days 00:00:03.983000	0
2018-07-06 18:56:47.205	0 days 00:00:05.706000	0
2018-07-06 18:56:52.500	0 days 00:00:05.295000	0
2018-07-06 18:57:02.266	0 days 00:00:09.766000	0
2018-07-06 18:59:32.380	0 days 00:02:30.114000	0
2018-07-06 19:09:13.651	0 days 00:09:41.271000	1

3.6.2 Forecasting With ARIMA

Our third and last use case forecasts prices using ARIMA and invests accordingly, establishing long positions when the value is predicted to rise or short positions if we anticipate a price descent. As explained previously, the ARIMA model handles stationary time series and needs three arguments (p,d,q) where p represents the order of the AR, d is the order of differencing and q is the order of the MA. Therefore, it is mandatory to check if our input series is stationary which means that the mean, variance and autocovariance structure are constant. To verify this, we can use the Augmented Dicky-Fuller (ADF) test [39]. This test receives a time series as input and verifies the null hypothesis, general statement that there is no relationship between two measured phenomena [40], that a unit root is present in the series. The existence of an unit root refers to a stochastic trend in time series, meaning that it can change in each run due to the random component of the process so this procedure will be hard to predict and control.

In the ADF test case, the null hypothesis is that the time series is non-stationary. Specifically, if the probability value of the test is less than the significance level (0.05) then we reject the null hypothesis and deduce that the time series is stationary. Otherwise, we will need to apply differencing to the series so it becomes stationary, buy how do we find the required order of differencing? The appropriate order is the minimum differencing necessary so the Autocorrelation Function (ACF) plot reaches zero quickly and the mean has residual variations with time.

The next step is to analyze if AR terms are needed. To do so, we inspect the Partial Autocorrelation Function (PACF) plot. Partial autocorrelation gives the degree of association between two random variables, with the impact of a set of ruling random variables discarded. In the analysis of time series, the PACF presents the partial correlation of a stationary series with its own lagged amounts. An example using two lags is given by the following equation

$$Y_t = \alpha_0 + \alpha_1 Y_{t-1} + \alpha_2 Y_{t-2} \tag{3.2}$$

where Y_t is the current series and Y_{t-1} is the first lag of Y, then the partial autocorrelation of the second lag is the coefficient α_2 of in the above equation. To discover the order we should use, we plot the PACF and find the first lag well above the significance line. Figure 3.7 illustrates a case where we should choose *p* as 1.

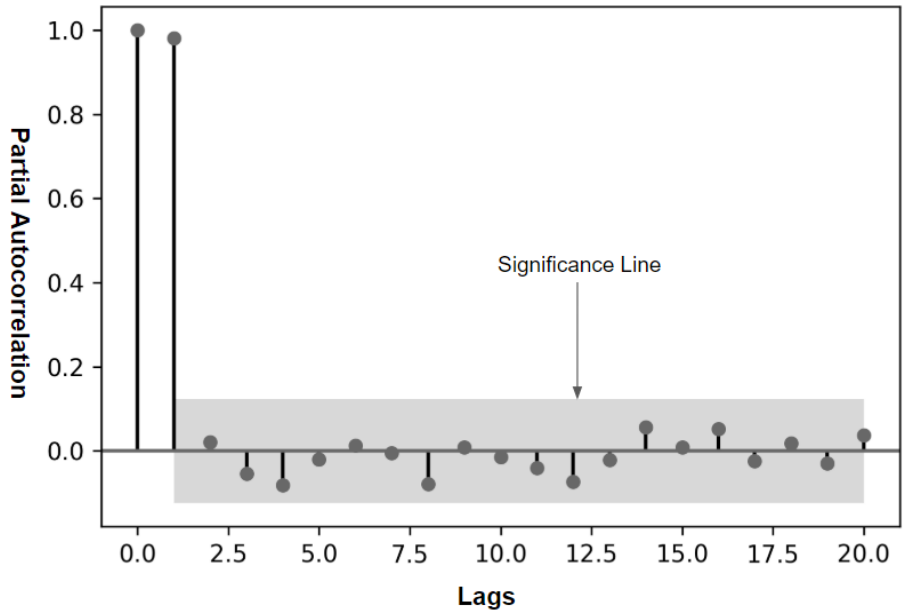

Figure 3.9: PACF Plot Example

As we examined the PACF plot to find the number of AR terms, we can do the same procedure with the ACF plot to find the MA order. Each MA element may be described as the error of the lagged prevision. They will be used to eliminate any autocorrelation in the now stationary time series.

Finally, we have all the elements required to build an ARIMA model. We used Python's *statsmodels* package since it has methods that helps us to rapidly implement an ARIMA model and test it. Figure 3.8 depicts how we the model was created using *statsmodels*. After all this configuration, we are now able to test our model. We selected the *forecast()* function to forecast future time steps since the alternative *predict()* method requires several specifications whilst the first performs a one-step forecast using the model. Additionally, we divided the input dataset (e.g., monthly order book mean buy rate and monthly order book mean sell rate) into train and test sets.

The train set will be used to fit the model and produce a prediction for each element on the test collection. For consistency purposes, we determined that always 75 percent of the input data would be used as the training collection and the remaining 25 percent would be the test set. A decision to use a rolling forecast was made due to the dependence on past observations for differencing. A simple method was adopted to implement this rolling forecast by re-creating the ARIMA model after each new observation is obtained. To keep track of all observations we used a list that initially contains the training set and to which new observations are added each iteration.

Lastly, we calculate a final Mean Squared Error (MSE) score for the forecasts, producing a point of comparison for different ARIMA configurations. After finding the best composition for the model, we can now use this predictions in our simulation algorithm to infer the market's trend and trade accordingly.

Afterwards, we verify if the forecast provided useful information.

```
from statsmodels.tsa.arima_model import ARIMA
```

```
# 1 for AR model parameter, 1 for differencing order,MA of 0
model = ARIMA(df['buy_meanRate'],order=(1,1,0))
model_fit = model.fit()
print(model_fit.summary())
```

```
                             ARIMA Model Results
==============================================================================
Dep. Variable:         D.buy_meanRate   No. Observations:                 8928
Model:                 ARIMA(1, 1, 0)   Log Likelihood              -32866.414
Method:                       css-mle   S.D. of innovations              9.605
Date:                Sun, 30 Dec 2018   AIC                          65738.828
Time:                        00:41:36   BIC                          65760.119
Sample:                    07-01-2018   HQIC                         65746.076
                         - 08-01-2018
==============================================================================
                     coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const                0.1481      0.139      1.067      0.286      -0.124       0.420
ar.L1.D.buy_meanRate     0.2681      0.010     26.294      0.000       0.248       0.288
                                    Roots
=============================================================================
                  Real          Imaginary           Modulus         Frequency
-----------------------------------------------------------------------------
AR.1            3.7297           +0.0000j            3.7297            0.0000
-----------------------------------------------------------------------------
```

Figure 3.10: ARIMA Implementation Example

3.6.3 Percentage Stop Order

After studying the standard types of limit and stop orders used, we thought that a trailing stop order was the fittest for our intentions of optimizing profits. Nevertheless, due to the volatility of the digital currency market, we needed more control over potential stop losses and be conservative to improve mean profit. The addition of dynamic limits presented itself like a solid idea, but how could we implement this concept? To solve this problem, we introduced a new variety of stop order called Percentage Stop Order (PSO).

Defining the adoption of percentages to the detriment of flat currency amounts allowed the PSO to have more flexibility and to sustain the considerable rate fluctuations of cryptocurrencies. Our stop order has conservative default percentages, using 0.5 percent as the stop gain percentage and a 2 percent stop loss. Also, the chosen threshold that provided the desired dynamic behaviour was 0.25 percent. To select this values, we tested several combinations and these produced the best results. Now we have the starting values, how does our stop order work during a trading simulation?

There are four courses of action that may be taken by the PSO algorithm. To clarify each one of them we will consider a simulation where we are placing long positions, which means that we bought currency and are expecting a growth in value. Figure 3.9 depicts the example where we start with 1 BTC bought at an original rate of 1000 USD. Point A illustrates a situation where the stop gain and stop loss orders were not reached nor the maximum sell rate went over the threshold to dynamically change both stop orders. When assessing the max rate sell at the B event, the PSO threshold was passed so both the upper and lower limits are updated. It is important to single out how to check if we need to update the

stop order limits, given by the equation

$$rate\_sell\_max \geq (original\_rate * (1 + threshold\_rate)) \tag{3.3}$$

Afterwards, the rate increases until reaching our current stop order limit in point C, at this point we sell our BTC having a profit of 0.75 percent. Finally, we decide to buy 1 BTC again in point D. The original limits are established and as we progress in time, the value of the digital currency held declines sharply so when the stop loss order is reached, at point E, we sell our position to minimize our loss.

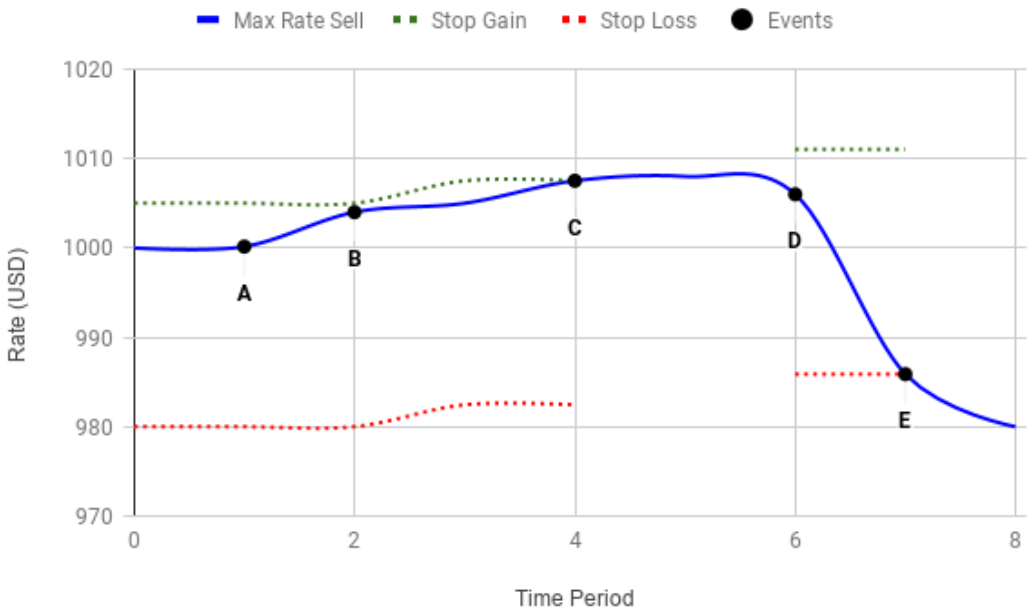

Figure 3.11: Percentage Stop Order Example

3.6.4 Simulation

The simulation component is crucial in this work since it allowed us to test all the mechanisms developed and verify the results produced by our three use cases. As input, it must receive two *DataFrames*, one with the real monthly trading information containing data about buys, sells (we can use minimum, mean or maximum values) in five minute intervals, and another one regarding the periods where we should invest according to the analysis performed by the selected use case. Furthermore, we define if we want to take into account fees that should be paid to the digital asset exchange and what type of positions we want to put on the market (long, short or both). A *DataFrame* containing the execution results will be output where each row referring to a time period will have a date, volume, number of trades, amount, rate, mock investor's money, mock investor's amount of digital currency, an indicator if an action was taken in that time frame and the profit percentage.

After gathering the initial configuration parameters, we use a method that generates the starting conditions of the simulation. Holding an amount of cryptocurrency bought at the starting market value of

the considered month or with money to invest and the original rate and amount are the variables handled by this function. These arrangements vary with the kind of position selected. We concluded the setup process, which allows us to start the simulation.

Let us consider a configuration where we start with money to invest (1000 USD), no amount of cryptocurrency and only intend to establish long positions with all the mock capital at our disposal. In this situation, the first course of action will be to find a suitable period to start investing. For each date in the real monthly *DataFrame*, we validate if it matches an action period obtained from the use case examination and if the MA of the last three periods display a tendency compatible with the trade we want to make (up trend for a long position and down trend for a short position). This is the first of four courses of action available in our simulation.

The second case occurs when we already have money invested and need to assess the next action. As described in the previous subsection, the PSO is used to define what is the step to be taken. Three options are available, keep the amount of digital currency while using the current stop limit orders, hold cryptocurrency but update stop limit orders and sell what we possessed. A dedicated function to deal with the last scenario was created to handle the change of position. But what if we sell and need to know when to invest although the next defined period has already passed in the simulation timeline?

Handling this issue is the goal of the third course of action. By verifying if the set investing date is older than the current date of simulation, we can select the next period in the *DataFrame*. We repeat this process until we find a period or we reach the end of the *DataFrame*. The fourth and last case just adds a new row to the results with information of the time period and occurs if none of the previous conditions were activated.

Finally, after reaching the end of the simulation, we validate if there is no money invested, otherwise we sell the amount of digital currency we have at the most recent rate to calculate the monthly profit. The *DataFrame* containing the results is now finished and saved in a CSV file. Table 3.10 displays an example of the last lines of the generated CSV file containing simulation results.

Table 3.8: Example of Last Lines of Results File of a July 2018 Simulation (BTC-USD)

date_begin	**total volume**	**trades**	**amount**	**rate**	**money**	**curr_held**	**action**	**profit_pct**
07-31 23:15	125172.12	164	16.31	7678.5	1073.87	0	1	0.074
07-31 23:20	282103.25	255	36.82	7654.04	1073.87	0	0	0.074
07-31 23:25	1685430	560	220.51	7629.48	1073.87	0	0	0.074
07-31 23:30	574214.34	162	74.35	7643.99	0	0.140486	1	0.074
07-31 23:35	642159.89	265	82.73	7730.43	0	0.140486	0	0.086
07-31 23:40	350464.33	355	45.22	7726.48	0	0.140486	0	0.085
07-31 23:45	106898.56	167	13.79	7749.98	0	0.140486	0	0.088
07-31 23:50	387562.07	264	49.97	7740.01	0	0.140486	0	0.087
07-31 23:55	171351.05	174	22.14	7727.28	0	0.140486	0	0.086
08-01 00:00	9115.31	19	1.18	7719.77	0	0.140486	0	0.0845
08-01-00:00	9115.31	19	1.18	7719.77	1084.52	0	1	0.0845

A lot of flexibility is given in the simulation component, since we can easily modify the parameters, conditions and use case that are employed. After obtaining the results, we had to establish how would we present the most relevant execution information since we might want to analyze the results with a

more user friendly interface than a CSV file. How these issues were addressed and how the presentation layer was organized will be explained in the next subsection.

3.6.5 Output

After running a simulation we discussed what type of information would be relevant to show and how would we implement the user interface. We decided to use a Python package called *Bokeh*. *Bokeh* is a visualization library that uses modern browsers for presentation. Its main objective is to provide exquisite,succinct construction of versatile graphics, and to extend this capability with high-performance interactivity over extensive datasets. This package allowed us to quickly develop interactive plots and to organize them so that our output could be compared to a report. An important feature was the possibility to test *Bokeh* methods directly in a Jupyter Notebook, which allowed us to reach our objective rapidly. Nevertheless, we have the possibility to save the generated output in Hypertext Markup Language (HTML).

The output designed for a simulation was five plots: profit variation, asks/bids minimum rate variation, daily actions, actions at each period and percentage variation of volume between the bid and ask sides. For the profit variation graphic, we use the *DataFrame* containing the simulation results, also providing the monthly profit as the plot's title. An example illustrating this graphic is depicted in Figure 3.11. We added an hover tool to offer the possibility to study in more detail the data present in the profit variation plot. This tool shows the date, money invested, amount of digital currency held and the profit percentage. Additionally, *Bokeh* supplies three more important features, box zoom, wheel zoom and the ability to reset our plots.

On the other hand, the original trading information *DataFrame* is utilized for the second plot. We might use this graphic to verify periods with high discrepancy between the minimum buy and sell rates. As the first plot, we are able to use an hover tool, but here it gives information about the minimum rates at each period. Figure 3.12 shows a sample plot, although it seems that bids and asks always have the same value, a closer inspection displays subtle differences.

Furthermore, we wanted to see where the trading algorithm decided to make actions during our simulation. To solve this need, we re-sampled the action column from the *DataFrame* resulting from the simulation to check in which days of the month the algorithm operated. This ended up as the first plot in Figure 3.13. The second graphic in the mentioned figure has all the monthly five minute time periods and whether an action took place in each one of them. Using the interactive tools provided by *Bokeh* is critical in the analysis of these plots since, at first glance, we might assume that there were actions in all periods present in Figure 3.13 and that is incorrect.

Lastly, a plot of the variation of volume between the bid and ask sides is presented. This plot was mainly used to verify the periods generated by the second use case method and check if the threshold applied was efficient. Figure 3.15 illustrates an example of this plot, using the box zoom tool to show only one day of the simulation. The last feature created regarding the generated output is the ability to convert the HTML file to Portable Document Format (PDF), but this is completely optional.

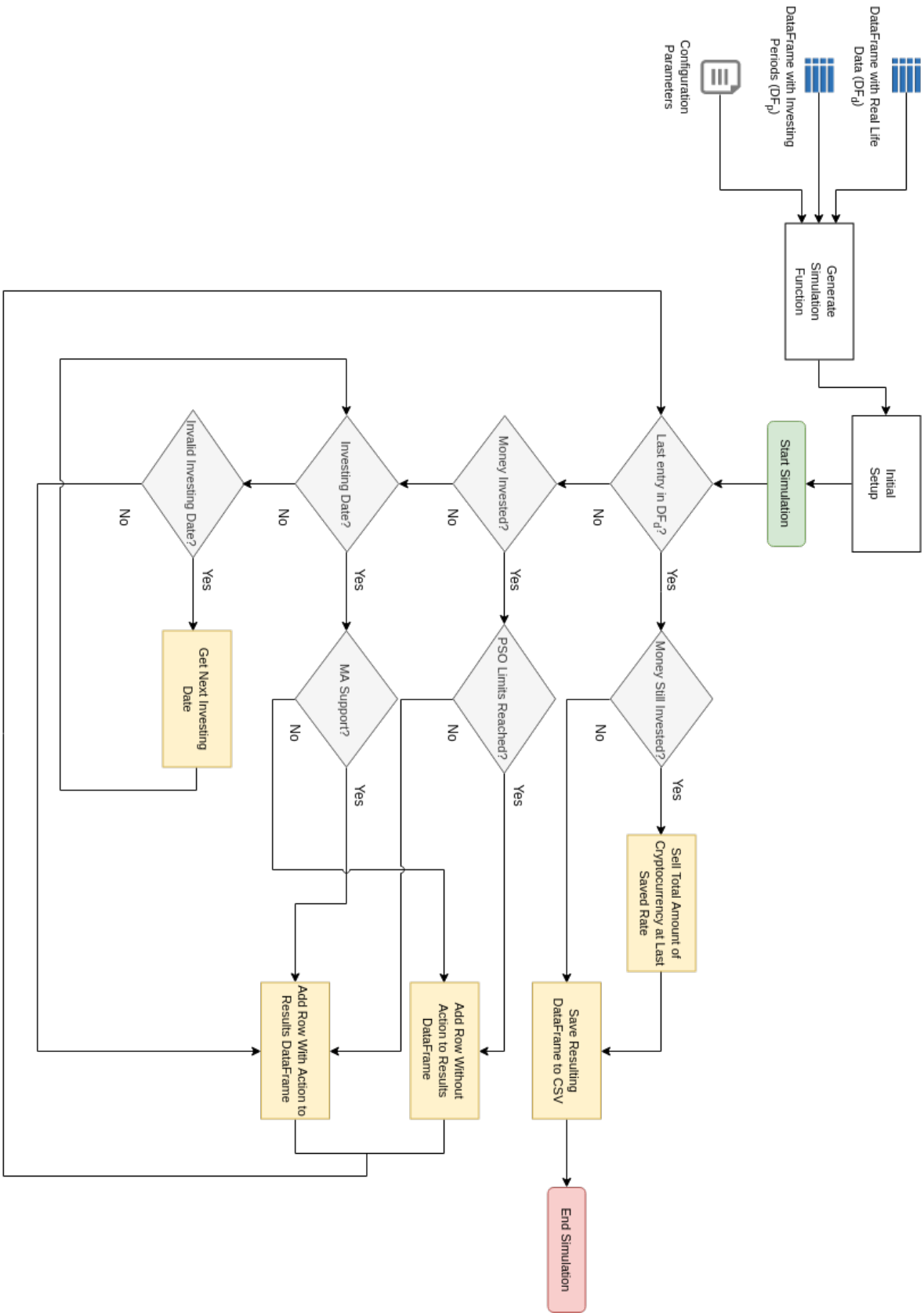

Figure 3.12: Simulation Overview

Figure 3.13: Profit Variation Plot

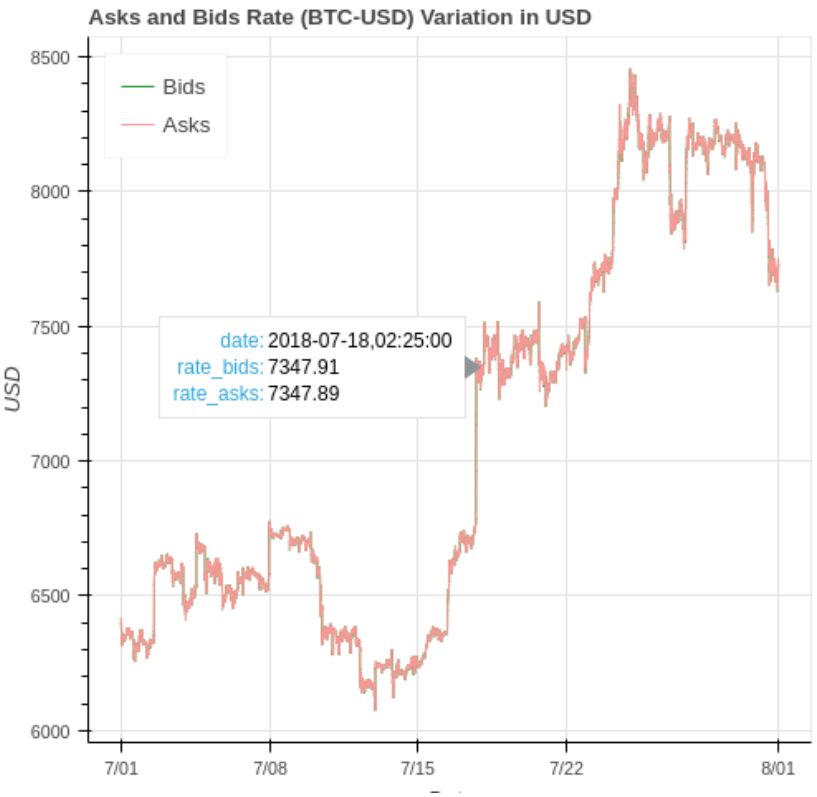

Figure 3.14: Minimum Buy/Ask Rates Variation Plot

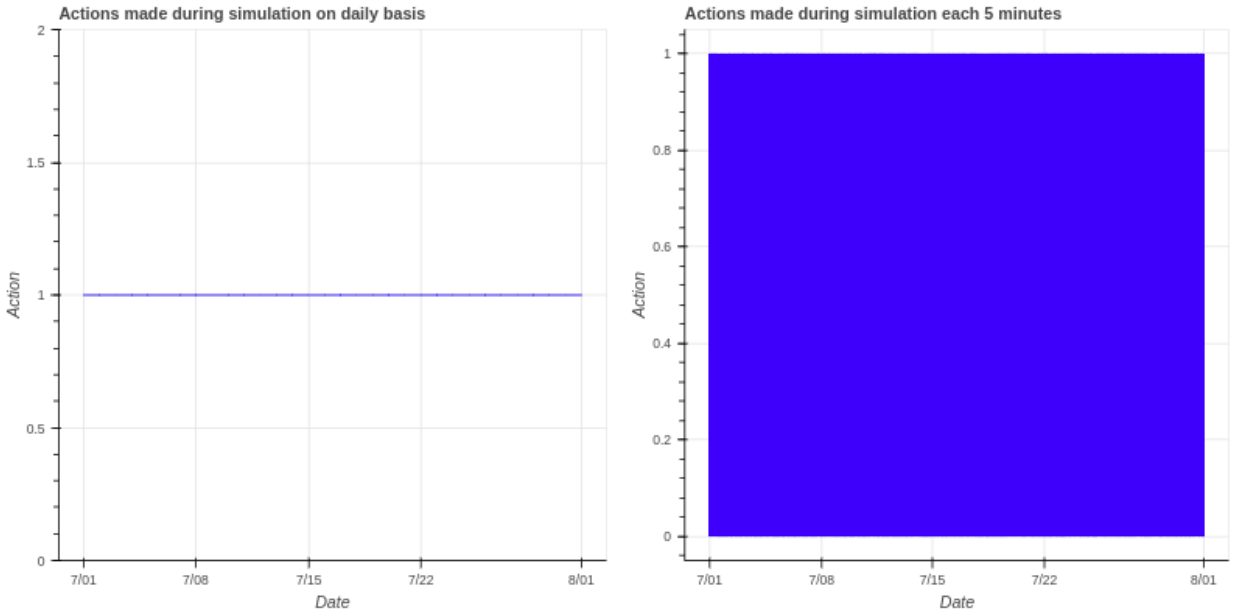

Figure 3.15: Action Plots

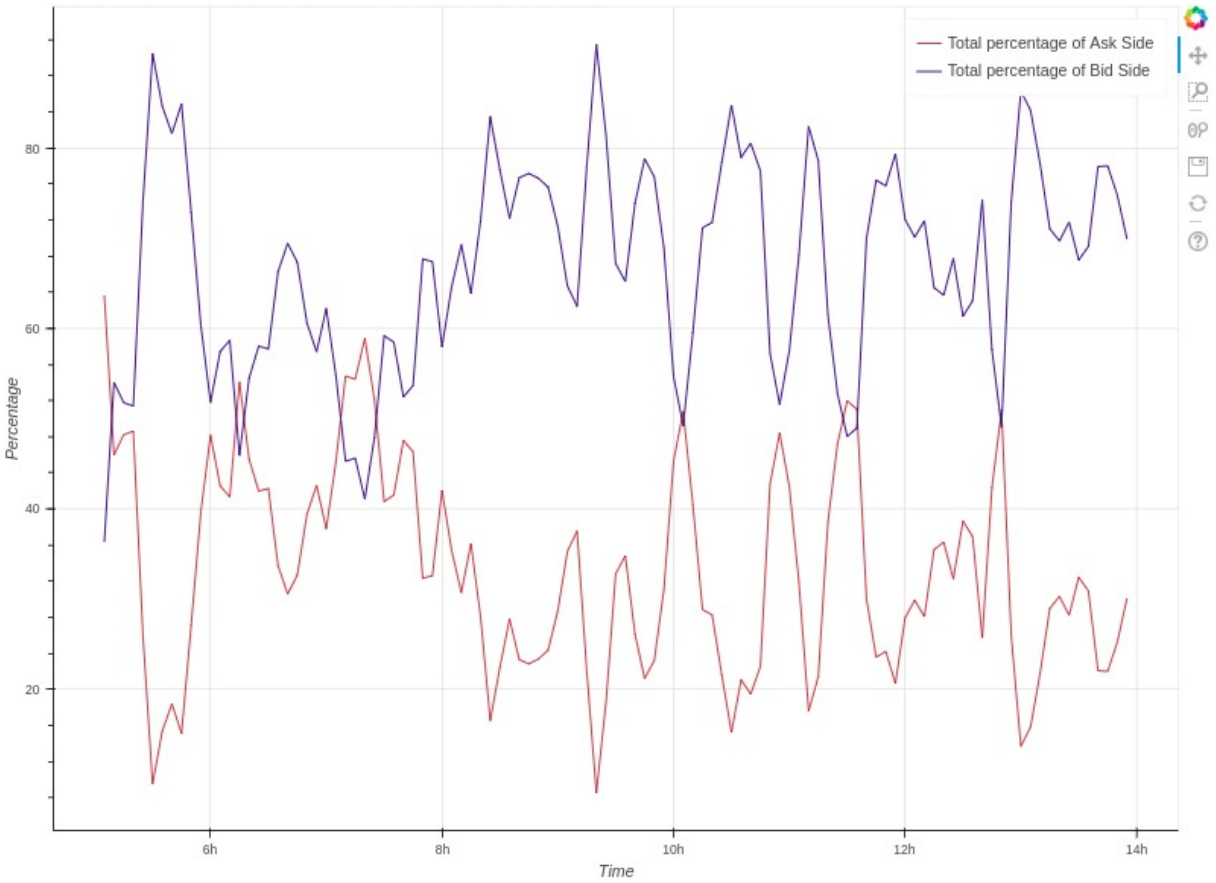

Figure 3.16: Daily Percentage Volume Side Variation

Chapter 4

Evaluation

In this chapter, the outcome of using strategies that extract information from the order book and use prediction over the digital currencies' market will be presented and discussed. The following sections have three use cases, a specific setup for each one of them, their purpose and lastly, an examination of results.

4.1 Tests Scenarios and Objectives

To validate the efficiency of the considered use cases, we used a method called Backtesting [41]. This strategy enables us, relying on previous trading data, to simulate our algorithm, assess the results and verify the mean profitability without using real money. Therefore, this method allows for better understanding of the risk associated with our approach. Additionally, if the simulation outcome is positive then our algorithm should be solid and generate profits when applied to the current market. Lastly, by adopting this procedure we can analyze several test scenarios and discover concept problems, areas for improvement and potential gains without wasting an extended period of time.

The following three use cases were performed:

1. Using a threshold over the maximum monthly order book volume to identify periods with a potential high volatility since a greater volume is correlated to a high demand for sales or buys of digital currency;

2. Discovering periods where a shift of the total volume percentage (over a defined threshold) from the asks side to the bids side or vice versa took place, which allows us to invest after an aggressive change of trend;

3. Using ARIMA to forecast prices and use the predictions as an indicator of market trend, subsequently applying this obtained information in the first two test scenarios and verify if it improves or reduces profitability.

These use cases were chosen because they provide a solid base for verifying if we can extract

information from order books and trade accordingly in four different digital currencies. Furthermore, we will validate if forecasting with ARIMA improves profitability.

4.2 General Configuration

Despite the different elements associated with each one of the three use cases, several processes and configuration parameters are identical throughout the evaluation phase. Table 4.1 displays the configuration values that remained the same during the whole testing operation.

Table 4.1: General Setup Parameters

Configuration Element	**Value**
Test Period	Monthly
Selected Periods	July and November 2018
Digital Currencies	4
Resampling Data By	5 Minutes
Starting Money	1000 USD
Rate	Mean
Moving Average	200 Days
Considered MA Periods	3 (15 minutes)

The chosen test period is a month although we have more than a year of trading information because this time window already needs a considerable amount of hours (at least eight) to aggregate all the data collected from the digital asset exchange. Nevertheless, to improve the quality of our results, we considered two distinct months for each currency simulation on each one of the use cases. As previously mentioned, the cryptocurrencies tested were BTC, ETH, LTC and BCH.To compare our results with a standard trading method, we used the Buy & Hold strategy in the same test periods, this approach buys digital currency in the first time period and only sells at the last one. Despite using a monthly test period, we gathered more than half a million order books for each digital currency so we used a 5 minute resampling. This processed allowed us to significantly reduce the computation time needed to execute a simulation.

Regarding simulation parameters, we established that the algorithm has 1000 USD to invest in every single run that does not start with a short position (in those cases, we start with 1 unit of the specific cryptocurrency) and must always use the full amount of virtual money available in every trade. Unfortunately, due to time constraints, only the USD market was tested. Additionally, our trading method constantly uses the mean rate of the trades that took place in the five minutes of the considered period, alongside a 200 day MA that enables us to follow the market trend. Before defining a new short or long trade, our algorithm verifies the directions of the MAs of the three previous periods and allows the position if it respects the market flow. Lastly, all the investing periods determined by our use cases will be generated before we execute the simulation and we did not take into account possible transaction fees.

After running a simulation, the Return On Investment (ROI) will be calculated. In the event of finishing the test period with money invested, our algorithm uses the last saved rate of the considered cryptocurrency to sell the full amount owned. This step allows the ROI computation.

Table 4.2: First Use Case Thresholds For July 2018 Simulations

	Bids				Asks			
Currency	**Lvl1 Vol**	**Periods**	**Total Vol**	**Periods**	**Lvl1 Vol**	**Periods**	**Total Vol**	**Periods**
LTC	0.75	15	0.65	20	0.75	16	0.85	16
ETH	0.65	17	0.8	26	0.7	17	0.65	20
BTC	0.55	16	0.55	15	0.45	16	0.6	17
BCH	0.6	15	0.8	18	0.55	15	0.45	16

Table 4.3: First Use Case Thresholds For November 2018 Simulations

	Bids				Asks			
Currency	**Lvl1 Vol**	**Periods**	**Total Vol**	**Periods**	**Lvl1 Vol**	**Periods**	**Total Vol**	**Periods**
LTC	0.75	19	0.8	18	0.6	18	0.75	15
ETH	0.6	18	0.8	17	0.5	19	0.75	23
BTC	0.6	16	0.7	16	0.65	18	0.65	17
BCH*	NA	NA	NA	NA	NA	NA	NA	NA

4.3 Test Case 1 - Maximum Monthly Order Book Volume Period

This test case initially considers the maximum monthly volume values at the first level of the order book and the total volume in both bid and ask sides. Afterwards, it applies a threshold to those maximum values and finds periods having a volume greater or equal to the calculated value. Lastly, this process combines all the generated periods into a single *DataFrame* that will serve as input to the simulation.

4.3.1 Scenario Setup

For this tests, we selected the months of July and November of 2018. The first step taken was to find the thresholds for each digital currency. Our principle was to find the highest threshold that retrieved at least fifteen periods for investing. Table 4.2 illustrates the thresholds used with the corresponding number of periods. Despite not having any issues regarding the July data, we found a problem with the November BCH order book information (empty order books) on the database so we did not consider this currency for the second test month. The thresholds adopted for the November data are displayed on Table 4.3. After gathering the periods and saving them in CSV files, we are now ready to start the testing phase using the simulation component.

4.3.2 Performance

Figure 4.1 presents the performance achieved by our trading strategy regarding the monthly ROI for each one of the four cryptocurrencies in July 2018. We considered three options for types of trades used in our algorithm: short, long and both positions. Afterwards, we use the November 2018 data to simulate the behaviour of our trading method with the note that BCH was not considered due to errors in the extracted order book information. The results of this simulation are shown in Figure 4.2.

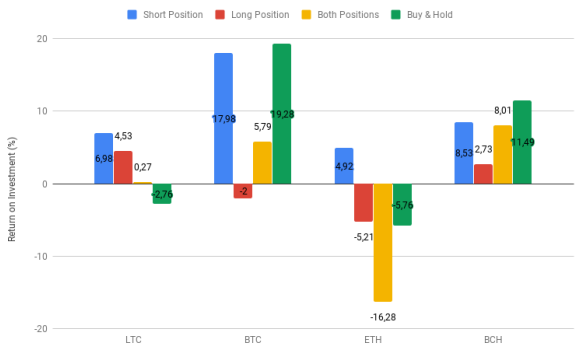

Figure 4.1: First Test Case Results on July 2018

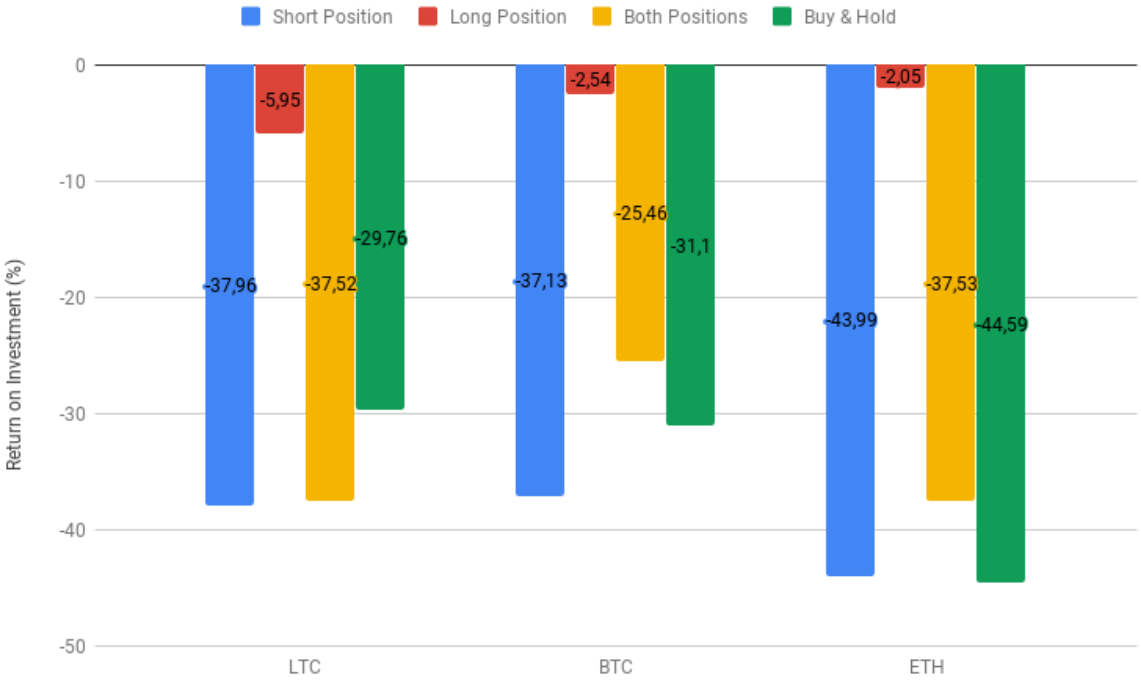

Figure 4.2: First Test Case Results on November 2018

4.3.3 Outcome Graphs and Discussion

The use of a percentage of the maximum order book volume to find suitable investing periods had the objective to provide more profit than the Buy & Hold trading strategy. Observing the plot illustrated in Figure 4.1, we can verify that our use case obtains better results, comparing to the mentioned strategy, with LTC and ETH ending the simulation with profit on July 2018. Nevertheless, the Buy & Hold ended the month with more significant gains on BTC and BCH. Another interesting note is the fact that all the test digital currencies provided solid results when running this scenario using only short positions.

Regarding the November 2018 results depicted in Figure 4.2, we may think that the scale is wrong since there are only negative percentages but this month brought important price decreases on several cryptocurrencies. The Buy & Hold strategy lost a major portion of the investment on all digital currencies. Only using long positions with this test case allowed us to significantly reduce our losses (e.g., lost a

minimum of 35% less than any other type of strategy for ETH).

The best test scenario, classified by profitability, is shown in Figure 4.3 and provided a 17,98% ROI. The plot on the right side of the figure displays the mean rate of asks and bids during our testing period. We can clearly observe that the most profitable trades happened after the two most important rate increases on the 18th an on the 24th of July. Another simulation that ended with profit is presented in Figure 4.4. in spite of having a considerable loss after the first week. In this test we are using both short and long positions, which might be an indicator on how our algorithm managed to have profit despite suffering a heavy loss. On the other hand, Figure 4.5 illustrates the use case situation that provided the worst results, with a staggering loss of 43,99% of our initial investment amount. This test used only short positions so it could not win money since ETH faced a steady decline in value along the month of November. Nonetheless, the five minute periods may be increasing our losses since we can't fully use the PSO capabilities if we are facing high volatility.

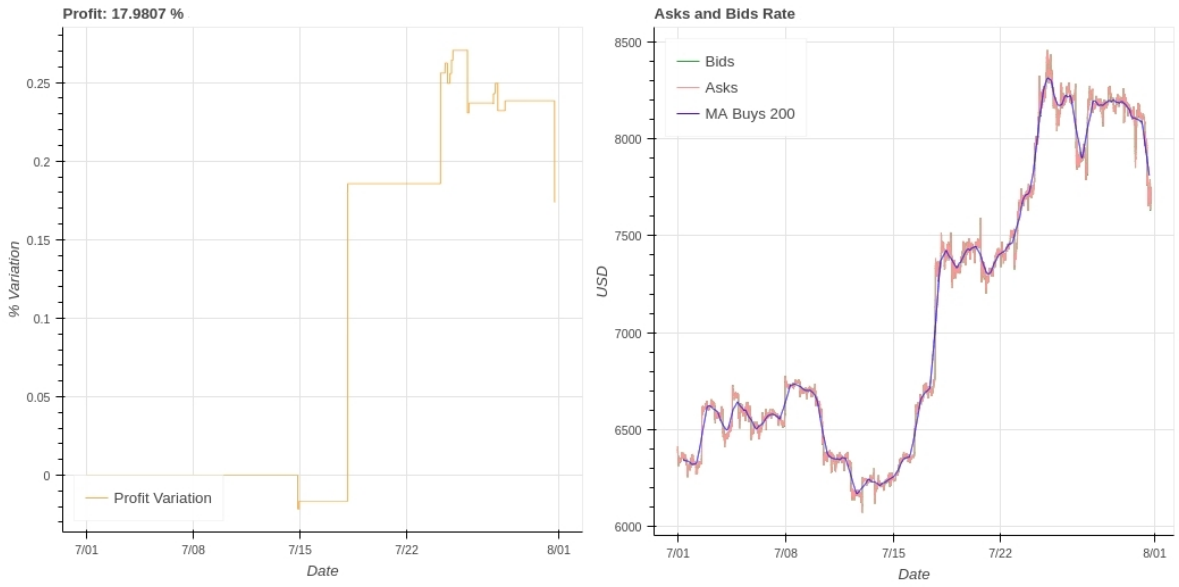

Figure 4.3: BTC First Test Results Using Short Positions on July 2018

4.4 Test Case 2 - Monthly Total Order Book Percentage Inversion Of Bids And Asks

This test case considers predefined thresholds to find inversions between the total order book percentage of bids and asks. Afterwards, it saves all time periods where the inversion matched the threshold. Lastly, this process combines all the generated periods into a single *DataFrame* that will serve as input to the simulation.

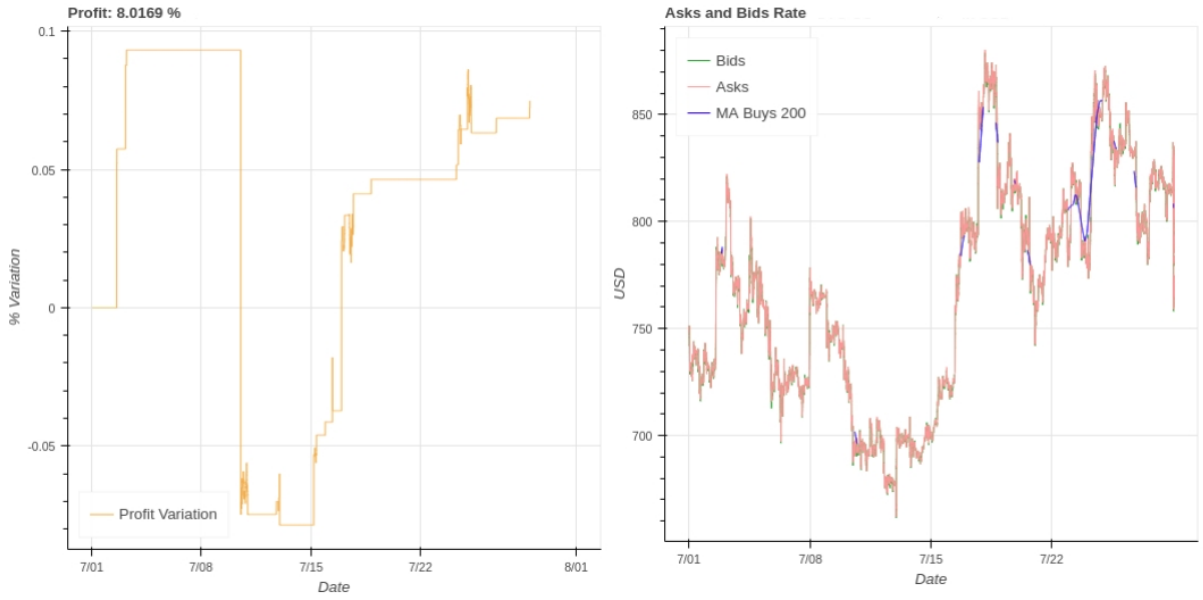

Figure 4.4: BCH Results Using Both Position Types on July 2018

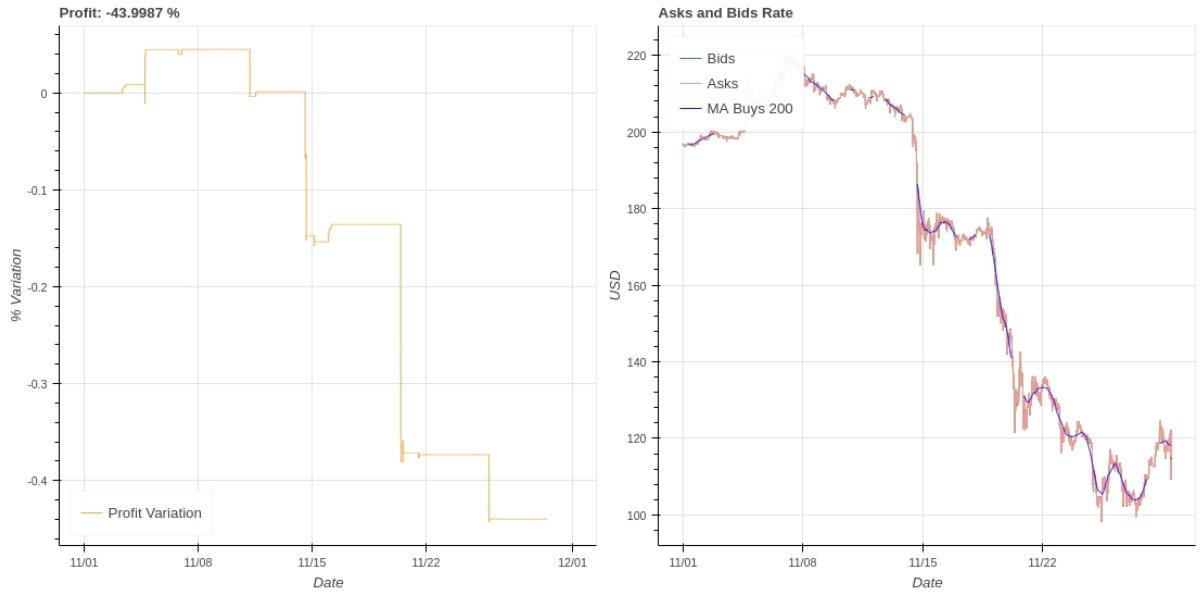

Figure 4.5: ETH First Test Results Using Short Positions on November 2018

4.4.1 Scenario Setup

For this use case, the same months were considered for the simulation and we also considered short, long and both types of trades. Our principle remained finding the highest threshold that retrieved at least fifteen periods containing opportunities for investment. Tables 4.4 and 4.5 present the considered thresholds for the months used in the simulation. Due to problems with BCH November's order book data, we only tested it in the July period. Lastly, after gathering the periods and saving them in CSV files, we are now able to execute this use case.

Table 4.4: Second Use Case Thresholds For July 2018 Simulations

Currency	Threshold_Bids	Periods	Threshold_Asks	Periods
LTC	0,3	15	0,3	19
ETH	0,45	17	0,45	23
BTC	0,7	16	0,7	28
BCH	0,45	15	0,65	18

Table 4.5: Second Use Case Thresholds For November 2018 Simulations

Currency	Threshold_Bids	Periods	Threshold_Asks	Periods
LTC	0,2	27	0,2	19
ETH	0,35	32	0,35	18
BTC	0,45	15	0,45	17

4.4.2 Performance

Figure 4.6 illustrates the performance achieved by using in our simulation periods where an total order book percentage inversion took place in July 2018. We considered three types of trade configurations in our algorithm: short, long and both positions. As in the previous use case, we use also tested this strategy on November 2018 data (except for BCH). The results of this simulation are shown in Figure 4.7. Lastly, our results were compared with the ones generated by the Buy & Hold method.

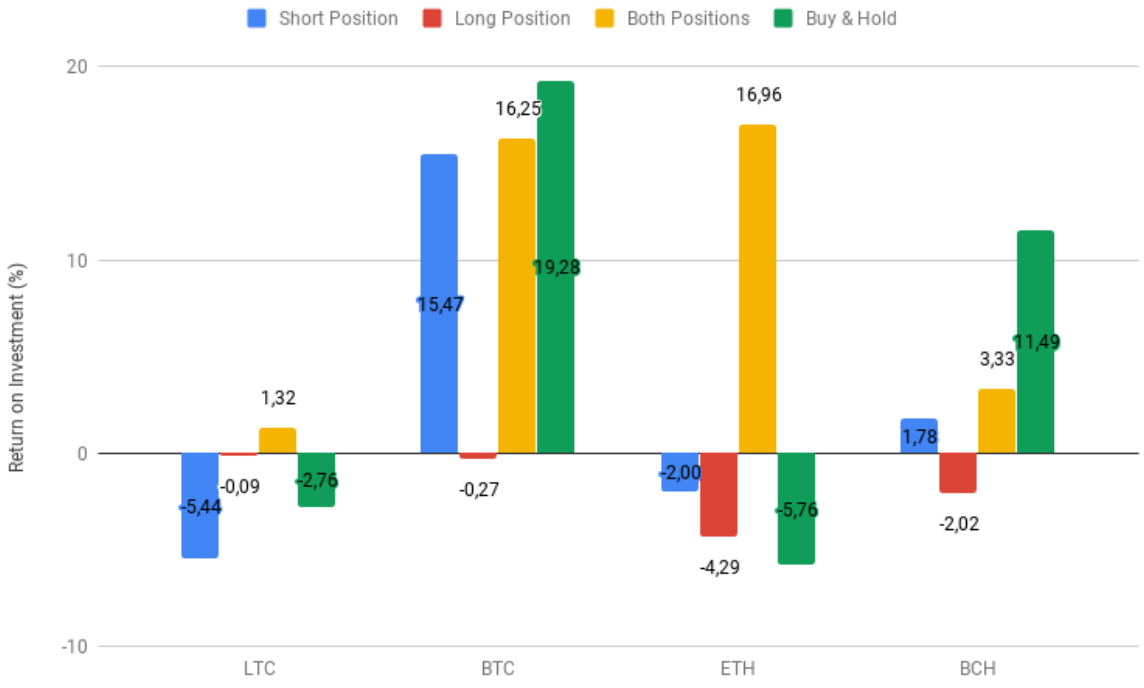

Figure 4.6: Second Test Case Results on July 2018

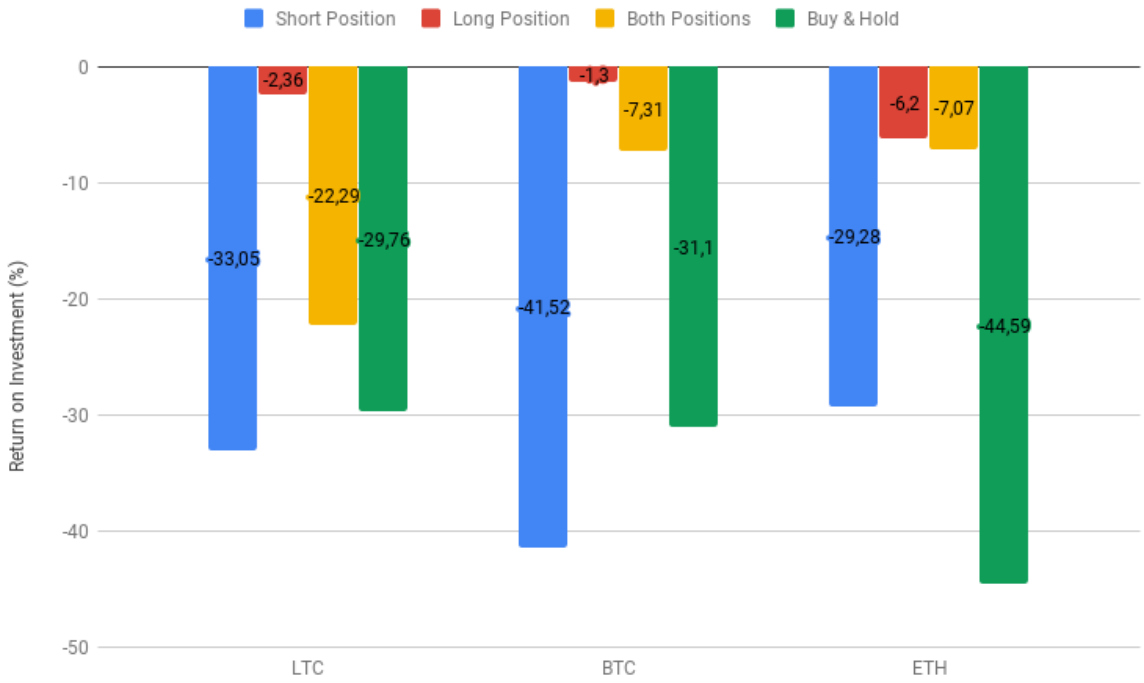

Figure 4.7: Second Test Case Results on November 2018

4.4.3 Outcome Graphs and Discussion

In this use case, we used total order book percentage inversions to extract periods to start positions with the goal of generating profit whilst comparing the results with the Buy & Hold method and the first test case. Analyzing the graph presented in Figure 4.6, we can conclude that while running with the two existing types of trades and the July 2018 data, this test case obtains profit on BTC and BCH unlike the Buy & Hold strategy. As the previous use case, the latter method has more profit with BCH and BTC. The type of trading strategy that provided the best outcome was using both position types since it allowed us to have profit with all four currencies. Comparing with the first use case, only in ETH while using both types of trades the results were considerably superior, a 33,24% increase in profit as shown in Figure 4.8.

Figure 4.7 depicts the results obtained after running our trading simulation with the November 2018 data. As the first case, we could not get profit in all currencies with the three types of trade mechanisms. Nonetheless, using long positions obtained a lower loss compared with the previous test case with the exception of ETH. Furthermore, for LTC, all results were better than those in the first use case. For BTC using short positions brought a more important loss but the other two trade configurations reduced the deficit obtained previously.

The worst test scenario for this use case is illustrated in Figure 4.10. We are in a similar case as the worst case initially presented as we are using only short positions on a steadily declining market which results in significant losses. On the other hand, Figure 4.9 shows a LTC simulation using both position types that managed to end the simulation with profit despite having a 10% loss before the 15th of July.

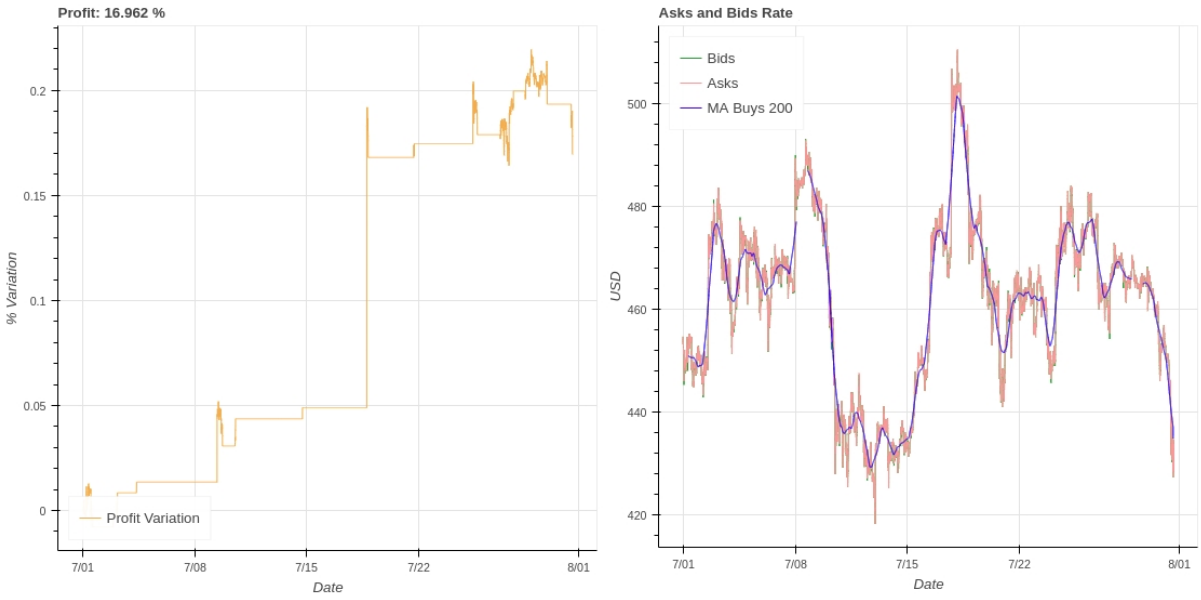

Figure 4.8: ETH Second Test Results Using Both Position Types on July 2018

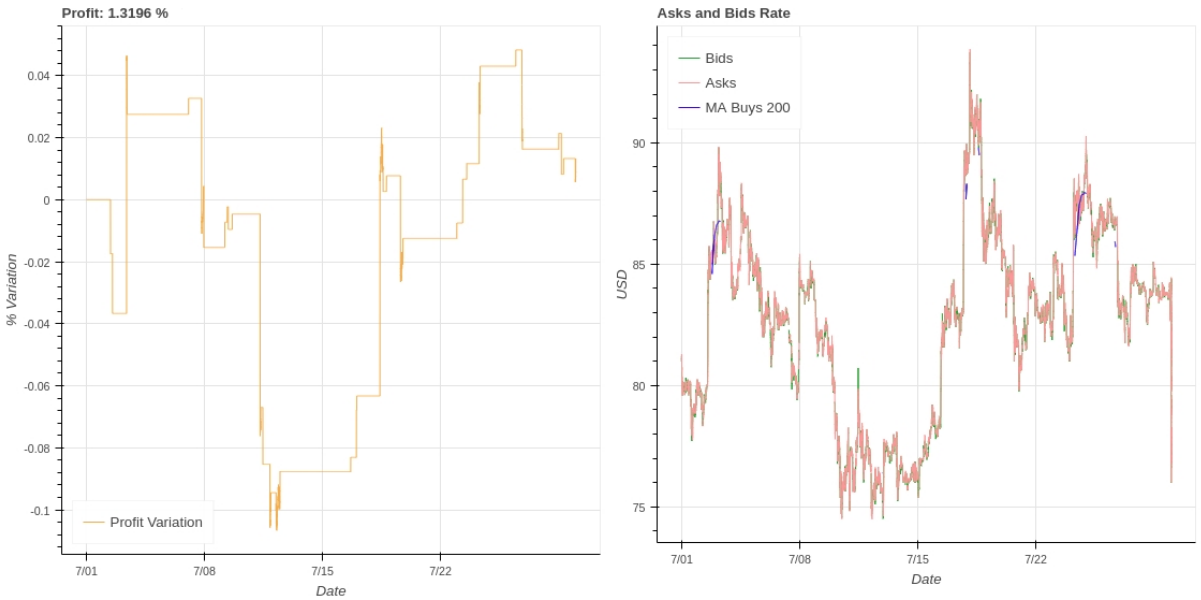

Figure 4.9: LTC Second Test Results Using Both Position Types on July 2018

4.5 Test Case 3 - Forecasting with ARIMA

The final test case uses ARIMA forecasting as a validation mechanism before opening positions in the simulations with the first two use cases. The projections will verify if the predicted market direction matches the type of position that we want to start. Lastly, this process combines all the generated periods into a single *DataFrame* that will serve as input to the simulation.

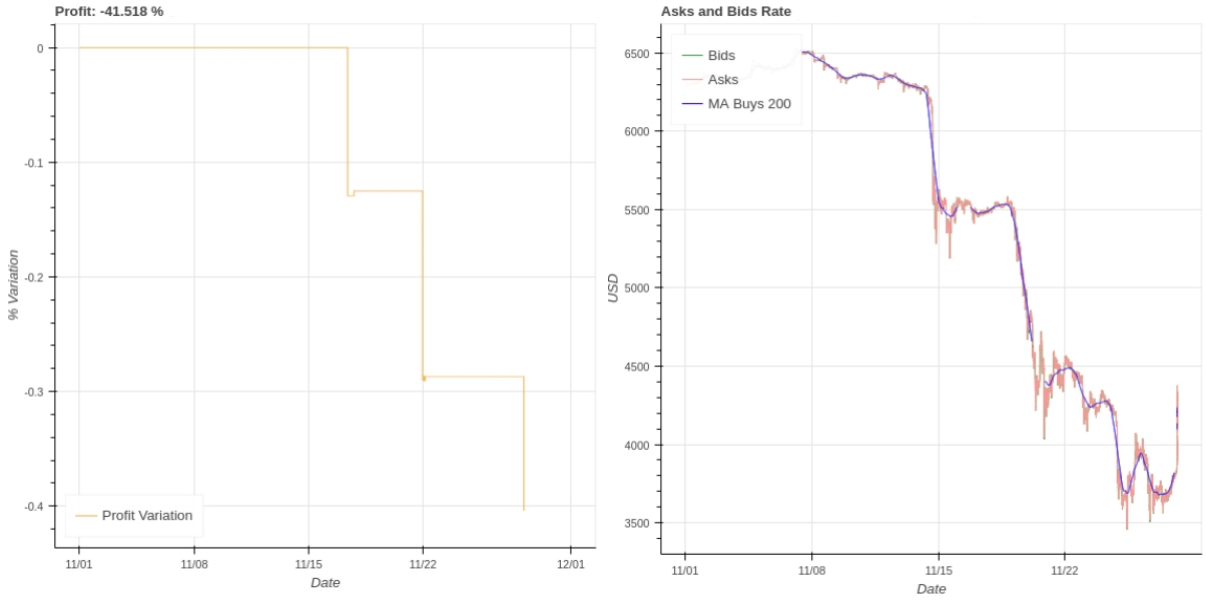

Figure 4.10: BTC Second Test Results Using Short Positions on November 2018

4.5.1 Scenario Setup

To test this use case, we started by taking each series containing the monthly trades of all the currencies from both July and November 2018 and applying a back fill to all missing values. Next, we defined our ARIMA model following the steps thoroughly explained in Chapter 3. After choosing the parameters for our mode, we decided to check the previsions made using a month of data, using 75% for training purposes and 25% for testing. We verified that the previsions were really distant from the real rates but we could gather and use information from a different perspective, using the direction of the forecast instead of the value itself. To check which direction the market is going, we compare our prediction to the last predicted value, if it is a positive number, we consider the rate should increase, other wise we will consider it will decrease. Table 4.6 presents the first five lines of a *DataFrame* example containing the forecasts made by our ARIMA model and the corresponding direction. We will save this information on CSV file and use this directions as an extra validation method before starting a new trading position on a simulation.

Table 4.6: First Five Lines of A Directions' *DataFrame*

date_begin	**predicted**	**delta**	**direction**
2018-11-23 12:00:00	4283.710894	0.000000	long
2018-11-23 12:05:00	4304.651584	20.940690	long
2018-11-23 12:10:00	4293.585618	-11.065966	short
2018-11-23 12:15:00	4273.753277	-19.832341	short
2018-11-23 12:20:00	4261.880173	-11.873104	short

4.5.2 Performance

Figure 4.11 illustrates the performance achieved by adding ARIMA previsions to the first use case in July 2018. We still considered three types of trade configurations in our algorithm: short, long and both

positions. Afterwards, we used the November 2018 data to simulate the behaviour of our trading method with the note that BCH was not considered due to the problems already mentioned. All the other test scenarios for both use cases generated the same results obtained previously.

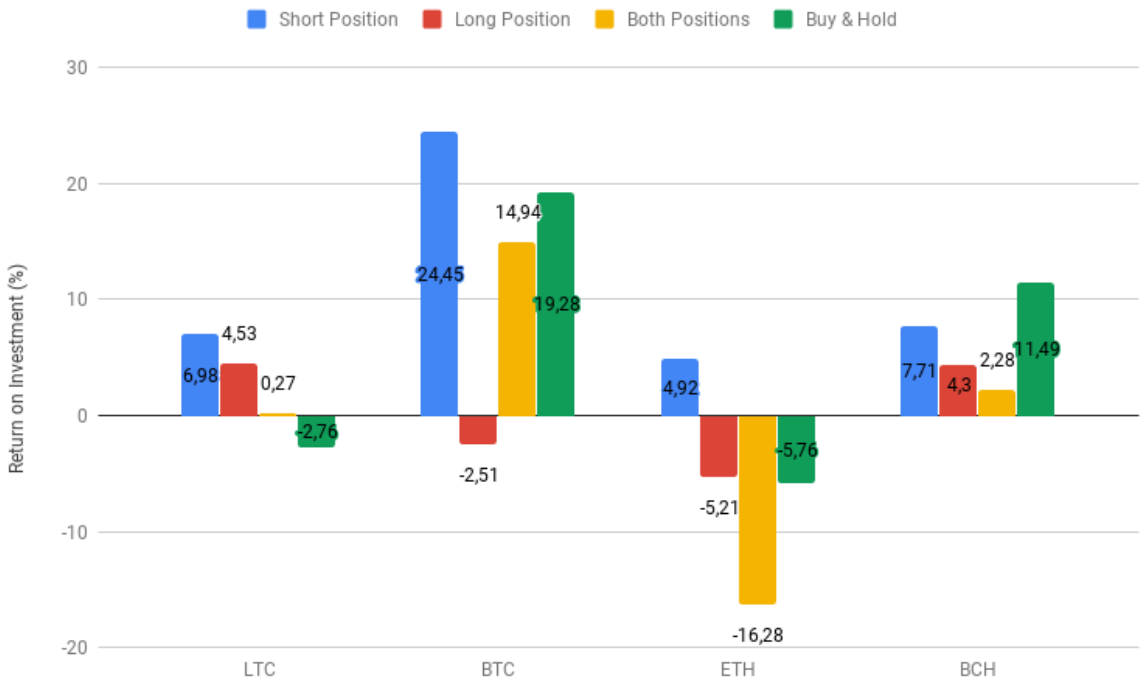

Figure 4.11: First Test Case Results With ARIMA on July 2018

4.5.3 Outcome Graphs and Discussion

This test case only generated different results (for BCH and BTC) when applied to the first use case on July 2018, the outcome is illustrated in Figure 4.11. Using only short trading positions in BTC generated the highest profit within all use cases (24,45%) and only two trades were needed, as presented in Figure 4.12. This result is superior to the 19,28% of profit gained by the Buy & Hold method. Contrarily, using both types of trades in BCH results in a 5,74% profit reduction compared to the first use case without forecasting. Figure 4.12 depicts the profit variation from this last test scenario. Unfortunately, this test did not generate a different outcome for the majority of the trade type / cryptocurrency pairs since we used three quarters of the month to train our ARIMA model so it could only be tested on the last week of the month. Obviously, if our trading algorithm does not take any action in the last week, our direction mechanism is unable to act.

4.6 Overall Results

Despite the lack of results of the last use case, the first two provided interesting outcomes that gave us a solid indication that there is value in extracting order book information and we can use it to improve

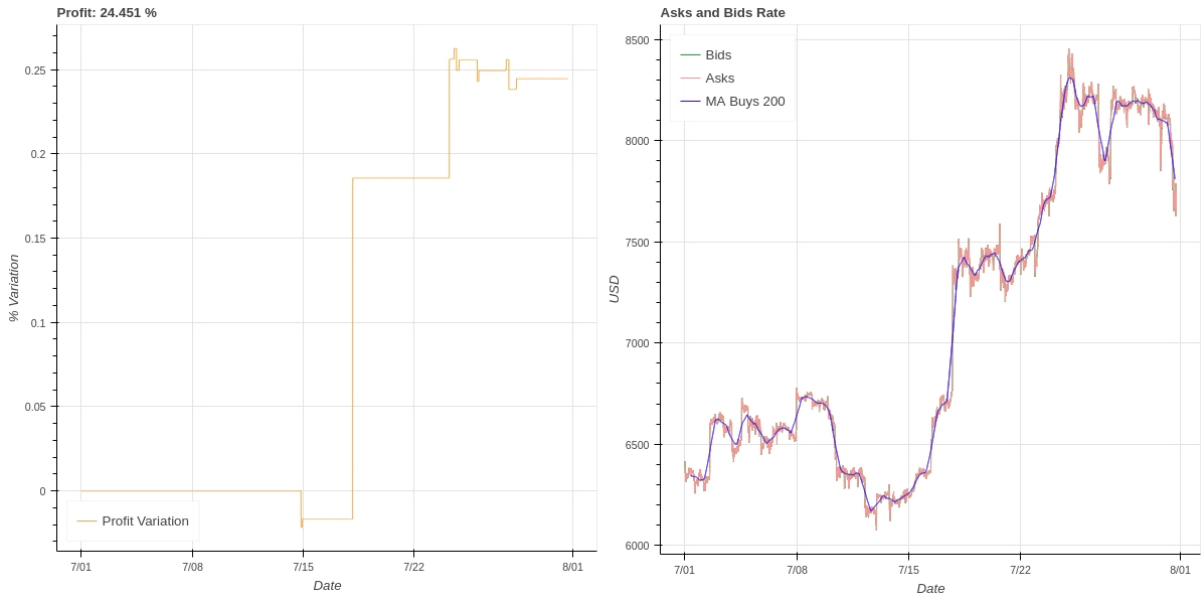

Figure 4.12: BTC First Test Results Using Short Positions and ARIMA on July 2018

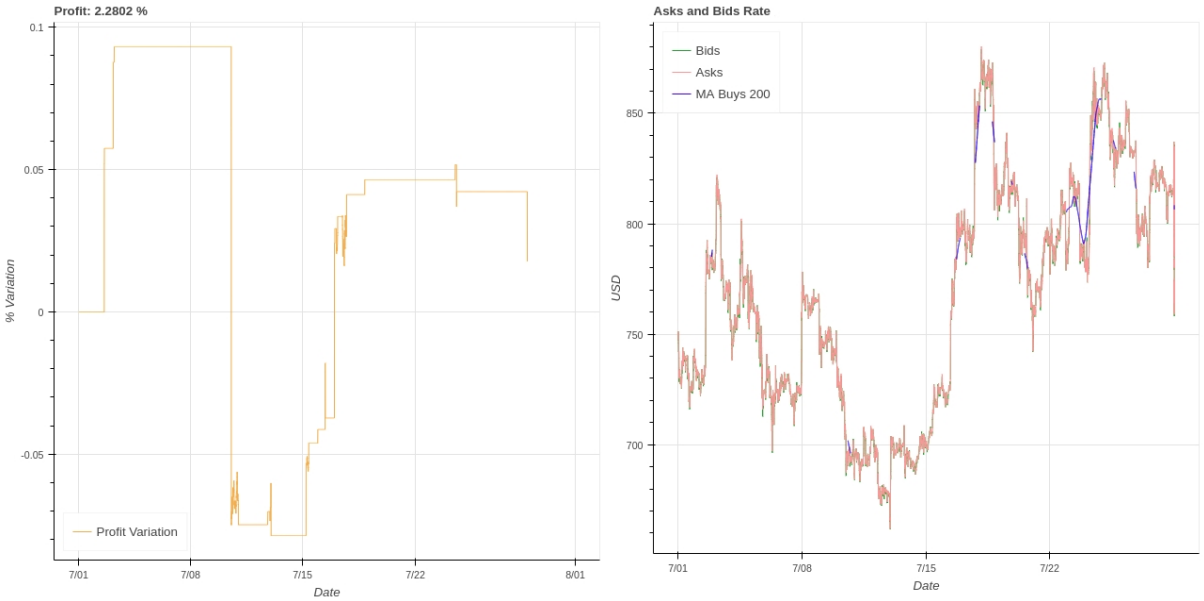

Figure 4.13: BCH First Test Results Using Both Position Types and ARIMA on July 2018

profits. Before going through the positive aspects, let us address the major problems that occurred during the evaluation phase. First of all, the time needed to extract data from the database and aggregate it because it made us consider monthly test periods which is not suitable for a trading algorithm. Additionally, we should have preemptively verified the information integrity to avoid situations like the inability to execute a simulation with BCH on the November 2018 data.

Observing the best performing trading methods using the July 2018 data in Table 4.7, we can infer that only establishing short positions with the first use case generates profit for all four digital currencies. Furthermore, using ARIMA for forecasting allows to improve our most profitable situation (short trades on BTC) by 6,47 %. Nonetheless, the third test scenario slightly reduced the gains with BCH while compared with the first use case. Measuring the efficiency of our strategies against the Buy & Hold method,

we can verify that in the first and second use cases we achieved better results in two cryptocurrencies whilst in the last scenario we generated more profit for LTC, BTC and BCH.

Table 4.7: Best Performing Trading Strategies For July 2018

Currency	1st Use Case	2nd Use Case	3rd Use Case	Buy & Hold
LTC	6,98 % (Short)	1,32 % (Both)	6,98 % (Short)	-2,76 %
BTC	17,98 % (Short)	16,35 % (Both)	24,45 % (Short)	19,28 %
ETH	4,92 % (Short)	16,96 % (Both)	16,96 % (Both)	-5,76 %
BCH	8,53 % (Short)	3,33 % (Both)	7,71 % (Short)	11,49 %

Table 4.8 presents the outcome of the most performing trading strategies for November's 2018 data. Two notes must be addressed initially, the first one is why all the profit percentages are negative whilst the second is related to not having both third use case and BCH results. The latter remark is justified by a problem regarding the order book retrieval from the digital asset's store API since some of them were empty. We can explain the negative profits by visualizing the market trend on November 2018 because the majority of cryptocurrencies suffered significant price drops as shown by the right side plot of Figures 4.5 and 4.10. In spite of these problems, our developed use cases managed to drastically reduce losses if compared to the Buy & Hold strategy results while using only long positions. This fact may feel strange since we are dealing with a month where the general market tendency was to decrease the rate of digital currencies but it is a proof that the PSO acts efficiently even in challenging periods.

Table 4.8: Best Performing Trading Strategies For November 2018

Currency	1st Use Case	2nd Use Case	Buy & Hold
LTC	-5,95 % (Long)	-2,36 % (Long)	-2,76 %
BTC	-2,54 % (Long)	-1,30 % (Long)	-31,10 %
ETH	-2,05 % (Long)	-6,20 % (Long)	-44,59 %

The results show that the first use case should be used in months with a bull market, which can be described as a period of generally rising prices, using only short positions. Contrarily, the second test scenario using only long positions should be select for periods with a bear market, where a general decline in the digital currencies rate happens over a period of time. An important aspect of the developed algorithm is the ability to overcome opposite market directions as illustrated in the left side plot of Figures 4.4, 4.9 and 4.13. The last test case did not produce enough data for us to assess if using the ARIMA model improves or reduces the profitability of the initial use cases. Nonetheless, the fact that we were only able to test two periods reinforces that these conclusions should be further tested to check that the outcome produced is consistent with our remarks.

Finally, this evaluation phase offered innumerable challenges and objections but we ended up creating three use cases that, in spite of needing additional examinations, substantiate our objective of extracting information from the order books of digital currencies and use machine learning to improve profits.

Chapter 5

Conclusions

5.1 Summary and Achievements

The presented work proposes an end to end system designed in a microservice architecture that contains a database with order books and trading information of the four major digital currencies (more than 70 Gigabytes of data), an algorithm that pre-processes and aggregates information from order books and trades to define trading positions, a trading simulator, a new type of trailing stop that optimizes profits and the usage of forecasting to check the market trend.

Three use cases were developed for our trading algorithm, the first two extract order book data to find suitable trading periods whilst the last one uses the ARIMA model with a 70% training and 30% testing periods to predict cryptocurrencies' rates and use that information on the initially mentioned test scenarios. We used monthly test periods and considered three trading configurations to find the most performing strategy according to the market's situation.

In addition to the trading algorithm and simulation, the output generated for all test scenarios allows the end user to easily check what happened in the simulation with an HTML dynamic report.

All in all, despite the reduced testing sample and innumerable challenges faced, the results were promising, surpassing the Buy & Hold strategy gains and significantly reducing losses on bear markets.

5.2 Future Work

In this section, various limitations of the developed solution are focused with enhancements that should be implemented to mitigate these restraints and improve the overall system performance.

- The information extracted to define our use cases (maximum monthly volume, total percentage order book inversions) was gathered from the month that will be used in our simulation so if we want this algorithm to be used in real time, the first step is to change this to use data from previous periods;
- Establish data integrity mechanisms when gathering trades and order book information to alert if

there are any problems;

- Longer and more test periods to validate if the results produced by our use cases are consistent;
- Test the EUR market data and compare the results with the ones obtained on the USD market;
- Feed more data to the ARIMA model to improve the precision of previsions;
- Compare the generated results with more standard trading strategies;
- The five minute time window chosen must be reduced to allow a better functioning of the PSO;
- Create a NARX model and verify if it improves the forecasts made for the third use case.

Bibliography

[1] S. Nakamoto: Bitcoin: A Peer-to-Peer Electronic Cash System (2008)

[2] O. J. Mandeng: Cryptocurrencies, monetary stability and regulation: Germany's Nineteenth Century Private Banks of Issue. Technical report, LSE Institute of Global Affairs (2018)

[3] J. Fleming, C. Kirby and B. Ostdiek: The economic value of volatility timing using "realized" volatility. Journal of Financial Economics (2003)

[4] P. K. Jain, P. Jain and T. H. McInish: The Predictive Power of Limit Order Book for Future Volatility, Trade Price, and Speed of Trading. SSRN Electronic Journal (2011)

[5] M. Crosby, Nachiappan, P. Pattanayak, S. Verma and V. Kalyanaraman: BlockChain Technology - Beyond Bitcoin. Technical report, University of California Berkeley (2015)

[6] J. P. Podolanko, J. Ming and M. Wright: Countering Double-Spend Attacks on Bitcoin Fast-Pay Transactions (2017)

[7] G. O. Karame, E. Androulaki and S. Capkun: Double-spending fast payments in Bitcoin. In: Proceedings of the 2012 ACM conference on Computer and communications security (CCS '12). (2012)

[8] K. Okupski: Bitcoin Developer Reference (2016)

[9] I. H. Witten and E. Frank: Data Mining : Practical Machine Learning Tools and Techniques. 4th edn. Morgan Kaufmann (2005)

[10] M. D. Rechentin: Machine-learning classification techniques for the analysis and prediction of high-frequency stock direction. PhD thesis, University of Iowa (2014)

[11] J. Fleming, C. Kirby and B. Ostdiek: Naive Bayesian Classification of Structured Data. Machine Learning **57**(3) (2004) 233–269

[12] M. Garofalakis, D. Hyun, R. Rastogi and K. Shim: Building Decision Trees with Constraints. Data Mining and Knowledge Discovery **7**(2) (2003) 187–214

[13] J. Menyhart and R. Szabolcsi: Support Vector Machine and Fuzzy Logic. Acta Polytechnica Hungarica (2016)

[14] G. Box and G. Jenkins: Time Series Analysis, Forecasting and Control. Wiley (1970)

[15] Sean McNally: Predicting the price of Bitcoin using Machine Learning. Technical report, School of Computing National College of Ireland (2016)

[16] J. L. Elman: Finding Structure in Time. Cognitive science **14**(2) (1990) 179–211

[17] R. Frigola: Bayesian Time Series Learning with Gaussian Processes. PhD thesis, University of Cambridge (2015)

[18] G. Bontempi, S. Ben Taieb and Y.-A. Le Borgn: Machine Learning Strategies for Time Series Forecasting. Business Intelligence: Second European Summer School (2013) 62–67

[19] M. Haferkorn and J. M. Quintana Diaz: Seasonality and Interconnectivity Within Cryptocurrencies - An Analysis on the Basis of Bitcoin, Litecoin and Namecoin. Lecture Notes in Business Information Processing (2015) 106–120

[20] A. Hencic and C. Gouriéroux: Noncausal Autoregressive Model in Application to Bitcoin/USD Exchange Rates. Econometrics of Risk (2015)

[21] J. Perdiguero: Exploring the determinants of Bitcoin's price: an application of Bayesian Structural (2017)

[22] M. Ordon, D. Urbach, M. Mamdani, R. Saskin, H. D'A and K. Pace: The surgical management of kidney stone disease: A population based time series analysis. Journal of Urology (2014) 1440–1456

[23] P. Cortez, M. Rio, M. Rocha and P. Sousa: Multi-scale Internet traffic forecasting using neural networks and time series methods. Expert Systems (2012) 143–155

[24] R. H. Shumway and D. S. Stoffer: Time Series Analysis and Its Applications: With R Examples. Springer (2011)

[25] Z. Boussaada, O. Curea, A. Remaci, H. Camblong and N. M. Bellaaj: A Nonlinear Autoregressive Exogenous (NARX) Neural Network Model for the Prediction of the Daily Direct Solar Radiation (2018)

[26] A. Lasfer: Performance Analysis of Artificial Neural Networks in Forecasting Financial Time Series (2013)

[27] T. Guo and N. Antulov-Fantulin: Predicting short-term Bitcoin price fluctuations from buy and sell orders (2018)

[28] R. J. Hyndman and G. Athanasopoulos: Forecasting: principles and practice. OTexts (2014)

[29] P. R. Hansen and A. Lunde: A forecast comparison of volatility models: does anything beat a Garch(1-1)? Jornal of Applied Econometrics (2005) 873–889

[30] J. H. Friedmann: Greedy function approximation: a gradient boosting machine. Annals of Statistics (2001) 1189–1232

[31] Y. Liu, A. Niculescu-Mizil, A. C. Lozano and Y. Lu: Learning temporal cause graphs for relational time-series analysis. ICML (2010) 687–694

[32] Y. Wu, J. M. Hernandez-Lobato and Z. Ghahramani: Gaussian process volatility model. NIPS (2014) 1044–1052

[33] J. Donier and J.-P. Bouchaud: Why Do Markets Crash? Bitcoin Data Offers Unprecedented Insights. PLOS ONE 10 (2015) 1–11

[34] Y. B. Kim, J. G. Kim, W. Kim, J. H. Im, T. H. Kim, S. J. Kang and C. H. Kim: Predicting Fluctuations in Cryptocurrency Transactions Based on User Comments and Replies. PLOS ONE 11 (2016)

[35] D. Garcia, C. Tessone, P. Mavrodiev and N. Perony: The digital traces of bubbles: feedback cycles between socio-economic signals in the Bitcoin Economy. Journal of The Royal Society Interface 11 (2014)

[36] M. Amjad and D. Shah: Trading Bitcoin and Online Time Series Prediction. IPS 2016 Time Series Workshop (2017) 1–15

[37] D. Jaramillo, D. V. Nguyen and R. Smart: Leveraging microservices architecture by using Docker technology. SoutheastCon (2016)

[38] L. Kumar, S. Rajawat and K. Joshi: Comparative analysis of NoSQL (MongoDB) with MySQL Database". International Journal of Modern Trends in Engineering and Research (2015) 120–127

[39] Y. W. Cheung and K. S. Lai: Lag Order and Critical Values of the Augmented Dickey-Fuller Test". Journal of Business & Economic Statistics (1995) 277–280

[40] B. Everitt: The Cambridge Dictionary of Statistics. UK New York: Cambridge University Press (1998)

[41] Jiarui Ni and Chengqi Zhang: An Efficient Implementation of the Backtesting of Trading Strategies. ISPA 2005: Parallel and Distributed Processing and Applications (2005) 126–131

Appendix A

Docker-Compose For Data Scrapping

```
version: "2"
services:
  mongo:
    container_name: mongo_db
    restart: always
    image: mongo
    volumes:
      - ./data:/data/db
    ports:
      - "27017:27017"
  python:
    container_name: python_app
    restart: always
    build: ./orders_scrapper
    depends_on:
      - "mongo"
    links:
      - mongo:mongo
  node:
    container_name: node_app
    restart: always
    build: ./trades_scrapper
    depends_on:
      - "mongo"
    links:
      - mongo:mongo
```