CM3203

Data Analysis of the Bitcoin Blockchain

Kieren Spear – C1535417

Table of Contents

List of Figures

List of Tables

Abstract

The emergence of Virtual, or Crypto-Currencies, based on a distributed-ledger or blockchain technology, has led to many forecasts of an imminent revolution across the financial sector[1] and beyond[2]. However, despite the increasing evangelism and significant flows of investment capital into these technologies, a number of obstacles to widespread adoption remain[3].

National and supra-national agencies globally continue to struggle to legislate for increased anonymity within such systems, and significant energy costs of maintaining the blockchain leads to node concentration, and therefore existential concerns within more mature networks. Recent academic analysis of such concerns conclude that a new ecosystem of blockchain analytics platforms might be the solution, as revenues generated by such platforms could incentivise increased active-node participation across blockchain networks.

This project explores the scope for such solutions by analysing the Bitcoin network, the first and largest blockchain-based system, and developing a prototype of a Bitcoin analytics platform. The developed platform displays balance data for Bitcoin users, along with information relating to transaction size and fees donated to Bitcoin miners, by parsing and analysing the data stored within the first 140,000 blocks of the Bitcoin blockchain which represents around one-third of blocks at the outset of the project.

1 Introduction

This section of the report will explain the purpose of the project in some detail to allow for a better understanding of what I intend to achieve within this project. I will be delving deeper into the aims and goals that I have set myself from the beginning of the project and explain how I set to achieve these goals. I will also be providing details on the scope of the project, intended project audience, assumptions made and a summary of the important outcomes. This information should provide more than enough information on what I set out to achieve throughout the duration of the project and should make the background reading section of this report much easier to follow and see its relevance to the project.

1.1 Aims and Goals

The main aim of this project was to develop a prototype of a Bitcoin analytics platform that would enable users of the platform to see various information on the first 140,000 blocks of the Bitcoin Blockchain. This information includes the likes of displaying balance data for users, information relating to transactions and fees donated to Bitcoin miners. The purpose of developing a platform for this data is to see if a new eco-system of blockchain analytics platforms could lead to an increase in active-node participation across blockchain networks. The reasoning behind only parsing the first 140,000 blocks is for the sake of the speed of the Bitcoin Analytic platform. At the time of writing this report, 140,000 blocks represents just over one quarter of the current amount of blocks in the Bitcoin blockchain.

Another goal of this project was to research a number of possible ways that this project could be completed. This would include researching what I need to do in order to have a foundation to build upon to develop a Bitcoin analytics platform, researching a few APIs which could be helpful and choosing how best to integrate any information retrieved into a user-friendly platform.

The third aim of this project was to make the Bitcoin analytics platform as bug-free and user friendly as possible to ensure the best possible experience whilst using the platform. This would be done in numerous ways, the first being to make the platforms navigation intuitive around the platform by keeping simple and well-known interactions with the platform. Graphs that are used within the platform will be clear and easy to read so that users are not lost when reading graphs. Core functionality of the platform is also extremely important as it makes for a more complete user experience.

1.2 Approach

From the very beginning of the project I knew that the approach I would use would be based from the Agile Software Development Model. This approach would allow me to be flexible with the system and design requirements due to the state of the system constantly changing as new challenges and goals would arise throughout the scope of the project.

At the beginning of the project I only had a basic understanding of how the Bitcoin Blockchain works so choosing to follow an agile development approach whilst undertaking the project best suited me as I knew I would run into new challenges throughout the course of the development of the system.

This approach to the project would be much more effective than the likes of the Waterfall Model due to the structured nature of the Waterfall Model. From the beginning of the project I knew that I didn't have a clear picture of how the Bitcoin blockchain analytics platform would turn out, so it would have been impossible and unreasonable to follow an approach that lacks the ability to change over time.

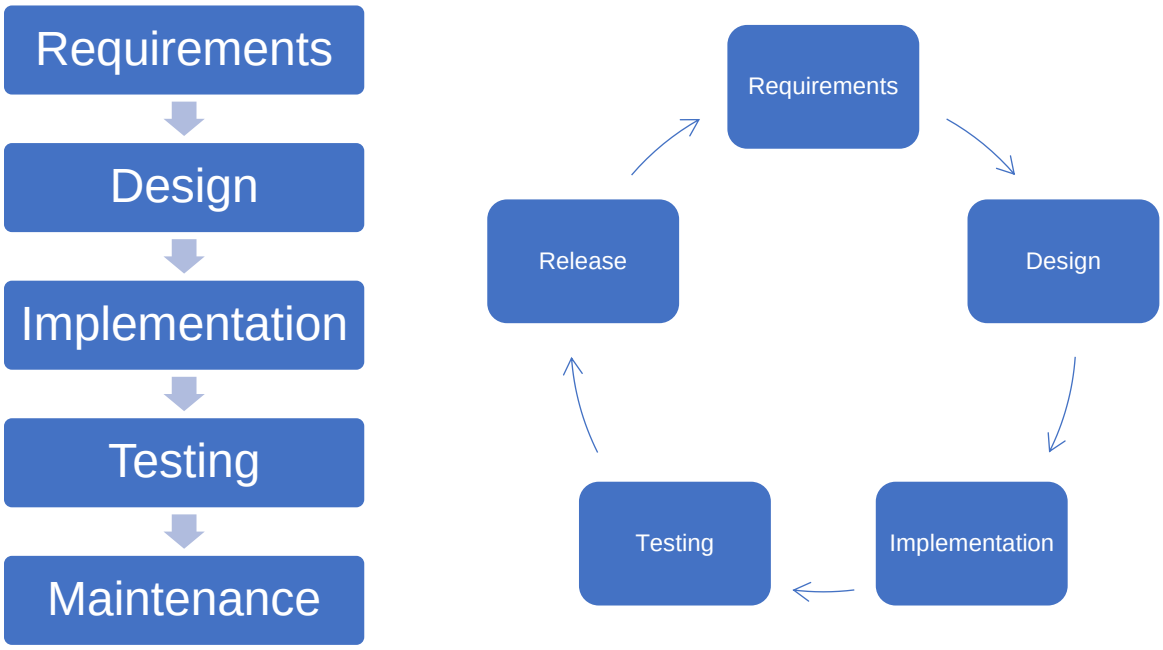


Figure 1: Waterfall Model vs Agile Model

1.3 Project Scope

Throughout the project there were several sub-goals that I set for myself to hit and achieve. This section of the report will detail those sub-goals as well as explaining the approach I took to achieve these goals.

The first major step I set out to achieve for this project was to fully research the Bitcoin blockchain, first at a technical level, then progress to a code-level of understanding which I will show later within the report. I achieved this step by first researching what a blockchain is, followed by how specifically the Bitcoin blockchain

works. Due to crypto-currency being a fairly new technology the best place to research about these were online.

The second step of the project I sought to achieve was to research how the code-level of analysis of the Bitcoin blockchain could be best implemented into a platform-based system to reach the final goal. The approach I took to achieve this step was to first learn how to extract data from the Bitcoin blockchain so that I was able to analyse and see what I had to work with. By this time, I knew that the programming language I would be using was Python 3.0, from here I was able to search for platforms that would allow me to use Python as the foundation when moving forward with this project.

1.4 Intended Audience

This project may be of interest to those that have an interest in crypto-currency technology or possibly to those who have an interest in the financial aspect of crypto-currency rather than the technology behind it. As mentioned above, the end-goal of this project was to develop a Bitcoin analytics platform, which could be of interest to a number of users e.g. general bitcoin users and organisations that are trying to learn more about the technology behind crypto-currency.

1.5 Assumptions

An assumption I will make for the purpose of reading is to assume that the reader is familiar with general purpose programming languages and some of the basic libraries. I shall also assume that the reader has a very low level of knowledge on the blockchain and how it works, specifically with the Bitcoin Blockchain.

2 Bitcoin’s Background

Within this section of the report I will be providing relevant background information that will help with the understanding of Bitcoin and its blockchain. I shall be avoiding long descriptions of details, so I will be providing references that you can follow to do further reading on the areas I will be mentioning.

2.1 Crypto-Currency Challenges

Before Bitcoin, there were a couple of major concerns with how crypto-currency worked, one being the Double Spending problem, and the other being Byzantine Fault Tolerance.

Double Spending[4] – This concern is associated with crypto-currencies that use a decentralized system. This problem arises because the digital files that are used within the crypto-currency system are easily replicated and sent back into the system. If a double spend is currently occurring within the decentralized system, the system can have trouble recognizing which of the transactions is the original transaction. If a system can do this, then double spending would not be effective due to duplicate transactions being rejected. This problem does not arise with Bitcoin due to a mechanism based on transaction logs within blocks to verify the authenticity of each transaction, preventing double spending to occur.

Byzantine Fault Tolerance[5] – This problem can happen if there is a conflict in agreement between nodes within a network, or in this case a blockchain. As the blockchain is a decentralized system, if there were to be a fault in the system, nothing can be done to correct illegitimate transactions if they were to be accepted. This would greatly reduce the trust within a crypto-currency if this was able to happen. The way the Bitcoin Blockchain addresses this problem is through a concept named Proof of Work (PoW). The main idea behind this is that in order to successfully validate an illegitimate transaction within the blockchain, at least 51% of computational power would be required in order to do so.

2.2 The Background of Bitcoin

This section will contain some of the fundamental details on Bitcoin, including the evolution of Bitcoin and how Bitcoin works.

2.3 Bitcoin’s Beginning

The first known attempt at creating a crypto-currency platform was created in the late 1980’s where ‘smartcards’ were given to truck drivers in the Netherlands instead of cash due to the amount of robbing that was occurring in rural areas at night. In the 1990’s, the concept of crypto-currency first evolved when DigiCash was founded. This platform would be the first platform that would have anonymous transactions.

Since then many crypto-currencies have been tried and tested however all ran into problems when trying to create a fully decentralized crypto-currency platform[7].

In October 2008 a paper titled “Bitcoin: A Peer-to-Peer Electronic Cash System” was posted to a cryptography mailing list. The author of this paper is under a pseudonym of “Satoshi Nakamoto”, and it is still unknown who the person(s) behind the name is. The post that was posted to the cryptography mailing list included a link to the widely known paper that details the workings of Bitcoin[6].

Within the previously mentioned paper, Satoshi Nakamoto had provided a solution to the problems that would occur within previous crypto-currencies by introducing an **“electronic payment system based on cryptographic proof instead of trust, allowing any two willing parties to transact directly with each other without the need for a trusted third party”** and that **“a solution to the double-spending problem using a peer-to-peer distributed timestamp server to generate computational proof of the chronological order of transactions”**. He also goes onto mention that **“the system is secure as long as honest nodes collectively control more CPU power than any cooperating group of attacker nodes”**.

2.4 Idea Behind Bitcoin

As previously mentioned, Bitcoin works using a decentralized system where no single person can control the entire Bitcoin network, this is known as the Blockchain. Users are incentivised to become part of the network so that the network becomes more secure by being rewarded Bitcoin for using their computational power (more detail on this later).

One major difference between fiat currencies, like the pound and dollar, is that they have full reign over how much of that currency can be in circulation and if they wanted to print more money then they are able to do so. Bitcoin on the other hand, has a limited amount of Bitcoin that will ever exist, which is 21,000,000 Bitcoin (BTC). The supply of Bitcoin is very tightly controlled using an underlying algorithm. New Bitcoin is made by ‘mining’ blocks that exist within the blockchain. These blocks contain important information on transactions (continued later). Once the 21,000,000 BTC limit has been reached, other methods for incentivising Bitcoin nodes and miners will be required[9].

Bitcoin or more commonly Satoshi's (one hundred millionth Bitcoin) can be spent by sending the amount from one unique wallet address to another. There are different types of wallet that a person can have, the first being a local wallet where the user must download the full Bitcoin Blockchain so that it can appear as a node within the network and place any transactions that come from the wallet and place the transaction information directly into the blockchain. The easier way and more widely used method is to use a third-party system or website to open a wallet for the user so that they don't have to act as a node on the network. The trouble with this is that the wallet's data is stored on third-party servers which the user has no control over.

Another difference between Bitcoin and fiat currencies is that the senders of transactions that include fiat currencies are normally identified. This is not the case with Bitcoin as users operate in semi-anonymity due to a decentralized system that does not require users to identify themselves in order to make a transaction. The transaction will be confirmed once all previous transactions have been verified so that the sender has enough Bitcoin to make the transaction as well as the right authentication to send them. The reason that Bitcoin is not fully anonymous is that users can be somewhat identified by their wallet's address, and some methods have been successful in identifying users[8].

Transactions through Bitcoin are unable to be reversed, unlike that of electronic fiat currencies. This is because when using a decentralized system there is no single node that can decide whether to validate or reverse a transaction.

As briefly mentioned, the smallest amount of Bitcoin that can be spent is called a 'Satoshi'. The value of a Satoshi is one hundred millionth of a Bitcoin (0.00000001) which is incomparable to that of fiat currency. At the time of writing this report, one Satoshi is currently valued at £0.000058 and where 1BTC is worth £5833.49.

2.5 Bitcoin Mining

As previously mentioned, the total amount of Bitcoin that will ever exist is exactly 21,000,000. At the time of writing this report, the current number of Bitcoin in circulation is nearly 17,000,000. This means that just over 4,000,000 Bitcoin is still left to be 'released' into circulation.

As Bitcoin is a decentralized system, nobody has the power to create new Bitcoin unlike that of a fiat currency. This means that a different method must be used in

order to release new Bitcoin into circulation. This is done through ‘mining’ Bitcoin where users are rewarded Bitcoin for using their computational power to create blocks of transactions that have been validated and released onto the Bitcoin blockchain[10].

The mining of the Bitcoin is completely guess work. Each block within the blockchain has a hash function which makes it impossible to match the output without having to guess the ‘nonce’ of the block. The nonce and the previous block header are then hashed together to provide another hash number. If this number satisfies the pre-established number of zeroes for the block, then the block has been successfully mined.

Once the block has been mined, a reward of Bitcoin will be given to the user that mined the block. The value of this reward is pre-determined by how many Bitcoin is currently in circulation. The starting block reward was 50 BTC, however this is halved every 210,000 blocks that have been mined so at the time of writing this report the current block reward is 12.5 BTC.

As the current number of Bitcoin tends towards the total number that will be in circulation the difficulty of mining a block increases. This means that it will take longer for blocks to be solved if the computational power used is the same.

2.6 Bitcoin Block Transaction Details

All transaction details are included in blocks within the blockchain so that upon verification of a transaction, the block is able to confirm that the sender of the Bitcoin has a sufficient Bitcoin amount within their Bitcoin address to make the transaction.

The details that are included within the blocks are the version number, total amount of Bitcoin transacted, list of inputs, list of outputs and the lock time of the transaction. The list of inputs and the list of outputs are what is most important in the context of this project.

Input – The input of a transaction references the output of the previous transaction, where transactions usually include multiple inputs. The total amount of Bitcoin sent within a transaction is calculated by adding up the individual inputs that are stored inside of the transaction. The script signature is also stored within the transaction’s input information which is needed to verify that the transaction was made by the owner of the account[11].

Output – The output of a transaction details the instructions for the sending of Bitcoin between addresses. The output contains a value of Satoshi that will be given if the output is claimed, which will always happen if 2 real Bitcoin addresses are used. Multiple outputs are also possible where they share the combined value of the inputs. Any input that has not been redeemed by the recipient's address is collected by the miner who created the block. This is considered as a transaction fee[11].

2.7 Bitcoin Transactions Between Wallets

The first thing that is required is for both people to have a wallet which can be locally stored or provided by a third-party as previously mentioned in the report. A Bitcoin wallet holds a unique Bitcoin address which is used to store a record of all transactions. From this information we can gather the balance of the Bitcoin address. The address has a long value of 34 which is comprised of numbers and letters, this is known as the 'public key' for the specific address. For every valid public key that exists, there must also co-exist a 'private key' which has a length of 64 numbers and letters[13].

As the value of public keys are readily available on the Bitcoin blockchain, it is extremely important that the corresponding private keys to these addresses are kept secret and secure for the security of the Bitcoin that the address holds. This is because any transactions that are sent from the public key require a signature from the private key so that a transaction can be validated. However, a private key isn't the only thing that is needed for validation; the public key must have enough Bitcoin associated with it to make the transaction take place. Once the nodes within the network have confirmed that the Bitcoin address has enough Bitcoin to make the transaction and that the digital signature of the transaction is correct, the transaction can take place between the addresses.

For a more detailed explanation an article named "Bitcoin Private Keys: Everything You Need To Know"[12].

2.8 Relevancy of Crypto-Currency in Real-Life

Now that most of the technical side of Bitcoin and the technology behind it has been covered for the understanding of the project, it is now important to see some of both the positive, and negative impacts that this upcoming technology can have outside of computing.

Crypto-currencies have been heavily associated with the Dark Web[14] – a section of the web that is not accessible when using normal search engines. This association has given crypto-currency a bad name as it is the preferred method of payment due to the semi-anonymity for many illegal services that exist within the Dark Web. A concern of crypto-currencies in this regard is that as they get more popular, an increase in cyber related crime may occur.

People's finances have also been affected through the trading of crypto-currency. There has been an enormous increase in the number of users of Bitcoin over the past few years with exchanges being busier than ever in recent times. As with trading stocks and shares, trading with crypto-currency definitely has high risk high reward potential, however this means there will always be a winner and a loser financially speaking.

Money laundering through crypto-currencies is a major concern amongst governing bodies due to things like tax evasions, which can make a country economically weaker. Because of such concerns and the uncontrollable and unpredictable nature of crypto-currency, governments and banks alike are cautious about the adoption of the technology.

2.9 Relevancy of the Project

For crypto-currencies to be fast and reliable, node participation within the decentralized network is very important. As the popularity of crypto-currencies increases so does the required computational power so that transaction speeds can stay the same. The more participating nodes that are active within the blockchain the faster the validation of transactions can happen, it also brings an added level of security to the crypto-currency's network[15].

The purpose of this project as mentioned in the second paragraph of the abstract is to explore the solution of node concentration as the increased popularity of crypto-currency rises. One solution of this problem is to give users an incentive to become an active node within the network thus decreasing node concentration and therefore improving the speed and security of transactions.

This project explores how an uprising in Bitcoin Analytic Platforms may provide the users with the incentive to participate as an active node. At the time of writing this report, there is a total of there are currently 9880 nodes running on the Bitcoin

network[18]. Figure 2[18] shows the amount of active-node participation using the Bitcoin Core client which makes up for over 90% of total active node participation[19]. The recent value of Bitcoin may have been the cause of such an increase in active-node participation however analytic platforms may have also played a part through increased popularity.

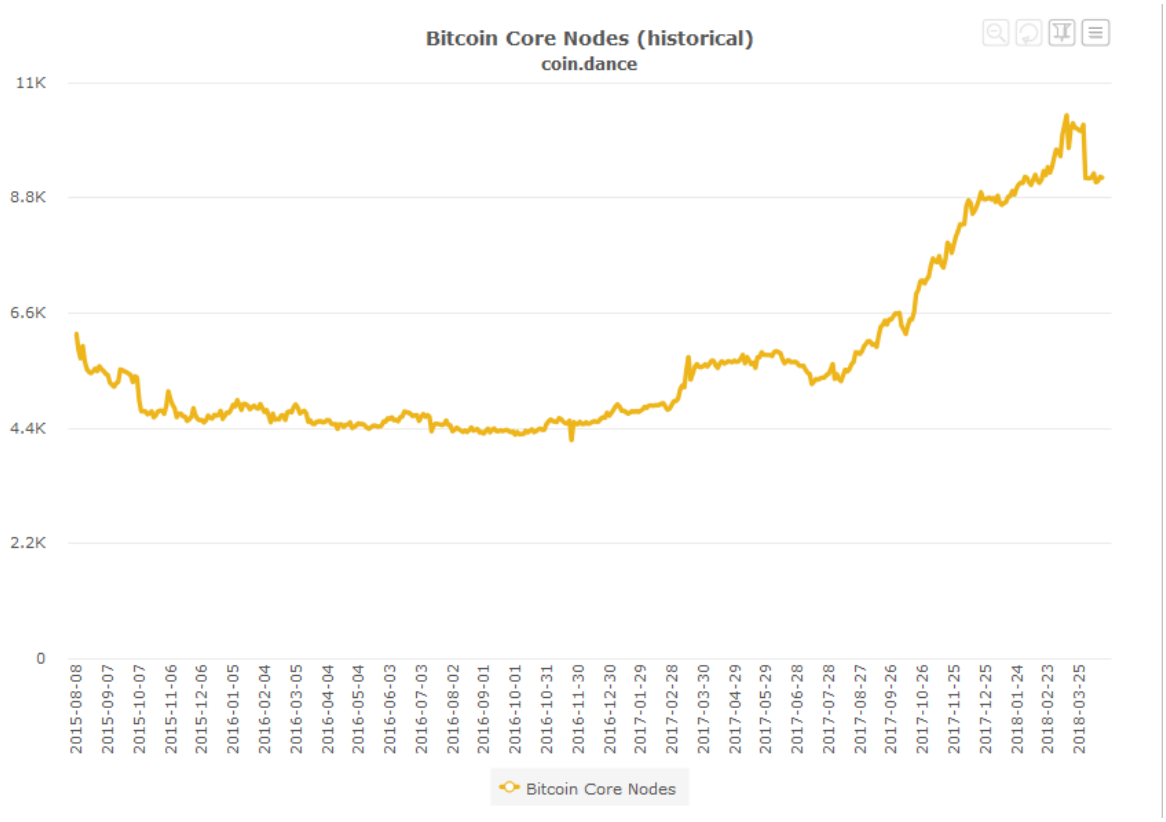


Figure 2[18] – Bitcoin Core Node Participation

2.10 Existing Bitcoin Analytic Platforms

There is a variety of analytic platforms for the most popular crypto-currencies that enables the user to explore almost every aspect of the chosen crypto-currency. Examples of some information that can be retrieved from these platforms is the block information (block height, transaction details etc), popularity of the network over time and even the volume of transactions that have taken place in different countries[16].

2.10.1 Example 1 - coin.dance[17]

The first example of a Bitcoin analytic platform is coin.dance. This website is a community driven Bitcoin analysis platform that allows users to view several types of

data that can be gained from the blockchain as well as other resources. This Bitcoin Analytics Platform is rather unique and clever with its analytic software however is lacking the ability to show users the balance of their Bitcoin wallet, therefore this solution doesn't fully meet the requirements of the project.

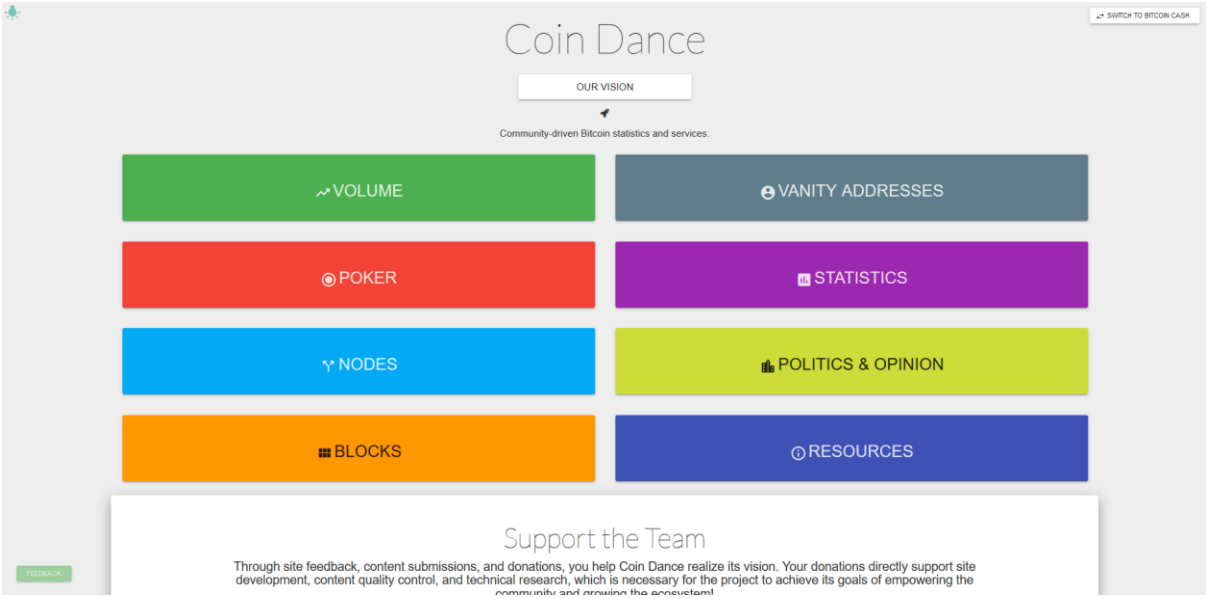


Figure 3[17] – coin.dance Homepage

2.10.2 Example 2 – blockchain.info[20]

Another example of another Bitcoin Analytics platform as a partial solution to the problem is the blockchain.info website platform. This platform has more statistical features than that of the previous example and is probably the most popular and complete Bitcoin Analytics platform that currently exists. This platform has all major functions that almost every user will need and want including the ability to break down every block and show all information related to the block; almost providing a complete solution to the requirements of the project.

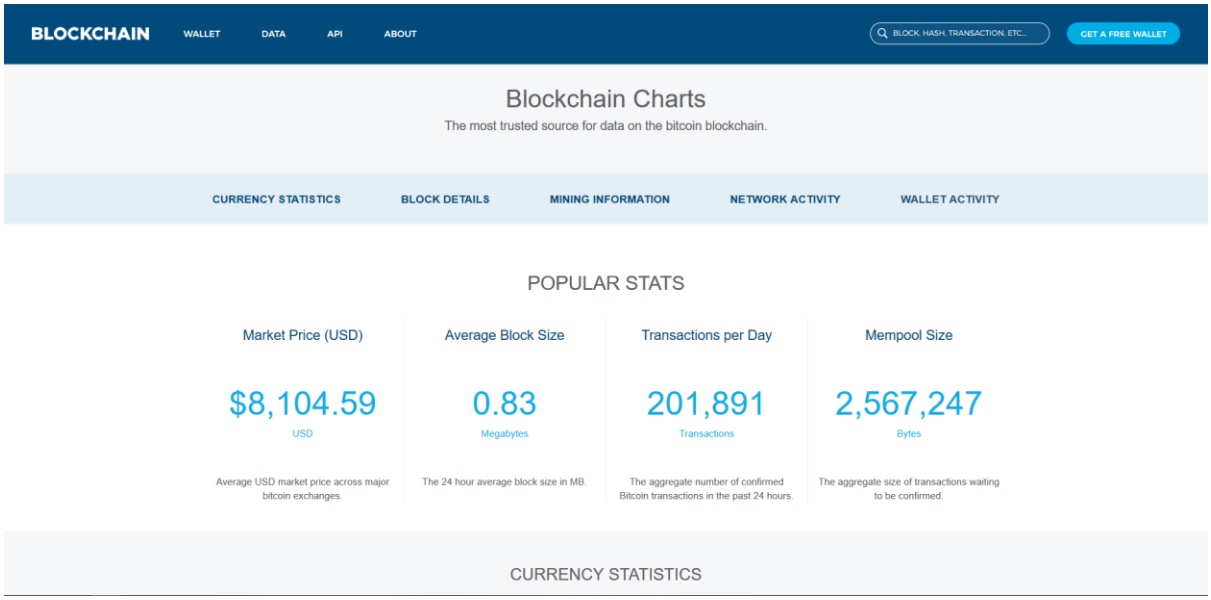


Figure 4 – blockchain.info Charts Page[21]

2.10.3 Example 3 – blockcypher.com[22]

The third and final example of a Bitcoin Analytic platform is blockcypher's website. This website isn't exclusive to Bitcoin but does however include in-depth statistical tools as well as proving an API that can be used by developers. Although this website isn't as complete for Bitcoin as the previous example, it does still prove to be a partial solution to the project requirements.

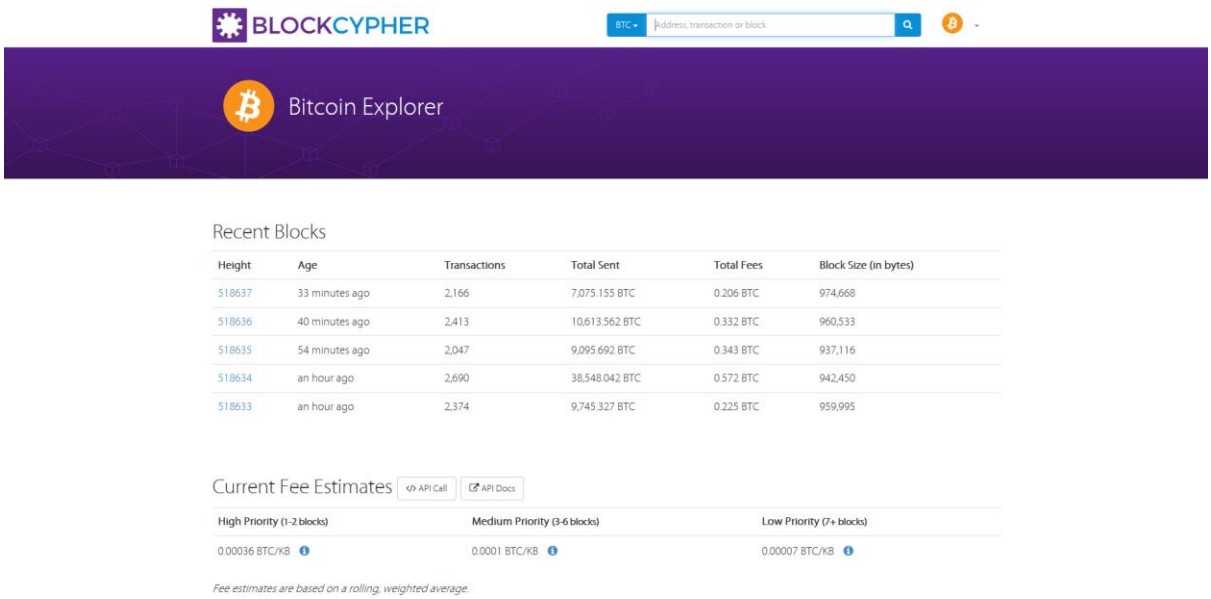


Figure 5 – blockcypher.com Bitcoin Explorer[23]

2.11 How this Project Differs

Although the end goal of this project is somewhat similar to the examples that were previously mentioned, this project differs by providing statistical data that is not available on the other Bitcoin Analytic platforms e.g. the average value of transactions per block within the first 140,000 blocks. Bitcoin statistics like this can be helpful and interesting to view, especially as the value of Bitcoin increases as these statistics are going to look very different.

Expecting to reach the functionality of the complete platforms mentioned above is however unreasonable considering the lack of time and resources that this project is given. However, as the purpose of the project was to explore and develop a prototype Bitcoin Analytics platform, incomplete tasks can be implemented at a later date.

2.12 Conclusion of Aims & Research Questions

This section of the report will make a clear aim of the project as well as providing what the research questions will need to be answered to meet the requirements of the project aim.

Aim – The aim of this project is to develop a prototype of a Bitcoin Analytic Platform to incentivise users to join participate as an active-node within Bitcoin's decentralized network, thus reducing the node-concentration within the network.

Research Questions – To develop a prototype of a Bitcoin Analytic Platform it will be necessary to identify how the blockchain works, how blocks are created, what information is stored within blocks, how this information can be used and manipulated and how to create an analytic platform from this data.

3 Specification & Design

This section will provide a clear picture of the Bitcoin Analytics platform that I had planned to create in order to satisfy the requirements of the project. The specification section will clearly state what the platform is required to do to satisfy these requirements, and the design section will provide more refined details of how this platform meets the requirements of the project.

3.1 Specification

This section will provide a list of requirements and functions that the platform will meet. The platform that was designed consists of two major parts, the system and the user interface. For ease of reading and following, I will be providing separate lists of functions and requirements for each.

As mentioned within the introduction of the report, the Agile software development method was what I was going to using due to the ability to make changes in the project at any time. This model is however unreasonable to follow without having a list of requirements that need to be met to satisfy the project's requirements.

3.1.1 MoSCoW[24] Requirements

Below I have listed the requirements for both the system, and the user interface. I have listed the requirements using the MoSCoW[24] method that I had used before. This method is helpful for providing a clear understanding of the project's requirements and their priority. This is done by first developing a clear set of requirements, and then ranking them based on priority:

Must – Requirement must be met to satisfy project's requirements

Should – Requirement should be met

Could – Requirement could be met but doesn't affect the project much

Would – Requirement would be met at a later time

System MoSCoW:

Must	• Analyse the Bitcoin network • Store balance data for each address • Store transaction information • Parse first 140,000 blocks of the Bitcoin blockchain • Output data that can be used for graphs • Be consistent and efficient with outputs
Should	• Provide different types of outputs • Store data locally • Be modular to help with implementation of new features • Be able to run on all OS's and platforms
Could	• Parse more than 140,000 blocks • Add cache to improve performance
Would	• Use external API's to enhance system • Run parallel code to have accurate order of blocks

Table 1: MoSCoW Requirements – System

User Interface MoSCoW:

Must	• Display Bitcoin balance for users • Display transaction information and transaction fees • Display parsed data of first 140,000 blocks • Develop platform that shows collected data in a visual form • Be consistent and efficient with graphs • Be user friendly
Should	• Provide different types of statistics/graphs • Make graphs and statistics easy to read • Be able to run on all OS's and platforms
Could	• Make graphs interactive • Add description to graphs
Would	• Allow users to form their own statistics or graphs

Table 2: MoSCoW Requirements – User Interface

To relate this back to the agile software development model, the important reason to have a clear set of requirements is so that the direction of the project always revolves around the set of 'Must' requirements as they are non-negotiable.

3.1.2 Non-Functional Requirements

Functional requirements are somewhat similar to the MoSCoW requirements in the way that they state what the system should do, however non-functional requirements state how the system should work. This information is important for the project specification as it shows the functionality of both the system and the user interface.

The non-functional requirements for the system and the user interface are based on quality attributes the Bitcoin Analytic platform should have in order to identify what needs to be done in the development of the platform to create a quality Bitcoin Analytic platform.

Table 3 (below) shows some sub-categories that can be used to provide the non-functional requirements of the system and user interface:

Quality Factors	Detail
Reliability	• The system parses and returns the correct data corresponding to the statistic the user wants to see 100% of the time providing that the system is not currently parsing any data. • An error message is displayed by the user interface if any errors occur whilst parsing data.

	• The system should not crash when parsing stored data as data should be easily readable before-hand. • The user interface should clearly display Bitcoin analytical data whenever a return from the system has been successful. • All data should be stored and calculated before creating graphs from the data. • Navigation through user interface shouldn't crash and bugs that occur are not displayed to the user.
Maintainability	• Source code is modular where available to allow for easier development in the future. • All source code is commented, and descriptions are provided where more complicated code arises to explain their function(s).
Ease of Use	• All the user's previous experience with graphs and analysing data should be carried across to the platform to make the users experience more intuitive so that users are not confused. • The user interface uses colour schemes that are suitable for users that are colour blind.
Compliance	• All graphs produced from data has been from the output of the system. • None of the data that is stored or used breaches any copyright laws.

Table 3: Non-functional Requirements

3.2 Design

In this section I will discuss the design choices that I have made throughout the development of the system and user interface as well as providing explanations and alternative methods that could have been chosen. I will do this by explaining the design choice I made at three stages of development; initial, interim and final. These design choices will relate closely to the overall aims of the project and requirements mentioned in the specification section above.

3.2.1 Design Phases

During the initial design phase, I planned to create a Bitcoin Analytic platform that can meet all project requirements including all of those that were mentioned in the

MoSCoW section above. This was planned to start with tasks that had the highest priorities (must on MoSCoW method) and then to work down through the other tasks with a lower priority.

The interim design phase of the system and user interface allowed me to judge and reflect on what the state of the system was at the mid-point of the project. Tasks that were complete or mostly complete at this stage was left out of any future work in order to make effective design choices.

The final state of the system shows what I was able to achieve at the end of this project but reflection on this will come at a later section. The relevance of the final state in this section is to show how the design changed over-time.

Due to the number of tasks that needed to be completed in order to achieve full completion of the project, I shall only be discussing the 'must' requirements from the MoSCoW method. Below are tables that describe the state of the system and user interface at the initial, interim and final states; based from the 'must' requirements and alternative methods that could have been chosen.

The first and second tables of this section (tables 4 & 5) shows how the system design changed for some of the requirements of the system as the project progressed.

System Requirements: -Analyse the Bitcoin network -Output data that can be used for graphs		
Initial Design	Interim Design	Final State
These requirements were initially designed to be met through using APIs that already existed to analyse the Bitcoin network. The data that had been retrieved from the API could then be manipulated and put into graphs that the user could view on a website. An alternate method of this design could have been to download the entire Bitcoin blockchain and gain data based from this. This alternate	At this design stage, I had decided to parse the first 140,000 blocks of the blockchain and retrieve useful data depending on the type of graph/statistic that the user wanted to view on their browser. This could have been done alternatively by using the API data from the initial design; however, this could put restrictions on the system due to the lack of data that is made available through a single API.	The final state of the system analysed the Bitcoin network the same way as the interim design however differed slightly in that data from the blocks were stored locally, reducing the run-time of the platform. An alternative method could have been to store a cache of previously run graphs so that no data would have to be parsed and a graph could just be displayed. This would negatively impact the

method would be slightly more complicated due to the parsing of the blockchain files.	Using multiple APIs would mean that the system would rely heavily upon other resources.	system's ability to expand as the amount of data within the graph would be restricted by only 140,000 blocks.

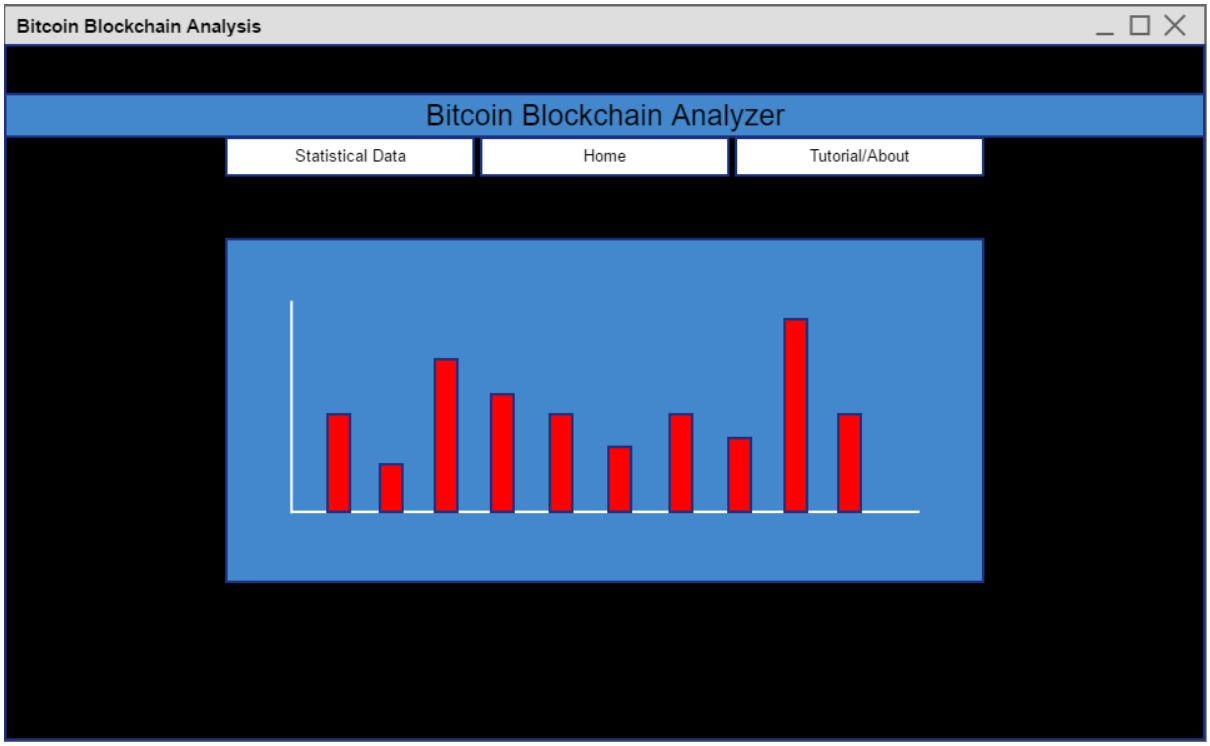
Table 4: First set of requirements design justification (System)

System Requirements: -Store balance data for each address -Store transaction information -Parse first 140,000 blocks of the Bitcoin blockchain		
Initial Design	Interim Design	Final State
Within the initial design phase, this was designed to be done through using an API to retrieve data about addresses and transactions within the first 140,000 blocks. This was because I thought it'd be the simplest way to implement this data. An alternate design could and would end up being to store all addresses and transaction data that were found when the first 140,000 blocks had been parsed.	At the interim stage, it had been decided that the best method of storing balance and transaction data would be to store it locally rather than retrieve it from APIs. The reasoning behind this is so that it would lead to the most efficient run-time of the platform. Alternatively, this could have been done by parsing the transaction data within all blocks each time the user requested the balance information. This however would make the platform much slower to use.	The final state for these requirements are the same as the interim design.

Table 5 – Second set of requirements design justification (System)

Below is a mock-up design of the website at the initial design phase of the website.

This is relevant to the table (*table 6*) that follows it.



User Interface Requirements: -Display Bitcoin balance for users -Display transaction information and transaction fees -Display parsed data of first 140,000 blocks -Develop platform that shows collected data in a visual form -Be consistent and efficient with graphs -Be user friendly		
Initial Design	Interim Design	Final State
All user interface requirements were initially designed to use data output that was retrieved from APIs and use HTML and JavaScript to display this data. The justification of doing so was so that only data relevant to the graph being shown would be retrieved by the API and processed into graphs by JavaScript and displayed using HTML. Alternatively, this could be done by not using APIs and using parsed data instead. This was decided against as the system was initially designed to use APIs as its main source of data.	At the interim stage of the project, the design of how these requirements would be met changed due to a change in system design. It was decided that the best way to display data would be to use a Python library named 'Bokeh'. This is because Python would be the language of choice on the back-end of the system, so displaying this data would be a better suited for implementation (see Implementation section). The output of 'Bokeh' is also very user friendly and the output of graphs are easily readable and understandable for users. This could have still been done using HTML and JavaScript, however for the data that needs to be displayed, a more mathematical platform would be required. Using HTML and JavaScript would have however allowed more freedom and customisation with the user interface of the website.	The final state of these requirements was met by following the design changes that were made in the interim stage of the project. The reasonings for this are the same as in the interim design section.

Table 6 – Set of requirements design justification (User Interface)

4 Implementation

This section of the report is somewhat similar to the ‘Specification and Design’ section, however it differs in that this section will describe the project’s system and user interface in more detail, right down to code level. The ideas and concepts that had been developed throughout the project will be included within this section as well as any problems which I came across during the implementation phase of the project.

4.1 Analysing the Blockchain – Initial Approach

As previously mentioned, the Bitcoin Analytic platform was planned to use API queries for the majority of the output that would be used to generate graphs for users. One of the APIs that I was looking into was the ‘blockchain.info’ API. A sample of the output of an API call for a single block is shown below, this example can also be found on the API’s documentation[25].


```
{
   "hash":"0000000000000bae09a7a393a8acded75aa67e46cb81f7acaa5ad94f9eacd103",
   "ver":1,
   "prev_block":"00000000000007d0f98d9edca880a6c124e25095712df8952e0439ac7409738a",
   "mrkl_root":"935aa0ed2e29a4b81e0c995c39e06995ecce7ddbebb26ed32d550a72e8200bf5",
   "time":1322131230,
   "bits":437129626,
   "nonce":2964215930,
   "n_tx":22,
   "size":9195,
   "block_index":818044,
   "main_chain":true,
   "height":154595,
   "received_time":1322131301,
   "relayed_by":"108.60.208.156",
   "tx":[--Array of Transactions--]
{
```


Figure 6 – Output of API call for block data

As can be seen from the output of the API call, a lot of data would have to be retrieved for 140,000 blocks to be parsed and analysed. This not only would take a lot of time, but there is also a limit as to how many calls you can make to the API. This is problematic due to one of the major requirements being that the first 140,000 blocks of the blockchain would need to be parsed and analysed. This is where a change in approach was needed so that the blockchain could be analysed.

4.2 Analysing the Blockchain – Final Method

The implementation of analysing the blockchain would be done by completed by downloading the full Bitcoin blockchain. To do this, I first had to download the 'Bitcoin Core[26]' client (*Figure 7*) which enables me to become a node on the network. As discussed earlier, nodes are responsible for keeping a record of all transactions that take place, as well as validating transactions. This means that to become a node on the Bitcoin network, you will first have to download the entire history of the blockchain.

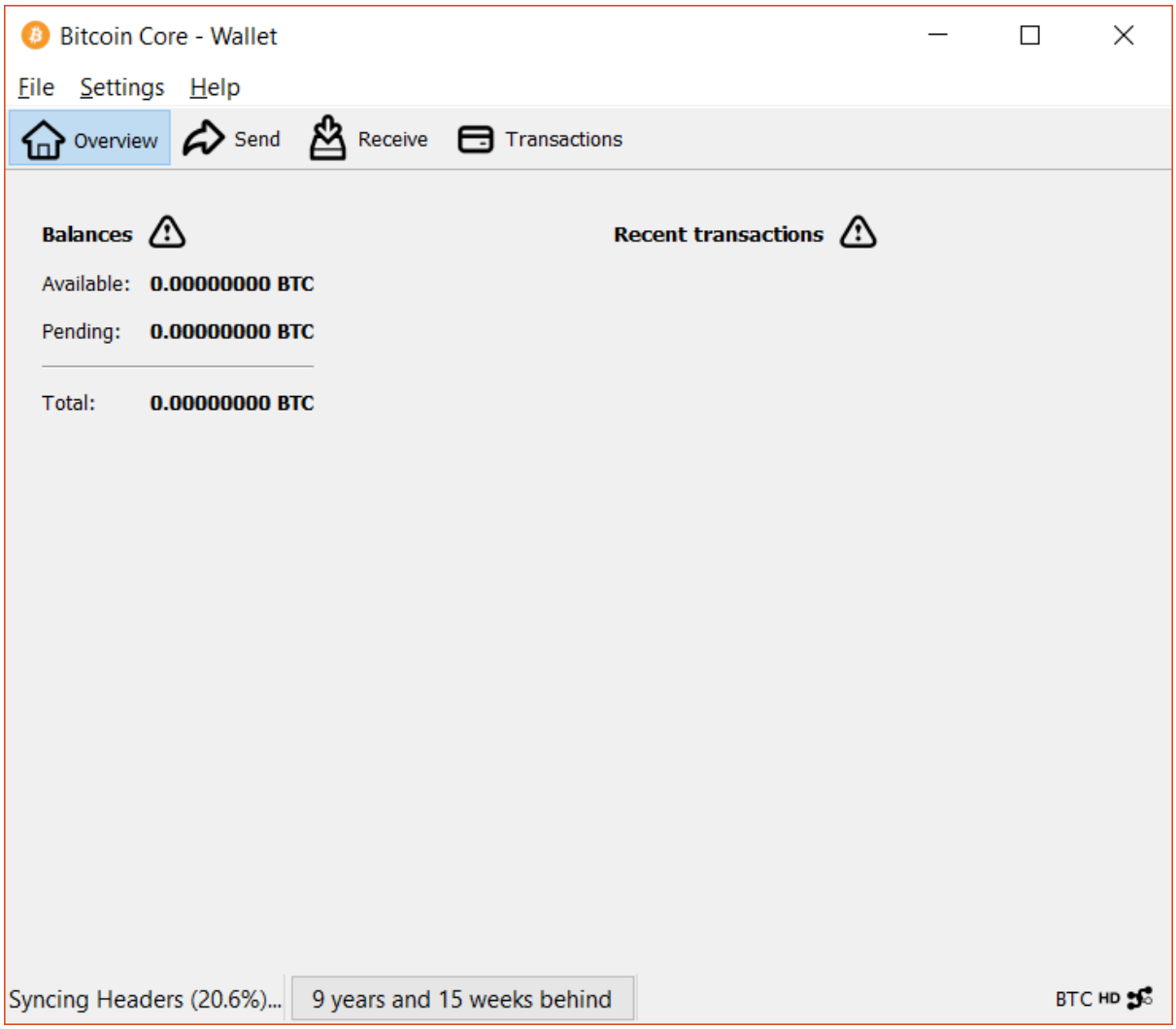


Figure 7 – Bitcoin Core Client

At the time of downloading, the Bitcoin blockchain consisted of just under 500,000 blocks which required around 150 gigabytes of storage. However, only the first 3 block files are needed for to analyse the first 140,000 blocks of the blockchain, which needs just over 3 gigabytes of storage space.

As I was not using an API to retrieve block data anymore, I had to find an alternative method to extract the same information from the downloaded raw block data. Whilst researching how to extract the information I was looking for, I cam across a website by 'coinlogic[27]' which looks at extracting the raw data from the blocks and making them human readable. This website really helped shape the core of the system as some of the ideas from the website were implemented into the Bitcoin Analytic platform that I had created.

The figure below (*Figure 8*) will help understand the structure of a block and how data is stored within a block; this will be very helpful to refer to when reading the upcoming section.

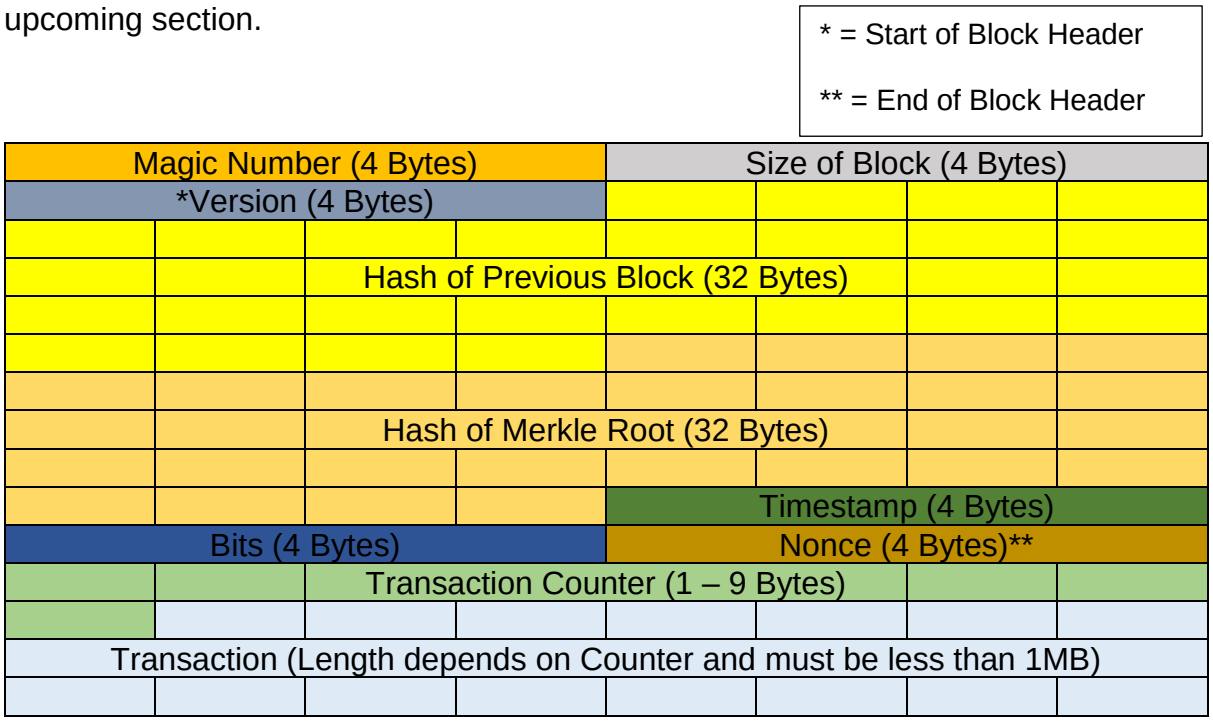


Figure 8 – Structure of a Block

The following sections will show and describe how all data from the blockchain can be extracted from the raw block data. All these methods were used to implement the features that can be found on the Bitcoin Analytic platform.

4.3 Reading the Block

Before continuing with the rest of the implementation information, it is important to note that the data within the blocks are stored in little-endian representation[30]; this means that to read the data that is stored within the block, the bytes must be read backwards. Luckily for the sake of implementation and simplicity, Python has a module[31] built into it that can unpack the string according to the format given.

The code below includes features such as how the system uses this module to convert the 1, 2, 4 and 8 byte little endian values into a python number. The implementation of this part of the system is crucial in extracting the correct data from the blocks.

```
def read_1bit(stream):
	return ord(stream.read(1)) #Reads 1 byte

def read_2bit(stream):
	return struct.unpack('H', stream.read(2))[0] #Converts 2 bytes of little-endian into an unsigned short integer

def read_4bit(stream):
	return struct.unpack('I', stream.read(4))[0] #Converts 4 bytes of little-endian into an ‘unsigned into’ integer

def read_8bit(stream):
	return struct.unpack('Q', stream.read(8))[0] #Converts 8 bytes of little-endian into an ‘unsigned long long’ integer

def reverse32(stream):
	return stream.read(32)[::-1] #Convert big endian --> little endian

def read_timeStamp(stream):
	utctime = read_4bit(stream)
	return utctime

def read_varint(stream):
	ret = read_1bit(stream)

	if ret < 0xfd: #One byte integer
		return ret
	if ret == 0xfd: #Read next two bytes
		return read_2bit(stream)
	if ret == 0xfe: #Read next four bytes
		return read_4bit(stream)
	if ret == 0xff: #Read next eight bytes
		return read_8bit(stream)
	return -1

def get_hexstring(bytebuffer):
	return(''.join(('%x' %i for i in bytebuffer)))
```

Figure 9 – Code for reading little-endian data

4.4 Magic Number

The magic number is a value that is 4 bytes in length which is used to mark the start of a new block within the blockchain. The value of the magic number is the same for every block – ‘0xD9B4BED9’; this is good to know for implementation as it is basically the ‘trigger’ point for the system to analyse the rest of the block.

Location – The magic number indicates the start of a new block, therefore it is the first 4 bytes at the very front of the block.

In order to read the magic number of the block, the first 4 bytes of the block must be read. Because the bytes are stored in little-endian format, the system makes use of the code in *Figure 9*. The code below (*Figure 10*) shows how the magic number can be retrieved from the block by parsing the block file and using the 'read_4bit' function.

```
self.magic_no = read_4bit(bf) #Reads and unpacks little-endian and returns the magic number
```

Figure 10 – Magic Number Code

Although the user for the Bitcoin Analytic platform won't see this information, it is important from a developers point of view so that the right bytes are being read by the system.

4.5 Size of Block

As you might be able to guess from the heading, this data will tell you the size of the block. The output retrieved when reading the size of the block is in bytes and there is also a limit as to how large a block can be which is 1 MB.

Location - The size of the block (blocksize) is also 4 bytes in length and is what follows the magic number within the block.

The code shown below (*Figure 11*) is very similar to that in the magic number. This is because the size of the block is stored in the 4 bytes after the magic number and needs the same method of conversion.

```
self.blocksize = read_4bit(bf) #Reads and unpacks little-endian and returns the size of the block
```

Figure 11 – Blocksize Code

Just like the magic number, the user does not see this data within he Bitcoin Analytic Platform, however this data could easily be implemented further down the line as future work.

4.6 Block Header

The 80 byte long block header holds a range of information that are associated with the block and where in the blockchain the block belongs[28]. The type of information held in these bytes includes: block version number, hash of the previous block, hash

of the merkle root, timestamp when block was mined, the target of the block and the nonce. As the version number is always the same value - 1, I will not include a section on implementation on this characteristic of the block due to the insignificant role it plays in the system.

Location – The block header is 80 bytes in length and follows the size of the block in location.

4.7 Hash of the Previous Block Header

The hash of the previous block header is an important value to know as it essentially tells you the block number of the block you are analysing. The only block to exist without a value for this is the genesis block (first block) as there is no prior block to this one.

Location - The 256-bit hash of the previous block header is contained from bytes 4-35 within the 80-byte block header. This is because it follows the version value which is located at the very beginning of the block header as seen in *Figure 8.*

The hash of the previous block is another important feature that was implemented within the system. This data is not yet part of the Bitcoin Analytic platform however could play a big part in the platform later in development. The code below (*Figure 12*) shows how the hash of the previous block is read by the system.

```
self.previousHash = reverse32(stream) #Reads and reverses 32 bytes of data to get hash of previous block
```

Figure 12 – Hash of Previous Block Code

4.8 Hash of the Merkle Root

This value is a 256-bit hash based on all transactions that occurred within the block and is updated as a new transaction is accepted by the block. This value is useful for keeping the integrity of the blockchain as a small change in any previously accepted transactions would completely change the hash value that is given.

Location - The 256-bit hash of the merkle root is contained within bytes 36-67 of the byte header.

The code for the hash of the merkle root is shown below (*Figure 13*). The merkle root hash doesn't play a part in the Bitcoin Analytic platform however it could be implemented if the site lets users analyse the blocks closer.

```
self.merkleHash = reverse32(stream) #Reads and reverses 32 bytes of data to get hash of the merkle root
```

Figure 13 – Merkle Root Hash Code

4.9 Timestamp

The timestamp of the block shows when the block had been successfully mined. It does this by calculating how many seconds it has been since 01/01/1970 00:00 UTC.

Location - This value is 4-bytes in length and is found in bytes 68-71 within the block header. The timestamp however doesn't have a role to play within the Bitcoin Analytic platform, so I will not be showing how this was implemented.

Just like the hashes of the previous block and the merkle root, this information is not shown to the user but could be utilised in the future. The code for extracting the timestamp is shown below (*Figure 14)*. As you can see, this uses the 'read_timeStamp' method seen in *Figure 9*.

```
self.time = read_timeStamp(stream) #Reads the timestamp of the block and returns UTC time value
```

Figure 14 – Block Timestamp Code

4.10 Bits

The purpose of the bits value is to show the current target that must be met by the nonce in order for the block to be successfully mined. This value is adjusted as the difficulty of mining a block is changed.

Location - This value is also 4-bytes in length and can be found in bytes 72-75 of the block header.

Similarly to the timestamp, this information doesn't play a major role within my system. The code for reading this information is shown in *Figure 15*. As you can see, this utilizes the 'read_4bit' method mentioned previously.

```
self.bits = read_4bit(stream) #Reads the target number of 0's for the block to be mined
```

Figure 15 – Reading the Bits Code

4.11 Nonce

The nonce of the block is a 32-bit value that is used to 'mine' the block. As previously mentioned, this is the value that is randomly input to try and complete the block. The block would be completed as soon as the target has been met.

Location - The nonce of a block is 4-bytes in length and can be found in bytes 76-79 of the block header; this also represents the end of the block header.

The nonce of the block is parsed the same way to that of the bits due to them both being 4 bytes in length; meaning that the 'read_4bit' method could be used once again.

```
self.nonce = read_4bit(stream) #Reads and returns the nonce of the block
```

Figure 16 – Reading the Value of the Nonce Code

4.12 Transaction Counter

This value holds the variable of the number of transactions that occur within a block which can hold the values of 1, 3, 5 and 9. The purpose of these encoded integers is to save the amount of space required for transactions by telling us how many transactions occur within a block.

Location – The transaction counter is stored in the next 1, 3, 5 or 9 bytes after the block header. How we read the number of transactions that occur depends on the value that is stored.

Implementing the transaction counter into the system was tricky due to the varying size of bytes that this value could be. This meant that a function needed to be created to find out the value of the transaction counter before going onto the finer details of the transaction.

Below (*Figure 17*) shows how the program did this. As you can see, this is done using the 'read_varint' method which takes the block file as input as shown in *Figure 9*. This method will use if statements to determine the number of bytes the system should read next.

```
self.transaction_count = read_varint(bf) #Returns the transaction count by parsing the block file
```

Figure 17 – Transaction Counter Code

4.13 Transactions

The final pieces of useful data can be found by using the value that we retrieved from the transaction counter. The first transaction of the block is known as that ‘coinbase’ transaction. This transaction is where the miner of the block gets their reward for using their computational power to successfully mine a block.

Each transaction that takes places hold certain data about the transaction. The data that is stored include: version number (always 1), input counter, input list, output counter, output list and the lock time of the transaction.

As the version number is always the same value - 1, I will not include a section on implementation on this characteristic of the transaction due to the insignificant role it plays in the system. I will describe the implementation and the rolls of all other transaction characteristics below.

Location – This set of data follows the transaction counter. The length of this data varies depending on the length of the attributes it has.

4.14 Input Counter

The input counter shows how many transaction inputs occur within a block. This data is used by the list of inputs so that it knows how many bytes to read.

Location – This data is located within the byte range of 4 – (4-12) within the transaction segment. As you can see, the size of this data can be 1 - 9 bytes in length.

4.15 List of Inputs

The list of inputs has a range of useful individual transaction information contained within it. The length in bytes of the list of input depend on the value retrieved from the input counter as mentioned above and contains information such as:

- Hash of previous transaction

  Location – This data can be found within the first 32 bytes of within the list of inputs allocated space

- Previous output transaction index – Indexes an output of a transaction that is to be used

Location – This data is found in the 4 bytes that follow the hash of the previous transaction

- Transaction Input Script length

  Location – Found in the next 1-9 bytes following the previous output transaction index

- Transaction Script[29] – A unique 2 part combination of a hashed public key and a signature to prove ownership of the private key that corresponds to the public key

  Location – Follows the transaction input script length and the size in bytes of the transaction script is determined by the value of the transaction input script length

- Sequence number – Irrelevant to the project

Location – The list of inputs data follows the input counter. The length of this data depends on the value of the input counter.

The implementation of the information above is shown below (*Figure 18*). The code must be implemented in this order otherwise the data that is returned will be incorrect and rendered useless. Although this data is not used by the system as much as the list of outputs; the data can be extremely useful for future development.

```
class tx_Input(object):

        def __init__(self):
                super(tx_Input, self).__init__()

        def parse(self, stream):
                self.previousHash = reverse32(stream) #Previous transaction hash
                self.prevTx_out_idx = read_4bit(stream) #Transaction output index
                self.txIn_script_len = read_varint(stream) #Transaction script length
                self.scriptSig = stream.read(self.txIn_script_len) #Transaction signature
                self.seqNo = read_4bit(stream) #Sequence number

        def __str__(self):
                return "\nPrevious Hash: %s \nTransaction out index: %s \nTransaction in script
lengh: %s \nscriptSig: %s \nSequence Number: %8x" \
                % (get_hexstring(self.previousHash), self.prevTx_out_idx, self.txIn_script_len,
get_hexstring(self.scriptSig), self.seqNo)

        def __repr__(self):
                return __str__(self)
```

Figure 18 – List of Inputs Code

4.16 Output Counter

The output counter shows how many transaction outputs occur within a block. This data is used by the list of outputs, this is because the size of the list of outputs depends on this number.

Location – This set of data follow the list of inputs, ranging from 1 – 9 bytes in length.

4.17 List of Outputs

The list of outputs holds the more relevant details of a transaction. The information that can be retrieved from the list of outputs is highly important to the system due to the data being used to generate graphs for the platform. The data that can be retrieved from the list of outputs include:

- Value – The amount of Bitcoin/Satoshis being sent within the transaction

  Location – This can be found in the 8 bytes that follow the output counter

- Transaction Output Script Length

  Location – Found within the next 1-9 bytes that follows the value

- Public Key – The public key of the address that the amount of Bitcoin/Satoshi is being sent to. This is essential in order to store the balances of all Bitcoin users

Location – This data follows the transaction output script length. The length in bytes of the public key varies as it depends on the value of the transaction output script length.

```
class tx_Output(object):

        def __init__(self):
                super(tx_Output, self).__init__()

        def parse(self, stream):
                self.value = read_8bit(stream) #Return value of transaction
                self.txOut_script_len = read_varint(stream) #Returns script length of transaction
                self.scriptPubKey = stream.read(self.txOut_script_len) #Returns public key

        def __str__(self):
                return "Value (Satoshis): %d (%f btc)\nTransaction out script lengh: %d\nScript
PubKey: %s" \
                % (self.value, (1.0*self.value)/100000000.00, self.txOut_script_len,
get_hexstring(self.scriptPubKey))

        def __repr__(self):
                return __str__(self)
```

Figure 19 – List of Outputs Code

The code above is what was implemented to use for most graphs on the Bitcoin Analytic Platform. This code is where the some of the most important requirements of the project are met that are mentioned in the Specification and Design section e.g. returning and getting balance data for users. The system heavily relies on the output of this data to use for graphs and analytical statistics.

4.18 Lock-Time

The value this holds shows the block height or timestamp when the transaction is final. The lock-time of a transaction had no role to play within the Bitcoin Analytic platform, so I will not show the implementation of this data.

Location – The lock-time of a transaction follows the list of outputs and is 4 bytes in length.

4.19 Block Parser

All the code mentioned above is called upon by the block parser. The block parser does this by taking an input that points to the location of the block file; an example of the block file's name is 'blk00000.dat'.

Once all data has been parsed, the blocks within 'blk00000.dat' have all had their data read and stored. To analyse the first 140,000 blocks within the blockchain, three block files will need to be parsed through the system; 'blk00000.dat', 'blk00001.dat' and 'blk00002.dat'.

4.20 Bokeh[32]

As briefly mentioned in the report, Bokeh is used for the user interface part of the Bitcoin Analytic platform. Bokeh is an open source Python interactive visualisation library which allows the output data of the blockfile to be represented visually on a web browser as a means of a graphical user interface.

Bokeh is the front end of the system and can produce some elegant graphs and interfaces to make the use of the Bitcoin Analytic platform and overall more complete and enjoyable platform. The figure below (*Figure 20*) provides a screenshot of the webpage of the Bitcoin Analytic platform that shows the transaction value ranges within the first 140,000 blocks.

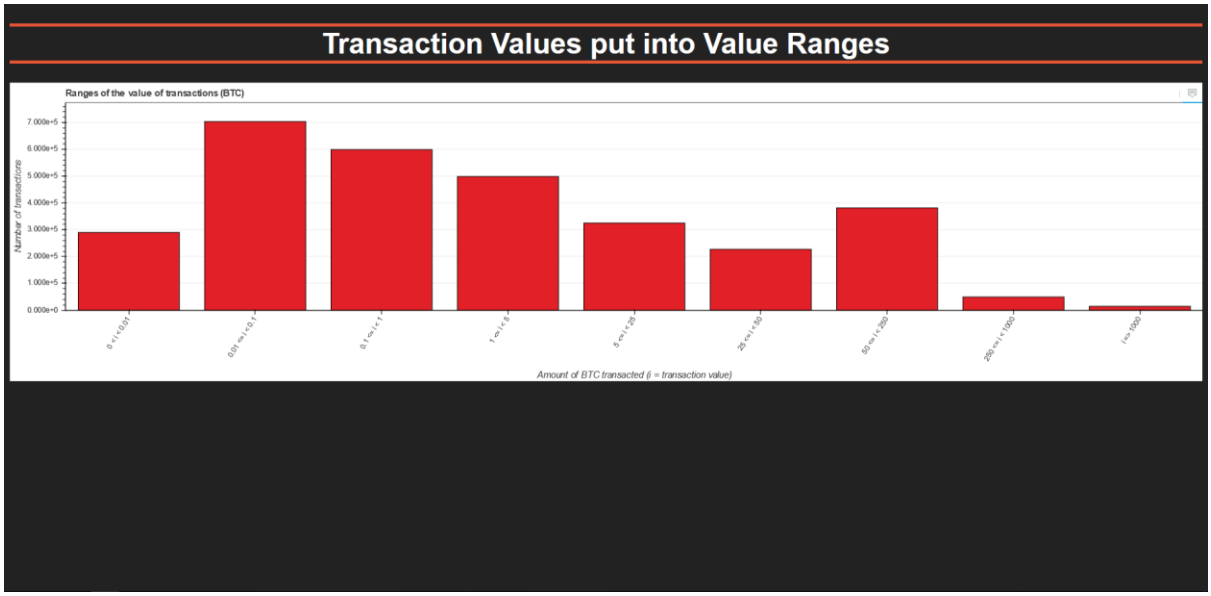


Figure 20 – Bitcoin Analytic Platform – Transaction Ranges

Bokeh does however have its downsides; the first one being that to be able to run and view the data on the Bitcoin Analytic program you must do one of two things. The first thing would be to run the page using a Bokeh server, or the alternative is to run it locally. Because of this a lot of complications can occur especially when first trying to set this up.

The code below shows how bokeh was implemented into the system using the same Python file that was used to parse blocks (*Figure 21*). The code also shows that bokeh uses HTML to create a hover tool that allows the user to pin-point data contained within a graph; this is also embedded within the main Python file (*Figure 22*).

```
def create_bar_chart(transactionAmountDict, title, x_name, y_name, hover_tool = None, width =
1200, height = 300):
        source = ColumnDataSource(transactionAmountDict)
        xdr = FactorRange(factors = transactionAmountDict[x_name]) #X-axis data values
        ydr = Range1d(start = 0, end = max(transactionAmountDict[y_name])*1.1) #Y-axis data
values

        tools = []
        if hover_tool:
                tools = [hover_tool,] #Creates hover tool

        plot = figure(title=title, x_range=xdr, y_range=ydr, plot_width=width,
            plot_height=height, h_symmetry=False, v_symmetry=False,
            min_border=0, toolbar_location="above", tools=tools,
            responsive=True, outline_line_color="#666666")
        #Defines the plot of the graph using the attributes mentioned

        glyph = VBar(x = x_name, top = y_name, bottom = 0, width = 0.8, fill_color = "#e12127")
        plot.add_glyph(source, glyph)

        xaxis = LinearAxis()
        yaxis = LinearAxis()

        plot.add_layout(Grid(dimension=0, ticker=xaxis.ticker))
        plot.add_layout(Grid(dimension=1, ticker=yaxis.ticker))
        plot.toolbar.logo = None
        plot.min_border_top = 0
        plot.xgrid.grid_line_color = None
        plot.ygrid.grid_line_color = "#999999"
        plot.yaxis.axis_label = "Transaction amount (BTC)"
        plot.ygrid.grid_line_alpha = 0.1
        plot.xaxis.axis_label = "Block Number Range (Range size = 20,000)"
        plot.xaxis.major_label_orientation = 1
        return plot
```

Figure 21 – Creating a Bokeh Bar Chart

```
def create_hover_tool(): #Creates a hover tool for user to highlight graph
        hover_html = """
                <div>
                        <span class = "hover_tooltip">Value of $x * 20,000 transactions</span>
                </div>
                <div>
                        <span class = "hover_tooltip">@Value (BTC)</span>
                </div>
        """
        return HoverTool(tooltips = hover_html)
```

Figure 22 – Bokeh Hover Tool

5 Results

In this section I will describe the extent to which I achieved my goals. I will do this by showing and describing how much of the Bitcoin Analytic platform works as intended

and what does not; this will be done by using screenshots as well as showing critical tests that were carried out on the system.

5.1 The Finished System

This part of the results and evaluation section will show you the final Bitcoin Analytic platform that was created to meet the requirements of the project. This will be done through using screenshots and providing a detailed description of what each screenshot is showing.

The first screenshot (*Figure 23*) and the first web-page that was developed shows how the average size of transactions within the first 140,000 blocks. To make the graph much more user friendly and easier to read, the blocks were grouped together in chunks of 1000 blocks.

The page makes for an interesting read as you can see the number of transactions rise as the number of blocks tend to 140,000; this is probably due to the increasing popularity of the blockchain.

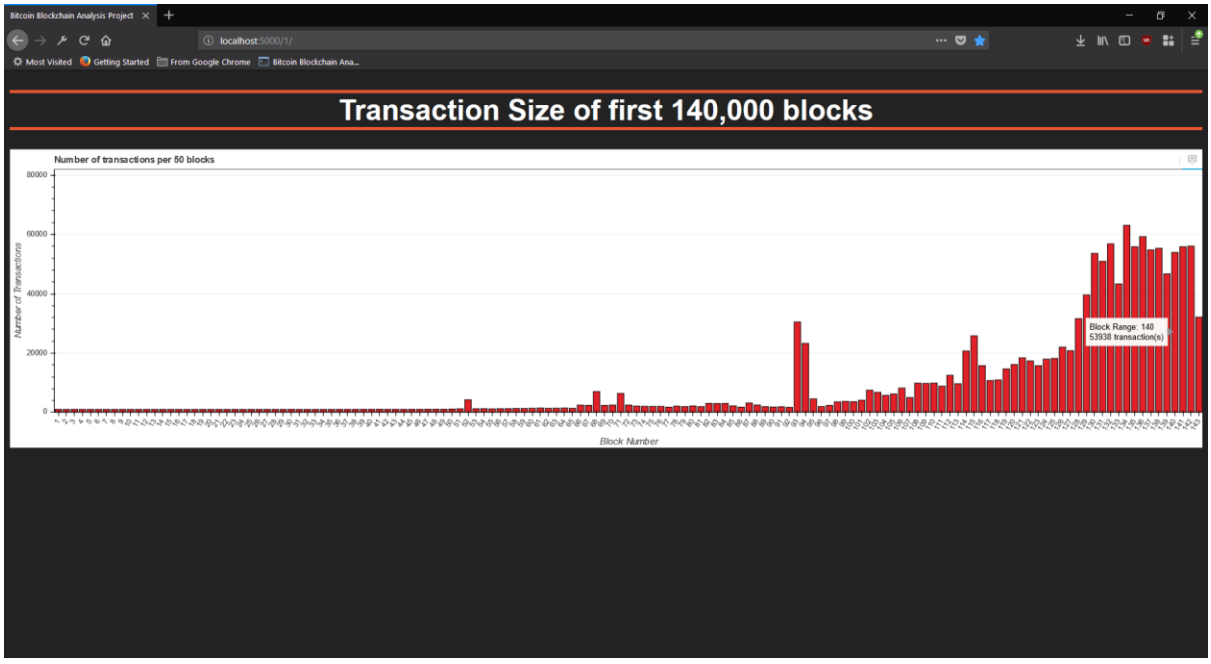


Figure 23 – Block's Transaction Size (Cumulative 1000 Blocks)

The next screenshot (*Figure 24*) shows another web-page on the Bitcoin Analytic platform; this page is showing the 10 blocks with the most valuable accumulative transaction value. The transaction value of these blocks are shown in Satoshis; this is not ideal but it was needed to increase the performance time of the system. The

system also does not show the user the most valuable individual transactions that occur within block which would make for a more interesting graph but this could be done in the future.

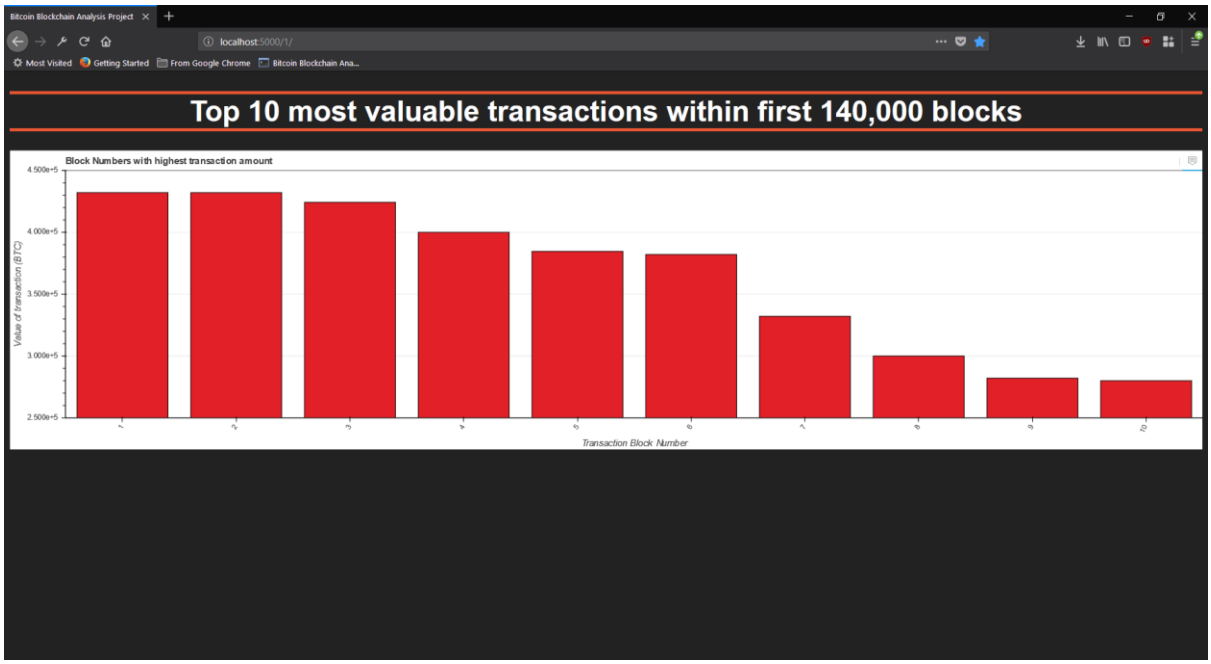


Figure 24 – Blocks with the Highest Value of Transactions

The third web-page (*Figure 25*) shows the total amount of Bitcoin that has been transacted over the first 140,000 blocks of the Bitcoin blockchain. The block data has again been grouped into chunks of 1000 blocks; this makes the readability of the data easier.

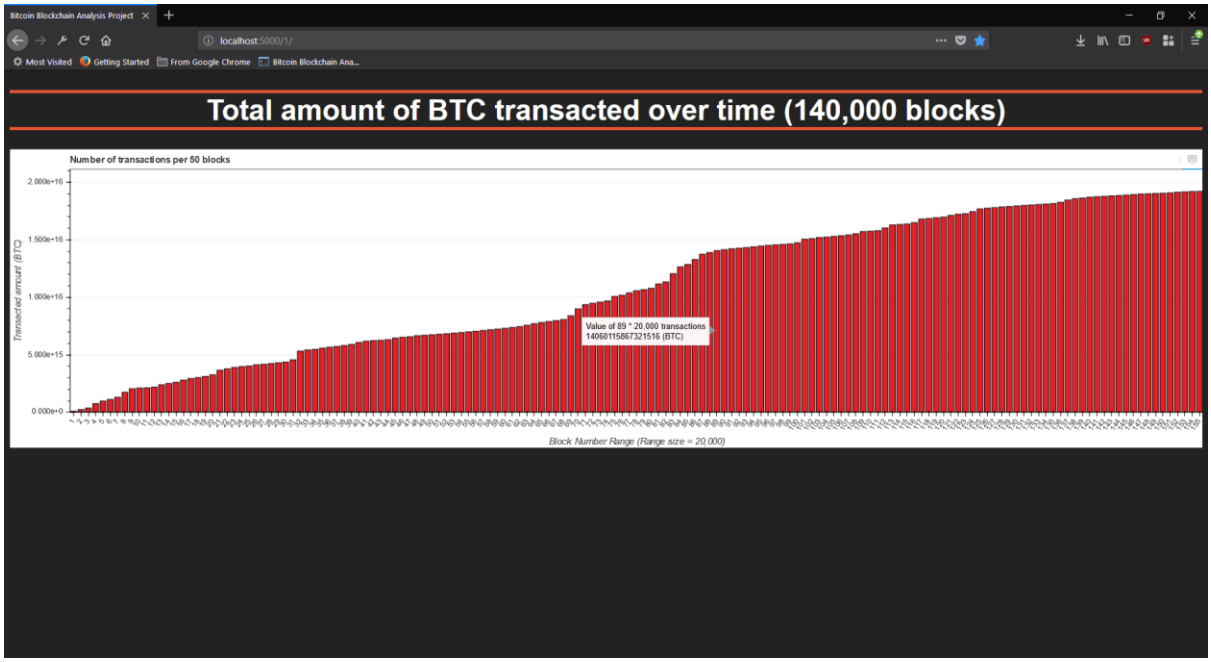


Figure 25 – Total amount of Satoshi Transacted over time

The fourth web-page (*Figure 26*) is a personal favourite of mine which groups transactions based on value and shows the number of transactions that lie within that value range.

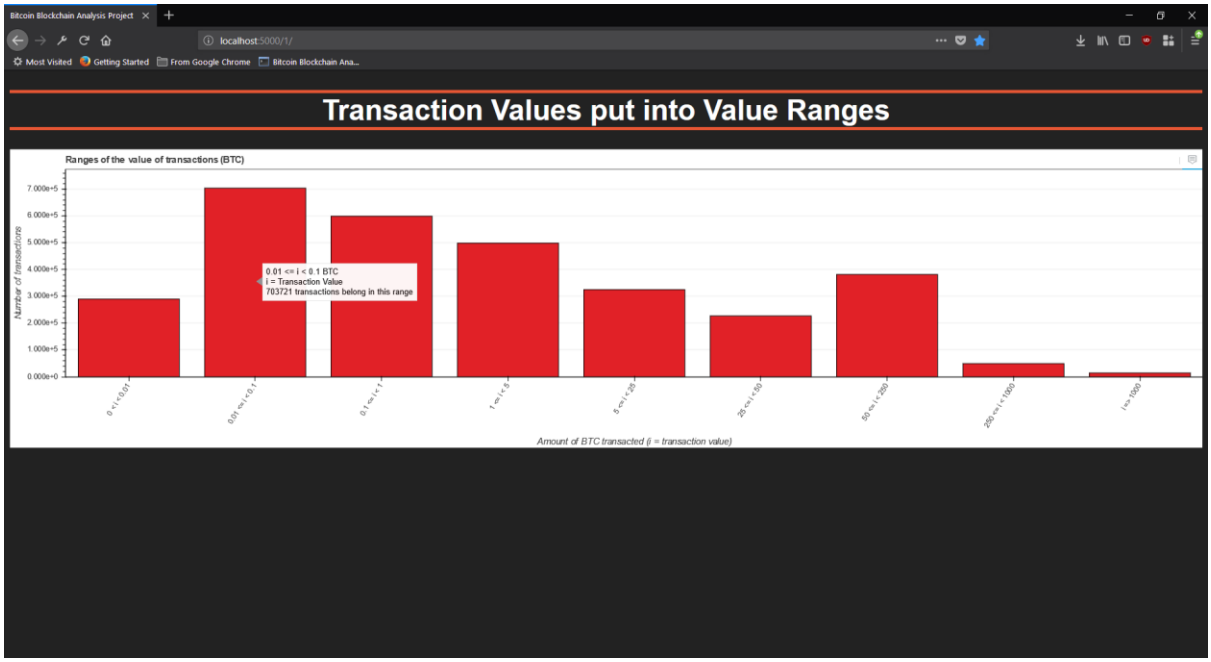


Figure 26 – Transaction Values

This graph makes for an interesting view as it shows where the most common transaction values lie within the first 140,000 blocks; the most popular range of transaction values being 0.01 – 0.1 BTC. If this graph would use the data from more

recent blocks, the graph would look extremely different due to the value of Bitcoin being significantly higher compared to what it used to be.

The fifth and final web-page (*Figure 27*) and graph of the system shows the amount of Bitcoin that the miner gets rewarded for creating a new block. As mentioned previously in the report, the value of the reward is halved every 210,000 blocks; if we're just talking about the first 140,000 blocks the reward value is always 50 BTC.

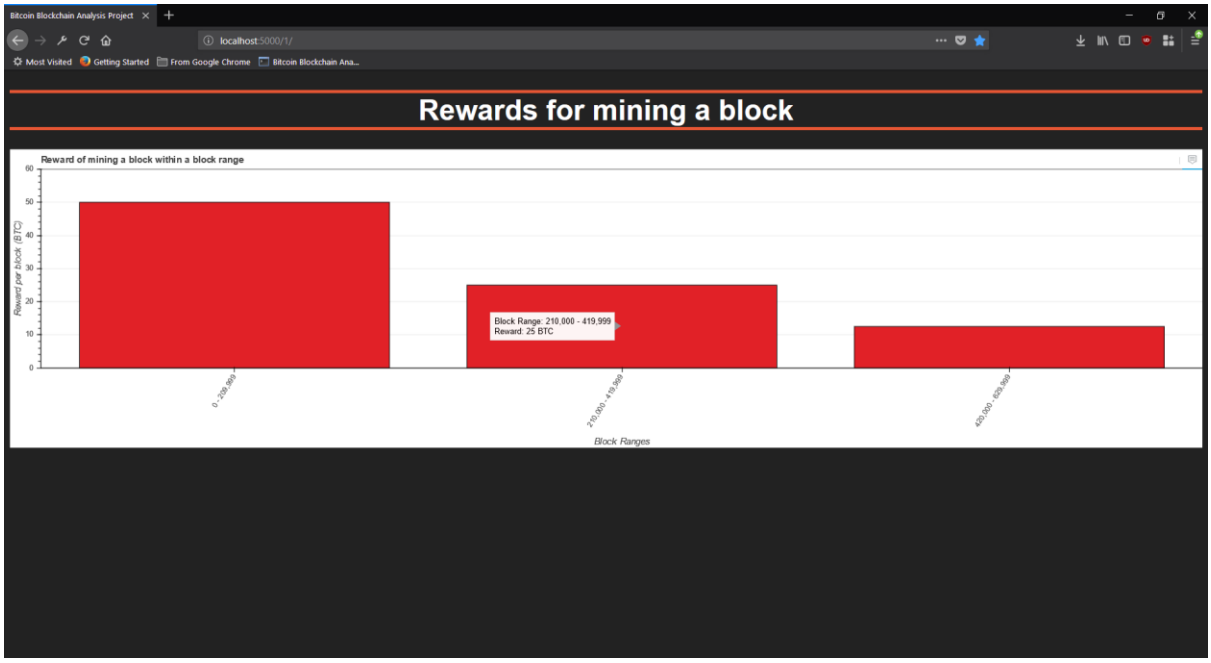


Figure 27 – Rewards for Mining a Block

5.2 Testing

Now that the finished Bitcoin Analytic platform has been shown, I will carry out extensive testing of the system to see where the system meets and fails the project's requirements. I will give reasoning behind the tests that have been used to evaluate the results of the project so that the tests serve a purpose to the Bitcoin Analytic platform.

Developing any type of system requires high quality code; this is even more true when the system includes a user interface as the speed of the platform highly influences the user's opinion on the system. To ensure that high quality code has been used, the Bitcoin Analytic platform has been tested from a developer's point of view as well as from a user's point of view.

To ensure that high quality code has been used, the code must crucially do the job that it is required to do, however this does not ensure that the quality of the code is of a high standard. High quality will also be easily readable, commented in detail, use appropriate naming and ensures the stability of the system if some aspects of the system were to fail by providing error handling.

At the time of writing this report, the system and the user interface have been tested from the point of view of the developer and of the user. The tests from the developer's perspective will be mostly testing of the system; testing from the user's perspective will be tests based on what the user can see and do on the system.

The table below (*Table 7*) includes the test reports of the features of the Bitcoin Analytic platform that have been implemented into the system.

Test Case ID: 1	Test Purpose: Analyse first 140,000 blocks of the Bitcoin blockchain. Test Reasoning: Show to what extent the system has met the project's requirements.	
Testing Environment: Windows 10		
Preconditions: 1) User has web paged open 2) User has Bokeh installed correctly 3) System has been pre-loaded for user		
Test Number	Test Test Reasoning Expected Response	Pass/Fail
1	Test: Run 'Number of transactions per 1000 blocks' Python file. Test Reasoning: To ensure that the system returns correct output to user. Expected Response: The user is shown correct web-page (*Figure 23*).	Pass
2	Test: Run 'Blocks with most valuable transactions' Python file. Test Reasoning: Ensure that the system returns the correct output. Expected Response: User is shown the correct web-page output (*Figure 24*).	Pass
3	Test: Run the 'Total amount of Bitcoin transferred over time' Python file. Test Reasoning: Ensure that the output of the system is correct and shown to the user.	Pass

	Expected Response: User is shown the correct web-page output (*Figure 25*).	
4	Test: Run the 'Value of Transactions' Python file. Test Reasoning: Ensure that the output of the system is correct and shown to the user. Expected Response: User is shown the correct web-page output (*Figure 26*).	Pass
5	Test: Run the 'Mining Rewards' Python file. Test Reasoning: Ensure that the output of the system is correct and shown to the user. Expected Response: User is shown the correct web-page output (*Figure 27*).	Pass
6	Test: Parse 140,000 blocks of blockchain. Test Reasoning: Ensure that the system meets the requirement of parsing he first 140,000 blocks of the Bitcoin blockchain. Expected Response: Data on the first 140,000 blocks is output by the system.	Pass
7	Test: Collect balance data for users. Test Reasoning: Ensure that the system meets the requirement of collecting balance of users for the first 140,000 blocks. Expected Response: Balance data is collected using the public keys of users.	Fail
8	Test: Display balance data for users. Test Reasoning: Ensure the system meets the requirement of 'Must display Bitcoin balance for users'. Expected Response: Web-page shows balance data for users.	Fail
9	Test: Display outputs of data in a visual form. Test Reasoning: Ensure that the system meets the requirement of providing the user with visual representation of data. Expected Response: Users are provided with visual forms of data when system is run.	Pass
	Test: Store transaction information.	

10	Test Reasoning: Ensure that the system meets the requirement of storing transaction information. Expected Response: Transaction information has been collected and stored sensibly.	Pass

Table 7 – System and User Interface Testing

5.3 Summary of Test Results

The results from the tests that have been carried out on this system show that the majority of requirements for the project have been met. The Bitcoin Analytic platform did however fail on some tests due to lacking the ability to store and display balance data for users which was one of the 'must' requirements for the system. The reason that this aim was not achieved was because I underestimated the amount of work that it would require whilst implementing the system.

To summarise, the Bitcoin Analytic platform has met most of the project's requirements however there is still a lot of room for improvements especially in terms of additional feature implementation as well as the speed in which the system can analyse the blocks and its data.

6 Evaluation

In this section of the report I will critically evaluate the results shown above in the light of the tests; this will be done by providing strengths and weaknesses of the Bitcoin Analytic platform as well as evaluating the methodology I used to carry out this project. The weaknesses of the platform will somewhat carry over to the 'Future Work' section of the report that follows.

6.1 Evaluation of Agile Methodology

As mentioned within the introduction, the project was carried out using the agile methodology to develop the Bitcoin Analytic platform. This methodology had many positives about it, however it was not perfect, so it did have some negatives too.

The positives of using the agile methodology approach for this project was that setting long term and short terms goals made it easier to focus on what tasks needed to be completed and the time they needed to be completed by. Meeting my supervisor every two weeks also enabled me to set reasonable goals to be

completed for the next time we met. Another positive of using the agile methodology for this project is that due to the lack of documentation on how to analyse the blockchain, the project needed to adapt to the information that the system is able to retrieve; using the agile methodology allowed for these adaptations within the system.

The negative of using an agile methodology for this project is that there was no clear end goal for the project until the bulk of development had been completed. This made it hard to work towards something specific as the project could be completed in many forms.

Overall, I feel that the agile methodology is definitely suited for this kind of project as the benefits of this approach definitely outweigh the downsides, especially when a project has a certain lack of documentation to use for implementation.

6.2 Strengths of the Platform

The first strength of the Bitcoin Analytic platform that was developed is that the results that are returned upon request are consistent and reliable. The system has proved to stay consistent with the output of data and no errors occurred whilst testing the platform. The graphs in which the users can view are also very consistent with the outputs which makes for no confusion when reading the data.

Another strength of the Bitcoin Analytic platform is that the user interface of the web pages is very user friendly and the simplicity of the site makes it intuitive for users. This was a priority when developing the platform as it would not make sense to overload the user in data; this would decrease the user-friendly nature of the site as well as increasing frustration. The lack of un-needed features makes the platform simple and not complicated which is different compared to other analytic platforms that often have unnecessary features.

The appearance of the web-pages show that design was carefully thought of through development. The pages are consistent in terms of colour schemes and graphs throughout the site. The same colour is used for the background throughout the system and the same goes for the colour of the text.

6.3 Weaknesses of the Platform

The biggest weakness of the platform at the time of writing this report is the speed of the system. The quickest method for the system to currently output data is to read previously stored data from the blockchain. This still affects the performance of the system considerably and some pages can take much longer to load compared to others. This can however be refined in the future by caching the web-pages.

Another weakness of the system is that although it is simplistic, it does lack additional graphs and helpful information that users could make use of. The lack of graphical data means that the users miss out on important and interesting data that could be implemented. This could be improved on in the future where time is not a limitation.

A third weakness of the system is that by using Bokeh, it limits what can be achieved visually within the platform. The virtual environment that is needed for Bokeh applications is also tricky to set-up for first time users, but this could be mitigated by using a Bokeh server to run the application from.

The fourth weakness of the system is that it still lacks the ability to store and retrieve data balance for users from the first 140,000 blocks of the blockchain. This was a ‘must’ goal as seen in the MoSCoW method, so failing to achieve this means that the system still has not reached core functionality. This part of the system was failed due to underestimating the amount of time that implementing this feature would take; the lack of documentation on how this could have been achieved is also not helpful, but ultimately the underestimation of this requirement is the reason this was not achieved.

7 Future Work

This section of the report will detail what can still be implemented within the Bitcoin Analytic platform after the end of the project. The point of this section is to express how the platform can grow whether its implementing features that were planned from the start or stating ideas that were thought of throughout the course of the project. If someone were to carry on with this project, this section should be a good place to start as these features would benefit both users and developers alike.

The first and most important piece of future work that I would implement is to store and display the balance data for users within the first 140,000 blocks of the Bitcoin blockchain. This feature would enable the system to fully meet the requirements of the project meaning that the core functionality of the system would be complete. This additional feature would also mean that users would have a wider variety of visual data in which they could explore.

Another piece of future work that could be done on the Bitcoin Analytic platform is to create web-pages based on an individual block that the system would retrieve. This feature would increase the complexity of the system; however, it would also make the system much more interesting and interactive for the user. Another idea that follows this would be to implement block-exclusive graphs such as graphs that include transactions from a specific block.

Converting Satoshis into Bitcoin would also be an improvement to the current system as the current graphs use Satoshis as the value. This can often lead to messy graphs as well as hard to read graphs, especially if the user is not good at reading high numbers.

As previously mentioned, the speed of the system is a big weakness in the current state of the project. There are a number of ways that this could potentially be improved; the first way being to look at how graphs or web-pages could be pre-loaded or cached. Another way in which the speed of the system could be increased is to run the Bitcoin Analytic platform using a Bokeh server which would also increase the usability of the system as the user wouldn’t have to run the system in a virtual environment to use.

8 Conclusion

In summary, the final version of the Bitcoin Analytic platform was of a fairly high standard that met all but two of the ‘must’ criteria that was shown in the MoSCoW section of the report. The output of information from the system was of a high standard where the only weakness it had was with performance. The data that was shown to the user was okay, but a lot of improvements could be made here in the future with the implementation of new features and a clean up of graph axis’. The

current usability of the system is somewhat limited especially with the use of Bokeh, but as mentioned previously this could be greatly improved in the future. Overall, I am happy with the Bitcoin Analytic platform that I have created; the platform is visually pleasing and will be more so with future development, and the blockchain parser is good at retrieving the information that is needed from the first 140,000 blocks.

Using the agile methodology seemed to be extremely helpful with this project as there was no clear goal of how the system would turn out from the beginning of the project. Regular meetings with my supervisor also proved to be helpful especially with using the agile methodology as it enabled the Bitcoin Analytic platform to take shape slowly throughout the course of the project.

9 Reflection on Learning

I shall end this report by showing what I have learnt whilst undertaking this project. This will allow me to reflect upon my own performance and identify how I can transfer this performance to future situations.

Throughout the course of this project, I have learnt valuable new skills as well as improving old skills that can be transferred and re-used in the future. One skill that I feel that I have vastly improved on is the time-management of tasks and allocation of time in general. At the start of this project I gave myself unrealistic time goals that I felt could be reached before I even started to do the background reading. Throughout the course of the implementation phase I set myself realistic goals to reach as well as realistic expectations of how much work I could put into the project each day. This was improved on over time as I adjusted aspects of my day to fit around the project; then I was able to find the most effective way to manage my time.

Another skill that can be carried over upon reflection is the ability to pick out important details within big bodies of texts. My ability to do this has grown significantly throughout the course of the project as well as my knowledge on the subject. I feel that I can learn best when making notes and asking questions on areas in which I do not understand. This project has enabled me to realise how important note-taking and questions are especially on un-familiar and new topics

such as blockchain technology, which I will be sure to use in the future. It has taught me that if there is a lack of documentation on something, I should invest more time than what was originally planned.

I also feel that I have improved when it comes to critical thinking. Throughout the project I have been faced with decisions that would change the outcome of the project. I feel as though I am able to see the advantages and disadvantages that these decisions will have on the impact of the platform much clearer than I would have before undertaking the project. This skill was improved through by analysing how previous decisions had affected the project and whether these decisions could have been thought through more from all aspects of the system.

The final reflection on learning will be about the ability to work independently on a project. Undertaking a project of this magnitude was a new experience for me as well as a real eye opener. At times throughout the course of the project I felt lost in what I was supposed to be doing; working independently has made me realise the value of doing sufficient research and asking the right questions based on what you need to know. Working independently also showed me that I demanded too much from myself from the outset of the project; I know this because the ‘future work’ section of the report is not empty.

To summarise this section, it is fair to say that I have further developed many skills that I have learnt from past experiences. New skills have also been learnt which shows that there is always something that can be learnt upon reflection; the reflection of my performance has shown this, and I will be sure to carry and continue to develop these skills in the future.

References

[1] https://www.law.ox.ac.uk/business-law-blog/blog/2017/04/how-blockchain-technology-will-impact-digital-economy - [Online] [Accessed – 12/03/2018]

[2] https://hbr.org/2017/03/how-blockchain-applications-will-move-beyond-finance - [Online] [Accessed – 12/03/2018]

[3] https://techcrunch.com/2017/04/20/whats-keeping-cryptocurrencies-from-mass-adoption/ - [Online] [Accessed – 12/03/2018]

[4] https://www.investopedia.com/terms/d/doublespending.asp - [Online] [Accessed – 14/03/2018]

[5] https://www.nasdaq.com/article/byzantine-fault-tolerance-the-key-for-blockchains-cm810058 - [Online] [Accessed – 14/03/2018]

[6] https://bitcoin.org/bitcoin.pdf - [Online] [Accessed - 16/10/2017]

[7] https://bitcoinmagazine.com/articles/quick-history-cryptocurrencies-bbtc-bitcoin-1397682630/ [Online] [Accessed - 14/03/2018]

[8] https://www.coindesk.com/danish-police-claim-breakthrough-bitcoin-tracking/ [Online] [Accessed – 15/03/2018]

[9] https://www.coindesk.com/information/what-is-bitcoin/ [Online] [Accessed – 15/03/2018]

[10] https://www.coindesk.com/information/how-bitcoin-mining-works/ [Online] [Accessed – 16/03/2018]

[11] https://en.bitcoin.it/wiki/Transaction#Data [Online] [Accessed – 16/03/2018]

[12] https://coinsutra.com/bitcoin-private-key/ [Online] [Accessed – 16/03/2018]

[13] https://www.coindesk.com/information/how-do-bitcoin-transactions-work/ [Online] [Accessed – 16/03/2018]

[14] https://www.newgenapps.com/blog/impact-of-bitcoins-on-the-economy-banks-finance [Online] [Accessed – 18/03/2018]

[15] https://www.coindesk.com/bitcoin-nodes-need/ [Online] [Accessed – 19/03/2018]

[16] https://coin.dance/volume/localbitcoins [Online] [Accessed – 19/03/2018]

[17] https://coin.dance/ [Online] [Accessed – 19/03/2018]

[18] https://coin.dance/nodes/core [Online] [Accessed – 17/04/2018]

[19] https://coin.dance/nodes/share [Online] [Accessed – 17/04/2018]

[20] https://blockchain.info/ [Online] [Accessed – 17/04/2018]

[21] https://blockchain.info/charts [Online] [Accessed – 17/04/2018]

[22] https://live.blockcypher.com/ [Online] [Accessed – 17/04/2018]

[23] https://live.blockcypher.com/btc/ [Online] [Accessed – 17/04/2018]

[24] https://www.projectsmart.co.uk/moscow-method.php [Online] [Accessed – 18/04/2018]

[25] https://blockchain.info/api/blockchain_api [Online] [Accessed – 29/01/2018]

[26] https://bitcoin.org/en/bitcoin-core/ [Online] [Accessed – 01/02/2018]

[27] https://coinlogic.wordpress.com/2014/02/18/the-protocol-1-block/#more-8 [Online] [Accessed – 04/02/2018]

[28] https://en.bitcoin.it/wiki/Block_hashing_algorithm [Online] [Accessed – 05/02/2018]

[29] https://en.bitcoin.it/wiki/Script [Online] [Accessed – 07/02/2018]

[30] https://en.wikipedia.org/wiki/Endianness#Little [Online] [Accessed – 05/02/2018]

[31] https://docs.python.org/2/library/struct.html [Online] [Accessed – 05/02/2018]

[32] https://bokehplots.com/pages/about-bokeh.html [Online] [Accessed – 07/02/2018]