# Prediction of Bitcoin prices
# FYS-STK4155 Project 3

Oskar Våle, Mia Synnøve Frivik, Andrea Myrvang and Max Jan Willem Schuringa
https://github.com/Mia-F/FYS_STK_Project_3
(Dated: December 18, 2023)

In this project we are going to use different techniques in machine learning to try to predict Bitcoin prices. The different methods we have applied in this project are Long Short-Term Memory (LSTM) neural network, decision trees and random forest. We started by first running through different parameters to find the best model, then we used that model to try predicting the price of bitcoins for different time windows. We found that LSTM got better accuracy when predicting bitcoin prices for the last 365 days in our dataset, and it seldom predicted prices that were more than 3000 USD from the actual price. For Decision tree we used a model that had an MSE of 0.00169 when the square error was used as a criterion and the splitter parameter was set to 'best'. This model was not the best at predicting Bitcoin prices, as it usually had a deviation of about 2500 USD from the actual price. Lastly we used random forest to obtain a model with an MSE of 0.0011 when the criterion used was 'absolute_error' and bagging was included. We found that while the model was worse than the LSTM NN, it preformed better than the decision tree with a usual deviation of about 5000 USD from the actual prices.

Figure 1. A DALL-E generated image of students trying to predict bitcoin prices with machine learning right before Christmas. (It should be mentioned here that I explicitly told DALL-E that I wanted four university students in the picture but it consequently drew six people or more : ( )

## I. INTRODUCTION

Metal has served as a currency since the ancient Babylonian era around 2000 BC and has remained a preferred means of payment for various services throughout history [3]. However, with the rise of the internet and increased accessibility to computers, alternative payment methods have been developed over the last few decades. Cryptocurrency, a digital form of currency existing exclusively online, has emerged as one such alternative. It enables users to verify their financial holdings independently, eliminating the need for a third-party intermediary like banks, resulting in anonymous transactions often utilized by criminal networks [8].

Over the years, cryptocurrencies, especially Bitcoin, have gained popularity among the general public, transforming Bitcoin into a lucrative asset. At the moment of writing a single Bitcoin holds a value of 477 056,34 Norwegian kroner [2], yet these values are subject to constant fluctuations. Anticipating these fluctuations in Bitcoin prices has become a crucial aspect for those looking to capitalize on potential gains or losses.

In this project we are going to use a Neural network, decision trees and random forest to see if we can create models that can accurately predict future bitcoin prices, testing them on a dataset of historic prices.
First, in Section 2, we will outline the methods and models employed. Section 3 presents our results, followed by

a discussion of them in Section 4. The report concludes in Section 5.

## II. THEORY AND METHOD

### A. Bitcoin

If you are like some of us in this group and don't now what a Bitcoin is, fear not! Here comes a brief overview of what Bitcoins are, how it is "mined" and how it is used.

Bitcoin is a type of cryptocurrency, meaning it doesn't have a physical form and exists solely virtually on a computer or in a cloud. It was the first cryptocurrency created in 2009 by an antonymous person called Satoshi Nakamoto. Satoshi published a paper called "Bitcoin: A Peer-to-Peer Electronic Cash System" that describes what Bitcoins are and how they are mined. The reason Satoshi created Bitcoins was so that users can send payment directly from one party to another without going through a financial institution like a bank [21].

What Satoshi aimed to achieve was a solution to the double spending problem. Double spending is a potential issue unique to digital currencies, where a single digital coin could be copied and used to make multiple transactions, undermining the currency's reliability and value. Traditional digital payment systems avoid this problem by using a trusted central authority, like a bank, which verifies the balance before authorizing any transaction. Once a transaction is made, the bank updates its records, ensuring that the same money cannot be spent again. Cryptocurrencies, such as Bitcoin, address the double-spending problem without needing a central authority. Instead, they use a decentralized system where every transaction is publicly recorded on a ledger known as the blockchain. The blockchain is maintained by a network of computers, and before a transaction is added to the blockchain, these nodes verify the validity of the transaction. If a user tries to spend the same digital token twice, the network will reject the second transaction because it would conflict with the record of the first transaction. This decentralized verification process is crucial in maintaining the integrity and security of a cryptocurrency. Transactions are grouped together in blocks and added to the blockchain through a process called mining, which involves solving complex mathematical puzzles. This process ensures that once a transaction is recorded, it cannot be altered or deleted, making double spending highly difficult [7].

Over the last couple of years it has become really popular to buy Bitcoins. This have led to Bitcoins becoming very valuable and driving the prices of one Bitcoin upwards. This means that if one can predict if the price of one Bitcoin will increase or decrees one can now when to purchase or sell Bitcoins thereby increase the changes of getting more profit.

### B. Data

The data used are collected from https://finance.yahoo.com/quote/BTC-USD/history?p=BTC-USD&guccounter=1 and shows how the pricing of bitcoins has changed from November 15th 2014 to November 11th 2023. Information included in the dataset is Open, High, Low, Close, Adj Close, Volume, Label and Target. Open represents the Bitcoin price at the start of a trading day, while High and Low indicate the maximum and minimum Bitcoin prices within that day. The Close value refers to the Bitcoin price at the day's end, and 'Adj Close' denotes the adjusted closing prices. Volume measures the total transaction volume in the asset. The Label reflects daily price movement, marked as 0 for a decrease in price and 1 for an increase compared to the previous day . Finally, Target is used to evaluate the performance of the neural network. This is done by taking the next days close (what the nn should have predicted). The following figure shows how the prices of Bitcoin has changes since 15th of November.

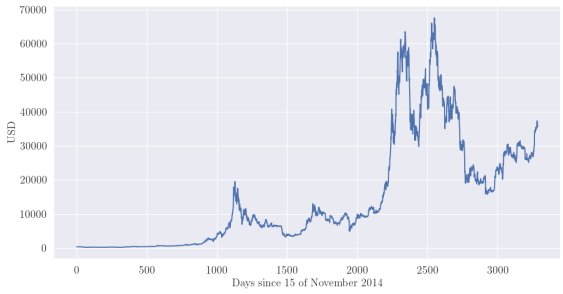


Figure 2. Plot over changes in Bitcoin prices from November 15th 2014 to November 11th 2023

#### 1. *Technical indicators*

To get some additional use-full input features we added technical indicators. These are calculated using the TA-Lib (Technical analysis library). To keep it simple we used a few common ones. The technical indicators used as input features are the Relative Strength Index (RSI), Simple Moving Average (SMA), Average True Range (ATR) and Moving average convergence/divergence (MACD) [4]. The RSI is a momentum oscillator that measures the speed and change of price movements. RSI oscillates between zero and 100. Traditionally, RSI is considered overbought when above 70 and oversold when below 30. The RSI aims to indicate whether a market is considered to be overbought or oversold in relation to recent price levels. It is used to identify potential reversal points in the market. The SMA is one of the

most basic technical indicators. It calculates the average of a selected range of prices, usually closing prices, by the number of periods in that range. The SMA is used to smooth out price data to identify the trend direction over a specified period of time, helping to cut down the amount of "noise" on a price chart. The ATR is a technical analysis indicator that measures market volatility by decomposing the entire range of an asset's price for that period. It provides a snapshot of the degree of interest or disinterest in a move. A high ATR value indicates a high level of volatility, and a low ATR value suggests a weaker trend. The MACD is a trend-following momentum indicator that shows the relationship between two moving averages of an asset's price. The MACD is calculated by subtracting the 26-period Exponential Moving Average (EMA) from the 12-period EMA. The result of that calculation is the MACD line. A nine-day EMA of the MACD called the "signal line," is then plotted on top of the MACD line, which can function as a trigger for buy and sell signals.

### C. Long short-term memory Recurrent neural network

Recurrent Neural Networks (RNN) are a class of neural networks with a distinctive feature: they possess backward-pointing connections, unlike the typical forward-only structure seen in feedforward neural networks [15]. The RNNs excel in processing time series data, such as predicting stock market trends and signaling optimal moments for trading. They are quite versatile, capable of handling variable-length sequences, which is a significant leap from the fixed-size input limitations of other network types[15].

#### 1. *Long short-term memory*

LSTM networks are a type of recurrent neural network (RNN) specialized in handling sequences of data, making them ideal for time-series analysis like Bitcoin price prediction. Unlike standard RNNs, LSTMs can capture long-term dependencies in data sequences, addressing the vanishing gradient problem common in traditional RNNs. They achieve this through their unique architecture, which includes memory cells and gates (input, output, and forget gates).
The LSTM gates are not explicitly defined in the code, as they are inherent components of the LSTM layers used in the Keras library. In Keras, when you add an LSTM layer to a neural network model, the gates are automatically included as part of the LSTM's internal architecture. These gates are responsible for controlling the flow of information within the LSTM cell, managing what information is stored, updated, or forgotten at each time step.

#### 2. *Look-back*

Look-back in the context of neural networks, particularly in time-series forecasting, refers to the number of previous time steps used to predict the future step. In simpler terms, it decides how much historical data the model should consider for making a prediction [6].
For example, in a stock price prediction model, if the look-back is set to 2, the model will use the stock prices from the previous 2 days to predict the price for the next day. The choice of look-back length can significantly impact the performance of the model. A too short look-back may not provide enough context, while a too long look-back might introduce noise and irrelevant information. The optimal look-back length is often determined empirically, based on the specific characteristics of the data.

#### 3. *Network topology*

We are using a Deep LSTM (DLSTM) RNN, by stacking several LSTM layers. A DLSTM RNN consists of an input layer, several LSTM layers and an output layer [13].

#### 4. *LSTM layer*

An LSTM layer consists of one or more memory blocks, each of which contains one or more memory cells that share the same input and output gate [18]. These memory cells contain gates. Below in figure (3) is a visualization of the high-level architecture of an LSTM model, with the middle-section ("hidden") showing the architecture of an LSTM layer [18].

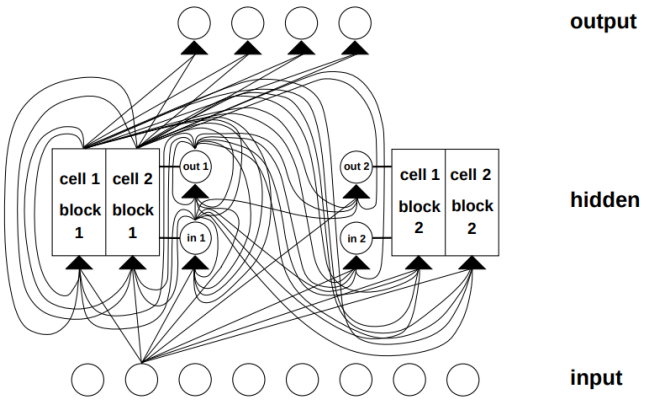


Figure 3. Visualization of LSTM layer architecture, created by the inventors of the LSTM model themselves [18]

We use 4 (hidden) layers, with two of them being LSTM layers and two being "Dense" layers. Dense layers in the Keras library are just regular neural network layers,

called "Dense" because all nodes in the layer are connected to all nodes in the previous layer [19]. See "FYS-STK4155 Project 2" for an explanation of regular neural network layers [11].

#### 5. *Memory cell*

Memory cells contain gates, and keep track of a cell state (long-term memory) and a hidden state (short-term memory). Having a long-term memory is one of the things that differentiates an LSTM from a standard RNN. As mentioned previously, having this long-term memory makes them better at dealing with long-term dependencies than vanilla RNNs, which only have short-term memory. Memory cells in Long Short-Term Memory (LSTM) networks are essential components that enable these networks to handle long-term dependencies effectively in data sequences [10]. Unlike standard Recurrent Neural Networks (RNNs), LSTMs are equipped with a unique structure that includes memory cells containing gates, which are crucial for the network's information flow [10]. The memory cell in an LSTM comprises three types of gates:

- **Forget Gate:** This gate decides what information in the cell state is remembered. It applies a sigmoid function to determine the extent to which each value in the cell state should be remembered, based on the current input and the previous hidden state.
- **Input Gate:** The input gate generates new information for the cell state, based on the hidden state and the input. This newly generated information is added to the cell state.
- **Output Gate:** The output gate determines the next hidden state, considering the current input, the previous hidden state, and the updated cell state.

Together, these gates control the information flow in and out of the memory cell, helping it keep only the relevant information.

#### 6. *Backpropagation in LSTMs*

As explained in project 2, backpropagation is "a method used for efficiently computing gradients of the cost function with respect to the weights and biases in the network. It adjusts the weights and biases in our network according to the errors, aiming to minimizing the error between the predicted output and the actual output"[11]. This explanation was made for backpropagation in regular neural networks (NN), but the quote still applies to our LSTM model. Backpropagation in an LSTM model does have some differences to backpropagation in a NN though. One of the key differences is that with LSTMs, we do backpropagation "through time" [18]. In this process, errors are propagated backwards through each timestep. This is made possible by the memory cells retaining their hidden states and cell states. Explaining the backpropagation of an LSTM in-depth is a complicated endeavor, and is quite frankly beyond the scope of this paper (and FYS-STK4155). We will therefore avoid this topic.

#### 7. *Activation functions*

We only used ReLU and Leaky ReLU in our layers. An explanation of these activation functions can be found in "FYS-STK4155 Project 2" [11].

### D. Decision trees

Decision Trees are a versatile supervised machine learning technique used for both classification and regression tasks. They model decisions and their possible consequences as a tree-like structure. The tree consists of three key components: the root node, internal nodes and the the terminal nodes, commonly referred to as leaf nodes or just leafs. The internal nodes represents a decision on an attribute while the leaf nodes represents the final output or decision made after computing all the attributes. The components are connected through branches, which represent the decision-making pathways. The paths from root to leaf represent classification rules or regression predictions [16].

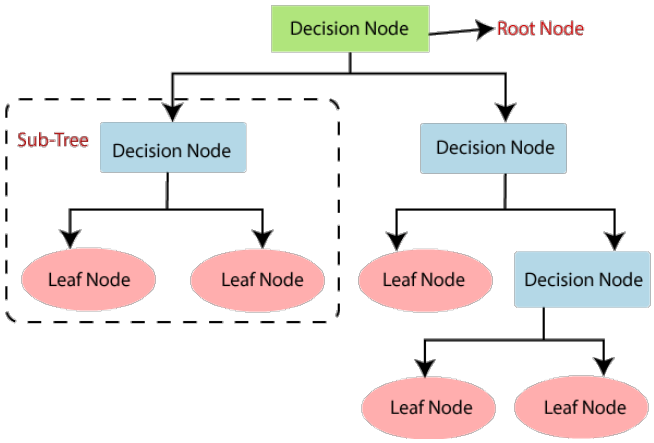


Figure 4. An illustration of the structure of a decision tree. Image obtained from https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm

#### 1. *Regression trees*

Regression Trees, a subset of Decision Trees, are used for predicting continuous numerical outcomes. They involve

two primary steps: Firstly, the predictor space is split into into $J$ distinct, non-overlapping regions. For every observation that falls into a region $R_j$, the prediction made is the mean of the response values for the training observations in $R_j$. The splitting is done to minimize the mean squared error (MSE) within each region [16].
In this approach, the predictor space is segmented into high-dimensional rectangles, or "boxes", to strike a balance between simplicity and the effectiveness of the predictive model. The primary objective is to configure these boxes, identified as $R_1, \ldots, R_J$, in a way that effectively minimizes the Mean Squared Error (MSE), thus enhancing the model's precision and reliability [16].

In this project we have used Scikit learns DecisionTreeRegressor class to create a model of our dataset. In this class there are a lot of options one can choose between to try and obtain the best model possible. We have chosen to look at two of these, criterion and splitter. Lets first look at what these different parameters do, and the possible values one can choose. Firstly, the criterion parameter specifies the function used to measure the quality of a split. It therefore plays an important role in the structure of the tree. The DecisionTreeRegressor class allows you to choose between the following values for criterion: "squared_error" (default), "friedman_mse", "absolute_error" and "poisson" [22]. The splitter parameter can have the values "best" (default) and "random".

#### 2. *squared_error*

The "squared_error" option refers to mean squared error [22]. What this method do is to average the square of the difference between the actual values and the predicted values calculate by the model [14], and is defined by the following equation:

$$\frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y}_i)^2 \tag{1}$$

where $y_i$ is the actual value and $\overline{y}_i$ is the predicted value [16]. This means that the algorithm will choose splits that minimize the variance within each node. It tries to create splits so that the values in each node are as close to each other as possible. This method is particularly useful in highlighting larger errors, as they are exponentially more impactful than smaller ones.

#### 3. *Friedman MSE*

The Friedman MSE is a modification of the traditional mean squared error, introduced by Jerome Friedman. The metric incorporates a statistical adjustment which makes it more suitable for datasets with a complex mix of categorical and continuous variables. The Friedman MSE is given by

$$\frac{K_1 K_2}{K_1 + K_2}(y_i - \overline{y}_i)^2 \tag{2}$$

where $K_1$ and $K_2$ are the number of examples in each subnode [24].

#### 4. *Absolute error*

The Absolute Error measures the absolute difference between the predicted and the actual values. Unlike the squared error, it does not emphasize larger errors, making it useful in scenarios where all errors are equally important. The absolute error for a prediction is calculated as:

$$|y_i - \hat{y}_i| \tag{3}$$

where $y_i$ is the actual value and $\hat{y}_i$ is the predicted value[14].

#### 5. *Poisson*

Poisson Error is commonly used in count data (integers of 0 or greater) where the data follow a Poisson distribution. This metric is especially relevant in cases where the response variable represents counts or frequencies. It measures how well the model predicts the frequency of events happening within a fixed period. The Poisson error can be useful in various fields, such as traffic flow prediction or natural event forecasting [23].

The next parameter we can change in the DecisionTreeRegressor class is the splitter parameter. What this parameter does is determine the strategy used to choose the split at each node in the tree [22]. Essentially, it decides how the algorithm selects the feature and the threshold to split the data. There are two options integrated in the class which are "best" and "random".

#### 6. *"best"*

This is the default value of the splitter parameter. When the splitter is set to "best", the algorithm searches for the best feature and the best threshold to split the data that results in the highest reduction in the impurity based on the chosen criterion. This approach involves checking every possible split of every variable, which can result in large computation times, especially for large datasets with many features. [1]

#### 7. *"random"*

When the splitter parameter is set to "random", the algorithm randomly selects a subset of features and

thresholds to find a split, rather than searching through all possible splits. This method is generally faster, particularly for large datasets, as it reduces the number of calculations needed. This speed increase can come at the cost of reduced accuracy however, since it is not guaranteed to find the optimal split. [1]

The code we have written for this part of the project is heavily inspired by Prashant Banerjee's code "Decision-Tree Classifier Tutorial" [5] and "Stock Market Prediction using Decision Tree" by Rishi Damarla [9].

### E. Random forest

Random Forest is a learning method that combines multiple Decision Trees to create a more accurate and robust model. It is particularly effective in dealing with over fitting, a common problem in decision tree models.
In the Random Forest model, a diverse collection of Decision Trees $\{T_1, T_2, \ldots, T_n\}$ is constructed, each trained on a unique bootstrap sample of the dataset $D$. This bootstrapping process involves randomly selecting data points with replacement from $D$ to form subsets $D_1, D_2, \ldots, D_n$, which are then used to train the corresponding trees $T_k$ [16]. Additionally, each tree in the forest considers a random subset of features when determining splits, further enhancing the diversity and robustness of the model. This strategic randomness ensures that while the bias of the forest remains comparable to that of a single Decision Tree, the variance across the forest is significantly reduced. The final prediction of the Random Forest, in the case of regression, is determined by averaging the predictions of all individual trees, represented mathematically as

$$\hat{y} = \frac{1}{n} \sum_{k=1}^{n} T_k(x), \tag{4}$$

where $\hat{y}$ is the predicted value for input $x$, and $T_k(x)$ denotes the prediction by the $k$-th tree. In classification tasks, the final output is typically the mode of the predictions from all trees.

#### 1. Bootstrap aggregating (Bagging)

Bootstrap Aggregating, commonly known as bagging, is an essential part in the Random Forest algorithm. By creating multiple subsets $\{D_1, D_2, \ldots, D_n\}$ of the original dataset $D$ through bootstrapping. Each Decision Tree within the forest is trained on these distinct subsets, introducing randomness and reducing variance in the model without increasing bias. This method enhances the robustness of the model, particularly in minimising overfitting [17].

### F. Model evaluation methods

Evaluation techniques for regression models are crucial for determining their performance and effectiveness. These methods provide insights into the model's accuracy, its fit to the data, and its overall predictive capabilities [14].

#### 1. Mean Square Error

Mean squared error (MSE) is a common metric used in regression analysis to measure the accuracy of a model. It calculates the square of the difference between the predicted values and the actual values [12]. Its mathematical expression is:

$$MSE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} (y_i - \tilde{y}_i)^2 \tag{5}$$

This formula determines the mean of the squared variances between the actual and predicted values. The smaller the MSE, the more accurate the predictions are compared to the actual outcomes. MSE is always a non-negative figure, as it is derived from the squared differences between the actual and predicted values. A zero MSE implies an ideal prediction that completely aligns with the actual data. A disadvantage with MSE is that it's not scaled. Outliers will thus contribute the same amount as any other point, which can lead to a skewed MSE value [12].

#### 2. Mean Absolute Error

The Mean Absolute Error (MAE) is a fundamental metric in regression analysis for measuring model accuracy. The MAE reflects the expected value of the absolute error loss, also known as the L1-norm loss [14]. It measures the average magnitude of errors between the model's predictions and the actual values, without considering the direction of these errors. The metric is calculated using the formula:

$$MAE(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = \frac{1}{n} \sum_{i=0}^{n-1} |y_i - \tilde{y}_i| \tag{6}$$

Here, $y_i$ and $\tilde{y}_i$ represent the actual and predicted values, respectively [14]. Unlike the Mean Squared Error, which squares the differences and thus gives greater weight to larger errors, MAE treats all errors on the same scale. This makes it particularly valuable in situations where each error, regardless of its size, holds equal importance. Furthermore, MAE's resistance to the influence of outliers makes it a robust choice for evaluating the performance of a regression model.

#### 3. R2-score

The $R^2$-score, a widely recognized metric for error measurement, calculates the coefficient of determination to gauge a model's predictive accuracy for future data points. This score, which typically falls between 0 and 1, can occasionally be negative. A score of 1 signifies an ideal model fit, while a negative score suggests a model's inadequacy. Models that consistently predict the mean of the target variable, disregarding the input features, would attain an $R^2$-score of 0 [12]. The $R^2$-score is mathematically defined as:

$$R^2(\boldsymbol{y}, \tilde{\boldsymbol{y}}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \tilde{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{7}$$

where we have defined the mean value of $\boldsymbol{y}$ as:

$$\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i \tag{8}$$

## III. RESULTS

### A. Neural network

Below, in figure (5), (6) and (7) you can see plots of the daily MSE, MAE and R2 scores of our LSTM model, plotted as a function of the amount of days since the start of our dataset. The R2 score for a given day is calculated by calculating the R2 score based on the past 7 days. There are some days without data at the beginning of each chart, due to the fact that we use a type of time series cross-validation, where the first part of the dataset is only used for training.

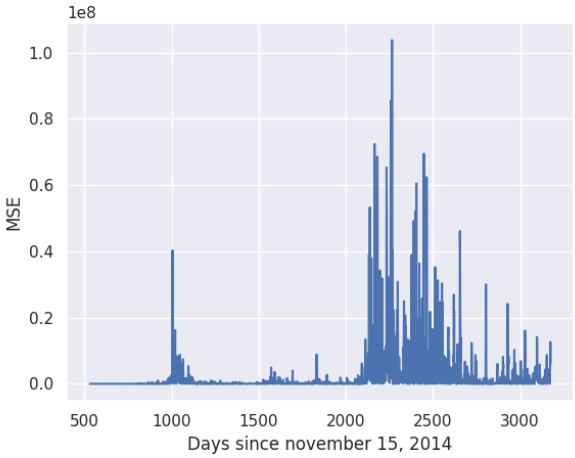


Figure 5. A plot of how the MSE changes as a function of days

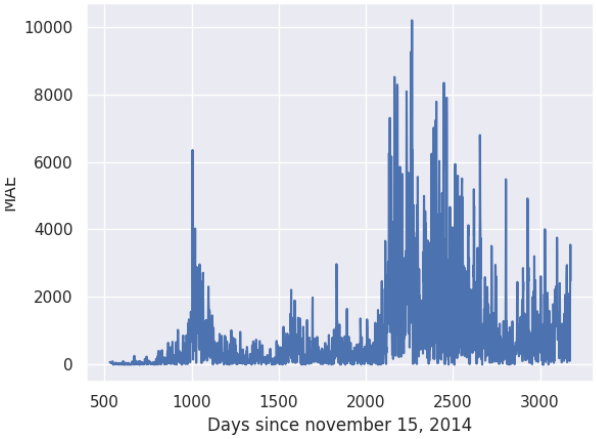


Figure 6. A plot of how the MAE changes as a function of days

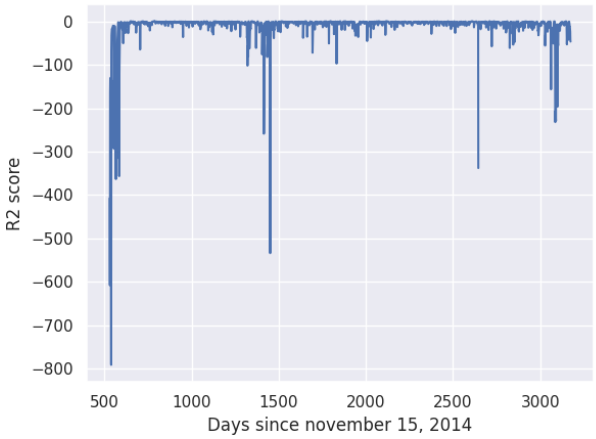


Figure 7. Daily R2 scores as a function of days. R2 is calculated using a rolling window approach, with window size = 7

Below, in figures (9) and (8) you can see a plot of the actual bitcoin prices and the predicted prices. The plot also has green crosses where the two lines cross, pointing out exact predictions. Figure (9) shows the predictions and actuals for the last 365 days, while figure (8) shows it for the last 100 days.

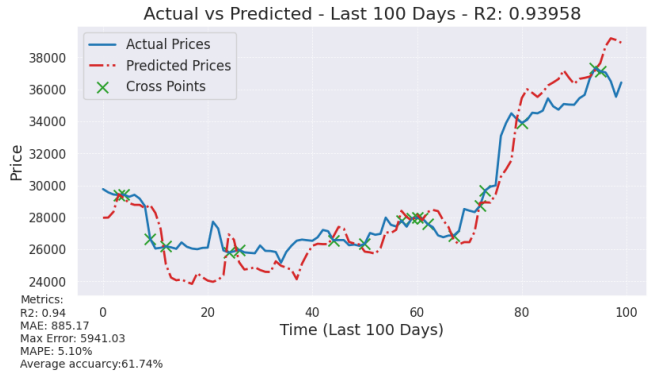


Figure 8. A plot showing the prediction vs the actual Bitcoin prices for the 100 last days of the dataset.

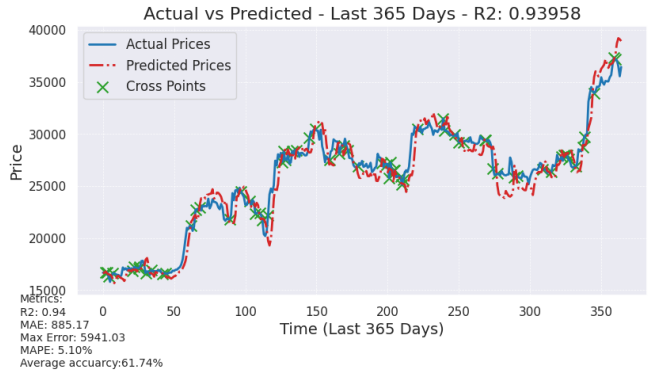


Figure 9. A plot showing the prediction vs the actual Bitcoin prices for the last year of the dataset.

We have also calculated the difference in the actual closing prices compared to the predicted prices as shown in figure (10) and (11).

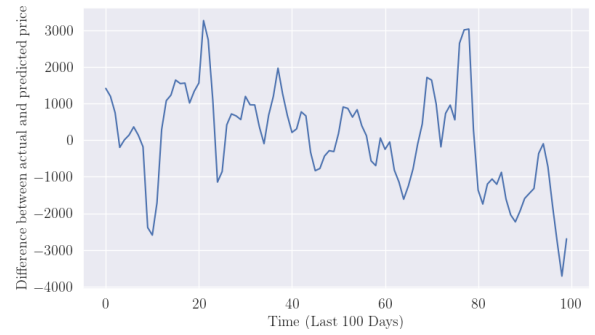


Figure 10. Shows the difference between the actual closing prices and the predicted ones for the last 100 days. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

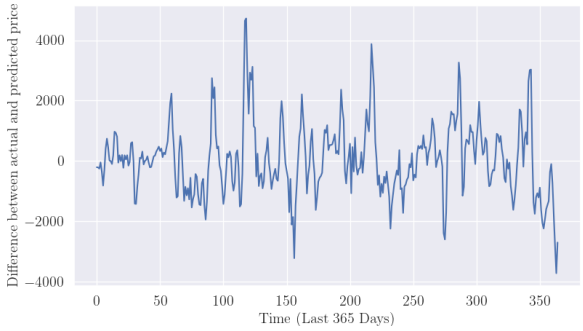


Figure 11. Shows the difference between the actual closing prices and the predicted ones for the last 365 days. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

### B. Decision tree

For the Decision trees we can start by looking at how the MSE changes as a function of the maximum depth of the decision tree. In this project we have used three different ways of weighting the splitting of our trees, namely mean squared error, Friedman mse and absolute error. We can start by looking at our result for how the MSE changes as a function of depth for these three criterions, including if the nodes are split either with the best or random choice. Figure (12) shows the result for these calculations.

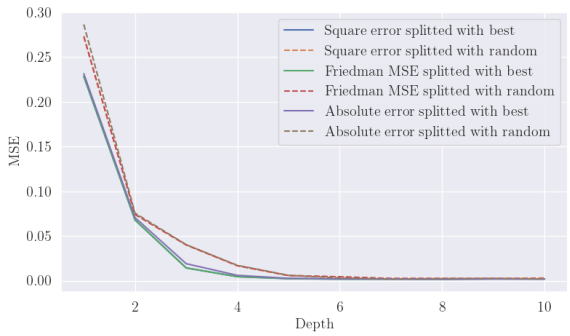


Figure 12. The best achieved MSE values for the three different methods, mean square error, Friedman MSE and absolute error as a function of depth. Including if the nodes are split using the best or random choice in the Scikit library

Next we can look at how the R2 score changed as a function of depth for these six different models, the results are shown in figure (13).

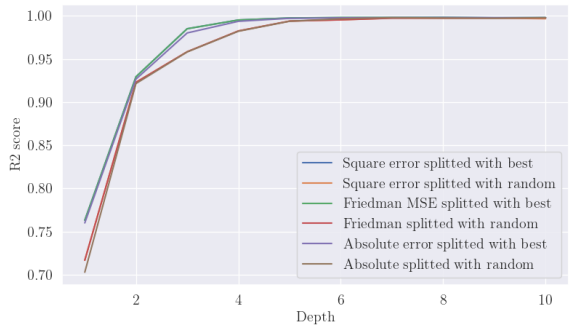


Figure 13. A plot of the R2 score as a function of depth for the six different models obtained from decision trees.

To be able to clearly see which of these methods are the best, we have chosen to plot the lowest MSE for each method in a bar plot shown in figure (14) for the nodes split with the best option, and figure (16) for the nodes split randomly.

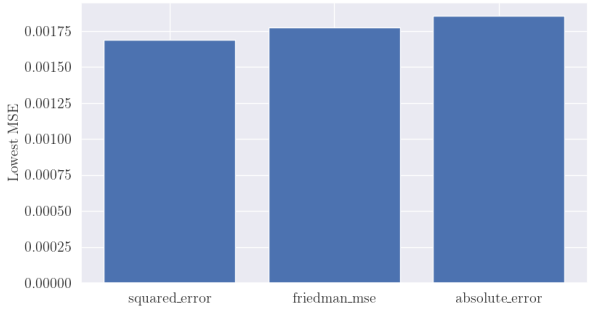


Figure 14. A bar plot of the lowest achieved MSE values for the three criterion's, where the nodes are split with the best option

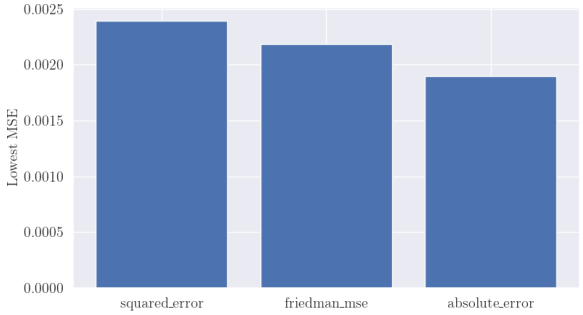


Figure 15. A bar plot of the lowest achieved MSE values for the three criterion's, where the nodes are split with the random option

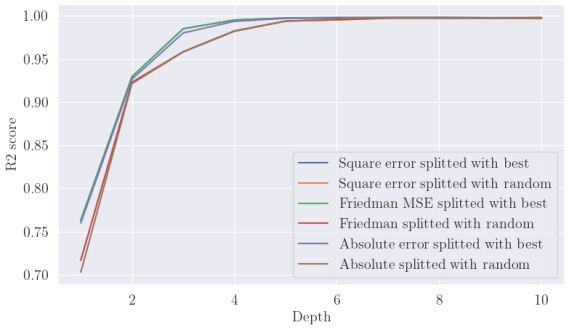


Figure 16. A bar plot of the lowest achieved MSE values for the three criterion's, where the nodes are split with the random option

To show the magnitude of the decision tree which gave the lowest MSE we have chosen to visualize the tree. Due to the size of it, it was impossible to get python to create a readable chart. But it is still interesting to look at. The Decision tree is shown in figure (17).

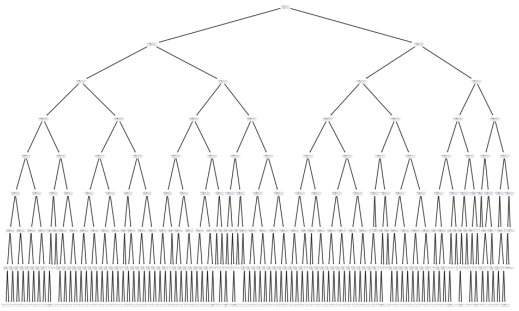

Figure 17. A visualization of the Decision tree that gave the lowest MSE value

Now that we have found the best model we can use this to try and predict some Bitcoin closing prices. Start with predicting the closing price of the last day in our dataset.

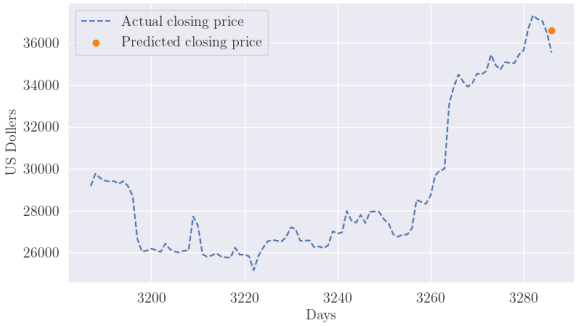


Figure 18. A plot of the predicted Bitcoin price done with the decision tree for the last day of the dataset shown in orange.

If we take the difference between the actual closing price and the predicted price we find that our model predicted the price to be about 1070 dollars higher than the actual price, as shown in figure (19).

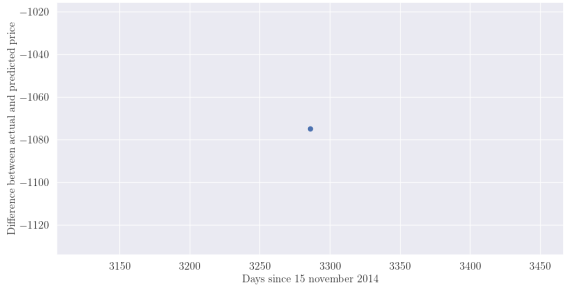


Figure 19. Shows the difference between the actual closing price and the predicted one. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

Next we can try and predict for the 10 last day in our dataset. Figure (20) shows how or model performed, compared to the actual closing prices.


Figure 20. A plot of the predicted Bitcoin price for the last 10 days of the dataset shown in orange

Figure (21) shows the difference between the actual prices and the predicted ones, for the 10 last days in the dataset.

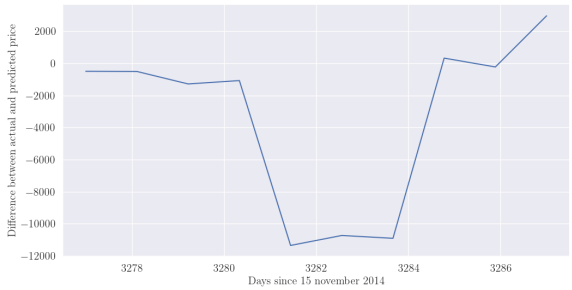


Figure 21. Shows the difference between the actual closing price and the predicted one for the 10 last days in the dataset. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

Next we can see how our model performs when trying to predict the last 100 days in our dataset. Figure (22) shows how it performs compared to the actual prices, and figure (23) shows the difference between the actual prices and the predicted ones.


Figure 22. A plot of the predicted Bitcoin price for the last 100 days of the dataset shown in orange

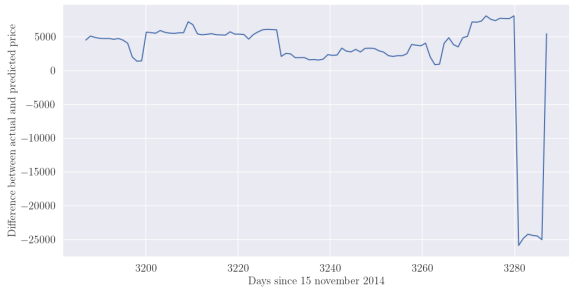


Figure 23. Shows the difference between the actual closing price and the predicted ones for the 100 last days in the dataset. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

Lastly we can see how it performs when predicting the Bitcoin prices for a whole year, so the 365 last days of the dataset. Figure (24) shows how it performs compared to the actual prices, and figure (25) shows the difference between the actual prices and the predicted ones.

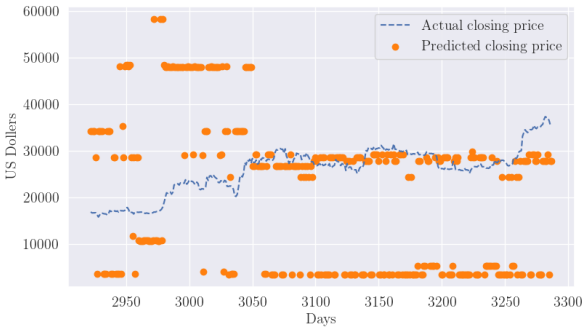


Figure 24. A plot of the predicted Bitcoin price for the last 100 days of the dataset shown in orange

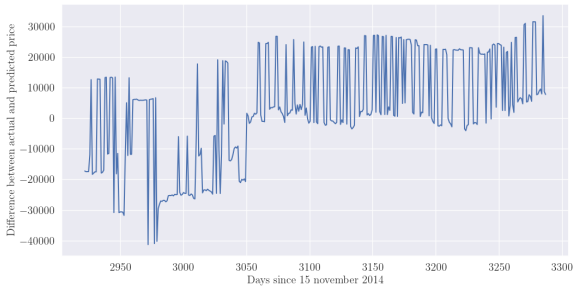


Figure 25. Shows the difference between the actual closing price and the predicted ones for the 100 last days in the dataset. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

### C. Random forest

The last thing we did in this project was to use scikit learns random forest regressor to try and predict bitcoin prices. We started in the same fashion as with decision tress and plotted how the MSE changed as a function of depth for the three criterions square error, Friedman MSE and absolute error. We also looked at how good our models were with and without bootstrap. The result for this is shown in figure (26).

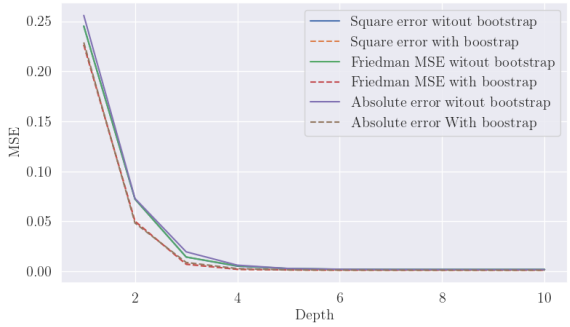


Figure 26. A plot showing how the MSE changes as a function of depth for the three different criterions with and without bootstrap

We did the same for the R2 score, and the result for this is shown in figure (27).

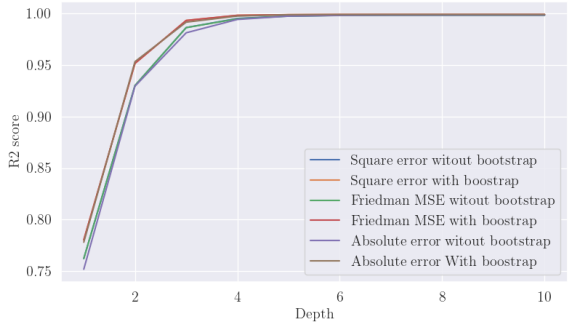


Figure 27. A plot showing how the R2 score changes as a function of depth for the three different criterions with and without bootstrap

To see more clearly the difference in the lowest MSE values achived, we have also here made a bare plot of the best result for each model, shown in figure (28) and (29).

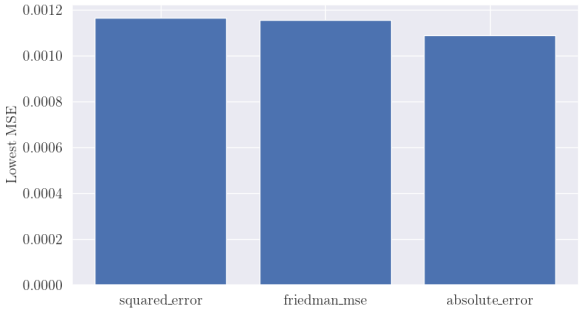


Figure 28. A bar plot of the lowest achieved MSE values for the three criterions, without bootstrap

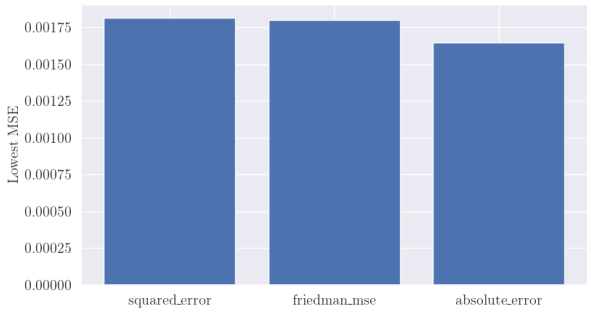


Figure 29. A bar plot of the lowest achieved MSE values for the three criterions, with bootstrap

Now that we have established a good model for our data set, we can see how well it predicts Bitcoin prices. We start with predicting the closing price for the last day in our dataset. The prediction compared to the actual price is shown in figure (30).

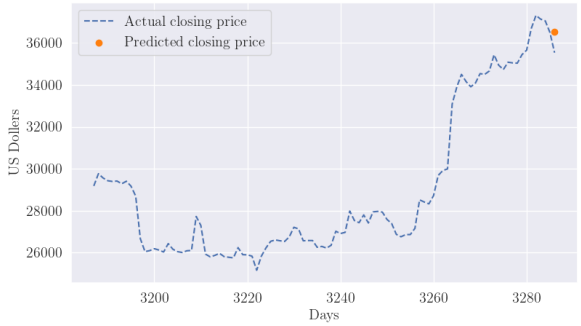


Figure 30. Prediction of the closing price for the last day in the dataset

If we take the difference between the actual closing price and the predicted price we find that our model predicted the price to be about 950 dollars higher than the actual price, as shown in figure (31).

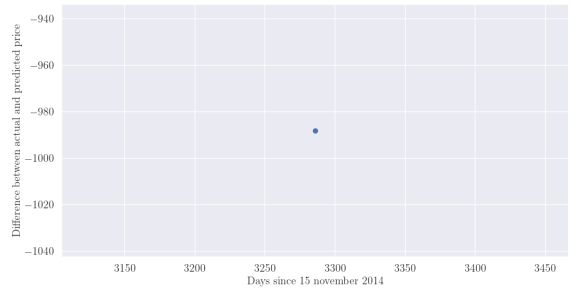


Figure 31. Shows the difference between the actual closing price and the predicted one. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

We can now look at how well our model predict the price for the last 10 days in our dataset. Figure(32) shows how our predicted prices compere to the actual ones.


Figure 32. Prediction of the closing prices for the last 10 days in the dataset

Figure (33) shows the difference between actual prices and our models predicted ones.

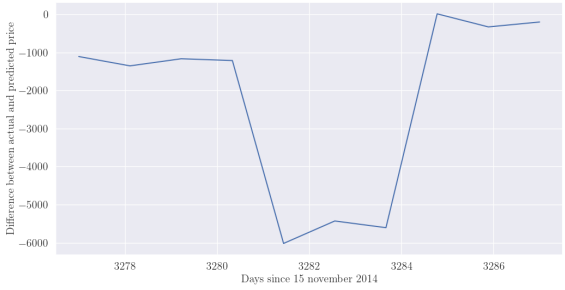


Figure 33. Shows the difference between the actual closing prices and the predicted ones. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

Next we tested how well our model performed when predicting the price for the 100 last days in our dataset. Figure (34) shows the predicted closing prices compared to the actual prices.


Figure 34. Prediction of the closing prices for the last 100 days in the dataset

Figure (35) shows the difference between the actual prices and predicted prices for the last 100 days of the dataset.

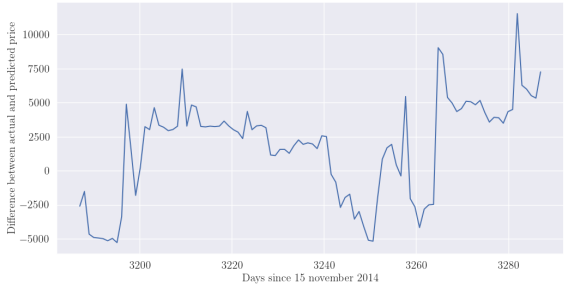


Figure 35. Shows the difference between the actual closing prices and the predicted ones. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

Lastly we looked at how well our model performed on the last 365 days of the dataset. Figure (36) shows the predicted prices compared to the actual closing prices.

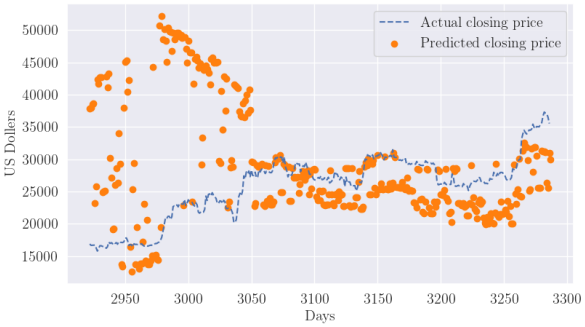


Figure 36. Prediction of the closing prices for the last 365 days in the dataset

The last figure, figure (37), shows the difference between the actual prices and predicted prices for the last 100 days of the dataset.

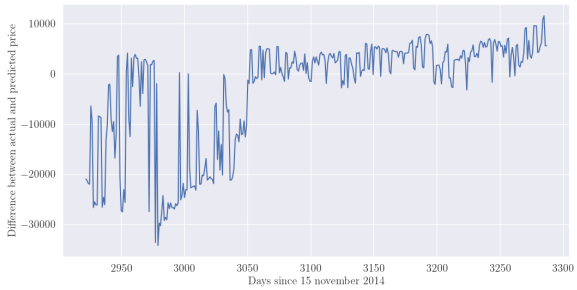


Figure 37. Shows the difference between the actual closing prices and the predicted ones. Negative numbers means a too high prediction, and positive numbers means a too low prediction.

## IV. DISCUSSION

### A. Neural network

Before we dive into the results of our neural network model, it's crucial to evaluate the relevance of MSE, MAE, and the R2-score as evaluation metrics. MSE emphasizes the mean of squared differences between our model's predictions and actual values, and is particularly sensitive to outliers. Given the unstable nature of the Bitcoin market where outliers are common, this sensitivity might result in a skewed perception of the model's efficacy. On the other hand, MAE calculates the average magnitude of errors, providing a more even-handed view. However, it tends to underplay the impact of substantial errors. Lastly, the R2-score, often used to assess model accuracy, may have limited utility in the context of Bitcoin's erratic price movements. This is because it fails to consider the inherent unpredictability of the market. With these considerations in mind, we can now start to discuss our neural network's results.

By examining figures (9) and (8), one can see that the neural network's predictions are consistently slightly "to the right" of the actual prices. The network captures the trends well, but it captures them a bit late. A possible cause of this could be that the model is learning to react to changes in prices, instead of anticipating them. This could happen if the model is primarily learning from the closest price values, instead of capturing the underlying patterns and factors that drive price changes. That being said, relying on recent prices is arguably essential here, due to the volatility and sudden changes that bitcoin prices often have. A possible way to mitigate this could be parameter tuning, for example taking a look at the lookback parameter or the amount of features. One could also consider adding a Kalman filter, shifting the predictions a tiny bit to the left (from a spatial perspective). In the article that is credited with creating the LSTM method, however, the authors mention the fol-

lowing in the "previous work" section:

> "Puskorius and Feldkamp (1994) use Kalman filter techniques to improve recurrent net performance. Since they use 'a derivative discount factor imposed to decay exponentially the effects of past dynamic derivatives,' there is no reason to believe that their Kalman Filter Trained Recurrent Networks will be useful for very long minimal time lags"[18].

The authors here suggest that using a Kalman Filter might not be optimal in cases where the network needs to account for patterns in the distant past. In our case, however, considering how little correlation there is between historic prices and current prices of bitcoin (see figure (2)), Kalman Filters potentially not being useful for "very long minimal time lags" is arguably not a big concern, making it a viable option for improving our model.

Examining the difference between the actual and predicted prices, we find that for the last 100 days' price forecasts, as depicted in Figure (10), the model typically deviated from the actual price by no more than 1000 USD in either direction. On the days when the model's accuracy diminished, the overestimations or underestimations did not exceed 3000 USD. Looking at the predictions for the entire past year, Figure (11) indicates a similar consistency, with deviations generally not surpassing 2000 USD. On the days where the model was less precise, the maximum overestimation reached around 4000 USD, while the most significant underestimation was approximately 5000 USD. Considering how much bitcoin prices can deviate, these are arguably very good results.

Looking at the MSE and MAE of the LSTM predictions over time in figures (5) and (6), we see that errors generally increase and decrease along with the price of bitcoin displayed in (2). This makes sense, as a larger price means potentially larger errors in absolute terms. With a high price of bitcoin, even a small percentage error can mean a large error value. It is also hard for the LSTM to adjust to these sudden changes in price, as it makes predictions looking at historic prices. Because of this, it makes sense that the LSTM sometimes struggles to capture market trends that significantly deviate from previously seen patterns.

We are not the first to apply an LSTM model to forecasting cryptocurrency/stock prices. Comparing our results to those of other researchers on comparable datasets could help us further understand how good our model is, and how applicable LSTM is to short-term bitcoin price prediction. In their 2023 study 'Comparative Performance of LSTM and ARIMA for the Short-Term Prediction of Bitcoin Prices' published in the AABFJ journal, Naveem Latif and colleagues investigate the performance of the LSTM model (and ARIMA) on short-term bitcoin price prediction [20]. This is exactly what we are doing with our LSTM, and these results are therefore relevant to us. Figure (38) below shows a graph from this study, comparing the actual bitcoin prices to the prices predicted by their LSTM model [20].

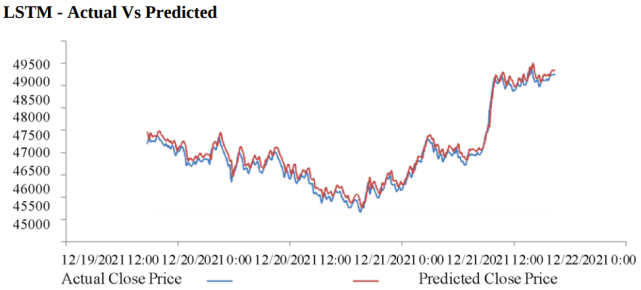


**FIGURE 16:** Actual vs. Predicted Prices of LSTM

Figure 38.

As one can see, their model captures the trend of the prices very well, with the predictions being a little bit "to the right" of the actual prices, just like our predictions. Their model is superior to ours though, achieving an impressive 126.97 MAE, while ours achieved a 885.17 MAE (see figure (9)) [20]. Nonetheless, their results are consistent with ours in that they manage to capture the trends of the prices over time. It should also be noted that we are not using the same training set as these authors, and are undoubtedly using different parameters for our LSTM model [20]. We also used different testing sets. Nonetheless both of our results show the ability to capture the general trend of the prices, which in itsself builds an argument for LSTMs being an appropriate model for cryptocurrency price prediction.

Another thing we could have given some more thought is the dataset we used. Latif et al. trained their model only on data from the end of 2020 until the end of 2021 [20]. We, on the other hand, use data from 2014 until november 2023. As one can see from the price graph (2), prices in 2014 were nothing like what they are now. This low magnitude that the prices used to have affects the cell state/long-term memory of our memory cells. This has probably lead to our model predicting lower than it otherwise would if our training dataset started in 2020 for example. Looking back at figures (8) and (9) with this in mind, one can actually see that the predictions are indeed often lower than the actual prices, perhaps for this very reason among other things.

### B. Decision tree

Before we start to discuss the results for decision tree and random forest, it is impotent to acknowledge that certain aspects of randomness inherent in the scikit-learn pack-

age may lead to slight variations in results upon each execution of the code. However, these differences are typically minor, ensuring that the overall conclusions remains the same. But we were careful to use the results from the same run throughout this report. One other thing to note is that in this project we have used scikit learns tree package to make the Decision tree model. In this package there are two main category's of decision trees, one using regression and one classifier. Since this project aims to predict bitcoin prices the desired output would be the price of one Bitcoin at closing hours. Therefor the regression tree is more favorable in this case. We have also in this part decided to use a standard scaler to scale the data so that features with higher magnitudes cannot dominate the model's learning, as this could lead to biased predictions.

We can start by discussing the MSE values obtained from the different criterion used in the decision tree. We see from figure (12) for all the six different models the MSE starts high and decreases with an increase in the max depth of the decision tree. This is as expected, when the max depth increases our decision tree can split the data in to more nodes creating a better fit to the training data, but also the test data. What we do expect is that when the depth increases we will come to a point were the MSE should start to increase due to overfitting to the training data, but that does not seem to happen in our case. Its difficult to say why this does not happen in our case. Since we do not do any pruning on our trees we shod expect the MSE to increase but maybe our training data is a good representation of the test data so that the increase in MSE is not notable. From figure (14) and (16) we see that the lowest achieved MSE was 0.00169, and was achieved using square error to calculate the quality of a split, and the best parameter to find the optimal feature and threshold to split the data. This is not surprising since the square error is designed to minimize the mean square error, which is the metric we are evaluating. It is also not surprising that we get the best result when splitting the data with the best option, since the algorithm will searches for the best feature and the best threshold in the whole training set to get the model with the smallest MSE. Regarding the other two criterions, we observe that both achieve notably low MSE values when utilizing either the best or random options for the splitter parameter. Notably, the lowest MSE values are consistently obtained when the best option is selected as the splitter. This trend underscores the effectiveness of the best splitter in optimizing the decision tree's performance across the different criterions. The reason the random splitter consequently returns worse result may be due to the random splitter, selecting a subset of splits and features randomly, which means that it might not always find the most optimal split that would minimize the MSE as effectively as the best option. But again, for very large datasets this option might decrease the computation time.

Next we can look at the variation of the R2 score as a function of depth across the six different models, as depicted in figure 13. Initially, with only a few layers in the decision tree, the R2 scores are closer to zero but as we add a few more layers the scores gets closer and closer to one, indicating a significant improvement in model performance. Typically, we would anticipate a point where the model begins to overfit the training data, marked by a subsequent decline in the R2 score. Interestingly, this expected decrease in the R2 score is not observed in our models. One possible explanation to this may be again that the training data representing the test data well, but this may not be the case.

Now that we have established a good model for our dataset we can use it to try and predict some Bitcoin prices! We can start by looking at the models preformed when trying to predict the closing price for the last day in the dataset. If we look at figure (18) and (19) we see that our model overestimates the price, and got a result that is about 1075 USD more than the actual closing price. When we look at final 100 days of the dataset, it becomes evident that there has been a downward trend in price during the last two days in the dataset. While our model has successfully grasped the necessity for a lower price compared to the previous day, it has underestimate the extent of this decrease. Overall this model is okay at forecasting the price for a single day.

We can further look at how our model is at forecasting the closing prices for the dataset's last 10 days. Referring to Figures (20) and (21), it's apparent that the model's predictions for the initial four days are relatively accurate, with a slight overestimation not exceeding 1500 USD for these days. However, for the following two days, the model's predictions deviate significantly, overestimating the price by approximately 11000 USD, which indicates a less reliable forecast. In contrast, the predictions for the final three days show improved accuracy. The model's forecasts for days 8 and 9 are nearly perfect, but it underestimates the price on the last day by around 3000 USD. Overall, the model performs well in predicting whether the price will rise or fall, but it is less accurate in determining the extent of the price increase or decrease.

Next we test our model on how well it preformed when predicting the price for a 100 days. From figures (22) and (23) we see that the first 80 days of the 100-day period, we notice that the actual prices were relatively stable. Similarly, our model forecasts stable prices throughout this initial 80-day span. However, it consistently underestimates the price during this period. The first 45 days the prediction was almost always about 5000 USD too low, but the following 45 days the model's prediction become more accurate, where the prediction was about 2500 USD too low. For the last 20

days the predictions become increasingly less accurate. And we see that for our last 10 days, the prediction was noticeably worse than when we only predicted the last 10 days and the last 1 day.

Next, we evaluated our model's performance over a 100-day prediction period. Analysis of figures (22) and (23) reveals that for the first 80 days of this period, the actual prices maintained a relative stable price. The model mirrored this trend, projecting steady prices during these initial 80 days, yet it consistently underestimated the actual prices. In the first 45 days, the model's predictions were typically around 5000 USD lower than the actual prices. However, its accuracy improved in the next 35 days, with a reduced average error margin of approximately 2500 USD. In contrast, the model's accuracy declined in the final 20 days, becoming increasingly imprecise. Particularly for the last 10 days, the predictive performance was noticeably poorer compared to its accuracy when we predicted the closing price for a time period of 1 and 10 days. For the last day the error was about 20000 USD which is worse than the 1 day prediction which gave an error of about 1075 USD, but it was a bit better at predicting the last day compared to the 10 day prediction which had an error of 3000 USD.

Finally, we assessed the model's performance in forecasting Bitcoin prices over an extended duration of one year (the last 365 days in the dataset). The outcomes of this analysis are depicted in Figure (24) and (25). From these results, it's observable that the model's predicted prices fluctuate, alternating between overestimations and underestimations. In the initial 150 days of our projection period, the model's forecasts were significantly off from the actual closing prices. However, in the subsequent 215 days, the model's accuracy improved, with alternate predictions aligning closely with the real prices, while the remaining estimates were typically around 2000 to 3000 USD lower than the actual price.

Our observations indicate that our model is more effective at predicting prices over shorter time frames. As the prediction window extends, the model's performance tends to approximate guessing, leading to lower accuracy. Another noticeable trend is the model have a tendency to underestimate prices, with overestimations being relatively rare. One potential approach to enhance the accuracy of predictions might involve sequential daily forecasts, using previous predictions as training data, similar to our approach with the neural network. Regrettably, due to time constraints, we were unable to implement this method in our current analysis.

Some might notice that there is actually four different criterion one can choose from when using scikit learns Decision tree, but in our project we have only used three of them. The reason for why we decided to exclude Poisson was that when we scaled the data some of our values become negative. From the theory were we mentioned Poisson, this method only works if the data consist of integers of 0 or higher, and that is not the case for our scaled dataset.

### C. Random forest

The last thing we implemented in this project was random forest. The thought behind this is that a random forest shod give better results than our model created with decision tree since the random forest will use multiple decision trees to create a more accurate model.

We initiated our analysis of the Random Forest model in a similar manner to the Decision Tree, by iterating over various depths to observe changes in MSE. However, instead of the splitter parameter relevant to Decision Trees, we had the option to either employ bagging in the Random Forest or not, essentially deciding whether to use bootstrapping. Bootstrapping, which involves creating multiple samples of the dataset through resampling with replacement, ensures that each tree in the forest gets a slightly different view of the data. This diversity among the trees leads to a more robust overall model that generalizes better to new, unseen data. As indicated in Figure (26), our hypothesis proved accurate. However, the distinctions in MSE values were not immediately apparent from the figure alone. Thus, we chose to illustrate the lowest MSE values for each of the six methods using bar plots, as shown in Figures (28) and (29). These visualizations clearly demonstrate that using bootstrap yields lower MSE values. Specifically, the best MSE value without bootstrap was approximately 0.00163, compared to 0.0011 when employing bootstrap. Notably, the lowest error rates, both with and without bootstrap, were obtained using the 'absolute error' criterion, which is different from what we found for decision tree. We have also plotted the R2 score as a function of depth, here we see exactly the same as for our models created with decision tree.

We have found that the best model achieved with random forest was with the use of bagging and a criterion of absolute error. We can now use this to predict Bitcoin prices and see if these predictions are better than that of decision tree. We started by predicting the price for the last day of our dataset, the results are shown in figure (30) and (31). We can here see that our model predicted a lower decrease in value than what actually happened, meaning that it overestimated the price with about 990 USD. While this prediction outperforms the one made by our Decision Tree model, it doesn't quite match the accuracy of our neural network.

Next we can look at how our model preforms if we

predict the closing prices for the 10 last days in our dataset, the result from these calculations are shown in figure (32) and (33). Here we see a similar result to that of our model created from decision tree, the model is good at predicting if the price will increase or decrease, but not the best to estimate how much it will increase or decrease with. We see that for the first 4 days and the last 3 the difference between the actual and predicted price is not that high, or model for the 4 first day our model overestimates the price a little but for the 3 last days the predicted prices is almost perfect. We see that for day 5, 6 and 7 the accuracy fell a lot and the overestimation was about 5500 USD, but this was still better than what was obtained with decision tree.

We then applied our model to forecast the closing prices for the last 100 days in the dataset. Observations from Figure (34) and (35) suggest that the model remains effective in predicting whether prices would rise or fall. However, it significantly struggles to accurately pinpoint the exact prices. There were only three instances where the predictions closely matched the actual prices. Notably, the disparity between the actual and predicted prices was smaller for the Random Forest model, particularly regarding price overestimations. The largest overestimation by this model was around 5000 USD, whereas the Decision Tree model's most significant overestimation reached about 25000 USD. In terms of underestimations, the Decision Tree model performed slightly better, with a maximum underestimation of approximately 7500 USD, compared to the Random Forest model's largest underestimation of around 11250 USD. Although the Decision Tree model showed greater accuracy in some instances, it also exhibited larger deviations in outlier predictions compared to the Random Forest model.

Lastly we predicted the closing price of bitcoin for the last year in our dataset. From figure (36) and (37) we see that this model also overestimate a lot of the prices the first 150 days, but this model does not fluctuate between overestimation and underestimation like the model from decision tree did. We also see that the last 215 days the prices started to oscillate between overestimations and underestimations but in this case the overestimations were extremely close to the actual prices and the underestimated prices were for the most part under 10000 USD which is a much better result than that of the the decision tree model.

## V. CONCLUSION

This project thoroughly explored the forecasting of Bitcoin prices using machine learning methods like long short-term memory neural networks, decision trees and random forests. Our findings indicate that while these models show promise, there are challenges in achieving high accuracy. This may be due to all the factors that contribute to increasing or decreasing the prices of Bitcoins. We may have gotten a more accurate result if we had included a few other features like the ones found at blockchain.com https://www.blockchain.com/explorer/charts/total-bitcoins, where one can find feature like how the blockchain size has changed over time and the average transaction per block, just to mention some. But it would have taken a lot of time and effort to get different datasets to mach with each other and that we unfortunately didn't have time for in this project.

In this project, we discovered that a Long Short-Term Memory (LSTM) recurrent neural network was exceptionally effective at predicting Bitcoin prices, even with our limited dataset. When forecasting over a year, the LSTM model rarely deviated more than 3000 USD from the actual price. This accuracy contrasts sharply with the Decision Tree, where predictions typically were about 25000 USD off the mark, and the Random Forest, which performed best within a 5000 range. Clearly, the LSTM emerged as the superior model for these price predictions.

This higher accuracy could be attributed to the differences in how the Decision Tree and Random Forest models operate, compared to the LSTM. Both the Decision Tree and Random Forest models were trained on the entire dataset minus the days to be predicted, and they made predictions for the entire time window simultaneously. In contrast, the LSTM also factored in previously predicted prices during its training.

For a fair comparison, we should have adopted a similar approach for the Decision Tree and Random Forest as we did for the LSTM. Unfortunately, due to time constraints, we were unable to implement this methodology.

To conclude this project we found that the LSTM neural network was the best at prediction Bitcoin prices. To get the full potential of the use of machine learning in finance, a better dataset and a Kalman Filter could be used to perhaps obtain a more accurate prediction.

## VI. ACKNOWLEDGEMENT

We would like to thank Morten Hjorth-Jensen for his amazing and well crafted jupyter notebooks. Additionally, we would like to acknowledge the valuable inspiration provided by ChatGPT throughout the report-writing process.

# REFERENCES


[1] What does "splitter" attribute in sklearn's decisiontreeclassifier do? https://stackoverflow.com/questions/46756606/what-does-splitter-attribute-in-sklearns-decisiontreeclassifier-do, 2019. Accessed: 2023-12-14.

[2] 1 btc to nok - convert bitcoins to norwegian kroner. https://www.xe.com, 2023. Accessed: 2023-12-14.

[3] When were coins first used as money? https://www.britannica.com/question/When-were-coins-first-used-as-money, 2023. Accessed: 2023-12-14.

[4] Manish Agrawal, Asif Ullah Khan, and Piyush Kumar Shukla. Stock price prediction using technical indicators: A predictive model using optimal deep learning. *International Journal of Recent Technology and Engineering (IJRTE)*, 8(2):2297, 2019.

[5] Prashant Banerjee. Decision-tree classifier tutorial. https://www.kaggle.com/code/prashant111/decision-tree-classifier-tutorial, 2020. Accessed: 2023-12-13.

[6] Jason Brownlee. Lstm for time series prediction in pytorch. https://machinelearningmastery.com/lstm-for-time-series-prediction-in-pytorch/, 2023. Accessed: 2023-12-16.

[7] Corporate Finance Institute. Bitcoin mining - overview, basics, why, 2023. Accessed: 2023-12-14.

[8] Council on Foreign Relations. Cryptocurrencies: Digital dollars and the future of money. https://www.cfr.org/backgrounder/cryptocurrencies-digital-dollars-and-future-money, 2023. Accessed: 2023-12-14.

[9] Rishi Damarla. Stock market prediction using decision tree. https://www.kaggle.com/code/rishidamarla/stock-market-prediction-using-decision-tree, 2021. Accessed: 2023-12-13.

[10] Databasecamp. Long short-term memory (lstm) networks. https://databasecamp.de/en/ml/lstms, 2023. Accessed: 2023-12-18.

[11] Mia Synnøve Frivik, Andrea Myrvang, Max Jan Willem Schuringa, Felix Hansen Haugland, and Oskar Våle. FYS-STK4155 Project 2, 2023. Accessed: 2023-12-18.

[12] Mia Synnøve Frivik, Andrea Myrvang, Max Jan Willem Schuringa, and Janita Ovidie Sandtrøen Willumsen. Project 1 FYS-STK4155. GitHub Repository : https://github.com/Mia-F/FYS_STK_Project_1.git, 2023. Accessed: 2023-12-14.

[13] Andrew Senior Hasim Sak and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. https://static.googleusercontent.com/media/research.google.com/no//pubs/archive/43905.pdf, 2014. Accessed: 2023-12-17.

[14] Morten Hjorth-Jensen. Week 34: Introduction to the course, logistics and practicalities. https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week34.html, 2023. Accessed: 2023-12-12.

[15] Morten Hjorth-Jensen. Week 45, Recurrent Neural Networks. https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week45.html#recurrent-neural-networks-rnns-overarching-view, 2023. Accessed: 2023-12-14.

[16] Morten Hjorth-Jensen. Week 46: Decision Trees, Ensemble methods and Rando Forests. https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week46.html, 2023. Accessed: 2023-12-13.

[17] Morten Hjorth-Jensen. Week 47: From Decision Trees to Ensemble Methods, Random Forests and Boosting Methods and Summary of Course. https://compphysics.github.io/MachineLearning/doc/LectureNotes/_build/html/week47.html, 2023. Accessed: 2023-12-13.

[18] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 11 1997.

[19] Keras. Dense layer class, Keras. https://keras.io/api/layers/core_layers/dense/, 2023. Accessed: 2023-12-18.

[20] Navmeen Latif, Joseph Selvam, Manohar Kapse, Vinod Sharma, and Vaishali Mahajan. Comparative performance of lstm and arima for the short-term prediction of bitcoin prices. *Australasian Accounting, Business and Finance Journal*, 17:256–276, 02 2023.

[21] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008. Accessed: 2023-12-14.

[22] scikit-learn developers. sklearn.tree.decisiontreeregressor. https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeRegressor.html. Accessed: 2023-12-14.

[23] Statology. A gentle introduction to poisson regression for count data, 2023. Accessed: 2023-12-13.

[24] Anna Åberg and Christine Sjölander. Building data classification and association. https://lup.lub.lu.se/luur/download?func=downloadFile&recordOId=8954429&fileOId=8954430, 2018. Page: 30, Accessed: 2023-12-13.