Cryptocurrency Predictor – Final Report

CPT_S 315 – Data Mining
Honghao Wei

Project Members:
Darron Li, Sai Gubba, Seth Trimberger, Ben Metzger
Josh Santiago, Travis Takushi, Jacob Fox, Giri Vignesh, Jacob Kolk

Abstract

Cryptocurrency has gained a lot of attention over the past few years due to the potential to make substantial financial returns. However, predicting cryptocurrency prices is extremely challenging due to the nature of the market. It is extremely complex and dynamic, and we are using a machine learning-based cryptocurrency price predictor that uses historical price data, technical indicators, and special analysis to improve forecasting accuracy. We will be using special data mining techniques such as data clustering, classification, the association rule, and much more to help us deploy a successful predictor model to generate reliable price predictions.

Introduction *[Josh]*

Cryptocurrency is a type of digital or virtual currency that uses methods such as cryptography for security and operates independently. This means that a central authority like a government or financial institutions like banks do not oversee cryptocurrencies. **Bitcoin** was the first of its kind, created by an anonymous individual or party under the alias of Satoshi Nakamoto in 2009. Concerned that currencies were too reliant on authoritative figures like the government or banks, Nakamoto wanted to bring a more decentralized currency, managed by a network of computers built on blockchain technology. This allowed for a transparent, secure, and anonymous transactional experience (Britannica, 2025).

Since the conception of Bitcoin in 2009, the technology behind it has evolved at an exponential rate and prices rose accordingly as cryptocurrencies entered the public eye. Alongside bitcoin, new cryptocurrencies have emerged. These include coins and currencies like Ethereum (ETH), Solana (SOL), USD Coin (USDC), and even "meme coins" like Dogecoin (DOGE).

When Bitcoin was created, the coin was as low as $0.10 during its early years. Bitcoin's price began to grow steadily during the 2010s before skyrocketing to tens of thousands of dollars during 2020. While there were low points where the coin lost thousands of dollars in value, recent trends of this volatile coin have shown Bitcoin's price has exponentially grown during the last year, breaking the $100,000 per Bitcoin mark during January of 2025.

The Causes of Crypto Volatility

Cryptocurrency prices are far more volatile than traditional financial assets like stocks, bonds, or index funds. There are multiple factors that contribute to the instability of cryptocurrency:

- Cryptocurrencies do not have a clear intrinsic value and are primarily driven by speculation and sentiment, not fundamentals. *When people believe in a coin's future, the price can skyrocket. If that faith falls, then the coin can crash hard.*
- The lack of market regulation can lead to market manipulation. Schemes like "pump and dump" can be coordinated, where groups artificially create hype and inflate prices, selling to make a profit and crash prices for many others.
- Low liquidity means that cryptocurrencies have smaller trading volumes, meaning a smaller amount of money could move the price significantly.
- Crypto markets are open 24 hours a day, 7 days a week, across all time zones as opposed to traditional stock markets which open and close at certain times.
- Social media can heavily influence a coin's value. Memes, viral Reddit threads, celebrity endorsements, and hype may cause massive price improvements. These platforms can either amplify or cause massive panic among investors which can easily sway the price.

Our Methods for Prediction

While the volatile nature of cryptocurrencies makes it difficult to predict, there are multiple methods and models that can be utilized to predict possible prices for cryptocurrencies, which may help investors make predictions and informed decisions to invest and hopefully turn a profit.

We explored several machine learning and data mining models to predict cryptocurrency prices:

- **Long Short-Term Memory (LSTM):** A type of *Recurrent Neural Network (RNN)* that is designed to process and analyze sequential data, like text, speech, and time series data like crypto prices.

- **Random Forests:** A machine learning algorithm that uses an ensemble of decision trees to make predictions. These work great to prevent overfitting and work well with noisy data like volatile cryptocurrency prices.
- **Linear Regression:** A method to predict the value of a variable based on the value of another variable. Models the relationship between independent variables and the dependent variable (future price).

Long Short-Term Memory (LSTM) *[Darron, Jacob K.]*

For the LSTM approach to the neural network, it was crucial to understand what LSTM was in the first place. LSTMs are a special type of recurrent neural network (RNN) that excels at learning patterns in sequential or time-series data. Because of how volatile and bullish cryptocurrency is, we figured that the LSTM approach would be a powerful approach to help predict cryptocurrency for both short- and long-term dependencies based on the historical pricings.

How it works is dependent on two types of memory: short-term memory (the ST in LSTM), and long-term memory, (the LM in LSTM). Short-term memory holds current inputs, while long term memory carries more accumulated patterns over time. Each LSTM cell uses gates to control how much information should be passed forward or remembered. Gates are a special component within each LSTM cell that controls the flow of information. It's like a valve that opens or closes to decide what information to keep, what information to get, and what information to add. The LSTM gates rely on sigmoid activation functions which output a value between 0 and 1, which is basically a percentage that determines the importance of the incoming information. For example, a value of 0.997 means nearly all of the short-term input would be transferred to long-term memory. The percentage that gets stored is learned during training with backpropagation. As the model is exposed to more data, it minimizes the prediction error by adjusting internal weights, which then influences the gate's decisions to keep, forget, and output. This is all extremely important, but it does come at its downsides.

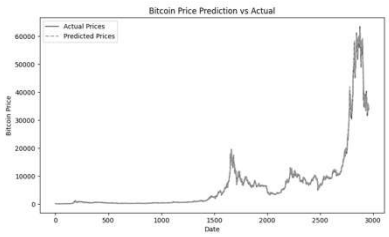
While LSTMS are great for our current project, they have several downsides such as complexity, overfitting, hyperparameter requirements, and limited interpretability. Throughout the project, we had to face many of these issues, with the most problematic one being the complexity. Due to the nature of how

LSTM works, they have a complex architecture with multiple gates per cell, which leads to weighing more parameters than simpler models such as linear regression. With datasets as large as historic prices of crypto currency, the computational cost of LSTMs grew exponentially. Speaking of large data models, it is a good thing that our crypto-currency project contains large sets of csv data for us to train and test. LSTMs naturally needs lots of sequential data to learn effectively and calculate an accurate price point. With smaller datasets, noisy data tends to be more persistent and may even overfit or perform worse than other models that are historically not as accurate. Not to mention, at the end of the output the data is extremely confusing. After we were done coding and testing, we were extremely confused on what the numbers meant. It took an extremely large amount of time going in to understand whether the data was accurate or not, whether it was useful, and even whether it was important. Unlike decision trees, it is difficult to explain why an LSTM made a specific prediction.

```
# training loop
for epoch in range(num_epochs):
    for batch_x, batch_y in data_loader:
        # forward pass: model predictions
        predictions = model(batch_x)
        loss = criterion(predictions, batch_y)

        # backwards pass: optimization
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()
```

Now, the reason we chose LSTM in the end to continue our future research on if we were given the chance is because it captures time dependencies quite well. Despite all the things we mentioned above, cryptocurrency naturally has large sets of data that already have large amounts of parameters. Cryptocurrency also is time series data, which depends on past trends, patterns, and market behavior. Unlike feedforward neural networks, LSTMs have memory, which means that they can actually remember what happened in the past and use that same context to make better predictions. We kind of saw it as the larger the data set, the more accurate our model became. Cryptocurrency markets also have cyclical trends, with monthly or even yearly volatility being relatively common. They were designed to retain the long-term dependencies, due to their gating mechanisms and back propagation, and this gives it an edge over simpler, smaller neural networks that tend to forget earlier data. Because of the complex, noisy, and sequential data that is cryptocurrency, we figured that the LSTM network was best suited for future usage even if the downsides can be costly.Now that our reasoning for choosing this model is understood, let us dive into our implementation and how we decided to make it. Our first step is to normalize the data. This is important to prevent exploding or vanishing gradients, ensure equal feature influence, and improve generalization to hopefully allow it to perform better unseen, volatile data. We used Z-Score normalization in order to form it into tensors, which we set up to be a thirty by six array. Thirty-two is the sequence length, or the number of days that we run through the model to get our prediction. Six is the number of features in our data set. We chose to include market cap, close price, open price, high price, low price, and volume. Thirty-two days of data and all the features are used to predict one day, and then each subsequent day is predicted with a tensor of the same size, but with the last thirty-two days of data. Now it was time to train our model. We used two layers, which means we have effectively two models stacked on top of each other. This works by passing data through one layer, which captures more short-term data, then directly into the other layer, which is usually better at forming long term predictions. Another hyper parameter we set was the loss function. We chose to go with the mean squared error. This is a function that penalizes error more heavily and is perfect for our model. One thing we did to prevent overfitting is shuffle the batches so they every epoch is a random order of thirty-two-day sequences. Now that everything is set up, we can begin the training. Our training loop is pictured. The line `predictions = model(batch_x)` is where the prediction comes from, then the error is calculated. Backpropagation then calculates gradients of the error with respect to model parameters, and the optimizer updates weights to reduce loss. After training it is now time to get out predictions and test our model.



To actually get our predictions, we just pass a tensor to out model and a prediction is returned. We do the same for all the thirty-two-day sequences, and we are left with a complete data set of predictions. Our mean absolute percentage error (MAPE) is calculated with the predictions and plot it. As you can see in our plot, our predictions were reasonably accurate, with about a 15% MAPE average throughout the runs. The chart shows it was in the general ballpark (Orange is actual value, Blue is predicted). We had better predictions at low volatility points and worse predictions at higher volatility points, which is to be expected. Although it was off by about fifteen percent on average, it usually was correct on predicting if the price would go up or down, which is enough accuracy for this model to have some use. Our model is comparable to others we found online; however, one major difference is ours has the potential to fail massively, at random. While our error was usually under 30%, occasionally it would exceed 70%, which if used in a live trading scenario would cause major losses. One way we could fix this is saving the model state that made the best predictions, instead of training a new one every time. This would result in less volatility in the error, however just because the model was good on the data it was trained on, does not mean it will do well on life prices, so further testing is needed if we would want to make this into a full trading bot. Based on our results, it is a viable concept.

Random Forests *[Sai, Seth, and Giri]*

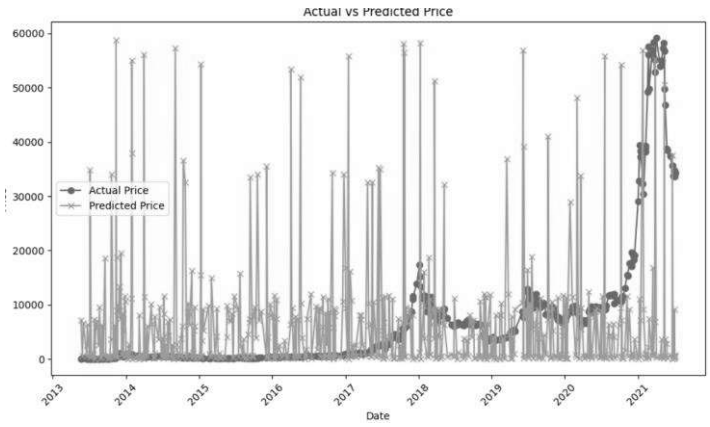
Random Forests are a machine-learning algorithm that processes large amounts of data into multiple decision trees to improve the accuracy of a prediction and control overfitting. Each decision tree in the forest is trained on a random subset of data and the final prediction is calculated through classification. Given the volatility of cryptocurrency, we wanted our Random Forest models to parse through our datasets to calculate patterns at random times and valuations of our selected cryptocurrencies to return a more accurate prediction of whether the value of a cryptocurrency will go up or down in price. Specifically, this prediction is based on calculating the Close price of a cryptocurrency one and two days from a given date and expecting the Random Forest models to keep track of these patterns.

Random Forests are useful since they gather data from a range of patterns that are random throughout a period rather than smaller sections, making this algorithm useful for volatile cryptocurrency prices. Machine learning classification algorithms for cryptocurrencies have a 55 to 65 percent accuracy (Akyildirim et. al.) given the unpredictable nature of cryptocurrencies, so any of our random forest models with accuracies within and around that range are deemed acceptable. We also calculate the R^2 value to indicate the fit of a prediction model on the Close testing data.

We are training our random forest models on the Bitcoin and Ethereum cryptocurrencies. Our first random forest model tests the prediction model on the first 20% of data and training on the remaining 80% of data. For Bitcoin, the model has an accuracy of 63.82% and an R^2 score of 0.9380, while for Ethereum, the model has an accuracy of 42.92% and an R^2 score of 0.3044. The low R^2 in the case of Ethereum is due to the training dataset containing high data values, while the testing dataset contains low data values. This can be solved by adding more low data values to the training dataset. The second random forest model tests the prediction model on the last 20% of the data and training on the first 80%

of the data. The second random forest model tests the last 20% of its data on the first 80% of the data. For Bitcoin, this model has an accuracy of 64.88% and an R^2 value of .1309. For Ethereum, it had an accuracy of 59.95% and an R^2 value of .4331. The third random forest model tests a random 20% of the data while training on the remaining 80%. For Bitcoin, the model has an accuracy of 78.43% and an R^2 value of 0.9965, while for Ethereum, the model has an accuracy of 66.44% and 0.9932.

Our preferred random forest model is testing a random 20% of data and training on the remaining 80%. On average, it has an accuracy of 72.4% compared to testing the first 20% of data (53.4%) or testing the last 20% of data (62.4%), and an R^2 value of 0.995 compared to 0.621 from testing the first 20% of data or 0.282 from testing the last 20% of data. Right now, random sampling across the full time series best captures the inherent fluctuations in price due to outside factors. This is with the limited data set we had we chose the random forest model. In the future, however, we would like to switch our model to 80-20% split. Right now, our current model has random spikes throughout the entire data set due to testing data that is taken from as low as .10 cent USD to all the way to $60,000 USD. The same is said for our training data. This is why our graphs have predictions all over the place.



We know this is causing unstable and erratic predictions and produces jumps widely whenever the model thinks there should be an extreme price adjustment. Right now, this model has extreme overlap between prices, and a rough output on predicting future prices that make sense. But based off the limited data we had this model produced a higher accuracy and R^2 score making it our preferred model until we get new testing data.

Even with its high predictive performance, our Random Forest method has a number of significant drawbacks and obvious areas for development: training and evaluating hundreds of deep trees over a price series that swings from pennies to tens of thousands of dollars demands substantial CPU/GPU resources and memory bandwidth, which slows down both batch retraining and any hope of near–real-time inference; sampling uniformly across all price extremes makes the model overly sensitive to rare outliers small dips or massive spikes that don't necessarily reflect broader market trends; price-up and price-down events aren't equally frequent, so the forest gravitates toward the majority class unless we rebalance; and despite feature-importance scores, it remains difficult to interpret why individual trees vote the way they do. To mitigate these issues, we'll explore parallelized tree-building libraries (e.g., scikit-learn's n_jobs or Spark MLlib), incremental learning techniques, and Extremely Randomized Trees; introduce robust pre-processing steps to dampen anomalies; implement SMOTE or class-weighting and custom sampling strategies to force equal exposure to up/down labels; and build surrogate models, partial-dependence plots, and integrate SHAP or LIME explanations for greater transparency. On the data side, we plan to enrich our inputs with engineered time-series features technical indicators such as rolling-

window volatility measures, on-chain activity metrics, and social-media sentiment—while replacing random splits with rolling and expanding windows plus nested cross-validation to respect chronology. We'll also upgrade hyperparameter tuning from grid search to Bayesian optimization to find the ideal number of trees, depth, and leaf sizes, and benchmark against gradient-boosted frameworks and LSTM networks to determine which architecture best captures temporal dependencies. Finally, a continuous monitoring pipeline will track live-production performance and drift metrics, automatically triggering retraining jobs whenever accuracy or R^2 falls below defined thresholds. Some of the drawbacks of using random forests include the complexity of using hundreds of trees, potential overfitting on noisy data, bias toward dominant classes in classification, and less interpretability.

Random Forests, ensembles of decision trees trained on random subsets of data are well suited to the volatility of cryptocurrency prices because they reduce overfitting and capture a wide range of market patterns. In our study on Bitcoin and Ethereum, we compared three data-splitting strategies first 20% vs. last 20% vs. random 20% test sets finding that the random split achieved the highest average accuracy (72.4%) and R^2 (0.995) compared to 53.4%/0.621 and 62.4%/0.282 for the first- and last-20% splits, respectively. Despite these strong results, the method is computationally intensive, can overreact to extreme outliers, exhibits bias toward the majority price-movement class, and offers limited interpretability at the individual tree level. To address these drawbacks, we will enrich and rebalance our dataset (adding low-price observations and using SMOTE or class weighting), engineer additional features such as RSI/MACD and sentiment signals technical indicators, and implement robust preprocessing. We'll also adopt time series aware validation schemes, upgrade hyperparameter tuning via Bayesian optimization, and benchmark against gradient-boosted trees and LSTM models to better capture temporal dependencies. Finally, integrating explainability tools such as SHAP or LIME and deploying a live monitoring pipeline for drift detection and automated retraining will ensure our predictions remain transparent, adaptive, and accurate in a production setting.

Linear Regression [Ben, Travis, Jacob]

Linear regression is a supervised machine-learning algorithm which uses linear relationships between a dependent variable and multiple independent variables to predict future values on datasets. This means that the model is time-based, which was perfect for predicting cryptocurrency, which heavily relies on time-output data. Typically, someone who wants to invest in cryptocurrency will look for indicators known as bear(decreasing) and bull(increasing) markets. They do this through looking at market trends over time, specifically open price, close price, volume, and market capitalization, to know when to invest in a specific cryptocurrency. However, cryptocurrencies can be very volatile and can change drastically on a day-to-day basis. This is why linear regression is perfect for this specific type of prediction problem, as it is excellent at predicting anywhere from 1-2 days in the future.

Linear regression achieves this sort of prediction pattern through looking at labeled datasets. It then uses these datasets to create a linear regression function, through a simple given function:

Linear Regression Equation

$$Y = a + bx$$

$$a = \frac{[(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)]}{[n(\Sigma x^2) - (\Sigma x)^2]}$$

$$b = \frac{[n(\Sigma xy) - (\Sigma x)(\Sigma y)]}{[n(\Sigma x^2) - (\Sigma x)^2]}$$

The model can then use this pattern to predict one to two days into the future, based off the slope of this line, fitting the created data to match what the slope predicts.

To achieve this in our project, we used what's known as "lag" on our variables, which is essentially shifting the close data 1 to 2 days back, which then becomes the open for the next day, offsetting the values, and creating a situation where our model needs to compare how close the open and "lagged" close values are. The reason we want to "lag" data is because while cryptocurrency does shift dramatically day-to-day, the largest margins are created over years and years. For example, in our dataset, the highs and lows of Bitcoin ranged anywhere from $200 to $1000 over a time period of about 5 years. By shifting the "lag" further back, this allowed our model to have more data to predict on and therefore allow it to become more accurate at predicting. This is a simple but effective model, as it gets continuous feedback, allowing it to improve dramatically over a small dataset.

With this in mind, we labeled the data as either part of a bear or bull market, using the average close over a 30-day period. If the average was negative, we would label it as a bear market, as the value was decreasing; if the average was positive, we would label it as a bull market, as the value was increasing. In the future, we would like to add more variance by including labels like sideways markets, which shows that the market is stagnant, so the average would be 0; as well as including a shorter trend period of around 5-10 days, increasing the accuracy of our trends. We then introduced our lag to the model, pushing the close dates back two days.

With our data ready to be input into the model, we first defined our features as close_lag_1, close_lag_2, volume_lag_1, market_cap_lag_2, market_label; meaning our predicted data would mainly have a relationship between the close, and these lagged variables. We then input this into our model, with the training size of our dataset being around 80% and testing size being around 20%.

From our model, we extrapolated the Mean Squared Error and R^2 of our model, with the MSE telling us how close our regression line was to the actual data and the R^2 telling us how much influence our independent variables had on our dependent variable. Our R^2 for both Bitcoin and Ethereum came out as positive, indicating a high correlation between variables at around 98-99% each. However, our MSE scores for either of them fluctuated dramatically. With a highly volatile cryptocurrency like bitcoin, the mean squared error shifted a lot, coming out to an error of about 1,630,601. This sounds like a lot, but compared to the actual values of about 1.0e9-10, the error is relatively small. The same goes for Ethereum, but even smaller, as Ethereum tends to be a relatively stable coin, with an MSE of about 12,719, compared to an average value of about 100,000.

Taking this trained model, we then had it predict future data, about 7 days into the future. We found that when comparing it to real world data, the model was accurate to about 1-2 days out, with anything after that being an overfit prediction. The model tended to think that whatever direction it was heading towards, it would keep on that track, meaning that high highs and low lows were the only thing it could predict in the future, rather than bounce backs or crashes. This is due to the fact that it was only able to use the previous two days as data, as when it lags, in order to fill in the missing data (as there is no data there yet), it just uses the previous two days close, resulting in overfitting.

In the future, we want to change this outcome by decreasing the time period for labeling, as well as increasing the amount of lag we introduce into our model. We also want to increase the number of labels that we have, so that our model can be more accurate when trying to predict a real-world market. We also want to include a way for our model to dynamically predict bear and bull markets, rather than just using the last known one; that way, it will ensure that our data won't be a result of overfitting. Overall, our model was very successful in what we aimed to do, as it was able to predict against given data, as well as predict future data, at least 1-2 days in advance.

Using linear regression works well for predicting values 1-2 days in the future but is unable to find long-term patterns and will not be able to work with external sources such as current politics and economic regulations. One drawback with this model for this problem is that it assumes linearity in the relationship between our attributes, when these values are non-linear and sensitive to volatile changes at any point. Our prediction model works well for predicting next day prices with a high-ranking accuracy compared to the other models, but for long-term predictions of market sentiment and volatility prediction, we should opt for using RNN as our selected model.

Conclusion/Summary

We aim to build a project that predicts cryptocurrency prices using data mining techniques. We will collect the data from an online database and use libraries such as pandas to clean and preprocess the data. Then, we will use classifying and clustering techniques to build the machine-learning model. To evaluate our model, we will use regression techniques and back-testing to ensure that our model performs up to standard. While this project contains limitations and challenges, we have plenty of ways to overcome those and ensure our project is successful. Finally, we will address any ethical concerns or challenges by ensuring that we prioritize transparency, fairness, and robust data protection.

Group Member Contribution

Each group member contributed an equal amount to the research of our problem statement, development of our models, compilation of data for our findings, and the presentation and paper of our findings.

References

[1] Dunn, J. C. (1973). A fuzzy relative of the ISODATA process and its use in detecting compact well-separated clusters. *Journal of Cybernetics*, *3*(3), 32–57.

https://doi.org/10.1080/01969727308546046

[2] G.M. Caporale, A. Plastun, Price overreactions in the cryptocurrency market, J. Econ. Stud. 46 (5)(2019) 1137–1155. https://www.researchgate.net/publication/335943647_Price_overreactions_in_the_cryptocurrency_market

[3] Britannica, T. Editors of Encyclopaedia (2025, April 26). cryptocurrency. Encyclopedia Britannica. https://www.britannica.com/money/cryptocurrency

[4] T. Singh, S. Mishra, D. Pandey, C. Sharma, S. Koli and K. Joshi, "Crypto currency-bitcoin Price Predictor using Linear Regression and random forest," *2024 International Conference on Electrical Electronics and Computing Technologies (ICEECT)*, Greater Noida, India, 2024, pp. 15, doi: 10.1109/ICEECT61758.2024.10738906.

[5] Akyildirim, E., Goncu, A. & Sensoy, A. Prediction of cryptocurrency returns using machine learning. *Ann Oper Res* **297**, 3–36 (2021). https://doi.org/10.1007/s10479-020-03575-y