# Trading and Finding Arbitrages in Bitcoin Markets

**Anvita Panandikar | Mani Sawhney | Riya Sheth**

**Abstract**

Cryptocurrency markets exhibit periods of large, recurrent arbitrage opportunities across exchanges. These price deviations are much larger across than within countries, and smaller between cryptocurrencies, highlighting the importance of capital controls for the movement of arbitrage capital. Price deviations across countries co-move and open up in times of large bitcoin appreciation. Countries with higher bitcoin premia over the US bitcoin price see widening arbitrage deviations when bitcoin appreciates. Upon utilizing various machine learning models, we achieved the lowest root-mean-squared error using a XGBoost model (Non-Parametric), Forward Stepwise Ordinary Least Squares model (Parametric), and an Autoregressive Integrated Moving Average model (Time Series), leading to a realized profit of $8,951,693.79 across the span of 5 years.

## 1 Introduction

Bitcoin markets have gained significant attention over the past few years due to their unique characteristics and potential for high returns. The decentralized nature of bitcoin, coupled with its limited supply, has led to a surge in demand from investors looking to diversify their portfolios and capitalize on the cryptocurrency's volatility. One strategy that has emerged in the bitcoin market is the concept of arbitrage, which involves buying and selling bitcoin on different exchanges to take advantage of price discrepancies and generate profits.

Arbitrage is a well-known strategy in traditional financial markets, but its application in the bitcoin market is particularly interesting due to the decentralized nature of the cryptocurrency. Unlike traditional financial markets, bitcoin is not regulated by a central authority, which has led to a fragmented market with significant price discrepancies across different exchanges and markets. This creates opportunities for arbitrageurs to exploit these price discrepancies and generate profits by buying bitcoin on one exchange and selling it on another at a higher price.

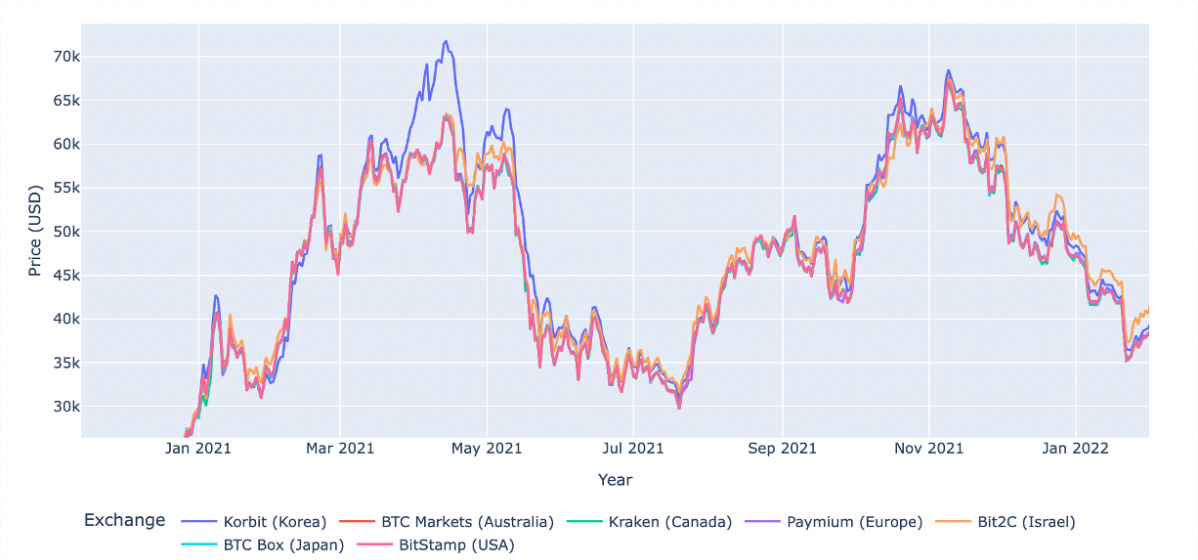


Figure 1: Evidence of Arbitrage Opportunities

## 2 Expectation

In this paper, we explore the concept of arbitrage in the bitcoin market and its potential for generating profits. We examine the factors that contribute to price discrepancies across different exchanges and markets, and the challenges and risks associated with arbitrage in the bitcoin market. We will be utilizing three groups of models - Non-Parametric, Parametric, and Time Series, to predict the prices on each exchange. Once that has been achieved, we will be transacting by buying bitcoin on the exchange with the lowest price and simultaneously selling it on the exchange with the highest price.

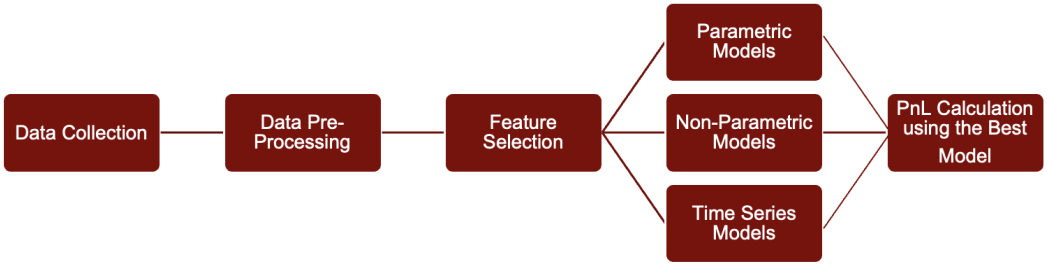


Figure 2: Models Deployed and Analysed

## 3 Procedures

### 3.1 Data Collection

For the purposes of our project, we imported data from Bitcoincharts[1] for Bitcoin across these 7 exchanges: BitStamp (USA), BTC Markets (Australia), Kraken (Canada), Paymium (Europe), BTC Box (Japan), Korbit (Korea), and Bit2C (Israel). The data collected ranges from January 1st, 2018 to December 31st, 2022.

To convert our foreign rates from these exchanges to our base currency, we imported FX rates from Nasdaq Data Link from the EDI Foreign Exchange Rates (CUR) data feed[2] with USD as the base currency. The data is constructed daily, aggregating executable quotes from major investment banks and FX dealers.

We also obtained hash rate and average block size from Nasdaq Data Link using the BChain data feed[3].

For the sentiment analysis of bitcoin, we imported data from Augmento[4]. Augmento ingests crypto related posts from Twitter, Reddit and Bitcointalk and turns the social sentiment of these posts into data. Augmento's AI is trained exclusively on cryptoasset- and trading-related language and uses a hybrid approach (machine learning and linguistic analysis) to ensure the highest achievable accuracy. While conventional sentiment analysis can merely differentiate between positive and negative sentiments, Augmento's AI can recognize and quantify the entire spectrum of crypto crowd psychology.

### 3.2 Data Preprocessing

In our implementation, we are aggregating trades that are happening within 10-minute intervals, by selecting the last data point within each interval, and filling any missing values with the last known value.

[1] http://api.bitcoincharts.com/v1/csv/
[2] https://data.nasdaq.com/data/CUR-foreign-exchange-rates
[3] https://data.nasdaq.com/data/BCHAIN-blockchain
[4] https://www.augmento.ai/

Our data from the seven exchanges include the date of the trade, timestamp (in UTC), price and volume of the cryptocurrency.

Once we get our trade data, we convert the prices using the FX rates to our base currency (USD).

We then perform the following preprocessing steps:

### 3.2.1 Splitting of the Date

The date is split into the following columns Year, Month, Day, Hour, and Minute. Splitting a date into its constituent parts can be helpful when building a regression model because it can help the model to capture time-based patterns or trends in the data.

For example, if the data being modeled has a strong seasonal component, splitting the date into its month or day components can help the model to capture the effect of seasonality. Similarly, if the data has patterns that vary by hour of the day, splitting the timestamp into hour components can help to capture these patterns.

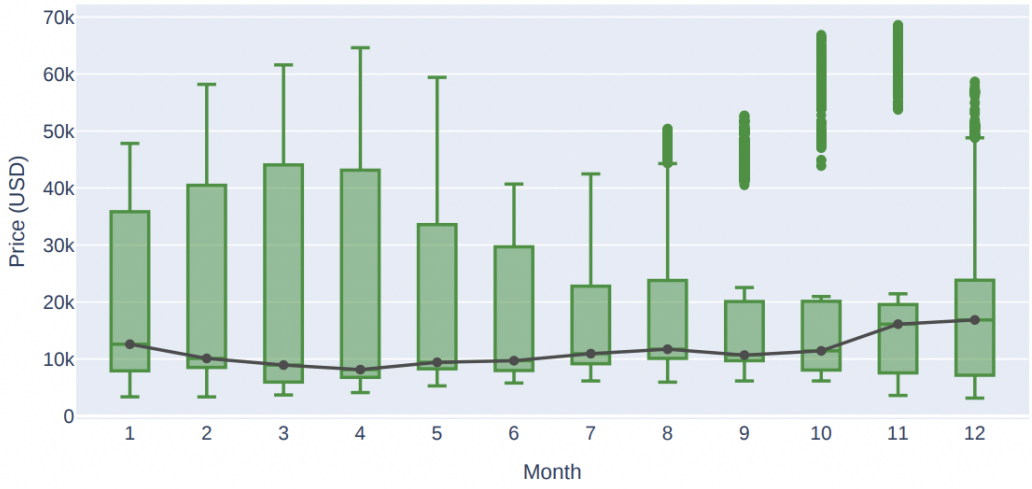


Figure 3: Variation of BTC Price by Month

### 3.2.2 Use of LabelEncoder in the Year

The LabelEncoder is a class in the scikit-learn library that is used to encode categorical variables into numeric values. Here, the LabelEncoder is used to convert the Year column, which is a categorical variable, into numeric values. The LabelEncoder assigns a unique integer value to each distinct category in the column. For example, if the Year column contains the categories "2019", "2020", and "2021", the LabelEncoder will assign the values 0, 1, and 2, respectively, to these categories. By converting the Year column to numeric values, we can use it as a feature in machine learning models to make predictions about future prices.

### 3.2.3 Logarithmic Mean of Lagged Prices

Lagged mean is used as a feature in time series analysis to capture the trend and seasonality in the data. The idea is that the previous mean prices (i.e., the mean prices from the previous days) can provide valuable information about the current price. For example, if the mean prices for the past few days have been increasing, it may be an indication that the current price is likely to increase as well.

By computing the lagged mean of the price variable for a range of lags (i.e., the mean of the price variable for each day, two days ago, three days ago, and so on), we can create new features that capture

the trend and seasonality in the data. These features can then be used in machine learning models to make predictions about future prices.

In our code, the natural logarithm of the mean of the price column for each date in the DataFrame is calculated, shifted by one day to seven days and 15 days, as well as 30 days ago, and stored as new columns. The natural logarithm is used instead of the raw mean to transform the data to a more normal distribution and to reduce the impact of large price fluctuations on the mean.

### 3.2.4 Normalization of Non-Price Data

The MinMaxScaler is a class in the scikit-learn library that is used to scale and translate each feature individually such that it is in the given range on the training set, between zero and one. Normalizing features in a regression model can enhance the accuracy and stability of the model. It can aid the optimization algorithm in converging quickly and reliably by avoiding issues related to different scales of input features. It can also prevent overfitting by reducing the magnitude of the features, and improving model performance by reducing the effect of outliers or extreme values in the input features.

### 3.2.5 Domain Knowledge for Lagged Prices

Domain knowledge is useful here in the context of dimensionality reduction for lagged prices by identifying the most relevant features that are likely to have a strong impact on the price. Since lagged prices are highly correlated with each other, we aimed to only keep the most relevant lags in our dataset to avoid multicollinearity. In order to do this, we have utilized Recursive feature elimination with cross-validation (RFECV) with XGBoostRegressor as the estimator. RFECV works by recursively removing variables from a model and evaluating its performance using cross-validation, in order to identify the set of variables that produces the best performance. We chose XGBoostRegressor as the estimator since the XGBoostRegressor provides feature importance scores, which can be used to identify the relative importance of each variable in the model. This information can be used to further refine the set of selected variables, by removing variables with low importance scores.

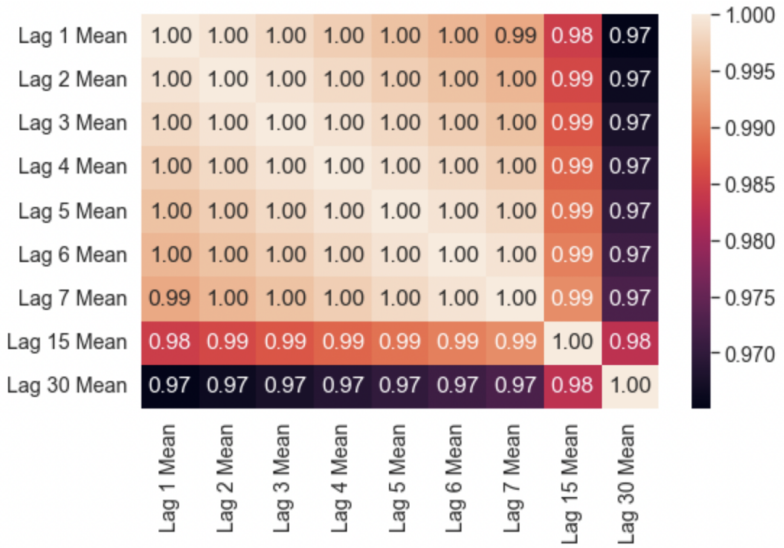


Figure 4: Evidence of Multicollinearity

# 4 Implementation

## 4.1 Splitting of data into train and test

We have incorporated a 80:20 train-test split for our data. This approach can help avoid overfitting, where the model memorizes the training data and performs poorly on unseen data. Secondly, it provides an estimate of the model's performance on new, unseen data, enabling us to compare different models and select the best one. Additionally, it allows us to tune the model's hyperparameters and optimize its performance. Finally, it can help identify data quality issues such as missing values, outliers, or data imbalance, allowing us to address these issues before deploying the model in production. In conclusion, a train-test split is a crucial technique that facilitates model evaluation, optimization, and deployment in real-world applications. Given that our data utilizes lagged prices, we decided to choose this approach versus other approaches such as Leave-One-Out and K-Fold data splitting since we can ensure that there will be no look-ahead bias in our train-test split.

## 4.2 Arbitrage

Arbitrage is the practice of buying an asset on one market and simultaneously selling it on another market for a higher price, thereby profiting from the price difference. The trading strategy the we employ is that the we predict prices from all the exchanges. Then we determine which exchanges shows the lowest price and which shows the highest price. We go long or buy the bitcoin at the shortest price and then short or sell the bitcoin at the highest price. Once we execute this trade, we determine the profit by using the actual prices from the dataframe. The trade is only executed if the profit is greater than 2% of the average prediction of the bitcoin prices to ensure that the trade is only done when there is a genuine possibility of an arbitrage instead of market volatility.

### 4.2.1 Assumptions

- The transaction costs for each of the trades is negligible.
- We do not hold a position for longer than a day, that is the buy and the sell trade is executed concurrently on the same day.
- We have no limitations on the capital.

## 4.3 Model Building

We have incorporated various regressive machine learning techniques from the three categories, namely non-parametric models, parametric models and time-series models. Models from the three different models were used because testing different types of models can help identify which model is best suited for a particular dataset and can provide insights into the underlying patterns and relationships in the data.

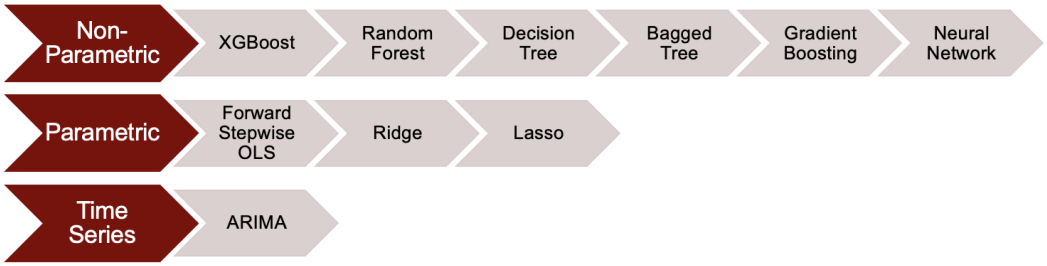


Figure 5: Models Deployed and Analysed

### 4.3.1 Non-Parametric Models

**XGBoost** In XGBoost, the algorithm fits a regression tree to the residual errors of the previous tree, where the residual is the difference between the predicted and actual values. This iterative process continues until a stopping criterion is met, such as a maximum number of trees or a threshold on the improvement in performance. XGBoost also incorporates regularization techniques such as L1 and L2 regularization, and also uses a technique called "tree pruning" to remove redundant nodes in the trees.

$$\hat{y} = \sum_{i=1}^{M} w_i h_i(x)$$

where $\hat{y}$ is the predicted value, $w_i$ is the weight of the $i$th tree, $h_i(x)$ is the output of the $i$th tree, and M is the total number of trees in the model.

**Random Forest** Random Forest uses an ensemble of decision trees to make predictions. In a random forest model, multiple decision trees are built independently and combined to make a final prediction. The trees in a random forest are constructed using a technique called "bagging," where each tree is trained on a randomly sampled subset of the data with replacement.

$$\hat{y} = \frac{1}{M} \sum_{j=1}^{M} T_j(x)$$

where $\hat{y}$ is the predicted value, $M$ is the number of trees in the forest, $T_j$ is the $j$th decision tree, and $x$ is the input feature vector.

**Decision Tree** In a Decision tree model, the algorithm starts with the entire dataset and selects the input feature that maximally separates the target variable into different groups. It then splits the data based on the selected feature, and repeats the process on each subset recursively until some stopping criterion is met, such as a maximum tree depth or a minimum number of samples in the leaf nodes. The regression model in each leaf node is typically the mean or median of the target variable in that node.

$$\hat{y} = f(x) = \sum_{m=1}^{M} c_m \cdot \mathbb{I}(x \in R_m)$$

where $\hat{y}$ is the predicted value, $f(x)$ is the regression function, $M$ is the number of leaf nodes in the tree, $R_m$ is the rectangular region corresponding to the $m$th leaf node, $c_m$ is the predicted value in the $m$th leaf node, and $\mathbb{I}(x \in R_m)$ is an indicator function that takes the value of 1 if $x$ belongs to $R_m$ and 0 otherwise.

**Bagged Tree** Bagged Trees, also known as Bootstrap Aggregating, is an ensemble machine learning method that involves training multiple decision trees on different subsets of the training data and combining their predictions to improve the overall model performance. In a bagged tree model, multiple decision trees are built independently using bootstrapped samples of the original training data. This means that each decision tree is trained on a random subset of the original data, with replacement.

$$\hat{y} = \frac{1}{B} \sum_{b=1}^{B} T_b(x)$$

where $\hat{y}$ is the predicted value, $B$ is the number of decision trees in the bagged tree model, $T_b$ is the $b$th decision tree, and $x$ is the input feature vector.

**Gradient Boosting** Gradient Boosting combines multiple weak learners, typically decision trees, to create a more powerful predictive model. The primary idea behind gradient boosting is to iteratively train decision trees that correct the errors of the previous trees in the ensemble. In other words, each subsequent tree is trained to predict the residual errors of the previous trees, rather than the actual target variable. One key difference between XGBoost is that XGBoost employs a technique called regularized boosting, which includes both a loss function and a regularization term to prevent overfitting. Additionally, XGBoost uses a second-order approximation to the objective function.

$$\hat{y} = \sum_{m=1}^{M} \gamma_m h_m(x)$$

where $\hat{y}$ is the predicted value, $\gamma_m$ is the weight or contribution of the $m$th decision tree, and $h_m(x)$ is the predicted value of the $m$th decision tree for input feature vector $x$.

**Neural Network** The basic architecture of a neural network for regression consists of an input layer, one or more hidden layers, and an output layer. Each layer consists of multiple neurons, with each neuron performing a weighted sum of its inputs, passing the result through an activation function, and outputting the result to the next layer.

The weights and biases of the neural network are initially set randomly, and the model is trained using an optimization algorithm such as stochastic gradient descent. During training, the weights and biases are adjusted to minimize the difference between the predicted values and the actual target values in the training data.

$$\hat{y} = f(W_2 f(W_1 x + b_1) + b_2)$$

where $\hat{y}$ is the predicted value, $x$ is the input feature vector, $W_1$ and $W_2$ are the weight matrices for the first and second layers, $b_1$ and $b_2$ are the biases for the first and second layers, and $f$ is the activation function. The output of the final layer is the predicted value $\hat{y}$.

### 4.3.2 Parametric Models

**Forward Stepwise OLS** Forward stepwise OLS (Ordinary Least Squares) is a feature selection method used in regression analysis to identify the subset of input variables that have the strongest association with the output variable. It is an iterative process that starts with an initial model containing no predictors and adds one predictor at a time, evaluating the improvement in model fit with each addition.

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + ... + \beta_p x_p + \epsilon$$

where $y$ is the output variable, $x_1$, $x_2$, ..., $x_p$ are the input variables, $\beta_0$, $\beta_1$, $\beta_2$, ..., $\beta_p$ are the regression coefficients, and $\epsilon$ is the error term.

**Ridge** Ridge regression is a regularized linear regression technique used in regression analysis to prevent overfitting by adding a penalty term to the cost function. The effect of the penalty term is to shrink the regression coefficients towards zero, which can reduce the variance of the model and improve its generalization performance. The optimal value of $\lambda$ can be determined using cross-validation or other model selection techniques.

The ridge regression model involves adding a penalty term to the cost function that is proportional to the sum of the squares of the regression coefficients:

$$\text{minimize} \left[ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} \beta_j^2 \right]$$

where $\lambda$ is the regularization parameter that controls the strength of the penalty term.

**Lasso** Lasso (Least Absolute Shrinkage and Selection Operator) regression is a regularized linear regression technique used in regression analysis to prevent overfitting by adding a penalty term to the cost function. The effect of the penalty term is to shrink some of the regression coefficients towards zero, effectively performing feature selection by setting some of the coefficients to exactly zero.

The lasso regression model involves adding a penalty term to the cost function that is proportional to the sum of the absolute values of the regression coefficients:

$$\text{minimize} \left[ \sum_{i=1}^{n} (y_i - \beta_0 - \sum_{j=1}^{p} \beta_j x_{ij})^2 + \lambda \sum_{j=1}^{p} |\beta_j| \right]$$

where $\lambda$ is the regularization parameter that controls the strength of the penalty term.

### 4.3.3 Time Series Model

**ARIMA** ARIMA (Autoregressive Integrated Moving Average) is a combination of autoregressive (AR) and moving average (MA) models with an added differencing step (I) to account for non-stationarity in the time series. The AR component of the model captures the linear dependence of the time series on its past values, while the MA component captures the linear dependence on past error terms. The differencing step is used to remove any trend or seasonality in the time series and make it stationary.

$$(1 - \sum_{i=1}^{p} \phi_i L^i)(1 - L)^d y_t = (1 + \sum_{i=1}^{q} \theta_i L^i)\epsilon_t$$

where $y_t$ is the value of the time series at time $t$, $L$ is the lag operator, $\phi_i$ and $\theta_i$ are the AR and MA coefficients, $p$ and $q$ are the orders of the AR and MA components respectively, $d$ is the order of differencing, and $\epsilon_t$ is the error term.

## 4.4 Hyperparameter Tuning

We have chosen RandomizedSearchCV to choose the optimal parameters for each model. RandomizedSearchCV replaces the exhaustive enumeration of all combinations of parameters by selecting them randomly. RandomizedSearchCV can help to increase the chances of finding global optima by randomly sampling across the hyperparameter space, compared to local optima that might be found through exhaustive grid search. We chose 10 iterations while using 3-Fold cross validation, totalling 30 fits for each model run on each exchange.

For the deep learning models, we choose to run a manual hyperparameter optimisation approach. Since the computational resources required for executing even one epoch was great, an optimisation of only 3 important parameters of the deep learning model was done; namely batch size, density units and drop out rate.

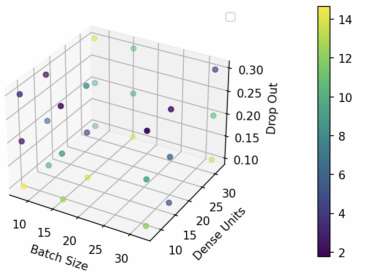


Figure 6: Hyperparamter Optimisation of Deep Learning Algorithms

# 5 Result

### 5.0.1 Root Mean Squared Error and R-Squared

| | Model_Type | RMSE_bitstampUSD | RMSE_korbitKRW | RMSE_btcmarketsAUD | RMSE_krakenCAD | RMSE_bcEUR | RMSE_bit2cILS | RMSE_btcboxJPY | Total_RMSE |
|---|---|---|---|---|---|---|---|---|---|
| 0 | XGBOOST | 1604.865754 | 1426.071271 | 1786.878828 | 1666.353047 | 1981.822197 | 1789.696391 | 1486.713305 | 11742.400792 |
| 1 | Random Forest | 3945.352825 | 4992.313033 | 4427.159617 | 3807.133711 | 4596.305898 | 6056.154328 | 3861.135924 | 31685.555335 |
| 2 | Decision Tree | 7532.495712 | 2938.281966 | 3373.342206 | 2569.060561 | 2947.764434 | 3981.456032 | 2244.305102 | 25586.706013 |
| 3 | Bagged Tree | 4211.883173 | 5644.254359 | 3763.147719 | 4250.077676 | 4638.628050 | 4393.146483 | 4068.454734 | 30969.592195 |
| 4 | Gradient Boosting | 1509.835039 | 1473.774466 | 1826.603708 | 2060.705056 | 2168.421471 | 2140.762616 | 1871.618872 | 13051.721228 |
| 5 | Sequential-Deep Learning | 2655.159103 | 2892.113178 | 9065.818177 | 1265.382890 | 6280.891692 | 1941.641257 | 7666.278487 | 31767.284784 |
| 6 | Forward Stepwise OLS | 885.327897 | 802.418740 | 837.890389 | 868.454366 | 897.651295 | 930.532256 | 909.777319 | 6132.052263 |
| 7 | Ridge | 900.782361 | 810.918567 | 858.820757 | 883.118540 | 909.219480 | 940.901735 | 906.731371 | 6210.492810 |
| 8 | Lasso | 934.313015 | 857.876574 | 892.918630 | 919.707873 | 954.719246 | 964.685943 | 936.026162 | 6460.247441 |
| 9 | Time Series | 2107.992184 | 4390.354114 | 2233.475439 | 2839.807533 | 1824.338388 | 2927.971504 | 3220.603678 | 19544.542840 |

Figure 7: Root Mean Squared Errors of Different Algorithms and Different Exchanges

| | Model_Type | bitstampUSD | korbitKRW | btcmarketsAUD | krakenCAD | bcEUR | bit2cILS | btcboxJPY | Average R-Squared |
|---|---|---|---|---|---|---|---|---|---|
| 0 | XGBOOST | 0.943924 | 0.947967 | 0.960474 | 0.962713 | 0.967585 | 0.960643 | 0.957157 | 0.957209 |
| 1 | Random Forest | 0.953359 | 0.933722 | 0.963068 | 0.95851 | 0.947217 | 0.947737 | 0.957453 | 0.951581 |
| 2 | Decision Tree | 0.76054 | 0.854066 | 0.886715 | 0.867085 | 0.892444 | 0.815104 | 0.848555 | 0.846359 |
| 3 | Bagged Tree | 0.865409 | 0.8993 | 0.863148 | 0.885814 | 0.887044 | 0.870263 | 0.880459 | 0.878777 |
| 4 | Gradient Boosting | 0.965406 | 0.93235 | 0.959161 | 0.923989 | 0.946629 | 0.925399 | 0.953669 | 0.943801 |
| 5 | Sequential Deep Learning | 0.91324 | 0.83701 | 0.67291 | 0.95813 | 0.81937 | 0.95381 | 0.82953 | 0.854857 |
| 6 | Forward Stepwise OLS | 0.992185 | 0.993721 | 0.99316 | 0.992486 | 0.991859 | 0.992019 | 0.992112 | 0.992506 |
| 7 | Ridge | 0.991865 | 0.99347 | 0.992887 | 0.992195 | 0.991471 | 0.99167 | 0.991818 | 0.992196 |
| 8 | Lasso | 0.991346 | 0.992867 | 0.992255 | 0.991605 | 0.990918 | 0.991456 | 0.991338 | 0.991683 |
| 9 | Time Series | 0.928472 | 0.93864 | 0.942873 | 0.91837 | 0.96272 | 0.91846 | 0.948272 | 0.936829 |

Figure 8: R-Squared Values of Different Algorithms and Different Exchanges

### 5.0.2 Feature Importances

Machine Learning models are often considered as black boxes but this time we wanted to execute a post-hoc explainability test using SHAP or the SHapely Additive Explanantion, a game theoretic approach which uses the classic Shapley values. From the feature importance graph we see that the Lag 1 mean was the most determinant factor in the prediction with high values of other lags such as Lag 5 mean and Lag 4 Mean also impacting the prediction of the models. To read the below Shapely graph, is that the absolute value of the SHAP values indicates how much of an impact the feature has on the prediction and the colour of the features indicates on which direction positive or negative the impact is. In the case of Lag 1 Mean, not suprisingly a low value of Lag 1 Mean would cause the prediction of the bitcoin price to swerve towards a low value too. This is in accordance with our intuition that the prices of bitcoins can be modelled by stochastic process and a Markovian process where the only information relevant for the prediction of the next state is the current state.

### 5.0.3 Trading Strategy

### 5.0.4 Intuition for the Performance of the Models

- **Machine learning:** Among the machine learning models, the ensemble learning model, XGBoost had the lowest RMSE. We postule that one of the reasons why XGBoost performs well in predicting

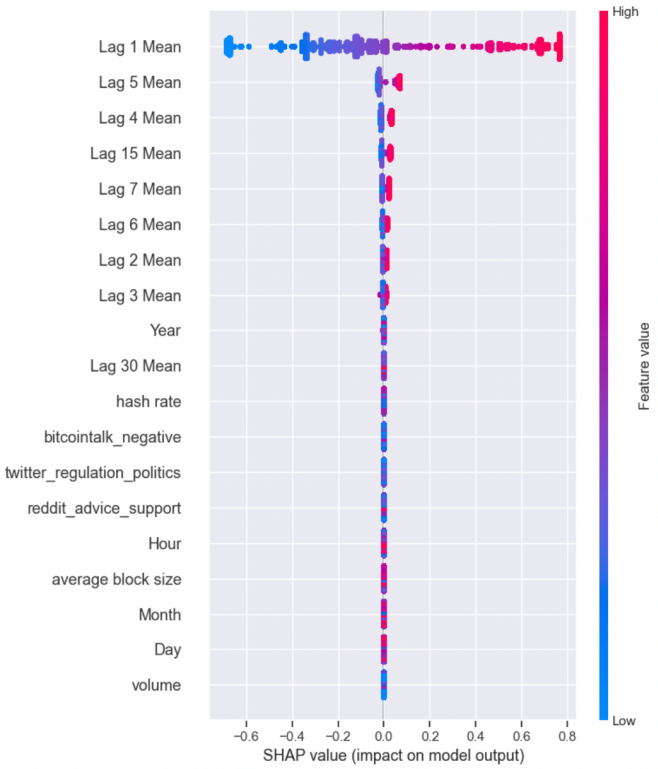


Figure 9: Feature Importances

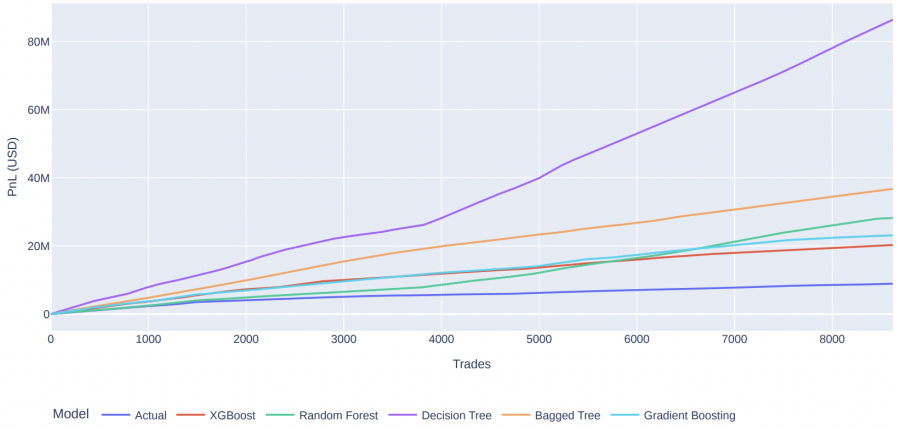


Figure 10: Actual vs Predicted Cumulative PnL (Non-Parametric)

Bitcoin prices is its ability to handle complex, non-linear relationships between the features and the target variable. Bitcoin price data used in the project can be affected by a wide range of factors, including market sentiment, historical data and market data. XGBoost can identify and incorporate these complex relationships into the prediction model, which can lead to more accurate predictions.

- **Deep Learning:** Deep learning algorithms do not perform too well on this dataset. As seen in BUSN 41204 lecture notes, the graph of model complexity versus the prediction error is a parabola and we should aim to balance between model complexity. The deep learning model was a little too complex and hence did not perform well.
- **Regression:** In the case of predicting Bitcoin prices, regression can be useful because it can help

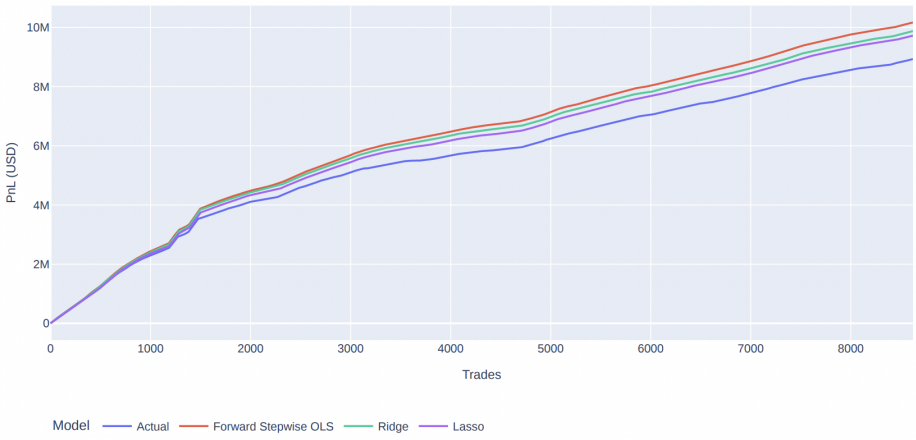


Figure 11: Actual vs Predicted Cumulative PnL (Parametric)


Figure 12: Actual vs Predicted Cumulative PnL (Time Series)

identify patterns and relationships between historical Bitcoin prices and other variables, such as trading volume, market sentiment, and other economic factors. The regression algorithms **Forward Stepwise OLS** and **Lasso Regression** performed the best.

- **Time Series Analysis:** One reason why time series analysis performs well in predicting Bitcoin prices is that it can capture the underlying patterns and trends in the data. For example, Bitcoin prices may exhibit seasonal or cyclic patterns that can be captured using time series analysis. Additionally, time series analysis can identify trends in the data, such as upward or downward trends in Bitcoin prices, which can be useful in predicting future prices.

## 5.1 Conclusion

Through the above project, we were able to train 7 machine learning models for the 7 exchanges with an average root-mean-squared-error of 2,212.5 and an average R-squared of 0.94. After deploying the models for each of the exchanges, we were able to capitalize on arbitrage trading opportunities due to the inefficiencies of the exchanges across the worlds which resulted in a realized profit of $8,951,693.79 across the span of 5 years.

### 5.2 Future Improvements

- To make the model applicable in real-world scenarios, the inclusion of real-time data through an API interface would be one of the improvements that could be made.
- One of the ways to improve the deep learning models would be to employ more sophisticated hyper-optimisation techniques. Due to the limited computational resources only a basic optimisation grid search was deployed in our code which did not give the highest results.

## 6 Reference

Makarov, Igor and Schoar, Antoinette, Trading and Arbitrage in Cryptocurrency Markets (April 30, 2018).