"Predicting the bitcoin price using trading indicators"

Thesis submitted for the B.Sc. Business Analytics,

Technological University Dublin, Aungier Street.

Course Code: TU912

by

James Freemantle

Supervisor: Neil O'Connor

Submission Date: 31/05/21

Declaration

I, James Freemantle, declare that the thesis entitled "*Predicting the bitcoin price using trading indicators"* was carried out by me for the degree of B.Sc. Business Analytics under the guidance and supervision of Neil O'Connor, Technological University Dublin, Aungier Street. The interpretations put forth are based on my reading and understanding of the original texts and they are not published anywhere in the form of books, monographs or articles. The books, articles and websites that I have made use of are acknowledged at the respective place in the text. This thesis is my own original work and contains no material that has been previously submitted, in whole or in part, for the award of any other academic degree.

James Freemantle Date: 31/05/2021

James Freemantle

Table of Contents

Abstract

Cryptocurrencies such as bitcoin are currently attracting traders due to volatile price movements that create trading opportunities. Traders often use indicators based on historical price and volume data to forecast the future price movement in an attempt to place a successful trade. The aim of this report is to study how capable these indicators are at predicting the price of bitcoin. Specifically, it investigates if the price of bitcoin can be predicted using solely indicators and machine learning algorithms.

To test the hypothesis – *'Indicators can accurately predict the price'*, two separate sets of indicators were selected and applied to machine learning algorithms. The first set of indicators includes custom indicators made by analysts in the bitcoin community using On-Chain data from the blockchain (MVRV ratio, NVT signal, Fee per Day). The second set of indicators are more commonly used traditional indicators in finance (RSI, MACD, OBV). The indicators were supplied to LSTM (Long-Short-Term-Memory) and MLP (Multi-Layer Perceptron) neural networks in an attempt to predict the price using indicators.

The models had varying results depending on the neural network and set of indicators. Based on my findings, indicators are able to predict the price of bitcoin with an accuracy of between 69.69% and 95.46%. I also found that On-Chain indicators outperformed the traditional indicators on their predictive capability. The On-Chain indicators on average returned a MAPE (Mean-Absolute-Percentage-Error) of 16.75% in comparison to the traditional indicators 24.32%. The On-Chain indicators also acted as better predictors of the direction of price movement than traditional indicators. The On-Chain indicators correctly forecasted the direction of price movement 83.41% of the time, whereas the traditional indicators were only able to predict the direction of the price 53.87% of the time.

Chapter One: Introduction

1.1 Problem Statement

Generally, the purpose of indicators are to monitor the performance of an asset. Some investors have developed trading strategies based on particular technical indicators values to help make a prediction on the price direction.

In this study I will identify and discuss properties of bitcoin which have an effect on the price, regardless as to whether or not the bitcoin price is justified, then investigate patterns and relationships between the price and indicators.

Specifically, I will investigate the accuracy in which indicators can predict the bitcoin price using neural networks. I aim to compare the usefulness of indicators traditionally used in finance with more recently established On-Chain indicators to determine their bitcoin price predictive capabilities.

1.2 Background Research and Motivation

Pseudonymous creator – Satoshi Nakamoto created the bitcoin network on January 9th 2009. In October 2009, an Internet exchange sold 5,050 bitcoins for $5.02, at a price of $1 for 1,006 bitcoins. This registers as the first purchase of bitcoin with money (Ammous, 2018). The 5,050 bitcoin purchased for $5.02 is now worth over $250 million. This is approximately a 50,000,000x price increase. Undoubtedly, cryptocurrencies such as bitcoin have become an attractive space for traders and speculators due to the opportunities that large volatility presents.

On February 19th 2021, the market capitalization of bitcoin reached 1 trillion US dollars, as the price per bitcoin rose to around $55,000. The market cap of bitcoin simply reflects the perceived value of the discovery of the first digital scarce monetary system. The true value of this discovery is debated in the context of many different fields of study such as software development, politics, economics and financial policy. For example, the ethics of printing money is debated politically, the scarcity element of bitcoin might attract investors who are concerned about inflation. Gold and bitcoin are both scarce and hard to reproduce, hence bitcoin is often sometimes referred to as a ‘digital gold’.

Michael Saylor, CEO of Microstrategy, has referred to the company’s treasury assets as ‘a melting ice cube’, a metaphor for the gradual loss of purchasing power of the dollar through inflation. Microstrategy has purchased over 91,579 bitcoin as a ‘store of value’ to protect their capital from inflation (Microstrategy, 2021, p.4). Other large companies have followed Microstrategy in purchasing large amounts of bitcoin on their balance sheets such as Square and Tesla. As of writing this report, Square has purchased a total of 8,027 bitcoin (Bursztynsky, 2021) (Square, Inc., 2020) and, according to an SEC filing, Tesla has purchased 43,200 bitcoin (Kovach, 2021). The new wave of institutional investors who are buying bitcoin for their treasury has brought a new wave of media attention as speculators and retail investors attempt to interpret the true value of bitcoin. A Forbes article posted in November 2020 reveals that a Citibank Report suggests that the bitcoin price could reach $300,000 by the end of 2021 (Bambrough, 2020).

A letter to shareholders of Aker from Kjell Rokke announced the establishment of Seetee, a company dedicated to investing in projects and companies in bitcoin (Rokke, 2021). In the

shareholder letter, Kjell notes that 'scarcity drives price', referring to the law of supply and demand (Gale, 1955). As will be further explained in the literature review, the 'halving' event in bitcoin occurs approximately every 4 years, this event gradually reduces the introduction of new bitcoin into existence. I find that there does not exist much research into the relationship between the price movements and the halving event. I find an analysis into the economics of bitcoin's cyclic supply reduction could give insight into the cyclic nature of the price and hence enhance the predictability of price and indicator values.

1.3 Aim of Research

The primary aim of the research is to study the relationship between price and traditional indicators, and to predict the price using machine learning algorithms. This report aims to answer the following research question:

Do technical indicators act as reliable predictors for the bitcoin price?

To achieve this aim, the following objectives have been set for the work:

- Identify indicators that are most commonly used by traders and investors to predict the bitcoin price.
- Identify traditional indicators that are most commonly used by traders and investors to predict stock prices.
- Investigate bitcoin's fundamentals in relation to price.
- Examine the relationship between halving cycles, price and chosen indicators.
- Apply Machine Learning Algorithms to the indicators to investigate if price can be accurately predicted.

1.4 Structure of the Study

This report consists of five chapters, namely Introduction, Literature Review, Methodology, Results & Findings and Conclusion. Chapter 1 provides the background to the study as well as the main aim and objectives. The Literature Review chapter is responsible for reviewing and discussing appropriate methods of valuing bitcoin and the metrics and indicators used by analysts to predict the bitcoin price. The Methodology chapter discusses the indicators and machine learning algorithms applied for this project. In Chapter 4, the obtained results and findings using the methodology of the previous chapter are presented, interpreted and evaluated. Finally, in Chapter 5, a critical discussion of the key findings and results are provided.

1.5 Scope of the Study

The scope of this study remains focused solely on indicators in relation to price. A particular indicator derives its usefulness from various parameters including time series data on price, volume and others. Regardless of which parameters are used to create an indicator, there will be a link between price and the indicator value. Since price is determined by supply and demand factors, it remains within the scope of the study to discuss bitcoin's fundamentals of supply and demand. However, the supply of bitcoin is guaranteed by the bitcoin protocol/code and therefore this report will have more of a focus on supply than demand.

Chapter 2: Literature Review

In conducting the literature review, I considered the scope of the study and therefore have researched the following particular areas; Bitcoin's fundamental properties in relation to price, metrics and indicators used by analysts to predict the bitcoin price, metrics and indicators used by analysts to predict the price of stock prices. This remains a narrow focus on the indicators due to relationship between an asset's economics, price and indicator values.

The definition of a technical indicator is a heuristic or pattern-based signal produced by the price, volume, and or/open interest of a security or contract used by traders who follow technical analysis (Chen, 2021). Since technical indicators patterns are conceived by price history patterns, it should be instructive to study bitcoin's price history from which the indicators were formed. The price of bitcoin is derived from perceived value, as Kristoufek (2015) shows that bitcoin price cannot be explained by economic theories, it is instead driven by speculation. In the same vein, Yermack (2015) concludes that bitcoin resembles more like a speculative investment than like a true currency.

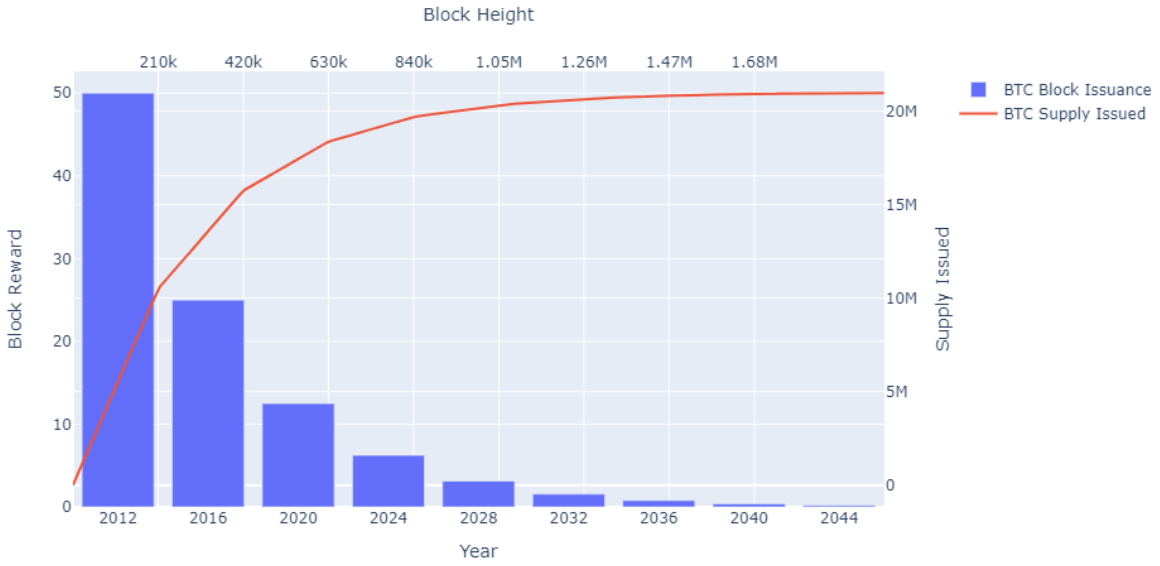
2.1 Bitcoin's Fundamentals in relation to Price

2.1.1 Supply

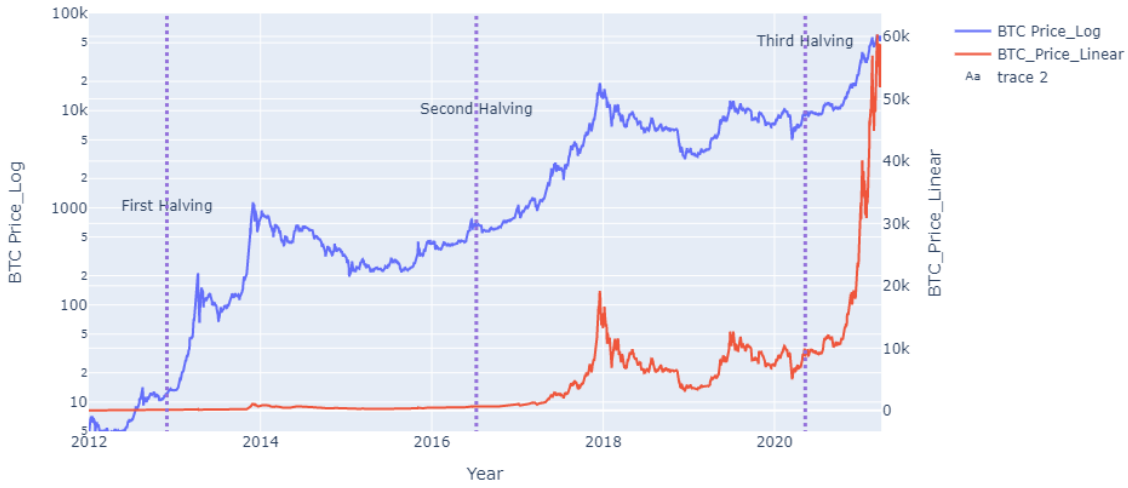
The price of any tradable asset is derived from the forces of law of supply and demand (Kramer, 2021). The supply of bitcoin is fixed and well understood by market participants. It is generally assumed to practically unchangeable as network consensus would be needed to change the supply (Nakamoto, 2009). The law of supply and demand suggests that consistent demand combined with a declining supply will result in a price increase. By comparison demand for bitcoin are less self-evident.

Buchholz et al. (2012) argued that one of the most important determinants of bitcoin price in the long-run is the interplay between bitcoins' supply and demand. (Meynkhard, 2019). Within the context of bitcoin, research results show that reducing issuance by half every four years leads to an increased market value of the cryptocurrency.

The halving (also known as the halvening) event is a reduction in the level of coin issuance. This event occurs every 210,000 blocks mined. A number of bitcoin is rewarded to the miner who successfully mines a new block of transactions. Mining blocks is the only way new bitcoin come into existence. Every 210,000 blocks (or approximately 4 years), the block reward is halved from the previous block reward. Bitcoin's starting block reward was 50 coins per block, so far three halving events have been completed.



As depicted in the visualisation, the block reward halves every 4 years. The new amount of bitcoin entering the network through miners is reduced in a logarithmic pattern. The reduction in the block reward is constant at 50% each halving, therefore investigating the price movements on a logarithmic scale would make sense.



Bitcoin miners expend large amounts of electricity in order to mine bitcoin. Miners profit from bitcoin when the value of the block reward is larger than the cost of electricity. When a halving occurs, the miners must adjust the price at which they are willing to sell in order to avoid incurring a loss. The halving event can result in miners accumulating bitcoin until the price increases to the target price at which they profit. This results in less coins reaching the exchanges and a rise in price where the exchange supply meets the demand (Hertig, 2020).

Some investors speculate that the halving cycle has an effect on price due to the history of a dramatic price increase after a halving. The brief review of the price history between halving events shows an obvious trend in which the price starts to increase. As previously stated, an

asset price is determined through supply and demand, since demand factors are less self-evident, the halving event is not guaranteed to have any effect on price.

Pseudonymous analyst Plan B created a model to predict the future price of bitcoin using solely existing supply and new issuance as parameters. This model is 'the Stock to flow model' which can be calculated on any asset where these parameters can be estimated. The Stock to flow ratio of bitcoin can be calculated by taking the existing amount of bitcoin in existince and dividing by annual flow/introduction of new supply, which the time of writing this report is over 18,713,000. The current annual flow of bitcoin is approximately 328,500btc (6.25 bitcoin is mined roughly every 10 minutes). The current stock to flow ratio can be calculated as 56.96 (18,713,000/328,500). However, with each halving the new inflow of bitcoin is reduced by half, thereby reducing the denominator by half every 4 years, the stock to flow ratio increases over time. Plan B's stock to flow model is a time series plot of the stock to flow ratio of bitcoin and the USD price. Investors refer to the Stock to Flow model often due to its accuracy so far (PlanB, 2019).

Later in 2020, Plan B published a new model titled "Bitcoin Stock-to-Flow Cross Asset Model". This model notes that as the market capitalisation of bitcoin increases, the narrative of bitcoin's purpose has transitioned. Plan B noted that the monthly data points plotted the stock-to-flow ratio and market capitalisation axis resulted in 4 distinct clusters. Plan B identified that these 4 four identified clusters of monthly data points are consistent with halvings and changing narratives. The transition periods that are consistent with halvings cycles and narratives are as follows: Proof of concept (S2F 1.3 and market value $1M); Payments (S2F 3.3 and market value $58M); E-Gold (S2F 10.2 and market value $5.6B); and Financial asset (S2F 25.1 and market value $114B). Gold and Silver were added to the chart which forms a perfect straight line between the 4 clusters and the precious metals (PlanB, 2020).

2.1.2 Demand

The demand for bitcoin is discussed by Tyler Winklevoss, CEO of regulated instituition Gemini in an article titled "The Case for $500K Bitcoin" (Winklevoss, 2020). The report discusses the problems with Oil, Gold and Fiat money (specifically the US Dollar) as reliable stores of value. The findings of the report state that bitcoin is superior to these other assets in terms of the following properties of money: Scarcity, Durability, Portability, Divisability, Storage and Counterfeit difficulty. According to Winklevoss, bitcoin's superiority to Gold as an asset in these listed properties would justify a bitcoin market cap of at least equal to gold's. At Gold's market cap, this would mean the bitcoin price would appreciate to approximately $500,000.

Cryptocurrencies such as bitcoin are still a new asset class, and are as yet unproven to be a reliable store of value. However, the demand for bitcoin extends beyond speculation where bitcoin is being used by citizens in countries experiencing hyperinflation. According to a BBC News report, bitcoin is being used by citizens in Venezula as an alternative to the Venezuelan Boilivar (Di Salvo, 2019). A study exploring the determinants of bitcoin's price found that "Bitcoin behaves as a financial instrument with a tendancy to be a speculative asset, but this study also shed light into the existance of an increasing mass of potential users as a currency or capital flight instrument"(Poyser, 2017). In the short term, speculation largely explains the demand for bitcoin. While in the long term, demand might be driven by expectations regarding bitcoin's future utility as a medium of exchange (de la Horra, de la Fuente, & Perote, 2019).

2.2 Overview of Similar Work

Attempts to predict the price movements of bitcoin using various machine learning algorithms and indicators have been attempted by scholars in the past. The accuracy of prediction varies based on the studied algorithm, algorithm configuration, indicators selected and the time-series data applied to the model. Based on my research, neural networks have been the most frequently used machine learning algorithm to predict the price movements of cryptocurrencies. A recent study investigated the suitability of neural networks for trend prediction in cryptocurrencies using technical indicators (Alonso-Monsalve, Suárez-Cetrulo, Cervantes, & Quintana, 2020). They found that six cryptocurrencies tested in this study were predictable using technical indicators, in terms of the models they applied, LSTM (Long Short Term Memory) neural networks outperformed other types of neural networks. LSTMs have also been used to predict the movement of cryptocurrencies. A particular paper investigated the price predictability of alternative cryptocurrencies, Litecoin and Monero using an LSTM neural network (Patel, Tanwar, Gupta, & Kumar, 2020). The results also showed that the model can predict the price movement of these currencies with high accuracy.

Neural Networks were used in a similar study by McNally, Roche and Caton (McNally, et. al, 2018) who implemented a Recurrent Neural Network (RNN) and a Long Short Term Memory network to predict the direction of the bitcoin price. All models used in that report struggled to effectively learn from the data, both neural networks models had close to a 50% classification accuracy, although their models used price history as training data instead of indicators. The predictability of indicators have been tested by another study which uses a large amount of indicators generated by a technical analysis library. The study looks into the ability to predict future trends of a stock using price and these technical indicators (Nelson, Pereira, & de Oliveira, 2017). The results from this study showed an average of 55.9% accuracy in predicting whether a stock was going to go up or down.

The indicators used in these studies were traditional and commonly used for stocks. However, I researched indicators that were created and used by the bitcoin community and I found that some indicators were custom created by analysts based on On-Chain (blockchain) data. Very little academic work exists in the area of using these custom indicators as predictors. One study analyses portfolio optimization of cryptocurrencies using custom indicators such as Network Value to Transaction Volume Ratio (NVT) and Market Value to Realized Value Ratio (MVRV) (Gu, Jiang, & Su, 2021). Through the use of models such as Convolution Neural Networks and a Visual Geometry Group Network, the study found that portfolio results improved 11.05% where NVT was used as a feature in comparison to a previous study where NVT was absent.

Based on my research from similar studies, I will compare the predictability of On-Chain indicators with more commonly used Traditional indicators using neural networks.

2.3 Indicators used by analysts to predict price movements in stock markets

In comparison to the custom indicators and metrics used by bitcoin, a larger amount of study exists investigating traditional indicators and their predictive ability in the stock market. Unlike the indicators produced by bitcoin analysts, the indicators discussed in this section can be used for many other asset types since they do not require On-Chain data. This section will introduce these more traditional and commonly used indicators, and will discuss studies in which they are used.

2.3.1 RSI

Relative Strength Index (RSI) is a trading indicator developed by J. Welles Wilder in the 70s. It is a momentum indicator that measures the velocity of price movements (rate of change between positive days and negative days). It is commonly used in trading to identify overbought/oversold assets. The output returns a value between 0 and 100, where high values are considered overbought (sell signal) and low values are considered oversold (buy signal). The RSI indicator compares the price on days that close higher than the previous day's close with the price on days that close lower than the previous days close. The formula takes the last 'n' periods and divides the last gross positive change per period by the gross negative change per period. This means that the RSI indicator will be high in the presence of frequent positive days within the specified period and vice versa. Increasing the time period value of 'n' will mean the indicator is likely to be smoother and less sensitive to violent price fluctuations.

The effectiveness of RSI to successfully trade assets has been studied extensively. The RSI indicator was tested on the Singapore Stock Exchange. The results of this study showed the RSI indicator returns a significantly positive profit (Wong, Manzur, & Chew, 2003). In another study that tested the validity of the RSI in Indian stock markets, in most cases the RSI proved positive (Bhargavi, Gumparthi, & Amith, 2017).

2.3.2 MACD

The Moving Average Convergence Divergence (MACD) trading signal was created by technician Gerald Appel in the 1970s (Appel, 2005). A line plot of the distance between the 26-period Exponential Moving Average (EMA) and the 12-period EMA forms the MACD line. The resulting line plot of this calculation is used in conjunction with the 9-period EMA as a signaling line. The convergence and divergence between the MACD line and the signal line is used by traders to identify trading opportunities.

The underlying theory is that the shorter 9-period moving average (signal line) will be more sensitive and act faster to price movements than the slower MACD line. When the signal line crosses the MACD line in either direction, this is seen by traders as a sign that the current trend is reversing. Hence, the crossover between the MACD and the MACD signal line is an opportunity for the trader to buy or sell the asset.

Like most momentum indicators, MACD analysis is more effective in trending markets rather than stable markets. Volatile markets such as cryptocurrencies that experience significant volatility could be an effective, and so for this reason MACD is selected.

A study was conducted that investigated the use of the MACD indicator in a trading strategy applied to the financial market of Serbia. It was concluded that from the results of this study that the MACD indicator "significantly contributes (to the) maximization of profitability on investments" (Eric, Andjelic, & Redzepagic, 2009). Many other studies have used MACD and RSI in conjunction with each other to develop and test trading strategies. For example, a study investigating the profitability of the RSI and MACD was tested on the London Stock Exchange FT30 Index. The results showed that the indicators can both generate higher returns than a buy-and-hold strategy (Chong & Ng, 2008).

2.3.3 OBV

The On Balance Volume (OBV) is momentum indicator developed by Joe Granville that uses volume flow to predict price movements (Granville, 1964). "Among the signals traders may add to online charts, one of the most popular is on balance volume. This indicator is of the greatest value when the indicator diverges from price. This indicates a likely coming reversal" (Thomsett, 2017). Granville theorized that volume precedes price and could therefore be used as an indicator to identify buying or selling pressure before the price has reacted. The purpose of OBV is to ascertain whether the price movement is supported by trading volume in the direction of the price.

The calculation of OBV is a running total of positive and negative volume. The period's volume is added to the OBV when the period's price finishes above the previous period's close. Inversely, the period's volume is subtracted from the OBV when the period's price finishes below the previous period's price. Despite being measured numerically, the OBV value itself is of little value to traders in comparison to the OBV line.

For example, a bullish (uptrend) signal occurs when the OBV diverges upward from the price. This would indicate that more volume is being traded on days where the price has risen from the previous day in comparison to the volume traded on days where the price has fallen. In other words, 'bearish' investors (investors expected a downtrend) who are selling are being exhausted and a price rise is expected.

On OBV's success as a trading indicator, Tsang and Chong conducted a study using OBV measuring the turnover of 9 major stock indices around the world (Tsang & Chong, n.d.).

2.4 Indicators used by analysts to predict price movements in bitcoin

New methods of modelling and predicting the price of bitcoin have been developed within the last few years. Considering that the asset class is still relatively new and the subsequent lack of long term data, many analysts are developing models which have so far been unproven to be accurate. In this section I will discuss models that have been created for the sole purpose of predicting the price of bitcoin. Analysts in the bitcoin community have created custom indicators to value bitcoin using On-Chain data (data publicly available on the blockchain) rather than indicators which are solely built from price.

2.4.1 The MVRV Ratio

In October 2018, Mahmudov and Puell introduced the Market-Value-to-Realized-Value (MVRV) ratio (2018). The MVRV Ratio was created to measure bitcoin's market value while accounting for coins that are lost or being held for the long term. To account for this, the MVRV ratio uses the 'Realized value' of coins instead of the 'Market value'. The MVRV-ratio is the market value divided by the realized value. The market value is the current last known price multiplied by the current circulating supply. The realized value differs significantly from the current market value. Specifically, it adjusts for lost and unmoved bitcoin and also acts as an indicator for when long-term holders bought bitcoin. Realized value is calculated by summing the products of price per bitcoin and UTXO (Unspent Transaction Output). The conventional method of calculating market capitalization is the last known price multiplied by circulating supply. However, the conventional method inflates the capitalization as there are many bitcoin

either unmoved for years or lost/unused. Hence, the realized market capitalization represents the current "real" value of bitcoin, by valuing bitcoin depending on when it was moved for the last time. By dividing market value by realized value, an indication of bitcoin's real value emerges. A 1:1 MVRV-ratio would indicate that bitcoin would be at its realized value. A MVRV-ratio below one indicates that bitcoin is undervalued, based on these principles. Mahmudov and Puell find that, historically, a MVRV-ratio above 3.7 denotes overvaluation and a MVRV-ratio below one indicates undervaluation (Puell, 2018)

2.4.2 Realized HODL Ratio

In 2020, the Realized HODL Ratio was published by Phillip Swift using On-Chain data to identify extremes in market psychology over time to accurately identify bitcoin global cycle highs and lows. The indicator uses the same 'Realized value' metric as the MVRV-ratio. The indicator first groups transactions by 'age' (the last time they were moved) and compares when shorter-term groups are worth considerably more or less verses longer-term groups, dividing the short term groups by the long term groups results in the Realised Hodl Ratio. The ratio closely mirrors the price because a bull market (uptrending) attracts more short-term traders which increases the numerator to the ratio. Inversely, as bitcoin enters a bear market (downtrending), the reduction of short-term traders decreases the numerator (Swift, 2020).

2.4.3 NVT Signal

The NVT-ratio (Network Value to Transactions) was first introduced as a tweet by bitcoin analyst Willy Woo in February 2017. But was further explained in an article in Forbes later that year in October (Woo, 2017). The indicator is formed from the ratio between Network valuation - also known as market cap, and the total Daily Transaction Value. The theory behind the NVT ratio is that it is a price-earnings ratio equivalent, Willy woo states that "Since bitcoin at its essence is a payments and store of value network, we can look to the money flowing through its network as a proxy to 'company earnings'." A high market cap relative to the transaction value flowing through the network would result in a higher NVT ratio.

The NVT ratio was improved by Dimitry Kalichkin in 2018, using Woo's work, Kalichkin developed the NVTS (Network Value to Transactions Signal). The NVTS provides more emphasis on predictive signaling ahead of price peaks than the original NVT ratio (Woo, 2018). The NVTS ratio uses the 90-day moving average (MA) of the Daily Transaction Value instead of the Daily Transaction Value as the denominator of the ratio. Consequently, the denominator – 90day MA is less reactive to movements of the numerator - Network Valuation (market cap), which in turn makes the NVTS a more volatile ratio than the NVT.

2.4.4 Network Fees

The miners purpose is to approve transactions, which is achieved approximately every ten minutes when a successful block of transactions is mined. However, there is limited space in each block, and therefore only a limited amount of transactions can be approved in each block. Transactions are not approved on a first-in-first-out basis, they are instead prioritized by highest fee per transaction. The fee sent to the miner for the transaction is determined by the sender, and subsequently reduced from the transaction amount.

Miners are financially incentivized to mine a block that contain transactions with the highest fees. Transactions that are sent with fees not large enough to be included in the next block of

transactions are sent to the Mempool (short for Memory Pool of transactions). When the network is encountering a high volume of transactions, users are competing for block space and therefore transaction fees can rise significantly. The total fees per block are an indication of the volume of transactions it is experiencing.

It is important to distinguish the difference between On-Chain transaction volume and exchange transaction volume. On-Chain volume reflects the sum of bitcoin addresses being transferred to other bitcoin addresses through mining. An exchange such as Coinbase, Bitstamp or Kraken record volume as the sum of bitcoin being bought and sold through their platform in exchange for fiat money or other coins. Exchange customer may choose to withdraw bitcoin to a personal address, this will then be recorded as an On-Chain transaction because the transaction will need to be mined in a block. Therefore On-Chain transaction volume and exchange volume are separate from each other.

2.5 Machine Learning Algorithms that are used for predicting price movements.

Stock market prediction using algorithms is an area of finance and economics that has been extensively explored by researchers. Numerous variations of algorithms and experiments have been applied in efforts to create accurate time-series forecasting models to predict the price of stocks. For the purpose of this report, I will be exploring previous work investigating the accuracy of various algorithms in stock markets. Furthermore, I will briefly discuss the composition and predictive features of each of these algorithms.

2.5.1 MLP – Multi Layer Perceptron

An Artificial Neural Network (ANN) is one of the least complex forms of neural networks because all of the features are processed in one direction. An ANN is a group of multiple perceptrons at each layer, where a perceptron is an artificial neuron. Each perceptron consists of weights and an activation function, which takes the weighted sum of inputs and outputs in order to calculate if weights need to be adjusted to match the expected output. A MLP is a type of ANN that has at least three layers: an input layer, one or more hidden layers and an output layer. Since the output layer of nodes has no connection to the input layer of nodes, the hidden layer is responsible for computing the value of the activation function from the input layer. Based on this computation, the neuron is either fired (activated) or not, which modifies the result value before proceeding to the succeeding neuron. Predictions are made on the output layer, also called the visible layer.

The predictive nature of MLPs and ANNs have been tested by numerous scholarly articles to forecast time series data. In terms of the price movements in stock markets, scholars in Malaysia created an MLP based stock forecasting model in order to help investors time buy and sell transactions to maximize profits. The results from the paper concluded that there was good agreement between the prediction model and the data (Yassin et al., 2017). An ANN classifier was used in a paper to discriminate between stocks that provide positive returns and those that provide negative returns. Their model correctly classifies the stocks 72% of the time.

2.5.2 LSTM – Long Short Term Memory

Recurrent Neural Networks (RNNs) are different from traditional feed-forward neural networks such as an ANN because they can store information (maintain an internal state) for a duration (Bengio, Simard, & Frasconi, 1994). This ability to remember the context of the data as data is being passed through the algorithm is what makes the algorithm more resistant to noise than other feed-forward neural networks.

A Long Short-Term Memory (LSTM) network is a type of recurrent neural network designed to be more capable of remembering values over larger intervals than an RNN. The architecture of an RNN network is less complex than an LSTM network. In an RNN network, each cell takes two inputs, the output from the hidden state and the new observation. The feature which gives LSTM the ability to remember lies in the 'cell state', information from previous intervals are stored in the cell state through recursion (Kang, 2017). A paper proposing an improvement to LSTMs architecture was published in 2000. The paper proposed that the cell state needed to also have the ability to forget as well as remember, this allows the cell to dispense of information at appropriate times (Gers, Schmidhuber, & Cummins, 2000).

2.6 Summary

This chapter discusses some of the literature available concerning bitcoin's fundamentals, indicators and machine learning algorithms. I feel that all the literature discussed in this chapter is relevant to the scope of the study. The discussion of literature in this chapter should give the reader an adequate understanding of the study in order to fully comprehend the logic behind the methodology and to be able to interpret results.

Chapter 3: Methodology

This part of the report will discuss the empirical approaches used for analysis, the indicators selected and the machine learning algorithms considered during the research analysis. The purpose remains to investigate the relationship between the bitcoin price and these indicators for the purpose of forecasting with machine learning algorithms.

3.1 Indicator Selection

The hypothesis for this report is that indicator values can predict the price, however different indicators are likely to have different predictive capabilities. For this reason, two sets of indicators were selected for the purpose of this report. Each of the indicators used are discussed in detail in the Literature Review.

The first set contains traditional indicators, namely the RSI (Relative Strength Index), MACD (Moving Average Convergence Divergence), and OBV (On Balance Volume).
The second set of indicators were produced using On-Chain data, namely the NVT signal, MVRV ratio and the Block Fee.

The indicator for the block fee used in this report is the sum of all fees paid to miners per day. In this context, the fee paid to miners does not include the block issuance because this value is constant for each halving cycle and therefore does not provide any insight to the predictive nature of the indicator.

3.2 Indicator Analysis

In order to effectively examine the predictability price and indicator values, it is necessary to first identify patterns that exist in the price data. The cyclical nature of bitcoin's halving cycle gives an opportunity to divide the total price history into subsets to identify patterns in price across these subsets. I will first run a correlation analysis between the halving cycles, calculating the difference between cycle tops and bottoms to compare halving cycle bull runs (uptrends).

Following analysis of bitcoin's price across halving cycles, attempts to recognize indicator value patterns across halving cycles are made.

The predictive capabilities of indicators such as MACD and RSI can be tested without the use of machine learning algorithms. A trader can use these indicator values alone as an indication for a trade opportunity. As discussed in the literature review, when the MACD line crosses the MACD signal line, this is an indication to either buy or sell. Based on this, the profitability of using this indicator alone is tested against a buy and hold strategy.

Similarly, the RSI value is used by traders to signal a buy and sell opportunity. As discussed in the Literature Review, the RSI indicator requires a 'look back' parameter to determine the amount of previous time periods to consider. The most commonly used 'look back' period for the RSI indicator among traders varies depending on the time unit selected. The most common daily and weekly look back period is set to 14, whereas the monthly look back period of 12 is most common. For this report, these most common look back periods were used. An RSI value above 70 is considered as an indication that the market is overbought, while an RSI value below

30 is considered oversold. The MACD and RSI values are tested separately against a buy and hold strategy, to determine their returned profit/loss (as a percentage of initial capital). As percentage of initial capital invested is measured as the profit or loss, the starting capital invested for each year is extraneous. The capital invested is reset at the start of each year (January 1st), this is to measure the indicators performance in both bull markets and bear markets. These year on year profit/loss figures will be compared to a year on year buy and hold strategy to offer a fair comparison.

OBV and On-Chain indicators do not have straight-forward buy/sell signals like the RSI and MACD. These indicators are instead often used in conjunction by traders to gain a deeper insight and understanding of the asset. For this reason, developing a trading strategy for these indicators is much more complex and outside the scope of this study. However, these indicator values are tested for correlations across halving cycles. The price correlations across halving cycles are compared to the indicator values correlations across halving cycle.

3.3 Data Collection

For the purpose of investigating all 4 halving cycles so far, this report requires price data from the first halving cycle. Since exchanges are a source of price and trade history, I investigated the most popular exchanges in bitcoin's history. The most popular exchange in the early years of bitcoin was Mt.Gox who handled 70% of all bitcoin transactions (Vigna, 2014). However this exchange was hacked and bankrupted in 2014. Another exchange, 'Bitstamp' was also operating as an exchange during bitcoin's first halving cycle and has been operating since. Thus Bitstamp data was used for this report.

The complete Bitstamp trade history was collected from data science platform Kaggle. The data was created by user 'Zielach' as a solution to bypassing the various exchange APIs. The data was then further cleaned by user 'kognitron' by removing Null values and adding a DateTime column. This dataset ranges from the 31st December 2011 to the 14th September 2020.

A different source of bitstamp's trade history that includes present data was used to make up for the lack of data after September 2020. These two datasets were filtered (no overlapping days) and concatenated together to form a single dataframe containing Bitstamps trade history from the 31st December 2011 to 22nd May 2021.

The exchange data was used for consistent and extensive price data for creating the first set of indicators. The indicators RSI and MACD were created using the library 'stockstats'. Whereas the OBV indicator is created manually because stockstats does not support OBV.

The second set of indicators cannot be generated from exchange data with the use of a library because the indicators require information from the blockchain. The second set of indicators were sourced from coinmetrics.io (https://coinmetrics.io/community-network-data/). Bitcoin price data from July 2010 is available on coinmetrics.io, however this data is not used as part of the report since exchange volume data is required to create the OBV indicator.

3.4 Neural Network Methodology

The literature review findings gave insight that neural networks are one of the most commonly used machine learning algorithms for stock market prediction and time series forecasting.
The MLP and LSTM neural networks discussed in the literature review are used for the methodology of this report. The machine learning libraries 'sklearn' and 'keras' are used in python to produce these models.

This report investigates whether the bitcoin price can be predicted using indicator values. The two separate sets of indicators supplied to these models are the 'Traditional Indicators' and 'On-Chain Indicators'. For each set of indicators, The MLP and LSTM models are trained using price (y) and indicator values (x) on each halving cycle. The model predictions of the price based on the test indicator data are plotted and evaluated.

The Mean Absolute Percentage Error (MAPE) is used for model evaluation. This is because the Mean Squared Error (MSE) will produce incomparable results between halving cycles. For example the 2009-2012 halving cycle's price is much lower than the 2020-2024 halving cycle price, so a model's predictions will likely incur a larger MSE for predictions on the 2020-2024 cycle than earlier cycles. In other words, since the price is relative to the halving cycle, the model's evaluation must also consider error relative to the price. Hence, MAPE is selected as an appropriate model evaluation method considering the problem.

The MAPE will calculate the difference between the test data and the model prediction. However, accurate model evaluation should credit the model where the predicted direction of price movement is correct, but the model either exaggerates or understates the amount in which the price moves. In this circumstance, the MAPE is less useful as an evaluation technique because the absolute error is calculated even though the direction of the price movement is accurate. To account for this, I use binary classification. For both the test and prediction data of the neural network, the direction of the price movement from the previous day to the next is represented by either a 0 or a 1. The accuracy score between test data's price movement and the model's predicted price movement is calculated.

Chapter 4: Results and Findings

This chapter will discuss the results of the report. The results are presented in three sections; halving cycle analysis, indicator analysis and machine learning. The halving cycle analysis discusses the notable patterns between the price movements during each cycle. Using these findings, the indicator values are discussed in the context of halving cycles. The RSI and MACD trading strategies are simulated to gauge their predictive capabilities. Following an indicator analysis, neural networks are used to predict the bitcoin price following each halving cycle.

4.1 Halving Cycle Analysis

Below is a chart of the bitcoin price (in USD) performance of each halving cycle. The block reward is also plotted alongside the halving cycle to allude to the possible effect of reduction of reducing the new supply. The final column calculates the percentage increase between the third column (The price at the start of each halving) and the fourth column (the price peak during the halving cycle). As of writing this report, all data is accurate.

Table 1: Relative performance of bitcoin price performance from the start of each halving cycle to the price peak during the same time period

Halving Cycle	Block Reward	Price at the start of the halving cycle	Price peak during the halving cycle	Price percentage increase
2009-2012	50 BTC	$0.17	$31.50	18,429%
2012-2016	25 BTC	$12.22	$1,178	9,539%
2016-2020	12.5 BTC	$657.61	$19,800	2,910%
2020-Present	6.25 BTC	$8705	$63,480	629%

Looking at the ‘Price percentage increase’ column, the percentage increase from one halving cycle to the next is a lesser increase from the cycle before it. The price percentage increase, during the 2012-2016 halving cycle is approximately half (51.8%) the price percentage increase from the 2009-2012 cycle.

Comparing this to the ‘Block Reward’ column which exactly halves (50%) every 4 years, we can see that there is a distinct pattern between the price percentage increase column and the reduction of supply.

Visualisation 1: Bitcoin's price movement in each Halving Cycle

The correlation between each halving cycle is shown in the table below. Note that the 2009-2012 and 2020-2024 halving cycles are significantly shorter than the other halving cycles. The correlation is calculated for these cycles using the first number of days in the halving cycle to match the length of the shorter cycle.

Table 2: Correlation matrix of the bitcoin price between halving cycles

	2009-2012 Cycle Price	2012-2016 Cycle Price	2016-2020 Cycle Price	2020-2024 Cycle Price
2009-2012 Cycle Price	1	0.53	0.75	0.87
2012-2016 Cycle Price	0.53	1	0.55	0.50
2016-2020 Cycle Price	0.75	0.55	1	0.89
2020-2024 Cycle Price	0.87	0.5	0.89	1

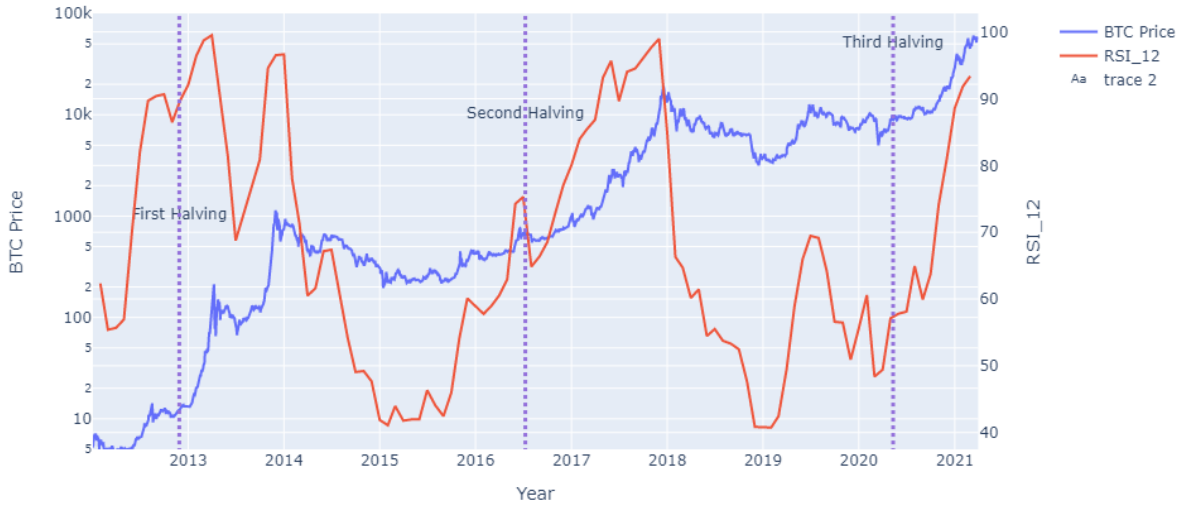
The correlation results show that the bitcoin price across halving cycles has medium – high correlation. It is impossible to explicitly prove that a reduction of newly issued supply has a direct effect on price because correlation does not prove causation. However, we have established that based on past data, the bitcoin price moves cyclically. Indicator values can now be examined as to whether they mirror a similar cycle to the price.

4.2 Indicator Analysis

4.2.1 RSI

Below is a visualization of the monthly RSI_12 data points in conjunction with the bitcoin price on a Logarithmic scale. The halving dates are also plotted vertically on the x axis.

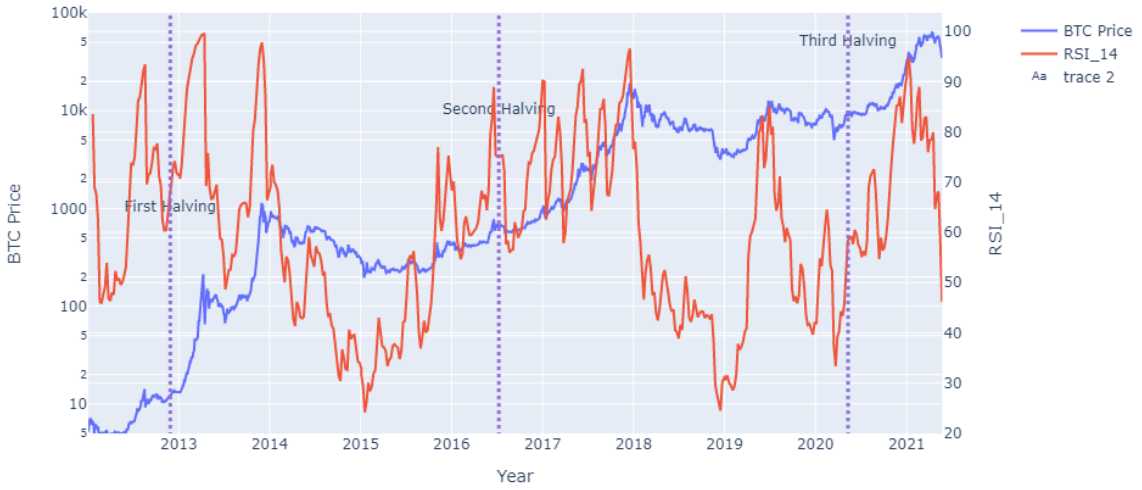
Visualisation 2: Bitcoin's price movement and RSI 12 values with monthly data



Monthly RSI_12 values have ranged from 40-100 during bitcoin's history. In traditional theory, an RSI value at 70 indicates that the asset is overbought (sell signal) and 30 indicates that the asset is oversold (buy signal). However, based on this theory, the monthly RSI for bitcoin has never produced a buy signal, and has spent the entire year of 2017 producing a sell signal. The correlation between the monthly RSI figures split by halving cycle range from a positive correlation of 95% to a negative correlation of 86% depending in the halving cycle. Overall the correlation between the monthly RSI and the price is 21%, which is relatively low.

The correlation analysis is repeated with RSI_14 Weekly and RSI_14 Daily figures.

Visualisation 3: Bitcoin's price movement and RSI 14 values with weekly data



Weekly RSI values decline below 30 for some time in 2013 and 2019, this is due to the reduction of the extreme price increases that are present in the lookback period, so the RSI values are more sensitive to such price changes. The RSI correlation across halving cycles continues to heavily vary from positive 61% to negative 67% based on halving cycle. The overall correlation between weekly RSI and price is also 21%, the same as monthly correlation. Daily RSI values are much more volatile and range from 10-100, this gives traders more of an opportunity to use RSI as in indicator for buy and sell signals. The correlation between daily RSI values and the price is 2% which is very low, but this is expected as RSI does not consider any price movements before the specified look back period – in this case, 14 days.

The predictability of the Daily RSI is tested using a simulated trading strategy as described in the methodology. Trades are executed if the RSI value crosses the upper or lower bound which are set to 70 and 50 respectfully. The lower bound is set at 50 to balance the amount of 'buy' signals with the amount of 'sell' signals. The chart is produced with the columns as follows:

'RSI Return'- The capital returned year on year from the RSI trading strategy.
'RSI P/L' – The profit or loss made from the RSI strategy (RSI Return – Capital).
'B&H Return' – The capital returned year on year through a simple Buy and Hold strategy.
'B&H P/L'–The profit or loss made from the Buy and Hold strategy (B&H Return – Capital).

Table 3: Chart of RSI 14 returns against a Buy and Hold Strategy

Year	Capital	RSI Return	RSI P/L	B&H Return	B&H P/L
2012	10,000	14,686	46.86%	25,375	153.75%
2013	10,000	14,520	45.20%	559,117	5491.17%
2014	10,000	11,894	18.94%	4,131	-58.69%
2015	10,000	18,647	86.47%	13,619	36.19%
2016	10,000	15,675	56.75%	21997	119.97%
2017	10,000	30,889	208.89%	130,359	1203.59%
2018	10,000	12,775	27.75%	2,730	-72.70%
2019	10,000	11,317	13.17%	19,083	90.83%
2020	10,000	21,333	113.33%	39,830	298.30%
2021	10,000	15,100	51.00%	11,924	19.24%

Looking at the 'RSI P/L' column, the trading strategy incurred a profit for each year, including those years where the buy and hold strategy incurred a loss. However the buy and hold strategy significantly outperformed the RSI strategy in the years with the largest price runs – 2013 and 2017. The RSI indicator is underperforming the buy and hold strategy in these years because the indicator value reaches 70 (upper threshold) and sells at the start of the bull run (uptrend). During these bull runs, the indicator is not expecting the price to continue to rise. A possible solution to fix this would be to increase the upper threshold to 80, however this would mean less trades are executed and is increasing the risk of missed trading opportunities.

4.2.2 MACD

As discussed in the literature review, the MACD indicator is used where the MACD line and the MACDS (signal) line cross. For visualization purposes, these two lines are plotted for the first 3 months of 2021 against the bitcoin price.

Visualisation 4: Bitcoin's price movement and MACD values for January 2021 – April 2021

The dotted vertical lines signal a crossover between the MACD and the MACDS, which are considering buying or selling opportunities. The green dotted vertical lines are plotted to indicate a 'buy' signal and the red dotted vertical lines are plotted to indicate a 'sell' signal.

The predictive capability of the MACD indicator is measured against a buy and hold strategy.

Table 4: Chart of the MACD returns against a Buy and Hold Strategy

Year	Capital	MACD Return	MACD P/L	B&H Return	B&H P/L
2012	10,000	17,517	75.17%	25,375	153.75%
2013	10,000	425,851	4158.51%	559,117	5491.17%
2014	10,000	11,509	15.09%	4,131	-58.69%
2015	10,000	22,449	124.49%	13,619	36.19%
2016	10,000	18,628	86.28%	21997	119.97%
2017	10,000	128,272	1182.72%	130,359	1203.59%
2018	10,000	8,992	-10.08%	2,730	-72.70%
2019	10,000	16,936	69.36%	19,083	90.83%
2020	10,000	28,456	184.56%	39,830	298.30%
2021	10,000	13,215	32.15%	11,924	19.24%

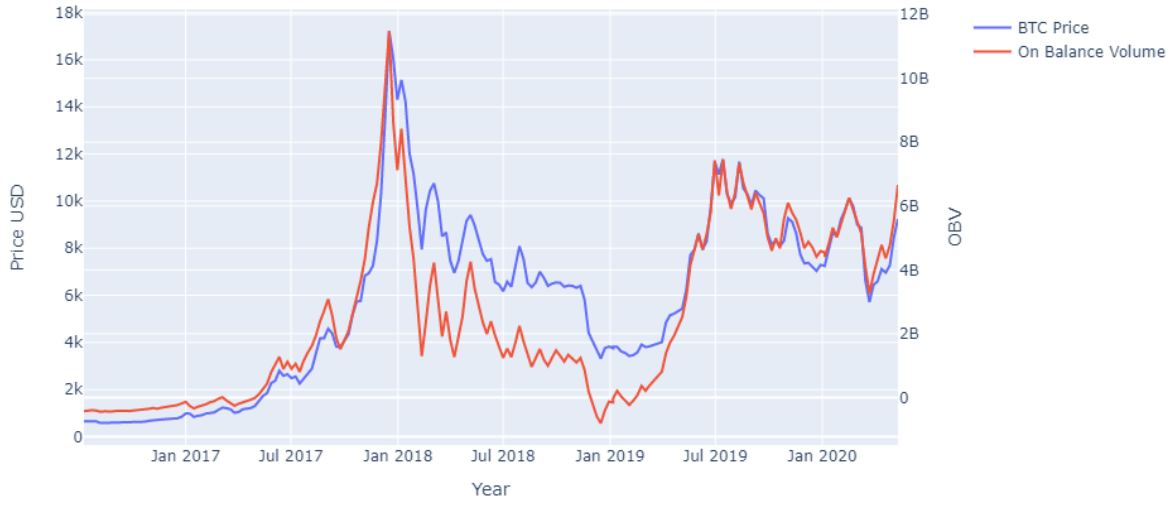
The MACD indicator performs well considering the profit in 'bull' (up trending) market years 2013 and 2017 are close to the profit made from a buy and sell strategy. The MACD indicator also performs reasonably well in 'bear' (down trending) market years 2014 and 2018. In the year of 2014, the MACD indicator returned a Profit of 15.09% whereas the buy and hold strategy lost 58.69% of the initial capital.

4.2.3 OBV

As discussed in the literature, the OBV indicator can be helpful for traders to recognize the presence of large institutional investors. In bitcoin's case, institutional adoption of bitcoin is becoming more popular with adoption from companies such as Microstrategy and Tesla but is still generally rare. The OBV can be used by traders to speculate on the presence of another large investor or company.

The OBV indicator by design will follow the price direction on a day by day basis. Therefore trader's measure OBV by the difference between the slope of the indicator line and the slope of price line.

Visualisation 5: Bitcoin's price movement and OBV values for the 2016-2020 halving cycle



During bitcoin's bull run (uptrend) of 2017, the OBV line closely followed the price action. This reflected that the high volume being traded justified the large price rise. However at the beginning of 2018, the OBV indicator fell sharply in comparison to the price, this indicated that a much higher volume was being traded in total than the volume of bitcoin being sold. The large volume of bitcoin being traded in comparison to the price (which is far less reactive) indicates the presence of many investors who are looking to both buy and sell. The price eventually fell indicating that the 'buying pressure' was exhausted by the 'selling pressure'. In March 2019, the OBV can be seen to gradually rise in comparison to price, indicating the presence of large amounts of volume or 'buying pressure'.

OBV predictive capability is not as straightforward to test as the RSI or the MACD because there is no specific buy or sell signal for traders based on the OBV value. The correlation between the price and the OBV is high at 92.3% because the direction of the price and OBV indicator will always be the same.

4.2.4 MVRV ratio

As described in the Literature Review, the MVRV ratio indicator is used to detect market highs and lows by accounting for the value of bitcoin when the coins were last moved. The total sum of the price when bitcoin's were last moved is referred to as the Realised value. The denominator of the MVRV ratio is the Realised value, whereas the numerator is the market cap. The ratio increases where the market cap is increasing relative to the realized value.

The full time series of the MVRV ratio with the price can be seen below.

Visualisation 6: Bitcoin's all-time price movement and MRVR ratio

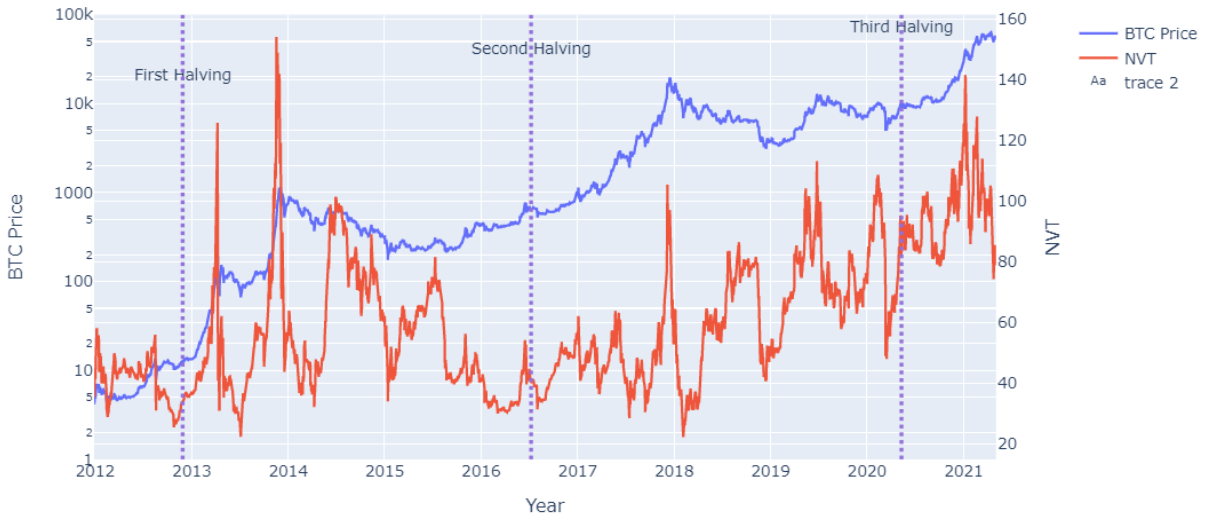
Puell set the upper threshold of to be 3.7 and the lower threshold to be 1 for identifying market tops and bottoms (Puell, 2018) . The MVRV ratio approximately follows the bitcoin price, which is somewhat expected since the price is reflective of the market cap. The MVRV ratio is correlated with price at 39%. The volatility of the ratio is decreasing with each halving cycle, this is due to the reducing difference between the market value and the realized value.

In terms of the predictive capability of the MVRV ratio, it has indicated that bitcoin was overvalued four times in all its history. These four points were when the price hit an all-time-high, and the price dropped in the months after. At these points, the ratio was over the upper threshold at 3.7 and considered bitcoin to be overvalued. Similarly, the ratio fell below the lower bound of 1 during the cycle lows and considered bitcoin to be undervalued. The ratio so far has correctly identified periods of overvaluation and undervaluation of the bitcoin price. Traders who have used MVRV ratio to identify time trading opportunities would have been successful. However the MRVR ratio reaches the upper and lower bound infrequently, and hence trading opportunities are rare.

4.2.5 NVT Signal

As described in the Literature Review, The NVT Signal is similar to the MVRV ratio as they both use Market cap as the numerator to their calculation. The NVT ratio's denominator is the transaction volume whereas the denominator for the MVRV ratio is the realized cap. The NVT signal improves on the NVT ratio by using a 90 day moving average of the transaction volume as the denominator, this helps to keep the denominator stable and the ratio more reactive to movements in the market cap.

Visualisation 7: Bitcoin's all-time price movement and NVT Signal ratio



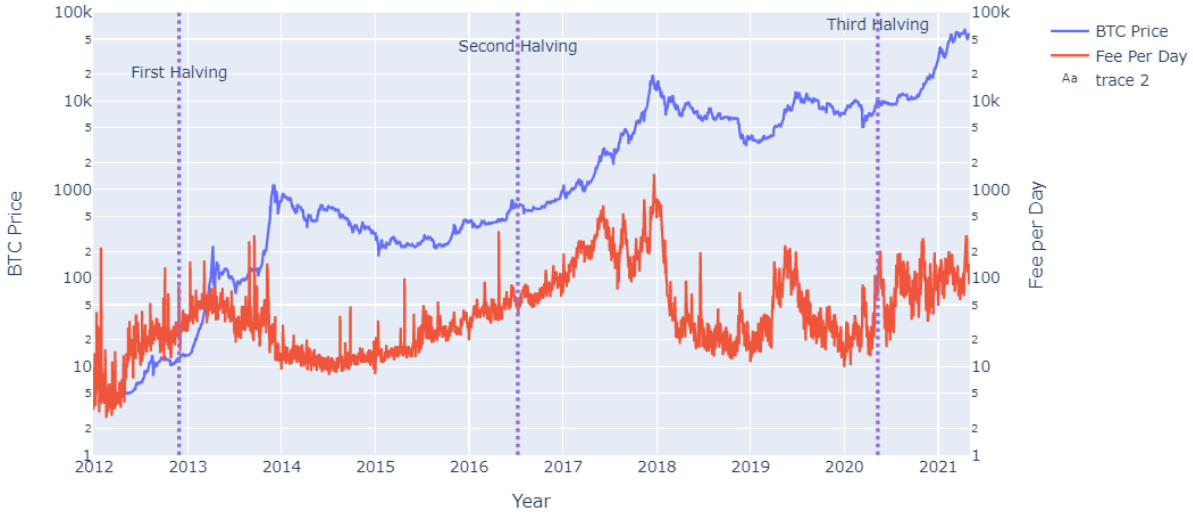
The NVT signal indicates that bitcoin is overvalued when the signal shows a value of 95 or above. Similar to the MVRV ratio, the NVT signal indicates that bitcoin is overvalued for the two large price increases between 2013 and 2014, the two indicators also agree that bitcoin was overvalued during the 2017 price. However, the NVT signal has been above 95 for most of the 2020-2024 cycle, indicating that bitcoin has been overvalued. This could have been caused by low transaction volume relative to an increasing market cap. The transaction volume could have been reduced as investors are more prudent with transacting as the fee per transaction in dollars is increasing.

The correlation of the NVT signal with price is 53.7%, which is higher than the MVRV ratio and perhaps reflective of a relationship between transaction volume and the price.

4.2.6 Fee per Day

The fee per day is the sum of all fees accumulated by miners during a single day. As discussed in the Literature Review, fees rise as users compete for space in a block, since blocks are responsible for verifying transactions, the fee accumulated by miners is equivalent to the user's demand for transaction verification. The fee is measured in bitcoin, so the price per transaction in dollars is the bitcoin price multiplied by the fee per day. Both of these metrics are visualized below.

Visualisation 8: Bitcoin's all-time price movement and MRVR ratio



The Fee per day appears to be much less volatile than the MVRV ratio and the NVT Signal. As can be seen during the 2017 price rise, the Fee per day also increased significantly with a peak of approximately 1000 bitcoin sent to miners in one day. Interestingly though, the fee per day did not have the large spike in values in the other discussed On-Chain indicators between 2013 and 2014. The fee per day is much more stable in comparison. Block space has been constant since bitcoin's inception. In 2013-2014, bitcoin had a much smaller user base than present, so there used to be fewer competition between the users for the limited block space.

In terms of the predictive capabilities of the fee per day as an indicator, the indicator can be seen to be reactive to price movements, such as the rapid decline in fees relative to price at the start of 2018. The correlation with the price is 21.8%.

4.3 Neural Network Analysis

The two sets of indicators were applied to the neural networks MLP and LSTM.

The indicators were evaluated through first a Mean-Absolute-Percentage-Error (MAPE) calculation, followed by a classification accuracy score for the directional movement in the price. The full collection of visualizations of the predictions can be found in the appendix.

MLP – Traditional Indicators

Cycle	MAPE	Classification Score
2009-2012	6.2%	100%
2012-2016	29.15%	12.25%
2016-2020	31.97%	31%
2020-2024	6.03%	68.18%

The Average MAPE across all halving cycles: 18.33%
The Average Classification Score across all halving cycles: 52.86%

The traditional indicators returned varying results depending on halving cycle. The halving cycle with the most accurate results is the 2009-2012 cycle. The indicators were able to predict the price with 6.2% error and correctly predict the movement of the price 100% of the days.

It is interesting to note that the indicators seemed to exaggerate the price movements in the prediction.
The halving cycle that the indicators found most difficult to accurately predict is the 2016-2020 halving cycle. The MAPE of the model is 31.97% and the model could only predict the direction of the price movement 31% of the time. Since 31% is less than 50% (random prediction), the indicators essentially failed to predict the direction of the price.

The MAPE is high for this halving cycle due to the model's prediction of the price being consistently lower than actual price. The MLP found this halving cycle especially hard to predict.

MLP – On-Chain indicators

Cycle	MAPE	Classification Score
2009-2012	10.61%	92.98%
2012-2016	6.2%	97.24%
2016 -2020	0.77%	100%
2020 -2024	0.59%	95.45%

The Average MAPE across all halving cycles: 4.54%
The Average Classification Score across all halving cycles: 96.42%

Overall the On-Chain indicators performed better than the Traditional indicators for the same model configuration. Surprisingly, the halving cycle results from the On-Chain indicators are opposite from the Traditional indicators, for example, the best performing model from the traditional indicators is the 2009-2012 Cycle, whereas for that same cycle and model configuration, the On-Chain indicators performed worse than traditional indicators.

Similar to the model error of the 'MLP- 2016-2020 - Traditional Indicators', the model's predictions are consistently higher than the actual price movement.

As visualized below, the best performing halving cycle for the On-Chain indicators is the 2016-2020 cycle. The model performs very accurately and seems overfit, however considering that the same model configuration and cycle for the traditional indicators performed poorly, perhaps the On-Chain indicators for this halving cycle are especially reactive and possess good predictive qualities.

LSTM – Traditional Indicators

Cycle	MAPE	Classification Score
2009-2012	16.37%	95.38%
2012-2016	35.86%	22.81%
2016-2020	30.66%	45.36%
2020-2024	38.35%	56%

The Average MAPE across all halving cycles: 30.31%
The Average Classification Score across all halving cycles: 54.88%

The LSTM model returns slightly poorer results than the MLP regressor. Similar to the MLP regressor the halving cycle that returned the best prediction with the LSTM is the 2009-2012 cycle.

Similar to the MLP regressor, the prediction exaggerates the price movement for the end of this cycle. Since both models have exaggerated the price movements for this cycle, perhaps the combination of traditional indicators are strongly exaggerating the price movement and the prediction is reflective of this.

The poorest performing cycle for the LSTM network is the 2020-2024 cycle. The LSTM model predicts a drastic reduction in price. The model is poor, however it correctly predicts the incoming sharp fall in price, only slightly earlier and sharper than reality.

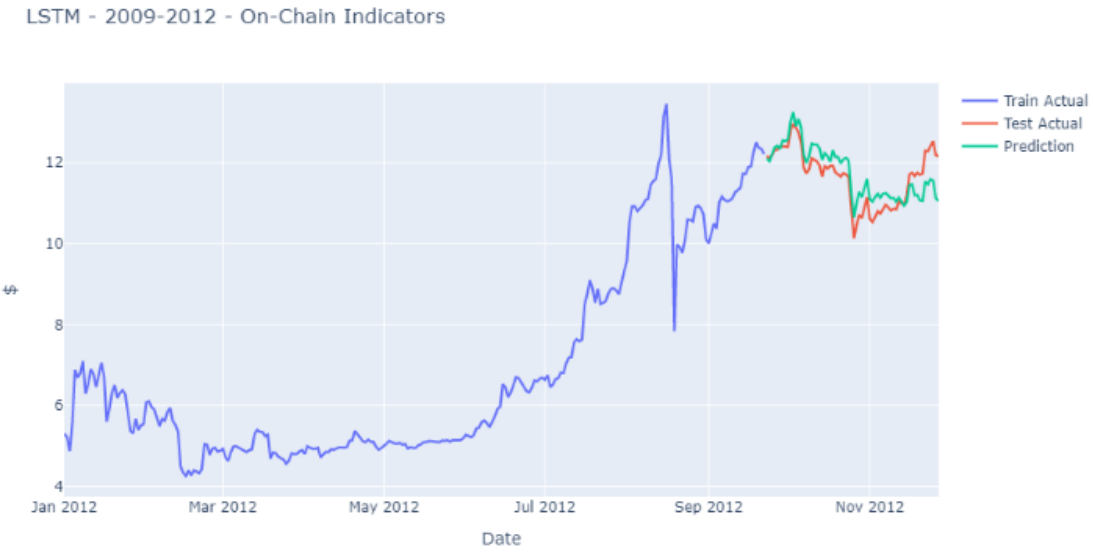
LSTM – On-Chain Indicators

Cycle	MAPE	Classification Score
2009-2012	2.5%	81.82%
2012-2016	59.9%	31.17%
2016-2020	15.88%	84.64%
2020-2024	37.54%	84%

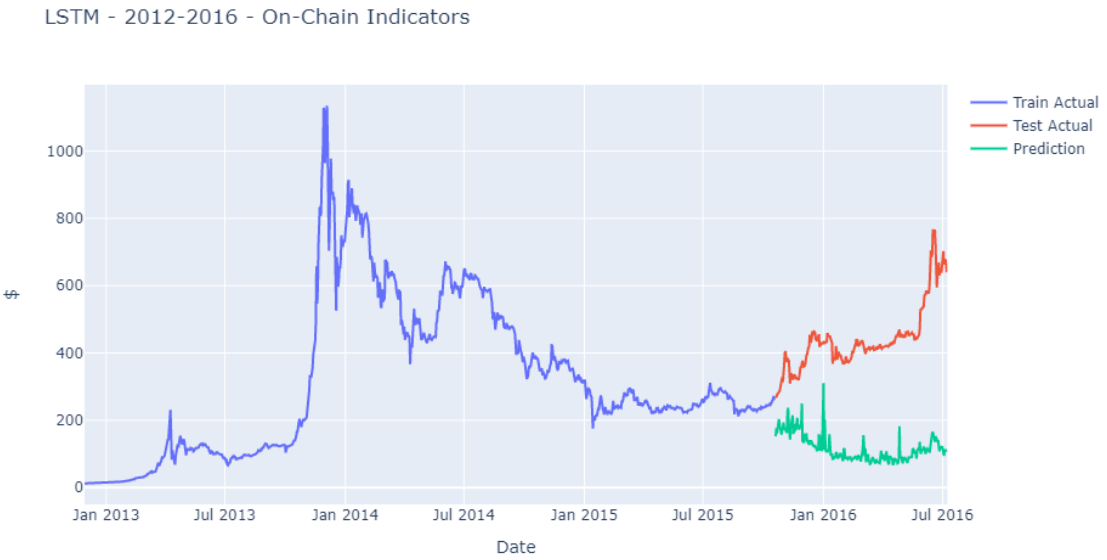
The Average MAPE across all halving cycles: 28.96%
The Average Classification Score across all halving cycles: 70.4%

The LSTM model for the On-Chain indicators performed better than the LSTM model for traditional indicators. Although in comparison to the MLP On-Chain indicators, the LSTM model performed slightly worse.

The best performing halving cycle for the LSTM On-Chain indicators is the 2009-2012 cycle. The model does not exaggerate the price movements as the traditional indicator model's do. The MAPE is especially low at 2.5%, while the model is correctly able to predict the direction of the price movement 81.82% of the time.



The poorest performing halving cycle for this model variation is the following 2012-2016 cycle. The prediction of this halving cycle is especially poor, it does not seem to anticipate any price movement while the prediction also steadily declines away from the actual price. The 2012-2016 cycle is one of the poorest performing halving cycles in both neural networks and indicator sets. This halving cycle is trained using the sharpest price increase (9,500%) across the halving cycle data used in this report. It is therefore a possibility that the model struggles the most to effectively learn from this halving cycle due to the price and indicator volatility.



4.4 Summary

This chapter has discussed the halving cycle analysis, indicator analysis and the neural network results. The Halving cycle price movements were shown to be correlated up to 89% between halving cycles. The indicators were described in terms of their values in relation to price movement. The predictive nature of each indicator was discussed, and tested in the case of RSI and MACD. The neural networks returned varying data depending on halving cycle. However, all models return significantly better results for On-Chain indicators than the traditional indicators. The average MAPE for On-Chain indicators is 16.75% whereas the MAPE for traditional indicators is 24.32%.

The MLP regressor also outperformed the LSTM based on the data and model configuration supplied. The MAPE for each set of indicators and neural network ranged from 30.31% to 4.54%. This is equivalent to an accuracy score of between 69.69% and 95.45% depending on indicator set and model.

The halving cycle which returned the best results on average for both sets of indicators is the 2009-2012 halving cycle, which from the data I have available (starting 2012), has the least drastic price increase – approximately 300% from bottom to top. In comparison the poorest performing halving cycle is the 2012-2016 halving cycle, this is also has the most drastic price increase – approximately 9,500% from bottom to top.

Chapter 5: Conclusion

5.1 Discussion of Findings

Based on my findings, indicators are able predict the price of bitcoin with an accuracy of between 69.69% and 95.46%. I came to this conclusion as the indicator set and neural network with the lowest MAPE is 4.54%, whereas the indicator set and neural network with the highest MAPE is 30.31%. These error are converted to accuracy by subtracting the MAPE from 100.

The On-Chain indicators selected for this report (MVRV ratio, NVT signal and Fee per day) outperformed the traditional indicators (RSI, MACD, OBV) on their predictive capability. The On-Chain indicators on average returned a MAPE of 16.75% in comparison to the traditional indicators 24.32%. The On-Chain indicators also acted as better predictors of the direction of price movement than traditional indicators. The On-Chain indicators correctly forecasted the direction of price movement 83.41% of the time, whereas the traditional indicators were only able to predict the direction of the price 53.87% of the time.

On average, the MLP neural networks outperformed the LSTM neural networks. The MLP's MAPE averaged 11.44%, the LSTM's MAPE averaged 29.64%. The MLP also performed better in classification than the LSTM. The MLP correctly forecasted the direction of price movement 74.64% of the time, whereas the LSTM correctly predicted the direction of price movement 62.64% of the time.

As discussed in the Results & Findings chapter, the results varied from halving cycle to halving cycle. The most consistently predictable halving cycle is the first halving cycle 2009-2012. The poorest performing halving cycle is the second 2012-2016 halving cycle. Perhaps coincidentally, based on the data starting 2012, the first halving cycle experiences the lowest price increase from cycle bottom to top – approximately 300%. Whereas, the second cycle experiences the most drastic price increase from cycle bottom to top – approximately 9500%. The neural networks may have struggled to accurately train a model with such drastic fluctuations in the target variable – price.

5.2 Limitations

I found it difficult to collect bitcoin exchange data that occurred before 2012. This meant I did not have access to the price and volume from the start of the first halving cycle. If this data were publicly available to me, perhaps more accurate predictions would be possible.

Neural Network configuration can be tedious and complex. In the case of the neural networks used in this report, little parameter tuning was performed in order to use the most straightforward model with default parameters. Advanced parameter tuning with parameters such as batch size, number of epochs and time-shifts would add an additional layer of complexity to the project. I believe that this report included enough model variations from indicator sets to halving cycles to neural networks and that comparing model parameters would detract from the value of the other areas of focus in this report.

Similar to neural networks, the RSI indicator has multiple possible configurations. I have used RSI 14 in this report because it is the most commonly used and referred to indicator among traders. Extensive analysis comparing the lookback period between RSI values or adjusting the threshold to test for optimum trading strategies is beyond the scope of this project.

5.3 Areas of Future Research

Bitcoin is a relatively new asset class with only 12 years of history. My analysis into halving cycles and the correlation with price would have been enhanced with the availability of further halving cycle data. Repeating the same experiments discussed in this report at some time in the future with additional halving data may be an area of interest and add to the robustness of the findings.

Bitcoin is not the only cryptocurrency that reduces its supply with a halving event. An investigation into other cryptocurrencies halving cycles such as Bitcoin Cash or Litecoin could be compared with my work in this report.

The traditional and On-Chain indicators used in this report were selected purely due to their popularity and simplicity. Numerous other traditional and on-chain indicators exist that could be tested on neural networks to investigate their predictive capabilities for bitcoin. Other variations of neural networks and machine learning algorithms can be investigated in terms of the halving cycle.

As previously mentioned, I find little academic research exists studying bitcoin's halving cycle despite its cyclic price pattern. I believe there are opportunities for further research into predictability of the price using halving cycle patterns.

Chapter 6: Bibliography

References

Alonso-Monsalve, S., Suárez-Cetrulo, A. L., Cervantes, A., & Quintana, D. (2020). Convolution on neural networks for high-frequency trend prediction of cryptocurrency exchange rates using technical indicators. *Expert Systems with Applications*, *149*(113250), 113250.

Ammous, S. (2018). *The bitcoin standard: The decentralized alternative to central banking*. Nashville, TN: John Wiley & Sons.

Appel, G. (2005). *Technical analysis: Power tools for active investors* (First). Upper Saddle River, NJ: Financial TImes Prentice Hall.

Bambrough, B. (2020, November 19). Leaked Citibank report reveals bitcoin could rocket to $300,000 price by end of 2021. *Forbes Magazine*. Retrieved from https://www.forbes.com/sites/billybambrough/2020/11/19/leaked-citibank-report-reveals-bitcoin-could-rocket-to-300000-price-by-end-of-2021/

Bengio, Y., Simard, P., & Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, *5*(2), 157–166.

Bhargavi, R., Gumparthi, S., & Amith, R. (2017). *Relative Strength Index for Developing Effective Trading Strategies in Constructing Optimal Portfolio*.

Buchholz, M., Delaney, J., Waarren, J., & Parker, J. (2012). *Bits and Bets Information, Price Volatility, and Demand for Bitcoin*. Retrieved from https://www.reed.edu/economics/parker/s12/312/finalproj/Bitcoin.pdf

Bursztynsky, J. (2021, February 23). Square buys $170 million worth of bitcoin. Retrieved May 26, 2021, from CNBC website: https://www.cnbc.com/2021/02/23/square-buys-170-million-worth-of-bitcoin.html

Chen, J. (2021, May 19). Technical Indicator. Retrieved May 26, 2021, from Investopedia.com website: https://www.investopedia.com/terms/t/technicalindicator.asp

Chong, T. T.-L., & Ng, W.-K. (2008). Technical analysis and the London stock exchange: testing the MACD and RSI rules using the FT30. *Applied Economics Letters*, *15*(14), 1111–1114.

Coin Metrics. (2021, February 12). Data file downloads. Retrieved May 26, 2021, from Coinmetrics.io website: https://coinmetrics.io/community-network-data/

de la Horra, L. P., de la Fuente, G., & Perote, J. (2019). The drivers of Bitcoin demand: A short and long-run analysis. *International Review of Financial Analysis*, *62*, 21–34.

Di Salvo, M. (2019, March 19). Why are Venezuelans seeking refuge in crypto-currencies? *BBC*. Retrieved from https://www.bbc.com/news/business-47553048

Eric, D., Andjelic, G., & Redzepagic, S. (2009). Application of MACD and RVI indicators as functions of investment strategy optimization on the financial market. *Zbornik Radova Ekonomskog Fakulteta u Rijeci Časopis Za Ekonomsku Teoriju i Praksu/Proceedings of Rijeka Faculty of Economics Journal of Economics and Business*, *27*(1), 171–196.

Gale, D. (1955). The law of supply and demand. *MATHEMATICA SCANDINAVICA*, *3*(1), 155–169.

Gers, F. A., Schmidhuber, J., & Cummins, F. (2000). Learning to forget: continual prediction with LSTM. *Neural Computation*, *12*(10), 2451–2471.

Granville, J. (1963). *Granville's New Key To Stock Market Profits*.

Gu, F., Jiang, Z., & Su, J. (2021). Application of features and neural network to enhance the performance of deep reinforcement learning in portfolio management. *2021 IEEE 6th International Conference on Big Data Analytics (ICBDA)*, 92–97. IEEE.

Guesmi, K., Saadi, S., Abid, I., & Ftiti, Z. (2019). Portfolio diversification with virtual currency: Evidence from bitcoin. *International Review of Financial Analysis*, *63*, 431–437.

Hertig, A. (2020, March 24). What is bitcoin halving? Here's everything you need to know – CoinDesk. Retrieved May 26, 2021, from CoinDesk website: https://www.coindesk.com/bitcoin-halving-explainer

Kang, E. (2017, September 1). Long Short-Term Memory (LSTM): Concept. Retrieved May 26, 2021, from Medium website: https://medium.com/@kangeugine/long-short-term-memory-lstm-concept-cb3283934359

Kovach, S. (2021, February 8). Tesla buys $1.5 billion in bitcoin, plans to accept it as payment. Retrieved May 26, 2021, from CNBC website: https://www.cnbc.com/2021/02/08/tesla-buys-1point5-billion-in-bitcoin.html

Kramer, L. (2021, May 19). How does the law of supply and demand affect prices? Retrieved May 26, 2021, from Investopedia.com website: https://www.investopedia.com/ask/answers/033115/how-does-law-supply-and-demand-affect-prices.asp

Kristoufek, L. (2015). What are the main drivers of the Bitcoin price? Evidence from wavelet coherence analysis. *PloS One*, *10*(4), e0123923.

McNally, S., Roche, J., & Caton, S. (2018). Predicting the price of bitcoin using machine learning. *2018 26th Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP)*, 339–343. IEEE.

Meynkhard, A. (2019). Fair market value of bitcoin: halving effect. *Investment Management and Financial Innovations*, *16*(4), 72–85.

Nakamoto, S. (2009). *Bitcoin: A Peer-to-Peer Electronic Cash System*.

Nelson, D. M. Q., Pereira, A. C. M., & de Oliveira, R. A. (2017). Stock market's price movement prediction with LSTM neural networks. *2017 International Joint Conference on Neural Networks (IJCNN)*, 1419–1426. IEEE.

Patel, M. M., Tanwar, S., Gupta, R., & Kumar, N. (2020). A deep learning-based cryptocurrency price prediction scheme for financial institutions. *Journal of Information Security and Applications*, *55*(102583), 102583.

PlanB. (2019, March 22). Modeling bitcoin value with scarcity. Retrieved May 26, 2021, from Medium website: https://medium.com/@100trillionUSD/modeling-bitcoins-value-with-scarcity-91fa0fc03e25

PlanB. (2020, April 27). Bitcoin stock-to-flow cross asset model - PlanB - medium. Retrieved May 26, 2021, from Medium website: https://medium.com/@100trillionUSD/bitcoin-stock-to-flow-cross-asset-model-50d260feed12

Poyser, O. (2017). *Exploring the determinants of Bitcoin's price: an application of Bayesian Structural Time Series*. Retrieved from https://www.researchgate.net/publication/317356728_Exploring_the_determinants_of_Bitcoin's_price_an_application_of_Bayesian_Structural_Time_Series

Puell, D. (2018, October 1). Bitcoin market-value-to-realized-value (MVRV) ratio. Retrieved May 26, 2021, from Medium website: https://medium.com/@kenoshaking/bitcoin-market-value-to-realized-value-mvrv-ratio-3ebc914dbaee

Rokke, K. (2021). Letter to Shareholders of Aker. Retrieved from Seetee.io website: https://www.seetee.io/static/shareholder_letter-6ae7e85717c28831bf1c0eca1d632722.pdf

Square, Inc. (2020). *Square, Inc. Bitcoin Investment Whitepaper*.

Swift, P. (2020, December 15). Bitcoin Realized HODL Ratio. Retrieved May 26, 2021, from Medium website: https://positivecrypto.medium.com/bitcoin-realized-hodl-ratio-9023db15a559

Thomsett, M. (2017). Chapter 10. Volume Indicators. In *Stock Market Math*. Berlin, Boston: De Gruyter.

Tsang, W. W. H., & Chong, T. T. L. (n.d.). Profitability of the on-balance volume indicator. Retrieved May 26, 2021, from Accessecon.com website: http://www.accessecon.com/pubs/eb/2009/volume29/eb-09-v29-i3-p87.pdf

Vigna, P. (2014, February 25). 5 things about mt. Gox's crisis. *Wall Street Journal (Eastern Ed.)*. Retrieved from https://www.wsj.com/articles/BL-263B-352

Winklevoss, T. (2020, August 27). The Case for $500K Bitcoin. Retrieved May 26, 2021, from Winklevosscapital.com website: https://winklevosscapital.com/the-case-for-500k-bitcoin/

Wong, W.-K., Manzur, M., & Chew, B.-K. (2003). How rewarding is technical analysis? Evidence from Singapore stock market. *Applied Financial Economics*, *13*(7), 543–551.

Woo, W. (2017, October 5). Introducing NVT Ratio (Bitcoin's PE Ratio), use it to detect bubbles. Retrieved May 26, 2021, from Woobull.com website: https://woobull.com/introducing-nvt-ratio-bitcoins-pe-ratio-use-it-to-detect-bubbles/

Woo, W. (2018, February 11). NVT Signal, a new trading indicator to pick tops and bottoms. Retrieved May 26, 2021, from Woobull.com website: https://woobull.com/nvt-signal-a-new-trading-indicator-to-pick-tops-and-bottoms/

Yassin, I. M., Abdul Khalid, M. F., Herman, S. H., Pasya, I., Wahab, N. A., & Awang, Z. (2017). Multi-layer perceptron (MLP)-based nonlinear auto-regressive with exogenous inputs (NARX) stock forecasting model. *International Journal on Advanced Science, Engineering and Information Technology*, *7*(3), 1098.

Chapter 7: Appendix

MLP Neural Networks – Traditional Indicators

2009 – 2012

2012-2016

2016 – 2020

2020-2024

MLP Neural Networks – On-Chain Indicators

2009-2012

2012-2020

2016-2020

2020-2024

LSTM Neural Networks – Traditional Indicators

2009-2012

20112-2016

2016-2020

2020-2024

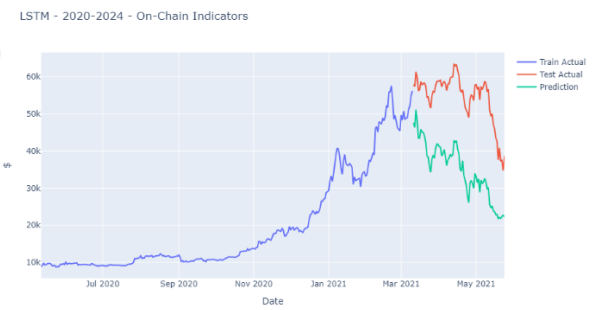
LSTM Neural Networks – On-Chain Indicators

2009-2012

2012-2016

2016-2020

2020-2024



Python Code

1. *Cleaning file*
2. *Analysis File*
3. *MACD File*
4. *RSI File*
5. *Volume File*
6. *On-Chain Indicators File*
7. *Machine Learning File*

1.Cleaning file

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

#bitstamp data found on kaggle @ https://www.kaggle.com/kognitron/zielaks-bitcoin-historical-data-wo-nan
import pandas as pd
df = pd.read_csv('C:/College/Final Year Project/bitstamp_cleaned.csv',sep=',')
df.head()

# In[2]:

len(df.index)
#number of rows in the data

# In[3]:

#This block of code takes 10 mins to run on my machine.
#sets the index to DateTime and changes tyoe to DateTimeIndex
df.set_index('DateTime',inplace = True)
df.index=pd.DatetimeIndex(df.index)

# In[4]:

print(df.index.dtype)
#check to make sure index of of tyoe datetime

# In[5]:

df

# In[6]:

#We want our data to have the Average prices, but the sum of volume durinf a period
mean= df.groupby(df.index.date).mean()
total = df.groupby(df.index.date).sum()

data = {'Open':mean['Open'],
        'High':mean['High'],
        'Low':mean['Low'],
        'Close':mean['Close'],
        'Volume_BTC':total['Volume_(BTC)'],
        'Volume':total['Volume_(Currency)'],
```

```
        'Weighted_Price':mean['Weighted_Price']}

df =pd.DataFrame(data,index = mean.index)
df

# In[13]:

#round to 4 decimal places.
df = df.round(decimals = 4)

# In[14]:

#data sourced from :     https://www.cryptodatadownload.com/data/bitstamp/
import pandas as pd
df2 = pd.read_csv('C:/College/Final Year Project/Bitstamp_BTCUSD_d.csv')
df2

# In[15]:

#check number of rows in data
len(df2.index)

# In[16]:

#reversing the order from newest to oldest -> oldest to newest
df2 = df2.iloc[::-1]

# In[17]:

#same process as previous dataset, detting index to DateTime
df2.set_index('date',inplace = True)
df2.index=pd.DatetimeIndex(df2.index)
df2.index.name = None

# In[18]:

df2

# In[19]:

#preparing to concatonate two datasets together by removing overlapping rows from df2
df2 = df2.loc['2020-09-15':]
del df2['unix']
del df2['symbol']

# In[20]:

#setting the weighted price column to the average between intraday high and low.
df2['Weighted_Price'] = df2[['high','low']].mean(axis=1)
df2.rename(columns = {'open':'Open','high':'High','low':'Low','close':'Close',
                      'Volume BTC':'Volume_BTC','Volume USD':'Volume'}, inplace = True)

# In[21]:

#concatonate the two dataframes together and set index type to DateTimeIndex
frames = [df,df2]
new_df = pd.concat(frames,axis=0)
new_df.index=pd.DatetimeIndex(new_df.index)

# In[26]:

new_df

# In[25]:

#removing last row becuase that's today and data such as total volume is lower
new_df = new_df[:-1]
```

In[27]:

```
#Writing this new df to a csv for accessibility reasons
new_df.to_csv('C:/College/Final Year Project/daily_bitstamp.csv')
```

Reading the preprocessed data

In[28]:

```
#read in the data that has just been sent to csv.
import pandas as pd
df = pd.read_csv('C:/College/Final Year Project/daily_bitstamp.csv')
#reading in the csv sets the index to a number, so I'm resetting it to a DateTimeIndex
df = df.set_index('Unnamed: 0')
df.index.rename('Date',inplace = True)
df.index=pd.DatetimeIndex(df.index)
df
```

In[29]:

```
#These are the dates of the previous 3 bitcoin halvings
halving_2012 = pd.to_datetime('2012-11-28')
halving_2016 = pd.to_datetime('2016-07-09')
halving_2020 = pd.to_datetime('2020-05-11')

# dividing total dataframe into 4 for halving cycle analysis
#'pre' in this context is short for previous
pre_2012 = df.loc[df.index < halving_2012]
pre_2016 = df.loc[(df.index < halving_2016) & (df.index > halving_2012)]
pre_2020 = df.loc[(df.index < halving_2020) & (df.index > halving_2016)]
pre_2024 = df.loc[df.index > halving_2020]

#creating a column called 'Cycle' in each dataframe specifying what halving cycle they're in, will be useful when they are co
ncatonated together
pre_2012['Cycle'] = '2012'
pre_2016['Cycle'] = '2016'
pre_2020['Cycle'] = '2020'
pre_2024['Cycle'] = '2024'

frames = [pre_2012,pre_2016,pre_2020,pre_2024]
new_df = pd.concat(frames,sort = False)
```

In[ ]:

In[30]:

```
#We want our data to have Average prices, but the sum of volume

#monthly figures
#transforming the dataframe so each row is grouped by the month
monthly_mean= new_df.groupby([new_df.index.strftime('%Y-%m')]).mean()
total= new_df.groupby([new_df.index.strftime('%Y-%m')]).sum()

data = {'Open':monthly_mean['Open'],
        'High':monthly_mean['High'],
        'Low':monthly_mean['Low'],
        'Close':monthly_mean['Close'],
        'Volume_BTC':total['Volume_BTC'],
        'Volume':total['Volume'],
        'Weighted_Price':monthly_mean['Weighted_Price']}

monthly_mean =pd.DataFrame(data,index = monthly_mean.index)
```

```
#weekly figures
#transforming the dataframe so each row is grouped by the week
weekly_mean = new_df.groupby([new_df.index.strftime('%Y-%W')]).mean()
total= new_df.groupby([new_df.index.strftime('%Y-%W')]).sum()

data = {'Open':weekly_mean['Open'],
        'High':weekly_mean['High'],
        'Low':weekly_mean['Low'],
        'Close':weekly_mean['Close'],
        'Volume_BTC':total['Volume_BTC'],
        'Volume':total['Volume'],
        'Weighted_Price':weekly_mean['Weighted_Price'],}

weekly_mean =pd.DataFrame(data,index = weekly_mean.index)

# In[31]:

weekly_mean

# In[32]:

#removing the last row in the data since it could be an incomplete week.
weekly_mean.drop(weekly_mean.tail(1).index,inplace=True)

# In[33]:

# to change the groupby index to a datetime index, I am specifiying what day of the week the row of data is. In strptime, '0' =
 1st day of week
weekly_mean.index = weekly_mean.index.astype(str)+'-0'

# In[34]:

weekly_mean.index

# In[35]:

from datetime import datetime

correct_date_list =[]

# '%w' is the symbol for the day (0-6) of the week
for i in weekly_mean.index:
    correct_date = datetime.strptime(i,"%Y-%W-%w")
    correct_date_list.append(correct_date)


# In[36]:

weekly_mean.index = correct_date_list

# In[37]:

print(weekly_mean.index.dtype)

# In[38]:

#data grouped by week with a datetime index
weekly_mean

# In[39]:

#repeating these steps with monthly figures
monthly_mean

# In[40]:

#removing the last row in the data since it could be an incomplete month.
```

```
monthly_mean.drop(monthly_mean.tail(1).index,inplace=True)

# In[41]:

#checking index type, need to change to DateTimeIndex
print(monthly_mean.index.dtype)

# In[42]:

# adding an '-01' for strptime to identify the datetime format
monthly_mean.index = monthly_mean.index.astype(str)+'-01'

# In[43]:

monthly_mean

# In[44]:

from datetime import datetime

correct_date_list =[]

for i in monthly_mean.index:
    correct_date = datetime.strptime(i,"%Y-%m-%d")
    correct_date_list.append(correct_date)


# In[45]:

monthly_mean.index = correct_date_list

# In[46]:

print(monthly_mean.index.dtype)

# In[ ]:




# In[ ]:




# In[47]:

#within monthly dataframes and weekly dataframes I am adding a column specifying which halving cycle the row belongs to

#These are the dates of the previous 3 bitcoin halvings
halving_2012 = pd.to_datetime('2012-11-28')
halving_2016 = pd.to_datetime('2016-07-09')
halving_2020 = pd.to_datetime('2020-05-11')

pre_2012_weekly = weekly_mean.loc[weekly_mean.index < halving_2012]
pre_2016_weekly = weekly_mean.loc[(weekly_mean.index < halving_2016) & (weekly_mean.index > halving_2012)]
pre_2020_weekly = weekly_mean.loc[(weekly_mean.index < halving_2020) & (weekly_mean.index > halving_2016)]
pre_2024_weekly = weekly_mean.loc[weekly_mean.index > halving_2020]

pre_2012_monthly = monthly_mean.loc[monthly_mean.index < halving_2012]
pre_2016_monthly = monthly_mean.loc[(monthly_mean.index < halving_2016) & (monthly_mean.index > halving_2012)]
pre_2020_monthly = monthly_mean.loc[(monthly_mean.index < halving_2020) & (monthly_mean.index > halving_2016)]
pre_2024_monthly = monthly_mean.loc[monthly_mean.index > halving_2020]

pre_2012_weekly['Cycle'] = '2012'
pre_2016_weekly['Cycle'] = '2016'
pre_2020_weekly['Cycle'] = '2020'
```

```
pre_2024_weekly['Cycle'] = '2024'

pre_2012_monthly['Cycle'] = '2012'
pre_2016_monthly['Cycle'] = '2016'
pre_2020_monthly['Cycle'] = '2020'
pre_2024_monthly['Cycle'] = '2024'

#This is to add new column called cycle to overall dataframe
weekly_frames = [pre_2012_weekly,pre_2016_weekly,pre_2020_weekly,pre_2024_weekly]
weekly_mean = pd.concat(weekly_frames,sort = False)

monthly_frames = [pre_2012_monthly,pre_2016_monthly,pre_2020_monthly,pre_2024_monthly]
monthly_mean = pd.concat(monthly_frames,sort = False)

# In[48]:

weekly_mean

# In[49]:

#to install stockstats, whic will generate the indicators.
pip install stockstats

# In[50]:

#adding the indicators from stockstats library.
from stockstats import StockDataFrame as Sdf
monthly_mean = Sdf.retype(monthly_mean)
monthly_mean.get('rsi_12')
monthly_mean.get('vr')
monthly_mean.get('macd')

weekly_mean = Sdf.retype(weekly_mean)
weekly_mean.get('rsi_14')
weekly_mean.get('vr')
weekly_mean.get('macd')

new_df = Sdf.retype(new_df)
new_df.get('rsi_14')
new_df.get('vr')
new_df.get('macd')

# In[51]:

#adding OBV manually because stockstats does not support OBV.
# The OBV is calulated using the 'Close' price and volume.
import numpy as np
monthly_obv = (np.sign(monthly_mean['close'].diff()) * monthly_mean['volume']).fillna(0).cumsum().to_frame()
weekly_obv = (np.sign(weekly_mean['close'].diff()) * weekly_mean['volume']).fillna(0).cumsum().to_frame()
daily_obv = (np.sign(new_df['close'].diff()) * new_df['volume']).fillna(0).cumsum().to_frame()

#renaming the column of the dataframe
monthly_obv = monthly_obv.rename(columns = {0:'obv'})
weekly_obv = weekly_obv.rename(columns = {0:'obv'})
daily_obv = daily_obv.rename(columns = {0:'obv'})

#since the dataframe 'monthly_obv' only has one column, it can be added as a column to another datframe as if it's a series
monthly_mean['Obv']=monthly_obv
weekly_mean['Obv']=weekly_obv
new_df['Obv']=daily_obv

# In[52]:

monthly_mean

# In[53]:
```

```
weekly_mean

# In[54]:

new_df

# In[ ]:



# In[55]:

#writing these datframes with indicators to csv files so they can be used for analysis in another notebook.
new_df.to_csv('C:/College/Final Year Project/daily_cleaned.csv')

# In[56]:

monthly_mean.to_csv('C:/College/Final Year Project/monthly_cleaned.csv')

# In[57]:

weekly_mean.to_csv('C:/College/Final Year Project/weekly_cleaned.csv')
```

2.Analysis File

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

import pandas as pd
#reading in the data
df = pd.read_csv('C:/College/Final Year Project/daily_cleaned.csv')
# setting the 'Date' column to be the index and setting its type to 'DateTimeIndex'
df.index=pd.DatetimeIndex(df['Date'])
# deleteing the now redundant 'Date' column
del df['Date']
df

# In[4]:

# import plotly library
import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import numpy as np

#plotting the price in Linear vs Log
fig1 =go.Figure([go.Scatter(x=df.index, y = df['weighted_price'],name = 'BTC Price_Log')])
fig2 =go.Figure([go.Scatter(x=df.index, y = df['weighted_price'],name = 'BTC_Price_Linear')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig1.data[0], secondary_y=False)
fig.add_trace(fig2.data[0], secondary_y=True)

##adding vertical lines to highlight the halving event
fig.add_shape(type="line",
    x0='2020-05-11', y0=1, x1='2020-05-11', y1=100000,
    line=dict(color="MediumPurple",width=3,dash = 'dot'))
```

```
fig.add_shape(type="line",
   x0='2016-07-09', y0=1, x1='2016-07-09', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2012-11-28', y0=1, x1='2012-11-28', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_trace(go.Scatter(
   x=['2020-05-11', '2016-07-09', '2012-11-28'],
   y=[50000,10000,1000],
   text=['Third Halving',
       'Second Halving',
        'First Halving'],
   mode="text",
))

fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'BTC Price_Log',type="log", range=[np.log10(5), np.log10(100000)]),
                  yaxis2=dict(title = 'BTC_Price_Linear'))

py.iplot(fig)

# In[5]:

#correlation dataframe of the prices between halving events
correlation_frame =pd.DataFrame()

#I entered cycle 2020 first, this creates an index large enough for the other halvings.
correlation_frame['2020 Cycle'] = pd.Series(df['weighted_price'].loc[df['cycle']==2020].values)
correlation_frame['2012 Cycle'] = pd.Series(df['weighted_price'].loc[df['cycle']==2012].values)
correlation_frame['2016 Cycle'] = pd.Series(df['weighted_price'].loc[df['cycle']==2016].values)
correlation_frame['2024 Cycle'] = pd.Series(df['weighted_price'].loc[df['cycle']==2024].values)
correlation_frame = correlation_frame[['2012 Cycle','2016 Cycle','2020 Cycle','2024 Cycle']]

# In[6]:

correlation_frame

# In[7]:

#since these columns have different length, the correlation is calculated in the rows filled.
# Example/ '2012 Cycle' only has 333 rows, so the first 333 rows of other cycles are used for correlation
correlation_frame.corr(method='pearson')

# In[8]:

#plotting the cycle in different figures

fig = go.Figure()

fig = make_subplots(rows=1, cols=4,subplot_titles=("Cycle 2009-2012", "Cycle 2012-2016", "Cycle 2016-
2020", "Cycle 2020-2024"))

fig.add_trace(go.Scatter(x=df['weighted_price'].loc[df['cycle']==2012].index, y = df['weighted_price'].loc[df['cycle']==2012]
,name = '2012 price'),row=1,col=1)
fig.add_trace(go.Scatter(x=df['weighted_price'].loc[df['cycle']==2016].index, y = df['weighted_price'].loc[df['cycle']==2016]
,name = '2016 price'),row=1,col=2)
fig.add_trace(go.Scatter(x=df['weighted_price'].loc[df['cycle']==2020].index, y = df['weighted_price'].loc[df['cycle']==2020]
,name = '2020 price'),row=1,col=3)
fig.add_trace(go.Scatter(x=df['weighted_price'].loc[df['cycle']==2024].index, y = df['weighted_price'].loc[df['cycle']==2024]
,name = '2024 price'),row=1,col=4)

# Update yaxis properties
fig.update_yaxes(title_text="Price", tickmode = 'auto',row=1, col=1)
fig.update_yaxes(tickmode = 'auto', row=1, col=2)
```

```
fig.update_yaxes(tickmode = 'auto', row=2, col=1)
fig.update_yaxes(tickmode = 'auto', row=2, col=2)

# Update yaxis properties
fig.update_xaxes(title_text="Price", tickmode = 'auto',row=1, col=1)
fig.update_xaxes(tickmode = 'auto', row=1, col=2)
fig.update_xaxes(tickmode = 'auto', row=2, col=1)
fig.update_xaxes(tickmode = 'auto', row=2, col=2)

fig.update_xaxes(showgrid=True)
fig.update_yaxes(showgrid=True)

fig.show()

# # Gold vs Bitcoin

# In[7]:

pip install yfinance

# In[9]:

import yfinance as yf
#downlaoding gold price data from yahoofinance
gold_df = yf.download('GLD','2019-01-01','2021-5-24',auto_adjust=True)
gold_df = gold_df[['Close']]
gold_df = gold_df.dropna()

# In[10]:

gold_df

# In[11]:

#comparing this to bitcoin over the same timeframe
fig1 =go.Figure([go.Scatter(x=gold_df.index, y = gold_df['Close'],name = 'Gold')])
fig2 =go.Figure([go.Scatter(x=df['2019-01-1':].index, y = df['weighted_price']['2019-01-1':],name = 'Bitcoin')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig1.data[0], secondary_y=False)
fig.add_trace(fig2.data[0], secondary_y=True)

#plotting vertical lines when microstartegy, telsa and square first bought bitcoin.
fig.add_shape(type="line",
   x0='2020-08-21', y0=110, x1='2020-08-21', y1=200,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2020-10-08', y0=110, x1='2020-10-08', y1=200,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2021-02-08', y0=110, x1='2021-02-08', y1=200,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_trace(go.Scatter(
   x=['2020-08-21', '2020-10-08', '2021-02-08'],
   y=[140,160,195],
   text=['Microstrategy',
        'Square',
         'Tesla'],
   mode="text",
))


fig.update_layout(xaxis = dict(title = '2019-Present'),
                  yaxis=dict(title = 'Gold Price (Oz)'),
```

```
            yaxis2=dict(title = 'Bitcoin price'))

py.iplot(fig)

# In[12]:

#a piece of code to calculate the total bitcoin in issuance between the halvings, will be used for visualisation

halving_2012 = pd.to_datetime('2012-11-28')
halving_2016 = pd.to_datetime('2016-07-09')
halving_2020 = pd.to_datetime('2020-05-11')

issued_supply_list = [(160000*50)] #Aproximately 160000 was the block height on 01-01-12 when my dataset started
total_supply =0
block_reward = 50

for i in df.index:
    total_supply = issued_supply_list[-1]
    if i < halving_2012:
        adjusted_block_reward =  block_reward

    elif (i < halving_2016) & (i > halving_2012):
        adjusted_block_reward =  block_reward/2

    elif (i < halving_2020) & (i > halving_2016):
        adjusted_block_reward = block_reward/4

    elif i > halving_2020:
        adjusted_block_reward = block_reward/8

    issued_supply_list.append(total_supply + adjusted_block_reward *144) #Approx 144 blocks mined per day - Approx One
 Block mined every 10mins

# In[13]:

# comparing the bitcoin price to the total supply in issuance

fig1 =go.Figure([go.Scatter(x=df.index, y = df['weighted_price'],name = 'BTC Price')])
fig2 =go.Figure([go.Scatter(x=df.index, y = issued_supply_list, name = 'BTC Supply Issued')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig1.data[0], secondary_y=False)
fig.add_trace(fig2.data[0], secondary_y=True)


fig.add_shape(type="line",
    x0='2020-05-11', y0=1, x1='2020-05-11', y1=100000,
    line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
    x0='2016-07-09', y0=1, x1='2016-07-09', y1=100000,
    line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
    x0='2012-11-28', y0=1, x1='2012-11-28', y1=100000,
    line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_trace(go.Scatter(
    x=['2020-05-11', '2016-07-09', '2012-11-28'],
    y=[70000,40000,20000],
    text=['Third Halving',
          'Second Halving',
           'First Halving'],
    mode="text",
))
```

```
fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'BTC Price',type="log", range=[np.log10(1), np.log10(100000)]),
                  yaxis2=dict(title = 'BTC Supply Issued'))

py.iplot(fig)

# In[14]:

#this is to find the supply curve
years = [2012]   #start at year 2012

#I've set the loop to stop at 32 becuase the last bitcoin will be mined in 2140. Which is 32 cycles of 4 years away
for i in range(32):
    years.append(years[-1]+4)

divide_50 = [50]
for i in range(32):
    divide_50.append(round(divide_50[-1]/2,6))
supply_frame = pd.DataFrame(divide_50,years)
supply_frame.columns =['Block Issuance']

# In[15]:

# again_these are 'previous to' figures. 50btc per block was rewarded previous to 2012.
supply_frame

# In[16]:

import plotly.graph_objs as go
#plotting block issuance over time
data = [go.Bar(x = years,y = divide_50 , name = 'BTC Block Issuance')]
fig = go.Figure(data=data)

fig.update_layout(xaxis = dict(title = 'Halving Events'),
                  yaxis=dict(title = 'Block Issuance', tickmode = 'array', tickvals = divide_50[:5]))

fig.add_trace(go.Scatter(
    x=['2050','2050','2050'],
   y=[28,15,9],
    text=['25 new BTC per block',
          '12.5 new BTC per block',
          '6.25 new BTC per block'],
    mode="text",
))

py.iplot(fig)

# In[17]:

#This is for calculating supply based on blocks, however blocks were mined on average slightly quicker than 10mins each, s
o
# in the code above it was simpler for visulaisation puposes to calculate total btc issued using date rather than block height

issued_supply_list = []
total_supply =0
block_reward = 50
starting_block_height = 210000

stop_loop_height = 2000000

for count in range(stop_loop_height):
    total_supply = total_supply + block_reward
    issued_supply_list.append(total_supply)
    if count == starting_block_height:
        starting_block_height = starting_block_height+210000
        block_reward = block_reward/2
```

```
issued_supply_frame = pd.DataFrame(issued_supply_list, columns=['supply'])

# In[18]:

issued_supply_list

# In[19]:

#plotting total suppy vs issuance

import plotly.graph_objs as go
#plotting first ten bars only (for visualisation purposes)
fig1 = go.Figure([go.Bar(x = years[:10],y = divide_50[:10] , name = 'BTC Block Issuance')])
fig2 =go.Figure([go.Scatter(x=issued_supply_frame.index, y = issued_supply_frame['supply'], name = 'BTC Supply Issued')
])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig1.data[0], secondary_y=False)
fig.add_trace(fig2.data[0], secondary_y=True)

fig.update_layout(xaxis = dict(title = 'Year',tickmode='array', tickvals= years[:9]),
                  yaxis=dict(title = 'Block Reward'),
                  yaxis2 = dict(title = 'Supply Issued'))

fig.update_layout(xaxis2= {'title':'Block Height','anchor': 'y', 'overlaying': 'x', 'side': 'top','tickmode':'array', 'tickvals' : [21000
0,420000,630000,840000,1050000,1260000,1470000,1680000]})
fig.data[1].update(xaxis='x2')


py.iplot(fig)
```

3. MACD File

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

#reading in datasets

import pandas as pd
df_daily = pd.read_csv('C:/College/Final Year Project/daily_cleaned.csv')
df_daily = df_daily.set_index('Date')
df_daily.index=pd.DatetimeIndex(df_daily.index)

df_weekly = pd.read_csv('C:/College/Final Year Project/weekly_cleaned.csv')
df_weekly = df_weekly.set_index('Unnamed: 0')
df_weekly.index.rename('Date',inplace = True)
df_weekly.index=pd.DatetimeIndex(df_weekly.index)

df_monthly = pd.read_csv('C:/College/Final Year Project/monthly_cleaned.csv')
df_monthly = df_monthly.set_index('Unnamed: 0')
df_monthly.index.rename('Date',inplace = True)
df_monthly.index=pd.DatetimeIndex(df_monthly.index)

# removing first few rows of the dataframes, I found that since indicator values are built on previous data,
# and there is no previous data to produce the firt indicator values, they defaulted to 100. These indicator values at 100 were
removed.

df_daily = df_daily.iloc[4:]
df_weekly = df_weekly.iloc[4:]
```

```
df_monthly = df_monthly.iloc[2:]

# In[2]:

#Zoomin in on one year only
df3 = df_daily.loc[(df_daily.index < pd.to_datetime('2021-12-31')) & (df_daily.index > pd.to_datetime('2021-01-01'))]

# In[3]:

df_monthly

# In[4]:

import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import numpy as np

#since this is just for visualisation purposes to explain how macd works,
# the last 50 rows are not included in the graph because it makes the chart look messy.
fig1 =go.Figure([go.Scatter(x=df3.index, y = df3['weighted_price'][:-50],name = 'BTC Price')])
fig2 =go.Figure([go.Scatter(x=df3.index, y = df3['macd'][:-50],name = 'MACD')])
fig3 =go.Figure([go.Scatter(x=df3.index, y = df3['macds'][:-50],name = 'MACD SIGNAL')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig1.data[0], secondary_y=False)
fig.add_trace(fig2.data[0], secondary_y=True)
fig.add_trace(fig3.data[0], secondary_y=True)

# the overlaps between the signal line and th macd line is represented by either a red or green line depending on the direction
 of overlap.
for i in range(1,len(df3['macds'][:-50])):
    macd = df3['macd']
    signal = df3['macds']

    if macd[i] > signal[i] and macd[i - 1] <= signal[i - 1]:
        fig.add_shape(type="line",
        x0=df3.index[i], y0=20000, x1=df3.index[i], y1=70000,
        line=dict(color="Green",width=3,dash = 'dot'))
    elif macd[i] < signal[i] and macd[i - 1] >= signal[i - 1]:
        fig.add_shape(type="line",
        x0=df3.index[i], y0=20000, x1=df3.index[i], y1=70000,
        line=dict(color="Red",width=3,dash = 'dot'))

fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'Price USD'),
                  yaxis2=dict(title = 'Volume'))

py.iplot(fig)

# In[5]:

#for the period of the start of 2021 to present, what is the end capital following macd trading signals.

capital = 10000
btc_bought = 0
for i in range(1,len(df3['macds'])):
    macd = df3['macd']
    signal = df3['macds']
    price = df3['weighted_price'][i]

    if macd[i] > signal[i-1] and macd[i - 1] <= signal[i - 1]:
        btc_bought = capital/price
    elif macd[i] < signal[i] and macd[i - 1] >= signal[i - 1]:
        if btc_bought != 0:
            capital = btc_bought *price
```

```
        btc_bought = 0
capital

# In[6]:

#macd strategy yearly
macd_df = pd.DataFrame(columns=['Year','Start Capital','End Capital','Profit/Loss'])
#start at year 2012
year =2012
# loop for each of the ten years
for i in range(10):
   #from the start of the year to the end of the year
   df3 = df_daily.loc[(df_daily.index < pd.to_datetime(str(year)+'-12-31')) & (df_daily.index > pd.to_datetime(str(year)+'-
01-01'))]
   #starting capital for each year is 10,000
   capital = 10000
   btc_bought = 0
   for j in range(1,len(df3['macds'])): #for each day
      macd = df3['macd']
      signal = df3['macds']
      price = df3['weighted_price'][j]

      if macd[j] > signal[j-1] and macd[j - 1] <= signal[j - 1]:
          btc_bought = capital/price
      elif macd[j] < signal[j] and macd[j - 1] >= signal[j - 1]:
         # ensuring that I have bitcoin to sell.
         if btc_bought != 0:
            capital = btc_bought *price
            btc_bought = 0
   macd_df = macd_df.append({'Year':str(year),'Start Capital':10000, 'End Capital': round(capital),'Profit/Loss': round(capital
 - 10000)},ignore_index = True)
   year = year+1
print(macd_df)

# In[34]:

#this dca strategy method is not used in final report
#10,000 to invest every year, resetting amount invested at start of every year
dca_df = pd.DataFrame(columns=['Year','Start Capital','End Capital','Profit/Loss'])
year =2012
for i in range(10):

   df3 = df_daily.loc[(df_daily.index < pd.to_datetime(str(year)+'-12-31')) & (df_daily.index > pd.to_datetime(str(year)+'-
01-01'))]
   btc_bought = 0
   for j in range(len(df3['macds'])):
      price = df3['weighted_price'][j]
      btc_bought = btc_bought + 27.4/price
      #10,000/365 = 27.4 invested per day
   capital = btc_bought * df3['weighted_price'][-1] #end
   dca_df = dca_df.append({'Year':str(year),'Start Capital':10000, 'End Capital': round(capital),'Profit/Loss': round(capital - 1
0000)},ignore_index = True)

   year = year+1
print(dca_df)

# In[29]:

df3

# In[38]:

# compared to buy and hold strategy
buyhold_df = pd.DataFrame(columns=['Year','Start Capital','End Capital','Profit/Loss'])
year =2012
capital =10000
for i in range(10):
```

```
    df3 = df_daily.loc[(df_daily.index < pd.to_datetime(str(year)+'-12-31')) & (df_daily.index > pd.to_datetime(str(year)+'-
01-01'))]

    btc_bought = capital/df3['weighted_price'][0] #start
    capital = btc_bought * df3['weighted_price'][-1] #end

    buyhold_df = buyhold_df.append({'Year':str(year),'Start Capital':10000, 'End Capital': round(capital),'Profit/Loss': round(c
apital - 10000)},ignore_index = True)
    capital =10000

    year = year+1
print(buyhold_df)

# In[9]:

#plotting the profit/loss returns of this strategy over each year
import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import numpy as np

fig1 =go.Figure([go.Scatter(x=macd_df['Year'][:], y = macd_df['Profit/Loss'][:],name = 'MACD')])
fig2 =go.Figure([go.Scatter(x=buyhold_df['Year'][:], y = buyhold_df['Profit/Loss'][:],name = 'BUY & HOLD')])
#fig3 =go.Figure([go.Scatter(x=dca_df['Year'][:], y = dca_df['Profit/Loss'][:],name = 'DCA')])

fig=go.Figure()
fig.add_trace(fig1.data[0])
fig.add_trace(fig2.data[0])
#fig.add_trace(fig3.data[0])

fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'Profit USD'))

py.iplot(fig)

# In[10]:

import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import numpy as np

fig1 =go.Figure([go.Scatter(x=macd_df['Year'][2:], y = macd_df['Profit/Loss'][2:],name = 'MACD')])
fig2 =go.Figure([go.Scatter(x=buyhold_df['Year'][2:], y = buyhold_df['Profit/Loss'][2:],name = 'BUY & HOLD')])
#fig3 =go.Figure([go.Scatter(x=dca_df['Year'][2:], y = dca_df['Profit/Loss'][2:],name = 'DCA')])

fig=go.Figure()
fig.add_trace(fig1.data[0])
fig.add_trace(fig2.data[0])
#fig.add_trace(fig3.data[0])

fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'Profit USD'))

py.iplot(fig)

# In[11]:

import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import numpy as np

fig1 =go.Figure([go.Scatter(x=macd_df['Year'][6:], y = macd_df['Profit/Loss'][6:],name = 'MACD')])
```

```
fig2 =go.Figure([go.Scatter(x=buyhold_df['Year'][6:], y = buyhold_df['Profit/Loss'][6:],name = 'BUY & HOLD')])
#fig3 =go.Figure([go.Scatter(x=dca_df['Year'][6:], y = dca_df['Profit/Loss'][6:],name = 'DCA')])

fig=go.Figure()
fig.add_trace(fig1.data[0])
fig.add_trace(fig2.data[0])
#fig.add_trace(fig3.data[0])

fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'Profit USD'))

py.iplot(fig)
```

4. RSI File

```
#!/usr/bin/env python
# coding: utf-8

# In[18]:

# reading csv files
import pandas as pd
df_daily = pd.read_csv('C:/College/Final Year Project/daily_cleaned.csv')
# setting index to date
df_daily = df_daily.set_index('Date')
# changing index type to DateTimeIndex
df_daily.index=pd.DatetimeIndex(df_daily.index)

df_weekly = pd.read_csv('C:/College/Final Year Project/weekly_cleaned.csv')
df_weekly = df_weekly.set_index('Unnamed: 0')
df_weekly.index.rename('Date',inplace = True)
df_weekly.index=pd.DatetimeIndex(df_weekly.index)

df_monthly = pd.read_csv('C:/College/Final Year Project/monthly_cleaned.csv')
df_monthly = df_monthly.set_index('Unnamed: 0')
df_monthly.index.rename('Date',inplace = True)
df_monthly.index=pd.DatetimeIndex(df_monthly.index)

# removing first few rows of the dataframes, I found that since indicator values are built on previous data,
# and there is no previous data to produce the firt indicator values, they defaulted to 100. These indicator values at 100 were
removed.
df_daily = df_daily.iloc[4:]
df_weekly = df_weekly.iloc[4:]
df_monthly = df_monthly.iloc[2:]

# In[19]:

df_daily.tail(5)

# In[20]:

import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import numpy as np

#comparing the price with monthly rsi_12
fig1 =go.Figure([go.Scatter(x=df_daily.index, y = df_daily['weighted_price'],name = 'BTC Price')])
fig2 =go.Figure([go.Scatter(x=df_monthly.index, y = df_monthly['rsi_12'],name = 'RSI_12')])
```

```
fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig1.data[0], secondary_y=False)
fig.add_trace(fig2.data[0], secondary_y=True)

#vertical lines when a halving event occurs
fig.add_shape(type="line",
   x0='2020-05-11', y0=1, x1='2020-05-11', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2016-07-09', y0=1, x1='2016-07-09', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2012-11-28', y0=1, x1='2012-11-28', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_trace(go.Scatter(
   x=['2020-05-11', '2016-07-09', '2012-11-28'],
   y=[50000,10000,1000],
   text=['Third Halving',
       'Second Halving',
        'First Halving'],
   mode="text",
))


fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'BTC Price',type="log", range=[np.log10(5), np.log10(100000)]),
                  yaxis2=dict(title = 'RSI_12'))

py.iplot(fig)

# # Rsi Monthly

# In[21]:

#plotting rsi_12 values side by side for each halving cycle

fig = go.Figure()

fig = make_subplots(rows=1, cols=4,subplot_titles=("Cycle 2009-2012", "Cycle 2012-2016", "Cycle 2016-
2020", "Cycle 2020-2024"))

fig.add_trace(go.Scatter(x=df_monthly['rsi_12'].loc[df_monthly['cycle']==2012].index, y = df_monthly['rsi_12'].loc[df_mon
thly['cycle']==2012],name = '2012 rsi_12'),row=1,col=1)
fig.add_trace(go.Scatter(x=df_monthly['rsi_12'].loc[df_monthly['cycle']==2016].index, y = df_monthly['rsi_12'].loc[df_mon
thly['cycle']==2016],name = '2016 rsi_12'),row=1,col=2)
fig.add_trace(go.Scatter(x=df_monthly['rsi_12'].loc[df_monthly['cycle']==2020].index, y = df_monthly['rsi_12'].loc[df_mon
thly['cycle']==2020],name = '2020 rsi_12'),row=1,col=3)
fig.add_trace(go.Scatter(x=df_monthly['rsi_12'].loc[df_monthly['cycle']==2024].index, y = df_monthly['rsi_12'].loc[df_mon
thly['cycle']==2024],name = '2024 rsi_12'),row=1,col=4)

# Update yaxis properties
fig.update_yaxes(title_text="Price", tickmode = 'auto',row=1, col=1)
fig.update_yaxes(tickmode = 'auto', row=1, col=2)
fig.update_yaxes(tickmode = 'auto', row=2, col=1)
fig.update_yaxes(tickmode = 'auto', row=2, col=2)

# Update yaxis properties
fig.update_xaxes(title_text="Price", tickmode = 'auto',row=1, col=1)
fig.update_xaxes(tickmode = 'auto', row=1, col=2)
fig.update_xaxes(tickmode = 'auto', row=2, col=1)
fig.update_xaxes(tickmode = 'auto', row=2, col=2)

fig.update_xaxes(showgrid=True)
```

```
fig.update_yaxes(showgrid=True)

fig.show()

# In[22]:

# checking correlation of halvings (monthly data points)

correlation_frame =pd.DataFrame()
correlation_frame['2020 Cycle RSI'] = pd.Series(df_monthly['rsi_12'].loc[df_monthly['cycle']==2020].values)
correlation_frame['2012 Cycle RSI'] = pd.Series(df_monthly['rsi_12'].loc[df_monthly['cycle']==2012].values)
correlation_frame['2016 Cycle RSI'] = pd.Series(df_monthly['rsi_12'].loc[df_monthly['cycle']==2016].values)
correlation_frame['2024 Cycle RSI'] = pd.Series(df_monthly['rsi_12'].loc[df_monthly['cycle']==2024].values)
correlation_frame['2020 Cycle price'] = pd.Series(df_monthly['weighted_price'].loc[df_monthly['cycle']==2020].values)
correlation_frame['2012 Cycle price'] = pd.Series(df_monthly['weighted_price'].loc[df_monthly['cycle']==2012].values)
correlation_frame['2016 Cycle price'] = pd.Series(df_monthly['weighted_price'].loc[df_monthly['cycle']==2016].values)
correlation_frame['2024 Cycle price'] = pd.Series(df_monthly['weighted_price'].loc[df_monthly['cycle']==2024].values)
correlation_frame = correlation_frame[['2012 Cycle RSI','2016 Cycle RSI','2020 Cycle RSI','2024 Cycle RSI']]
#correlation_frame = correlation_frame[['2012 Cycle RSI','2016 Cycle RSI','2020 Cycle RSI','2024 Cycle RSI','2012 Cycle p
rice','2016 Cycle price','2020 Cycle price','2024 Cycle price']]

correlation_frame.corr(method='pearson')
#correlation_frame

# In[23]:

correlation_frame =pd.DataFrame()
correlation_frame['RSI'] = pd.Series(df_monthly['rsi_12'])
correlation_frame['Price'] = pd.Series(df_monthly['weighted_price'])
correlation_frame.corr(method='pearson')

# # RSI weekly

# In[24]:

import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import numpy as np

#comparing the price with monthly rsi_12
fig1 =go.Figure([go.Scatter(x=df_daily.index, y = df_daily['weighted_price'],name = 'BTC Price')])
fig2 =go.Figure([go.Scatter(x=df_weekly.index, y = df_weekly['rsi_14'],name = 'RSI_14')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig1.data[0], secondary_y=False)
fig.add_trace(fig2.data[0], secondary_y=True)

#vertical lines when a halving event occurs
fig.add_shape(type="line",
    x0='2020-05-11', y0=1, x1='2020-05-11', y1=100000,
    line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
    x0='2016-07-09', y0=1, x1='2016-07-09', y1=100000,
    line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
    x0='2012-11-28', y0=1, x1='2012-11-28', y1=100000,
    line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_trace(go.Scatter(
    x=['2020-05-11', '2016-07-09', '2012-11-28'],
    y=[50000,10000,1000],
    text=['Third Halving',
        'Second Halving',
```

```
        'First Halving'],
    mode="text",
))


fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'BTC Price',type="log", range=[np.log10(5), np.log10(100000)]),
                  yaxis2=dict(title = 'RSI_12'))

py.iplot(fig)

# In[25]:

# plotting weekly rsi_14 values across halving cycles
import plotly.offline as py
import plotly.graph_objs as go
fig = go.Figure()

fig = make_subplots(rows=1, cols=4,subplot_titles=("Cycle 2009-2012", "Cycle 2012-2016", "Cycle 2016-
2020", "Cycle 2020-2024"))

fig.add_trace(go.Scatter(x=df_weekly['rsi_14'].loc[df_weekly['cycle']==2012].index, y = df_weekly['rsi_14'].loc[df_weekly[
'cycle']==2012],name = '2012 rsi_14'),row=1,col=1)
fig.add_trace(go.Scatter(x=df_weekly['rsi_14'].loc[df_weekly['cycle']==2016].index, y = df_weekly['rsi_14'].loc[df_weekly[
'cycle']==2016],name = '2016 rsi_14'),row=1,col=2)
fig.add_trace(go.Scatter(x=df_weekly['rsi_14'].loc[df_weekly['cycle']==2020].index, y = df_weekly['rsi_14'].loc[df_weekly[
'cycle']==2020],name = '2020 rsi_14'),row=1,col=3)
fig.add_trace(go.Scatter(x=df_weekly['rsi_14'].loc[df_weekly['cycle']==2024].index, y = df_weekly['rsi_14'].loc[df_weekly[
'cycle']==2024],name = '2024 rsi_14'),row=1,col=4)

# Update yaxis properties
fig.update_yaxes(title_text="Price", tickmode = 'auto',row=1, col=1)
fig.update_yaxes(tickmode = 'auto', row=1, col=2)
fig.update_yaxes(tickmode = 'auto', row=2, col=1)
fig.update_yaxes(tickmode = 'auto', row=2, col=2)

# Update yaxis properties
fig.update_xaxes(title_text="Price", tickmode = 'auto',row=1, col=1)
fig.update_xaxes(tickmode = 'auto', row=1, col=2)
fig.update_xaxes(tickmode = 'auto', row=2, col=1)
fig.update_xaxes(tickmode = 'auto', row=2, col=2)

fig.update_xaxes(showgrid=True)
fig.update_yaxes(showgrid=True)

fig.show()

# In[26]:

# correlation of rsi_14 across halvings (weekly values)
correlation_frame =pd.DataFrame()
correlation_frame['2020 Cycle'] = pd.Series(df_weekly['rsi_14'].loc[df_weekly['cycle']==2020].values)
correlation_frame['2012 Cycle'] = pd.Series(df_weekly['rsi_14'].loc[df_weekly['cycle']==2012].values)
correlation_frame['2016 Cycle'] = pd.Series(df_weekly['rsi_14'].loc[df_weekly['cycle']==2016].values)
correlation_frame['2024 Cycle'] = pd.Series(df_weekly['rsi_14'].loc[df_weekly['cycle']==2024].values)
correlation_frame = correlation_frame[['2012 Cycle','2016 Cycle','2020 Cycle','2024 Cycle']]

correlation_frame.corr(method='pearson')
#correlation_frame

# In[27]:

correlation_frame =pd.DataFrame()
correlation_frame['RSI'] = pd.Series(df_weekly['rsi_14'])
correlation_frame['Price'] = pd.Series(df_weekly['weighted_price'])
correlation_frame.corr(method='pearson')
```

```
# # RSI Weekly Strategy

# In[28]:

#How accurately can RSI predict price movements over weekly price
profit_df = pd.DataFrame(columns=['Year','Start Capital','End Capital','Profit/Loss'])
#starting year
year =2012
# number of years
for i in range(10):
    #between the start of the year and end of the year
    df3 = df_weekly.loc[(df_weekly.index < pd.to_datetime(str(year)+'-12-
31')) & (df_weekly.index > pd.to_datetime(str(year)+'-01-01'))]
    # starting capital
    capital = 10000
    # starting btc_bought
    btc_bought = 0
    # for each row
    for j in range(1,len(df3['rsi_14'])):
        rsi_14 = df3['rsi_14']
        price = df3['weighted_price'][j]
        # upper bound of 70, lower bound of 50
        # if the rsi is above 70, sell the bitcoin for capital, if the rsi is below 50, convert the capital into bitcoin
        if rsi_14[j] > 70 and 70 <= rsi_14[j - 1]:
            # chack that I have bitcoin to sell
            if btc_bought != 0:
                capital = btc_bought *price
                btc_bought = 0
        elif rsi_14[j] < 50 and 50 >= rsi_14[j - 1]:
            btc_bought = capital/price
    profit_df = profit_df.append({'Year':str(year),'Start Capital':10000, 'End Capital': round(capital),'Profit/Loss': round(capita
l - 10000)},ignore_index = True)
    year = year+1

print(profit_df)

# # RSI Daily

# In[29]:

import plotly.offline as py
import plotly.graph_objs as go
fig = go.Figure()

fig = make_subplots(rows=1, cols=4,subplot_titles=("Cycle 2009-2012", "Cycle 2012-2016", "Cycle 2016-
2020", "Cycle 2020-2024"))

fig.add_trace(go.Scatter(x=df_daily['rsi_14'].loc[df_daily['cycle']==2012].index, y = df_daily['rsi_14'].loc[df_daily['cycle']=
=2012],name = '2012 rsi_12'),row=1,col=1)
fig.add_trace(go.Scatter(x=df_daily['rsi_14'].loc[df_daily['cycle']==2016].index, y = df_daily['rsi_14'].loc[df_daily['cycle']=
=2016],name = '2016 rsi_12'),row=1,col=2)
fig.add_trace(go.Scatter(x=df_daily['rsi_14'].loc[df_daily['cycle']==2020].index, y = df_daily['rsi_14'].loc[df_daily['cycle']=
=2020],name = '2020 rsi_12'),row=1,col=3)
fig.add_trace(go.Scatter(x=df_daily['rsi_14'].loc[df_daily['cycle']==2024].index, y = df_daily['rsi_14'].loc[df_daily['cycle']=
=2024],name = '2024 rsi_12'),row=1,col=4)

# Update yaxis properties
fig.update_yaxes(title_text="Price", tickmode = 'auto',row=1, col=1)
fig.update_yaxes(tickmode = 'auto', row=1, col=2)
fig.update_yaxes(tickmode = 'auto', row=2, col=1)
fig.update_yaxes(tickmode = 'auto', row=2, col=2)

# Update yaxis properties
fig.update_xaxes(title_text="Price", tickmode = 'auto',row=1, col=1)
fig.update_xaxes(tickmode = 'auto', row=1, col=2)
fig.update_xaxes(tickmode = 'auto', row=2, col=1)
fig.update_xaxes(tickmode = 'auto', row=2, col=2)
```

```
fig.update_xaxes(showgrid=True)
fig.update_yaxes(showgrid=True)

fig.show()

# In[30]:

correlation_frame =pd.DataFrame()
correlation_frame['2020 Cycle'] = pd.Series(df_daily['rsi_14'].loc[df_daily['cycle']==2020].values)
correlation_frame['2012 Cycle'] = pd.Series(df_daily['rsi_14'].loc[df_daily['cycle']==2012].values)
correlation_frame['2016 Cycle'] = pd.Series(df_daily['rsi_14'].loc[df_daily['cycle']==2016].values)
correlation_frame['2024 Cycle'] = pd.Series(df_daily['rsi_14'].loc[df_daily['cycle']==2024].values)
correlation_frame = correlation_frame[['2012 Cycle','2016 Cycle','2020 Cycle','2024 Cycle']]

correlation_frame.corr(method='pearson')
#correlation_frame

# In[53]:

#How accurately can RSI predict price movements over daily price
profit_df = pd.DataFrame(columns=['Year','Start Capital','End Capital','Profit/Loss'])
year =2012
for i in range(10):
    df3 = df_daily.loc[(df_daily.index < pd.to_datetime(str(year)+'-12-31')) & (df_daily.index > pd.to_datetime(str(year)+'-
01-01'))]
    #print(df3)
    capital = 10000
    btc_bought = 0
    for j in range(1,len(df3['rsi_14'])):
        rsi_14 = df3['rsi_14']
        price = df3['weighted_price'][j]

        if rsi_14[j] > 70 and 70 <= rsi_14[j - 1]:
            if btc_bought != 0:
                capital = btc_bought *price
                btc_bought = 0
        elif rsi_14[j] < 50 and 50 >= rsi_14[j - 1]:
            btc_bought = capital/price
    profit_df = profit_df.append({'Year':str(year),'Start Capital':10000, 'End Capital': round(capital),'Profit/Loss': round(capita
l - 10000)},ignore_index = True)
    year = year+1
print(profit_df)

# In[54]:

df_daily

# In[55]:

# compared to buy and hold strategy
profit_df = pd.DataFrame(columns=['Year','Start Capital','End Capital','Profit/Loss'])
year =2012
capital =10000
for i in range(10):
    df3 = df_daily.loc[(df_daily.index < pd.to_datetime(str(year)+'-12-31')) & (df_daily.index > pd.to_datetime(str(year)+'-
01-01'))]

    btc_bought = capital/df3['weighted_price'][0] #start
    capital = btc_bought * df3['weighted_price'][-1] #end

    profit_df = profit_df.append({'Year':str(year),'Start Capital':10000, 'End Capital': round(capital),'Profit/Loss': round(capita
l - 10000)},ignore_index = True)
    capital =10000

    year = year+1
print(profit_df)
```

```
# In[ ]:



# In[9]:

correlation_frame =pd.DataFrame()
correlation_frame['RSI'] = pd.Series(df_daily['rsi_14'])
correlation_frame['Price'] = pd.Series(df_daily['weighted_price'])
correlation_frame.corr(method='pearson')
```

5.Volume File

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

# reading in csv files
import pandas as pd
df_daily = pd.read_csv('C:/College/Final Year Project/daily_cleaned.csv')
df_daily = df_daily.set_index('Date')
df_daily.index=pd.DatetimeIndex(df_daily.index)

df_weekly = pd.read_csv('C:/College/Final Year Project/weekly_cleaned.csv')
df_weekly = df_weekly.set_index('Unnamed: 0')
df_weekly.index.rename('Date',inplace = True)
df_weekly.index=pd.DatetimeIndex(df_weekly.index)

df_monthly = pd.read_csv('C:/College/Final Year Project/monthly_cleaned.csv')
df_monthly = df_monthly.set_index('Unnamed: 0')
df_monthly.index.rename('Date',inplace = True)
df_monthly.index=pd.DatetimeIndex(df_monthly.index)

# removing first few rows of the dataframes, I found that since indicator values are built on previous data,
# and there is no previous data to produce the firt indicator values, they defaulted to 100. These indicator values at 100 were
removed.

df_daily = df_daily.iloc[4:]
df_weekly = df_weekly.iloc[4:]
df_monthly = df_monthly.iloc[2:]

# In[2]:

import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
import plotly.express as px
from plotly.subplots import make_subplots
import numpy as np

#plotting the price against the volatility volume ratio
fig1 =go.Figure([go.Scatter(x=df_monthly.index, y = df_monthly['weighted_price'],name = 'BTC Price')])
fig2 =go.Figure([go.Scatter(x=df_monthly.index, y = df_monthly['vr'],name = 'Volatility Volume Ratio')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig1.data[0], secondary_y=False)
fig.add_trace(fig2.data[0], secondary_y=True)
```

```
fig.add_shape(type="line",
   x0='2020-05-11', y0=1, x1='2020-05-11', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2016-07-09', y0=1, x1='2016-07-09', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2012-11-28', y0=1, x1='2012-11-28', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_trace(go.Scatter(
   x=['2020-05-11', '2016-07-09', '2012-11-28'],
   y=[50000,10000,1000],
   text=['Third Halving',
       'Second Halving',
        'First Halving'],
   mode="text",
))


fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'Price USD',type="log", range=[np.log10(1), np.log10(100000)]),
                  yaxis2=dict(title = 'Volume',type="log", range=[np.log10(1), np.log10(10000)]))

py.iplot(fig)

# In[3]:

#focusing on the OBV value with price over 1 halving cycle

import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
import plotly.express as px
from plotly.subplots import make_subplots
import numpy as np

fig1 =go.Figure([go.Scatter(x=df_weekly['weighted_price'].loc[df_weekly['cycle']==2020].index, y = df_weekly['weighted_
price'].loc[df_weekly['cycle']==2020],name = 'BTC Price')])
fig2 =go.Figure([go.Scatter(x=df_weekly['Obv'].loc[df_weekly['cycle']==2020].index, y = df_weekly['Obv'].loc[df_weekly['
cycle']==2020],name = 'On Balance Volume')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig1.data[0], secondary_y=False)
fig.add_trace(fig2.data[0], secondary_y=True)

fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'Price USD'),
                  yaxis2=dict(title = 'OBV'))

py.iplot(fig)

# In[ ]:



# In[4]:

#checking correlation between the obv and the price.
correlation_frame =pd.DataFrame()
correlation_frame['2020 Cycle'] = pd.Series(df_monthly['vr'].loc[df_monthly['cycle']==2020].values)
correlation_frame['2012 Cycle'] = pd.Series(df_monthly['vr'].loc[df_monthly['cycle']==2012].values)
correlation_frame['2016 Cycle'] = pd.Series(df_monthly['vr'].loc[df_monthly['cycle']==2016].values)
correlation_frame['2024 Cycle'] = pd.Series(df_monthly['vr'].loc[df_monthly['cycle']==2024].values)
```

```
correlation_frame = correlation_frame[['2012 Cycle','2016 Cycle','2020 Cycle','2024 Cycle']]

correlation_frame.corr(method='pearson')
#correlation_frame

# In[5]:

#plotting the price over each halving cycle and the OBV value over each halving cycle.

fig = go.Figure()

fig = make_subplots(rows=2, cols=4,subplot_titles=("Cycle 2009-2012", "Cycle 2012-2016","Cycle 2016-
2020", "Cycle 2020-2024"))

fig.add_trace(go.Scatter(x=df_monthly['weighted_price'].loc[df_monthly['cycle']==2012].index, y = df_monthly['weighted_
price'].loc[df_monthly['cycle']==2012],name = '2012 price'),row=1,col=1)
fig.add_trace(go.Scatter(x=df_monthly['weighted_price'].loc[df_monthly['cycle']==2016].index, y = df_monthly['weighted_
price'].loc[df_monthly['cycle']==2016],name = '2016 price'),row=1,col=2)
fig.add_trace(go.Scatter(x=df_monthly['weighted_price'].loc[df_monthly['cycle']==2020].index, y = df_monthly['weighted_
price'].loc[df_monthly['cycle']==2020],name = '2020 price'),row=1,col=3)
fig.add_trace(go.Scatter(x=df_monthly['weighted_price'].loc[df_monthly['cycle']==2024].index, y = df_monthly['weighted_
price'].loc[df_monthly['cycle']==2024],name = '2024 price'),row=1,col=4)

fig.add_trace(go.Scatter(x=df_monthly['Obv'].loc[df_monthly['cycle']==2012].index, y = df_monthly['Obv'].loc[df_monthly
['cycle']==2012],name = '2012 Obv'),row=2,col=1)
fig.add_trace(go.Scatter(x=df_monthly['Obv'].loc[df_monthly['cycle']==2016].index, y = df_monthly['Obv'].loc[df_monthly
['cycle']==2016],name = '2016 Obv'),row=2,col=2)
fig.add_trace(go.Scatter(x=df_monthly['Obv'].loc[df_monthly['cycle']==2020].index, y = df_monthly['Obv'].loc[df_monthly
['cycle']==2020],name = '2020 Obv'),row=2,col=3)
fig.add_trace(go.Scatter(x=df_monthly['Obv'].loc[df_monthly['cycle']==2024].index, y = df_monthly['Obv'].loc[df_monthly
['cycle']==2024],name = '2024 Obv'),row=2,col=4)

# Update yaxis properties
fig.update_yaxes(title_text="Price", tickmode = 'auto',row=1, col=1)
fig.update_yaxes(tickmode = 'auto', row=2, col=1)

# Update yaxis properties
fig.update_yaxes(title_text="OBV", tickmode = 'auto',row=2, col=1)
fig.update_yaxes(tickmode = 'auto', row=2, col=1)

fig.update_xaxes(showgrid=True)
fig.update_yaxes(showgrid=True)

fig.show()

# In[6]:

correlation_frame =pd.DataFrame()
correlation_frame['2020 Cycle'] = pd.Series(df_monthly['Obv'].loc[df_monthly['cycle']==2020].values)
correlation_frame['2012 Cycle'] = pd.Series(df_monthly['Obv'].loc[df_monthly['cycle']==2012].values)
correlation_frame['2016 Cycle'] = pd.Series(df_monthly['Obv'].loc[df_monthly['cycle']==2016].values)
correlation_frame['2024 Cycle'] = pd.Series(df_monthly['Obv'].loc[df_monthly['cycle']==2024].values)
correlation_frame = correlation_frame[['2012 Cycle','2016 Cycle','2020 Cycle','2024 Cycle']]

correlation_frame.corr(method='pearson')
#correlation_frame

# In[7]:

# correlation matrix between the price at each halving cycle and the OBV at each halving cycle

correlation_frame =pd.DataFrame()
correlation_frame['2020 Cycle OBV'] = pd.Series(df_monthly['Obv'].loc[df_monthly['cycle']==2020].values)
correlation_frame['2012 Cycle OBV'] = pd.Series(df_monthly['Obv'].loc[df_monthly['cycle']==2012].values)
correlation_frame['2016 Cycle OBV'] = pd.Series(df_monthly['Obv'].loc[df_monthly['cycle']==2016].values)
correlation_frame['2024 Cycle OBV'] = pd.Series(df_monthly['Obv'].loc[df_monthly['cycle']==2024].values)
correlation_frame['2020 Cycle price'] = pd.Series(df_monthly['weighted_price'].loc[df_monthly['cycle']==2020].values)
```

```
correlation_frame['2012 Cycle price'] = pd.Series(df_monthly['weighted_price'].loc[df_monthly['cycle']==2012].values)
correlation_frame['2016 Cycle price'] = pd.Series(df_monthly['weighted_price'].loc[df_monthly['cycle']==2016].values)
correlation_frame['2024 Cycle price'] = pd.Series(df_monthly['weighted_price'].loc[df_monthly['cycle']==2024].values)
correlation_frame = correlation_frame[['2012 Cycle OBV','2016 Cycle OBV','2020 Cycle OBV','2024 Cycle OBV','2012 Cy
cle price','2016 Cycle price','2020 Cycle price','2024 Cycle price']]

correlation_frame.corr(method='pearson')
#correlation_frame

# In[8]:

correlation_frame = pd.DataFrame()
correlation_frame['OBV'] = df_daily['Obv']
correlation_frame['Price'] = df_daily['weighted_price']

correlation_frame.corr(method='pearson')

# In[9]:

correlation_frame = pd.DataFrame()
correlation_frame['OBV'] = df_weekly['Obv']
correlation_frame['Price'] = df_weekly['weighted_price']

correlation_frame.corr(method='pearson')

# In[11]:

correlation_frame = pd.DataFrame()
correlation_frame['OBV'] = df_monthly['Obv']
correlation_frame['Price'] = df_monthly['weighted_price']

correlation_frame.corr(method='pearson')
```

6 On-Chain indicators File

```
#!/usr/bin/env python
# coding: utf-8

# In[1]:

#reading in csv downloaded from https://coinmetrics.io/community-network-data/
import pandas as pd
df = pd.read_csv('C:/College/Final Year Project/btccoinmetrics.csv')

# In[2]:

print(list(df))

# In[3]:

#these are the indicators I am intrested in, removing other columns from dataframe
df = df[['date','PriceUSD','NVTAdj90','CapMVRVCur','FeeTotNtv']]
df.index = df['date']
#starting the dataframe from the start of 2012 so that a fair comparison can be made to other indicators
df = df[1093:]
df.index=pd.DatetimeIndex(df.index)

# In[4]:

df

# In[5]:
```

```
#adding a column to the data to specify which halving cycle the data is in

halving_2012 = pd.to_datetime('2012-11-28')
halving_2016 = pd.to_datetime('2016-07-09')
halving_2020 = pd.to_datetime('2020-05-11')

cycle1 = df.loc[df.index < halving_2012]
cycle2 = df.loc[(df.index < halving_2016) & (df.index > halving_2012)]
cycle3 = df.loc[(df.index < halving_2020) & (df.index > halving_2016)]
cycle4 = df.loc[df.index > halving_2020]

cycle1['Cycle'] = '2012'
cycle2['Cycle'] = '2016'
cycle3['Cycle'] = '2020'
cycle4['Cycle'] = '2024'

frames = [cycle1,cycle2,cycle3,cycle4]
df = pd.concat(frames,sort = False)

# # MVRV Ratio

# In[6]:

#plotting the MVRV ratio with the Price
import plotly.offline as py
import plotly.graph_objs as go
import plotly.figure_factory as ff
from plotly.subplots import make_subplots
import numpy as np

fig1 =go.Figure([go.Scatter(x=df.index, y = df['CapMVRVCur'],name = 'MVRV')])
fig2 =go.Figure([go.Scatter(x=df.index, y = df['PriceUSD'],name = 'BTC Price')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig2.data[0], secondary_y=False)
fig.add_trace(fig1.data[0], secondary_y=True)


fig.add_shape(type="line",
   x0='2020-05-11', y0=1, x1='2020-05-11', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2016-07-09', y0=1, x1='2016-07-09', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2012-11-28', y0=1, x1='2012-11-28', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_trace(go.Scatter(
   x=['2020-05-11', '2016-07-09', '2012-11-28'],
   y=[70000,40000,20000],
   text=['Third Halving',
        'Second Halving',
         'First Halving'],
   mode="text",
))

fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'BTC Price',type="log", range=[np.log10(1), np.log10(100000)]),
                  yaxis2=dict(title = 'MVRV Capitalisation'))

py.iplot(fig)

# In[7]:
```

```
#chceking the correlaion between the MVRV ratios across halving cycles
correlation_frame =pd.DataFrame()
correlation_frame['2020 Cycle'] = pd.Series(df['CapMVRVCur'].loc[df['Cycle']=='2020'].values)
correlation_frame['2012 Cycle'] = pd.Series(df['CapMVRVCur'].loc[df['Cycle']=='2012'].values)
correlation_frame['2016 Cycle'] = pd.Series(df['CapMVRVCur'].loc[df['Cycle']=='2016'].values)
correlation_frame['2024 Cycle'] = pd.Series(df['CapMVRVCur'].loc[df['Cycle']=='2024'].values)

correlation_frame.corr(method='pearson')
#correlation_frame

# In[8]:

#the MVRV ratio correlation to price
correlation_frame = pd.DataFrame()
correlation_frame['MVRV'] = df['CapMVRVCur']
correlation_frame['Price'] = df['PriceUSD']

correlation_frame.corr(method='pearson')

# # NVT Signal

# In[9]:

# Plotting NVT Signal to price
fig1 =go.Figure([go.Scatter(x=df.index, y = df['NVTAdj90'],name = 'NVT')])
fig2 =go.Figure([go.Scatter(x=df.index, y = df['PriceUSD'],name = 'BTC Price')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig2.data[0], secondary_y=False)
fig.add_trace(fig1.data[0], secondary_y=True)


fig.add_shape(type="line",
   x0='2020-05-11', y0=1, x1='2020-05-11', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2016-07-09', y0=1, x1='2016-07-09', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2012-11-28', y0=1, x1='2012-11-28', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_trace(go.Scatter(
   x=['2020-05-11', '2016-07-09', '2012-11-28'],
   y=[70000,40000,20000],
   text=['Third Halving',
        'Second Halving',
         'First Halving'],
   mode="text",
))

fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'BTC Price',type="log", range=[np.log10(1), np.log10(100000)]),
                  yaxis2=dict(title = 'NVT'))

py.iplot(fig)

# In[10]:

# checking correlation between NVT signals across halving cycles
correlation_frame =pd.DataFrame()
correlation_frame['2020 Cycle'] = pd.Series(df['NVTAdj90'].loc[df['Cycle']=='2020'].values)
correlation_frame['2012 Cycle'] = pd.Series(df['NVTAdj90'].loc[df['Cycle']=='2012'].values)
correlation_frame['2016 Cycle'] = pd.Series(df['NVTAdj90'].loc[df['Cycle']=='2016'].values)
```

```
correlation_frame['2024 Cycle'] = pd.Series(df['NVTAdj90'].loc[df['Cycle']=='2024'].values)

correlation_frame.corr(method='pearson')
#correlation_frame

# In[11]:

# correlation between NVT signal to MVRV ratio to Price
correlation_frame = pd.DataFrame()
correlation_frame['MVRV'] = df['CapMVRVCur']
correlation_frame['NVT'] = df['NVTAdj90']
correlation_frame['Price'] = df['PriceUSD']

correlation_frame.corr(method='pearson')

# # Fee per Day

# In[12]:

#checking correlation of the total Fees from blocks in a day to the price

fig1 =go.Figure([go.Scatter(x=df.index, y = df['FeeTotNtv'],name = 'Fee Per Day')])
fig2 =go.Figure([go.Scatter(x=df.index, y = df['PriceUSD'],name = 'BTC Price')])

fig = make_subplots(specs=[[{"secondary_y": True}]])
fig.add_trace(fig2.data[0], secondary_y=False)
fig.add_trace(fig1.data[0], secondary_y=True)


fig.add_shape(type="line",
   x0='2020-05-11', y0=1, x1='2020-05-11', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2016-07-09', y0=1, x1='2016-07-09', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_shape(type="line",
   x0='2012-11-28', y0=1, x1='2012-11-28', y1=100000,
   line=dict(color="MediumPurple",width=3,dash = 'dot'))

fig.add_trace(go.Scatter(
   x=['2020-05-11', '2016-07-09', '2012-11-28'],
   y=[70000,40000,20000],
   text=['Third Halving',
        'Second Halving',
         'First Halving'],
   mode="text",
))

fig.update_layout(xaxis = dict(title = 'Year'),
                  yaxis=dict(title = 'BTC Price',type="log", range=[np.log10(1), np.log10(100000)]),
                  yaxis2=dict(title = 'Fee per Day',type="log", range=[np.log10(1), np.log10(100000)]))

py.iplot(fig)

# In[13]:

#checking correlation between NVT ratio across halving cycles.

correlation_frame =pd.DataFrame()
correlation_frame['2020 Cycle'] = pd.Series(df['FeeTotNtv'].loc[df['Cycle']=='2020'].values)
correlation_frame['2012 Cycle'] = pd.Series(df['FeeTotNtv'].loc[df['Cycle']=='2012'].values)
correlation_frame['2016 Cycle'] = pd.Series(df['FeeTotNtv'].loc[df['Cycle']=='2016'].values)
correlation_frame['2024 Cycle'] = pd.Series(df['FeeTotNtv'].loc[df['Cycle']=='2024'].values)

correlation_frame.corr(method='pearson')
```

```
#correlation_frame

# In[14]:

#correlation between MVRV ratio / NVT signal / Block fees per day / Price
correlation_frame = pd.DataFrame()
correlation_frame['MVRV'] = df['CapMVRVCur']
correlation_frame['NVT'] = df['NVTAdj90']
correlation_frame['Fee'] = df['FeeTotNtv']
correlation_frame['Price'] = df['PriceUSD']

correlation_frame.corr(method='pearson')

# In[15]:

#Writing this new df to a csv so it can be used for machine learning in a different notebook
df.to_csv('C:/College/Final Year Project/new_indicators.csv')
```

7 Machine learning File

```
#!/usr/bin/env python
# coding: utf-8

# In[144]:

# importing libraries
import numpy as np
import pandas as pd

import plotly.graph_objs as go

from sklearn.preprocessing import StandardScaler
from sklearn.neural_network import MLPRegressor
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import r2_score
from sklearn.model_selection import TimeSeriesSplit

from keras.models import Sequential
from keras.layers import LSTM, Dense, Dropout
from keras.utils.vis_utils import plot_model

# # MLP (Neural Net) Traditional indicators

# In[145]:

# a method to calculate the Mean_Absolute_Percentage_Error (MAPE)

def percentage_error(actual, predicted):
    res = np.empty(actual.shape)
    for j in range(actual.shape[0]):
        if actual[j] != 0:
            res[j] = (actual[j] - predicted[j]) / actual[j]
        else:
            res[j] = predicted[j] / np.mean(actual)
    return res

def mean_absolute_percentage_error(y_true, y_pred):
    return np.mean(np.abs(percentage_error(np.asarray(y_true), np.asarray(y_pred)))) * 100
```

```
# In[146]:

# setting up an MLPRegressor function
def myMLPRegressor(x_train,y_train,x_test,y_test):
    # Initialising standard scaler
    scaler = StandardScaler()
    # Fitting the scaler with x_train data
    scaler.fit(x_train)
    #scaling x_train and y_train data
    x_train_scaled = scaler.transform(x_train)
    x_test_scaled = scaler.transform(x_test)

    #creating the model:
        # setting random_state to 1: This ensures the same result is produced across calls, by determining the same weight and b
ias initialisation
        # 10000 maximum number of iterations
        # 100 is the default hidden layer size. (number of neurons in hidden layer)
        # setting activation function to identity. returns returns f(x) = x
        # setting learning rate to adaptive, this means that the learning rate is constant if the training loss decreases.
    MLP = MLPRegressor(random_state=1, max_iter=10000, hidden_layer_sizes = (100), activation = 'identity',learning_rate
 = 'adaptive').fit(x_train_scaled, y_train)
    # use the model to precit the price based on the scaled x_test data
    MLP_pred = MLP.predict(x_test_scaled)

    #creating a dataframe for binary classification (up or down)
    classification = pd.DataFrame(data=y_test,columns =['test'])
    #if the price in price data increases from previous day, input 1. Else input 0
    classification['test'] = np.where(classification['test'].shift(-1) > classification['test'], 1, 0)
    classification['pred'] = MLP_pred
    classification['pred'] = np.where(classification['pred'].shift(-1) > classification['pred'], 1, 0)

    MLP_acc = accuracy_score(classification['test'],classification['pred'])

    # calculate the mean squared error
    MLP_MSE = mean_squared_error(y_test, MLP_pred)

    MLP_MAPE =mean_absolute_percentage_error(y_test, MLP_pred)
    # calculate the r2 score
    MLP_R2 = r2_score(MLP_pred, y_test)

    return MLP_R2 , MLP_MSE, MLP_pred, MLP_MAPE, MLP_acc

# In[147]:

def runMLP(pre_2012,pre_2016,pre_2020,pre_2024,indicator_set):
# list of frames, will be used to iterate through them
    frames = [pre_2012,pre_2016,pre_2020,pre_2024]
    # will be used to print out which cycle the data belongs to
    frames_string = ['2009-2012','2012-2016','2016-2020','2020-2024']
    MLP_results = pd.DataFrame({'Cycle':[],'R2 score':[],'MSE_score':[]})

    count = 0
    for df in frames:

        shift = -10
        df['Price_lag'] = df['Price'].shift(shift)

        train_pt = int(len(df)*.8)
        train = df.iloc[:train_pt,:]
        test = df.iloc[train_pt:,:]

        # x training on indicators only
        x_train = train.iloc[:shift,1:-1]
        # y training on what the indicators should predict after 10 days
        y_train = train['Price_lag'][:shift]
        x_test = test.iloc[:shift,1:-1]
        y_test = test['Price'][:shift]
```

```
        # changing y_test to an array, output of MLP_pred is array, so to compare MLP_pred to y_test, y_test needs to be an arr
ay
        y_test = np.array(y_test)

        # calling the MLPRegressor method and supplying inputs
        MLP_R2, MLP_MSE, MLP_pred, MLP_MAPE, MLP_acc = myMLPRegressor(x_train,y_train,x_test,y_test)

        #creating a row of the cycles scores
        new_row = pd.Series(data={'Cycle':frames_string[count],'R2 score':MLP_R2,'MSE_score':MLP_MSE, 'MAPE':MLP_
MAPE, 'classification':MLP_acc})

        # appending the results to the dataframe
        MLP_results = MLP_results.append(new_row,ignore_index = True)

        #plotting the prection vs the actual
        fig1 = go.Scatter(x=train.index,y=train['Price'],name = 'Train Actual')
        fig2 = go.Scatter(x=test.index[:shift],y=test['Price'],name = 'Test Actual')
        fig3 = go.Scatter(x=test.index[:shift],y=MLP_pred,name = 'Prediction')

        line = {'data': [fig1,fig2,fig3],
             'layout': {
                'xaxis' :{'title': 'Date'},
                'yaxis' :{'title': '$'},
                'title' : 'MLP' + ' - ' + str(frames_string[count]) + ' - ' + indicator_set
             }}
        fig = go.Figure(line)
        fig.show()

        count = count+1

    return MLP_results

# In[148]:

# Importing Training Set
#reading in csvs
df_daily = pd.read_csv('C:/College/Final Year Project/daily_cleaned.csv')
#setting index
df_daily = df_daily.set_index('Date')
# changing index type to DateTimeIndex
df_daily.index=pd.DatetimeIndex(df_daily.index)
# removing first 4 rows because indicator values are 100 or NaN (not accurate)
df_daily = df_daily.iloc[4:]

data = {'Price':df_daily['weighted_price'],
        'Macd':df_daily['macd'],
        'Macd_signal': df_daily['macds'],
        'RSI':df_daily['rsi_14'],
        'Obv':df_daily['Obv']}

df_daily =pd.DataFrame(data,index = df_daily.index)

halving_2012 = pd.to_datetime('2012-11-28')
halving_2016 = pd.to_datetime('2016-07-09')
halving_2020 = pd.to_datetime('2020-05-11')

#spitting into 4 halving cycle dataframe
pre_2012 = df_daily.loc[df_daily.index < halving_2012]
pre_2016 = df_daily.loc[(df_daily.index < halving_2016) & (df_daily.index > halving_2012)]
pre_2020 = df_daily.loc[(df_daily.index < halving_2020) & (df_daily.index > halving_2016)]
pre_2024 = df_daily.loc[(df_daily.index > halving_2020)]

# In[149]:

indicator_set = 'Traditional Indicators'
MLP_results = runMLP(pre_2012,pre_2016,pre_2020,pre_2024,indicator_set)
print(MLP_results)
```

```
# # MLP (Neural Net) On-Chain Indicators

# In[135]:

#repeating for On-Chain indicators
df_daily = pd.read_csv('C:/College/Final Year Project/new_indicators.csv')
df_daily = df_daily.set_index('date')
df_daily.index=pd.DatetimeIndex(df_daily.index)

data = {'MVRV' : df_daily['CapMVRVCur'],
   'NVT' : df_daily['NVTAdj90'],
   'Fee' : df_daily['FeeTotNtv'],
   'Price' : df_daily['PriceUSD']}

df_daily =pd.DataFrame(data,index = df_daily.index)

halving_2012 = pd.to_datetime('2012-11-28')
halving_2016 = pd.to_datetime('2016-07-09')
halving_2020 = pd.to_datetime('2020-05-11')

pre_2012 = df_daily.loc[df_daily.index < halving_2012]
pre_2016 = df_daily.loc[(df_daily.index < halving_2016) & (df_daily.index > halving_2012)]
pre_2020 = df_daily.loc[(df_daily.index < halving_2020) & (df_daily.index > halving_2016)]
pre_2024 = df_daily.loc[(df_daily.index > halving_2020)]

# In[136]:

#calling the same method as Traditional Indicators for completely fair comaprison between the two
indicator_set = 'On-Chain Indicators'
MLP_results = runMLP(pre_2012,pre_2016,pre_2020,pre_2024,indicator_set)
print(MLP_results)

# In[ ]:




# # LSTM (Neural Net) Traditional Indicators

# In[137]:

# cross validation technique which is a variation of k-fold
timesplit= TimeSeriesSplit(n_splits=4)

# In[138]:

#Defining the setup of the LSTM model
def LSTM_method(trainX,X_train1,y_train,X_test1,y_test):


   #LSTM mode: The number of units represnts the dimensionality
   #          The actiavtion function is set to 'relu' (smae as MLPRegressor)
   #          Return sequences set to true for returning the last output in output.
   lstm = Sequential()

   lstm.add(LSTM(units = 64, input_shape=(1, trainX.shape[1]), activation='relu', return_sequences=True))
   lstm.add(LSTM(units = 32, input_shape=(1, trainX.shape[1]), activation='relu', return_sequences=True))
   lstm.add(LSTM(units = 16, input_shape=(1, trainX.shape[1]), activation='relu', return_sequences=False))

   lstm.add(Dense(1))
   # the error is computed using MSE
   # Adam optimization is a stochastic gradient descent method
   lstm.compile(loss='mean_squared_error', optimizer='adam')

   # fitting the model on x_train and y_train over 50 epochs with batch size 5
   # verbose =1 shows the training progress pe epoch, setting to 0 hides the progress
```

```
    lstm.fit(X_train1, y_train, epochs=50, batch_size=5, verbose=1, shuffle=False)

    #predicting the price based on the test indicator data
    y_pred= lstm.predict(X_test1)
    #return a copy of the predicyion flattened into 1 dimension
    y_pred = y_pred.flatten()

    classification = pd.DataFrame(data=y_test,columns =['test'])
    classification['test'] = np.where(classification['test'].shift(-1) > classification['test'], 1, 0)
    classification['pred'] = y_pred
    classification['pred'] = np.where(classification['pred'].shift(-1) > classification['pred'], 1, 0)

    LSTM_acc = accuracy_score(classification['test'],classification['pred'])

    #calculate the MSE between the y_test and prediction from model
    LSTM_MSE = mean_squared_error(y_test, y_pred)
    MAPE = mean_absolute_percentage_error(y_test,y_pred)
    LSTM_r2 = r2_score(y_test,y_pred)

    return y_pred, LSTM_MSE, MAPE, LSTM_r2, LSTM_acc

# In[139]:

def runLSTM(features, pre_2012, pre_2016, pre_2020, pre_2024,indicator_set):

    # creating a list of frames to iterate through
    frames = [pre_2012,pre_2016,pre_2020,pre_2024]
    frames_string = ['2009-2012','2012-2016','2016-2020','2020-2024']
    #creating a dataframe for cycle, mean-squared error and r2 score.
    LSTM_results = pd.DataFrame({'Cycle':[],'MSE_score':[],'R2_score':[]})
    count = 0

    for df in frames:
        # initialise the scaler
        scaler = StandardScaler()
        #scale the indicators
        feature_transform = scaler.fit_transform(df[features])
        #create a dataframe of the scaled data
        feature_transform= pd.DataFrame(columns=features, data=feature_transform, index=df.index)
        # the desired output variable is price
        output_var = pd.DataFrame(df['Price'])

        #timesplit divides the data into folds
        for train_index, test_index in timesplit.split(feature_transform):

            #performing a train test split foe each fold in the timesplit
            X_train, X_test = feature_transform[:len(train_index)], feature_transform[len(train_index): (len(train_index)+len(test
_index))]

            y_train, y_test = output_var[:len(train_index)].values.ravel(), output_var[len(train_index): (len(train_index)+len(test_
index))].values.ravel()
        # transforming to numpy array
        trainX =np.array(X_train)
        testX =np.array(X_test)

        #shaping the data to be accepted by the model
        X_trainfit = trainX.reshape(X_train.shape[0], 1, X_train.shape[1])
        X_testfit = testX.reshape(X_test.shape[0], 1, X_test.shape[1])

        #running the LSTM model with parameters
        LSTM_pred, LSTM_MSE,MAPE, LSTM_r2, LSTM_acc = LSTM_method(trainX,X_trainfit,y_train,X_testfit,y_test)

        #creating a row from model output
        new_row = pd.Series(data={'Cycle':frames_string[count],'MSE_score':LSTM_MSE,'MAPE':MAPE, 'classification':LS
TM_acc})

        #append the results to a dataframe
```

```
        LSTM_results = LSTM_results.append(new_row,ignore_index = True)


        #methods to calaculate the accuraccy, f1 score and ROC curve score
        #print(' Accuracy: { :0.3 }'.format( 100*accuracy_score(y_test, 1 * (LSTM_pred > 0.5))) )

        #print(' f1 score: {:0.3}'.format( 100*f1_score( y_test , 1 * (LSTM_pred > 0.5))))

        #print(' ROC AUC: {:0.3}'.format( roc_auc_score( y_test , LSTM_pred)) )

        #plotting the prediction vs actual
        fig1 = go.Scatter(x=X_train.index,y=output_var['Price'],name = 'Train Actual') # Training actuals
        fig2 = go.Scatter(x=X_test.index,y=y_test,name = 'Test Actual') # Testing actuals
        fig3 = go.Scatter(x=X_test.index,y=LSTM_pred,name = 'Prediction') # Testing predction

        # Combine in an object
        line = {'data': [fig1,fig2,fig3],
             'layout': {
                'xaxis' :{'title': 'Date'},
                'yaxis' :{'title': '$'},
                'title' : 'LSTM' + ' - ' + frames_string[count]+ ' - ' + indicator_set
             }}
        # Send object to a figure
        fig = go.Figure(line)

        # Show figure
        fig.show()

        count = count+1

    return LSTM_results

# In[140]:

# Importing Training Set
df_daily = pd.read_csv('C:/College/Final Year Project/daily_cleaned.csv')
df_daily = df_daily.set_index('Date')
df_daily.index=pd.DatetimeIndex(df_daily.index)
df_daily = df_daily.iloc[4:]

data = {'Price':df_daily['weighted_price'],
        'Macd':df_daily['macd'],
        'Macds':df_daily['macds'],
        'RSI':df_daily['rsi_14'],
        'Obv':df_daily['Obv']}

df_daily =pd.DataFrame(data,index = df_daily.index)

halving_2012 = pd.to_datetime('2012-11-28')
halving_2016 = pd.to_datetime('2016-07-09')
halving_2020 = pd.to_datetime('2020-05-11')

pre_2012 = df_daily.loc[df_daily.index < halving_2012]
pre_2016 = df_daily.loc[(df_daily.index < halving_2016) & (df_daily.index > halving_2012)]
pre_2020 = df_daily.loc[(df_daily.index < halving_2020) & (df_daily.index > halving_2016)]
pre_2024 = df_daily.loc[(df_daily.index > halving_2020)]

# In[141]:

#specifying the indicator names
features = ['Macd','Macds','RSI','Obv']
indicator_set = 'Traditional Indicators'
#passed into the model with halving cycle data
LSTM_results = runLSTM(features, pre_2012,pre_2016,pre_2020,pre_2024, indicator_set)
#output results dataframe
print(LSTM_results)
```

```
# # LSTM (Neural Net) On-Chain Indicators

# In[142]:

#repeating with O-chain indicators
df_daily = pd.read_csv('C:/College/Final Year Project/new_indicators.csv')
df_daily = df_daily.set_index('date')
df_daily.index=pd.DatetimeIndex(df_daily.index)

data = {'MVRV' : df_daily['CapMVRVCur'],
   'NVT' : df_daily['NVTAdj90'],
   'Fee' : df_daily['FeeTotNtv'],
   'Price' : df_daily['PriceUSD']}

df_daily =pd.DataFrame(data,index = df_daily.index)

halving_2012 = pd.to_datetime('2012-11-28')
halving_2016 = pd.to_datetime('2016-07-09')
halving_2020 = pd.to_datetime('2020-05-11')

pre_2012 = df_daily.loc[df_daily.index < halving_2012]
pre_2016 = df_daily.loc[(df_daily.index < halving_2016) & (df_daily.index > halving_2012)]
pre_2020 = df_daily.loc[(df_daily.index < halving_2020) & (df_daily.index > halving_2016)]
pre_2024 = df_daily.loc[(df_daily.index > halving_2020)]

# In[143]:

indicator_set = 'On-Chain Indicators'
features = ['MVRV','NVT','Fee']
LSTM_results = runLSTM(features, pre_2012,pre_2016,pre_2020,pre_2024,indicator_set)
print(LSTM_results)
```