[Comparison Of Machine Learning Methods For Bitcoin Price Prediction]

Final Report



Information Technology Capstone Project

COMP5703

Group 23-2

**Members**

1. Quang Trung Nguyen (470518197)
2. Mingxuan Li (470325230)
3. Bin Liu (460130198)
4. Jun Xiong (470540420)
5. Jiaming Wei (460466754)

**Supervisor**: Raghavendra Chalapathy

**Abstract**

Bitcoin is a decentralised digital currency which was invented in 2009 and now has become the world’s most famous cryptocurrency with millions of daily transactions. In recent years, the price of Bitcoin has attracted growing amount of global attention from the public due to its frequent fluctuation. This project aims to train 14 predictive models using traditional machine learning, deep learning, and time-series approaches to predict the latest daily Bitcoin Price in 2018 and make a detailed comparison of all models in order to propose the best forecasting model. We collected a variety of features including Bitcoin trading data, currency exchange rates, commodity prices, stock market indexes, google trends, Blockchain technical data and Bitcoin price technical indicators, then relevant features are selected to train our models. We use daily data from March 2018 to June 2018 as training data and from July 2018 to September 2018 as test data. In addition, the project also attempts to build visualisations of predicted price movements as well as model comparison on a public static webpage hosted by Amazon Web Service.

**Table of Contents**

[1. Introduction 1](#_Toc530062818)

[2. Related Literature 1](#_Toc530062819)

[3. Project Problems 2](#_Toc530062820)

[3.1. Project Objectives 2](#_Toc530062821)

[3.2. Project Questions 2](#_Toc530062822)

[3.3. Project Scope 2](#_Toc530062823)

[4. Methodologies 3](#_Toc530062824)

[4.1. Exploratory Data Analysis 3](#_Toc530062825)

[4.2. Feature Engineering 3](#_Toc530062826)

[4.3. Day-Forward-Chaining Cross Validation 4](#_Toc530062827)

[4.4. Traditional Machine Learning Methods 4](#_Toc530062828)

[4.5. Deep Learning Methods 6](#_Toc530062829)

[4.6. Time-Series Method 7](#_Toc530062830)

[4.7. Visualisations 7](#_Toc530062831)

[5. Datasets 7](#_Toc530062832)

[6. Experiments & Results 9](#_Toc530062833)

[6.1. Exploratory Data Analysis 9](#_Toc530062834)

[6.2. Feature Engineering 11](#_Toc530062835)

[6.3. Lagging Predictions 12](#_Toc530062836)

[6.4. Model Evaluation 12](#_Toc530062837)

[7. Discussion 16](#_Toc530062838)

[7.1. Predictive Accuracy 17](#_Toc530062839)

[7.2. Forecast Bias 17](#_Toc530062840)

[7.3. Residual Autocorrelation 18](#_Toc530062841)

[7.4. Residual Distribution 18](#_Toc530062842)

[7.5. Conclusion 19](#_Toc530062843)

[8. Limitations and Future Works 20](#_Toc530062844)

[8.1. Limitations 20](#_Toc530062845)

[8.2. Future Works 20](#_Toc530062846)

[9. Project Resources & Outcomes 20](#_Toc530062847)

[9.1. Resources 20](#_Toc530062848)

[9.2. Project Deliverables 21](#_Toc530062849)

[10. Milestones & Risk Assessment 22](#_Toc530062850)

[10.1. Milestones 22](#_Toc530062851)

[10.2. Risk Assessment 24](#_Toc530062852)

[References 25](#_Toc530062853)

[Appendix 1 – Project Gannt Chart 28](#_Toc530062854)

# Introduction

Bitcoin is the most famous cryptocurrency invented by Satoshi Nakamoto, a purely peer-to-peer transaction system and network using digital signatures to timestamps the transactions with an ongoing chain of hash-based proof-of-work (Nakamoto, 2008). Since its creation, Bitcoin and other cryptocurrencies have been identified as economic bubbles or ponzi schemes by the public and its price fluctuates significantly by speculation, FUD (Fear, uncertainty and double) and FOMO (Fear of missing out). Since Bitcoin can be seen as an investment while its price keeps changing, it is meaningful to compare different forecasting methods to build an efficient and accurate model to predict future prices of Bitcoin. The potential beneficiaries are cryptocurrency investors and academic researchers. The investors can use our models to make investment choices and academic researchers can use our model comparison results to do further analysis.

# Related Literature

**Traditional Machine Learning Methods**

Several traditional machine learning models have been implemented successfully for Bitcoin price prediction over the years. In general, tree-based models such as Random Forest, Decision Tree and eXtreme Gradient Boosting (XGBoost) and regularised regression model like Elastic Net have been shown to outperform both simple linear regression model as well as time-series models (Guo & Antulov-Fantulin, 2018). In general, tree-based model performs best when predicting short-term Bitcoin price of within 10 days and XGBoost was able to achieve the lowest test error (Alessandrettia et al., 2018). Although in many studies, deep learning techniques such as Recurrent Neural Network and Bayesian Neural Networks outperform traditional machine learning models with lower test errors (Jang & Lee, 2018), they have higher computational costs to train the models which is a disadvantage. Since our training dataset is relatively small, that difference might not be significant. In 2014, Shah and Zhang (2014) applied Bayesian regression algorithms to predict Bitcoin price and was able to double their investment in 50 days. However, they used data from 2014 which has fewer fluctuations, so their model might not be as effective on today highly volatile data. That said, it would be interesting to test this model on today data and compare its performance with other machine learning models.

**Deep Learning Methods**

A feedforward network is the most basic deep learning architecture, but it has no conception of order in time (“A beginner’s Guide to LSTMs”, n.d.) as it only considers the current input it has been given to, hence it is not suitable for time-series data in our case. Comparing to feedforward network, the recurrent network (RNN) not only considers the current input but also inputs it has been given previously. In his paper, Chung et al. (2014)indicates that RNN is an extension of a conventional feedforward neural network, the activation function of RNN is in a recurrent hidden state that can handle sequence which dependents on that of previous time. RNN with Long Short-Term Memory (LSTM) units works well on sequence-based tasks with long-term dependencies, according to a research blog on understanding LSTM networks (Olah, 2015). Traditional neural networks cannot perform like humans because it is unclear how to use its reasoning about previous works. However, RNN addresses this issue because they have loops in the network that allows information to be persisted throughout.LSTM has been applied in several studies for predicting Bitcoin price and it has shown to achieve very good test errors. One study comparing its performance to RNN using Gated recurrent units (GRU) shows that single-layered GRU outperforms single-layered LSTM in prediction accuracy, however adding more layers to LSTM would improve its performance (Bobriakov, 2018).

**Time-Series Methods**

In time-series analysis, ARIMA - AutoRegressive Integrated Moving Average (Asteriou & Hall, 2011) is selected to fit to the dataset to better understand the data or to predict future points for forecasting. This method is proposed by Box and Jenkins in the early 1970s, ARIMA models aim to describe the correlations in the data with each other, it can also take into account the seasonality of dataset which is highly suitable to Bitcoin price data.

**Visualisation**

D3.js is an open source tool for visualisation where D3 means Data-Driven Documents. It is a JavaScript library that provides interactive data visualization displayed on the internet via web browsers. (Jain, 2014) Most data visualisation tools are just drag-and-drop type of software. The difference of D3.js is that it requires coding. Therefore, the development time using D3.js is longer than using other data visualisation tools. The input data format of D3.js can be csv, JSON and GeoJSON. In D3.js, data can be bound to a Document Object Model (DOM). On this document, data manipulations and data-driven transformations can be performed to create visualisation (Lekha et al., 2016). This is a powerful and flexible tool for us to do visualisation for this project.

# Project Problems

## Project Objectives

The project aims to (1) implement 14 machine learning methods to predict Bitcoin price and compare them based on prediction accuracy and residual analysis, (2) develop a static webpage that visualizes the prediction results and performance comparison of these methods. The project costed $0 to implement and was completed within 15 weeks from 8 August 2018 to 16 November 2018.

## Project Questions

The project addresses the following questions:

* What is the most efficient and accurate machine learning method for predicting Bitcoin price?

To answer this question, we train 14 predictive models based on different methods, using daily Bitcoin trading data, and compare their performance using different criteria.

* Should all Bitcoin price data from 2011 be used as training data to predict the latest Bitcoin price in 2018?

We answer this question by performing stationarity analysis on Bitcoin price time-series to evaluate if stationarity exists in its time-series of various time periods.

## Project Scope

**Part 1 - Modelling**: We implement and compare the results of the following 14 models to predict latest daily Bitcoin price in 2018, using data from 2011 to 2018.

* Traditional Machine Learning: (1) OLS Linear Regression, (2) Ridge Regression, (3) Lasso Regression, (4) Elastic Net, (5) Bayesian Regression, (6) K-Nearest Neighbours, (7) Support Vector Regression, (8) Decision Tree, (9) XGBoost, (10) Extra Trees
* Deep Learning: (11) Multilayer Perceptron Network – MLP, Recurrent Neural Networks: (12) Long Short-Term Memory - LSTM, (13) Gated Recurrent Unit - GRU.
* Time-Series: (14) ARIMA

**Part 2 - Visualisation:** We develop interactive visualisations of the following and deploy them on a static webpage:

* Predicted Bitcoin price movements from 14 models.
* Predictive accuracy bar chart of 14 models
* Mean Forecast Error (Bias) bar chart of 14 models
* Residual Quantile-Quantile plots of 14 models
* Residual Autocorrelation plots of 14 models

# Methodologies

In this section, we describe in detail our chosen methods for data analysis, feature engineering, time-series cross validation and 14 approaches for Bitcoin price prediction divided into three categories: traditional machine learning, deep learning, and time-series. We also explain the visualisation frameworks for developing the webpage.

## Exploratory Data Analysis

The main purpose of data analysis in our project is to study the characteristics of our dataset to understand its underlying structure and also to clean and prepare it for modelling.  First, we handle any missing and null values in the data. Next, due to its time-series nature, we examine if our data is stationary and is affected by trend or seasonality by performing stationarity analysis. We then visualise the correlation between all variables to identify any non-linear relationships and multicollinearity, and to discover how correlated each feature is to Bitcoin price. Since our features are highly diverse in scale, we rescale the features to a given range of 0-1 by using *sklearn.preprocessing.MinMaxScaler*. We use sklearn, pandas and matplotlib to perform data processing and analysis on our dataset.

## Feature Engineering

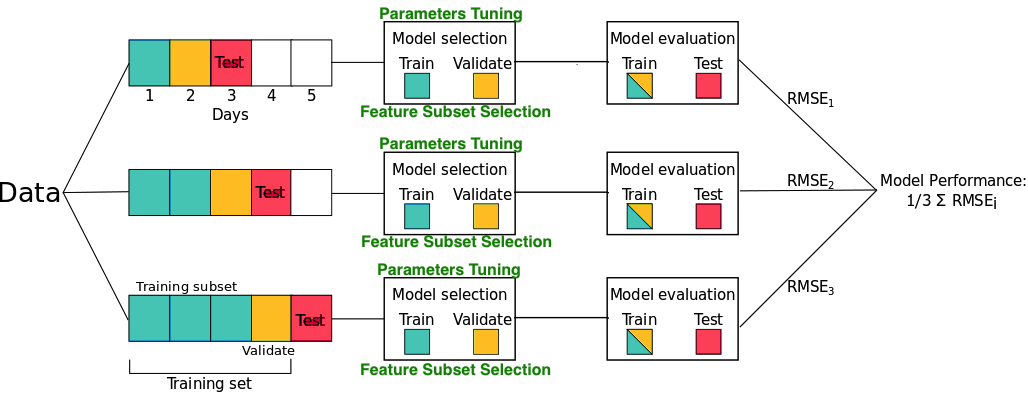
Feature Engineering can enhance the performance of predictive models as this process transforms raw data into meaningful and relevant features as inputs, and also it can prevent our models from overfitting. We divide our Feature Engineering into three main steps:

Step 1: Feature Generation. We use Python’s TA-lib library to generate five technical indicators from Bitcoin Price: Williams' %R, Moving Average Convergence/Divergence (MACD), MACD signal, MACD history, Exponential Moving Average.

Step 2: Filter Feature Selection. We reduce the number of features in our original dataset by half using Elastic Net. The parameters of an Elastic Net model are fine-tuned on the training set and the coefficients of the final model determine the selected features. The reduced feature set is then used to train all models in step 3.

Step 3: Wrapper Feature Selection. For each model, we use backward feature selection to recursively select the best subset for each test sample using the corresponding training set.

## Day-Forward-Chaining Cross Validation



**Figure 1.** Day-Forward-Chaining Cross Validation

Day-Forward-Chaining is a cross validation method for time-series data in which the dataset is divided into many folds where each fold contains just one sample as the test set and allocate all previous samples to the training set, demonstrated in Figure 1. As the number of folds depends on the number of samples in the test set, using this method can be computationally expensive. However, it leads to more robust predictions as it optimises the model parameters and feature subset for each individual test sample. As we focus on optimising the predictive accuracy of our models, we use this method to train our models despite its computational costs. Since our test dataset has 90 samples, we divide our dataset into 90 folds.

## Traditional Machine Learning Methods

**4.1.1. Parametric Methods:**

1. **Ordinary Least Squares (OLS)**

Regression problem belongs to the supervised learning area. It is often used to predict a quantitative value, for example, housing price or weather temperature. Linear regression is a basic technique to solve regression problem in machine learning. It uses least square method define the cost function and the goal is to minimise this cost function to estimate the appropriate coefficients for the model. To minimise the cost function, there are two algorithms available: gradient descent and normal equation.

1. **Ridge Regression**

Linear regression is a basic technique to solve regression problems in machine learning, however it is prone to overfitting issue when dealing with datasets with many features such as ours (James et al., 2013). We use regularised regression methods to overcome overfitting and also because they can handle multicollinearity in data more effectively. Ridge regression works by minimising the impacts of irrelevant features to avoid overfitting. It modifies the cost function of linear regression by adding a regularisation term to control the trade-off between two different goals. The first goal is to fit the training data set well by minimising the residual error. The second goal is to minimise the penalty by shrinking the coefficients of irrelevant features. We need to fine-tune parameter alpha, the regularisation term, of this method.

1. **Lasso Regression**

Lasso regression is another regularised technique to avoid overfitting, however it does not just minimise the coefficients of irrelevant features like Ridge regression but sets them to zero to completely eliminate these features. Its regularisation term is different than that of Ridge regression as it uses the absolute value of the coefficient instead of squaring it, hence, Lasso regression performs feature selection. Similar to Ridge regression, we need to fine-tune the regularisation term alpha.

1. **Elastic Net**

This method can be regarded as the combination of Ridge and Lasso regression as it uses regularised terms of both methods. It modifies the cost function of linear regression by adding two regularisation terms of Ridge and Lasso regressions. We choose this method as it was shown to be quite robust to redundant features in data, hence it is less prone to overfitting issues. Apart from the two regularisation terms, we also need fine-tune another parameter called lambda which determines the weights of these two regularisation terms.

1. **Bayesian Regression**

Contrary to regular linear regression methods which give a single estimate of the model parameters, Bayesian Regression outputs a distribution of the estimate which helps us infer the uncertainty of our model. This method was successfully applied to predict short-term Bitcoin price quite well as it automatically discovers important patterns in the data (Shah & Zhan, 2014). This method assumes the probability distribution of the response variable is normal which takes the form: *y ~ N(βTX, σ2)* where the mean is the product of the inputs features and their coefficients. The following hyper-parameters need to be tuned: *alpha\_1, alpha\_2, lambda\_1, lambda\_2*.

**4.1.2. Parametric Methods:**

1. **K-Nearest Neighbours (KNN)**

Assigned a value *K* for the number of neighbours and a prediction feature input *x0*, KNN regression first finds the *K* training observations that are nearest to *x0*, these *K* observations belong in neighbourhood *N0* (James et al., 2013). Then, it predicts *f(x0)* by averaging all training responses in neighbourhood *N0*. There are two parameters that need to be specified and tuned: the number of neighbour observations *K* and a distance metric to calculate the distance between observations. The most basic distance metric is Euclidean but Mahalanobis has been proven to be more effective in practice.

1. **Support Vector Regression (SVR)**

SVR defines a linear regression model for predicting the response variables, but instead of using OLS method to estimate the model’s coefficients, it minimises a loss function that is the sum of basis functions on all residual errors and a regularisation term for the coefficients (James et al., 2013). The basis function is called an epsilon-insensitive error measure where it ignores and assigns zero to the any residual error less than the value of the epsilon. SVR is a non-linear method because it solves this minimisation problem by using non-linear kernels. There are two parameters that needs to be fine-tuned for SVR: the value of epsilon and the regularization term on the coefficients.

1. **Decision Tree**

Decision Tree regression works by first partitioning the feature space into a series of non-overlapping regions or rectangles, then a simple model is fitted in each region (James et al., 2013). It predicts the response variable *y* in each region by a constant - averaging the *y* values of all training samples belonging in that region. The partition is based on successive splits of the feature space such that the Residual Sum of Squares (RSS), defined as the sum of squared difference between the predicted and actual values, is minimised using a greedy algorithm called recursive binary splitting. The following parameters need to be tuned: maximum number of features and depth of the tree, minimum number of splits and leaf.

1. **Gradient Boosting Trees (XGBoost)**

XGBoost (eXtreme Gradient Boosting) is an implementation of Gradient Boosting Algorithm for regression and classification problems. For regression, it uses weak decision trees to first allocate input data to one of the leaves which contains a continuous score (Chen & Guestrin, 2016). After that, it continues to add new trees which are combined with previous trees such that a regularised loss function for the residual errors is minimised. This loss function can be minimised using a gradient descent algorithm (“How XGBoost Works”, n.d). Its advantages over other tree-based models is computational speed and superior model performance (Chen & He, 2015). Indeed, XGBoost was shown to be the best performing tree-based model in predicting Bitcoin Price. Moreover, XGBoost also generates feature importance automatically. The following parameters of XGBoost need to be tuned: learning rate, gamma, regression alpha, maximum tree depth, number of estimators, and booster type.

1. **Extremely Randomised Trees (Extra Trees)**

Extra Trees is a method that contains strongly randomised decision trees whose structures are independent on the response values of training samples (Geurt, Ernst & Wehenkel, 2006). It makes prediction by averaging the predictions made by individual trees. Extra Trees has been proven to be more accurate than the single decision tree method and is less computationally expensive than other ensemble methods like XGBoost. In addition, it can handle data with a large number of features very well. Extra Trees is a good method for our dataset as it has relatively many features compared to the number of samples. These parameters of Extra Trees need to be tuned: maximum number of features and tree depth, number of estimators and bootstrap.

## Deep Learning Methods

We use Long Short-Term Memory (LSTM) and Gated Recurrent Unit (GRU) architectures of the recurrent neural network (RNN) to demonstrate the prediction of the Bitcoin price by neural networks. We choose RNN as it has been shown to work well with time-series data and has been previously applied successfully to predict Bitcoin price. We use Keras for training the model and fine-tune the following hyper-parameters: different number of layers and units, activation functions, optimizers, epoch and batch size. We then compare their performance with a simple Multilayer Perceptron model.

1. **LSTM & GRU**

LSTM is a variation of RNN. LSTMs can preserve the error that can be back-propagated through time and layers, which allows current nets to continue to learn over and over again (“A beginner’s Guide to LSTMs”, n.d.). A report from Chung et al. (2014)indicates that GRU will make each recurrent unit to adaptively capture dependencies of different time scales. Similar to a LSTM unit, GRU has gating units that regulate information flow inside the unit without needing separate memory cells.

## Time-Series Method

The ARIMA models can be estimated as Box-Jenkins Approach (Makridakis & Hibon, 1997) which is a systematic method of identifying, fitting, checking and using integrated autoregressive, moving average time-series models. A time-series is a set of values observed sequentially through time. The series maybe denoted by *X1*, *X2*…*Xt* , where *t* refers to the time period and *X* refers to the value. If the *X*’s are exactly determined by a mathematical formula, the series is said to be deterministic. If future values can be described only by their probability distribution, the series is said to be a statisticalor stochasticprocess. The identification phase determines the values of *d* (differencing), *p* (autoregressive order), and *q* (moving average order). We estimate them by calculating the autocorrelation plots.

## Visualisations

In order to visualise the results from different models we choose three frameworks for the visualisations, we use *Javascript’s D3.js* library to plot the actual prices and predicted prices from each model; for the residual autocorrelation and quantile plot we chose *Python’s plotly* library; we also use *Tableau* for plotting the predicting accuracy and Mean Forecast Error. There are plenty of examples from *D3js.org* and *plotly* to create an interactive SVG bar chart with smooth transitions and interaction so we use it to plot the price predictions against the time series; *Tableau* gives nice bar charts, so we use it to plot the comparison of the error rates from different models.

# Datasets

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Features** | **Description** | **Value Type** |
|  | date | Date | Datetime |
| **Bitcoin Trading Data** | | | |
|  | open | Bitcoin daily opening price (USD) | Float |
|  | high | Bitcoin daily highest price (USD) | Float |
|  | low | Bitcoin daily lowest price (USD) | Float |
|  | close | Bitcoin daily closing price (USD) | Float |
|  | volatility | Bitcoin Volatility Index | Float |
|  | volume | Amount of Bitcoin exchanged (USD) | Float |
|  | marketcap | Bitcoin total market value (USD) | Float |
| **Sentiment Data** | | | |
|  | google\_trends | ‘Bitcoin’ Google search interest measure | Float |
| **Commodity Prices** | | | |
|  | gold | Gold price (USD) | Float |
|  | silver | Silver price (USD) | Float |
|  | platinum | Platinum price (USD) | Float |
|  | palladium | Palladium price (USD) | Float |
|  | oil | Oil price (USD) | Float |
| **Major Currency Exchange Rates** | | | |
|  | usd\_eur | USD to EUR exchange rate | Float |
|  | usd\_jpy | USD to JPY exchange rate | Float |
|  | usd\_gbp | USD to GBP exchange rate | Float |
|  | usd\_cny | USD to CNY exchange rate | Float |
|  | usd\_chf | USD to CHF exchange rate | Float |
| **Major Stock Market Indexes** | | | |
|  | SP500 | U.S. stock market index | Float |
|  | Dow\_Jones | U.S. Industrial stock market index | Float |
|  | Nasdaq | U.S. stock market index | Float |
|  | Russell\_2000 | U.S. stock market index | Float |
|  | FTSE\_100 | London Stock Exchange index | Float |
|  | Nikkei | Tokyo Stock Exchange index | Float |
|  | SSE | Shanghai Stock Exchange index | Float |
|  | Eurostoxx | Eurozone Stock index | Float |
|  | VIX | Market Volatility index | Float |
|  | NVDA | Nvidia Corporation stock index | Float |
|  | GOOG | Alphabet Inc. (Google) stock index | Float |
| **Blockchain Technical Data** | | | |
|  | num\_of\_TXN | Total number of Bitcoin transactions | Int |
|  | TXN\_per\_block | Number of Bitcoin transactions per block | Int |
|  | est\_TXN\_vol | Total estimated volume of transactions on the Bitcoin blockchain (USD) | Float |
|  | cost\_per\_TXN | Cost per transaction = miners\_revenue / number of transactions (USD) | Float |
|  | total\_TXN\_fees | Total transaction fees miners earn (USD) | Float |
|  | usd\_trade\_vol | Trade volume from the top exchanges (USD) | Float |
|  | hash\_rate | Measure of a Bitcoin miner's performance | Float |
|  | avg\_block\_size | Bitcoin average block size (MB) | Float |
|  | difficulty | Measure of how difficult to find a new block | Float |
|  | num\_unique\_addr | Number of unique addresses used on the Bitcoin blockchain | Int |
|  | miners\_revenue | Total value of coinbase block rewards and transaction fees paid to miners | Float |
| **Bitcoin Price Technical Indicators** | | | |
|  | WilliamR14 | Williams' %R | Float |
|  | MACD | Moving Average Convergence/Divergence | Float |
|  | MACD\_signal | Moving Average Convergence/Divergence signal | Float |
|  | MACD\_hist | Moving Average Convergence/Divergence history | Float |
|  | EMA30 | Exponential Moving Average | Float |

**Table 1.** Dataset Feature Description

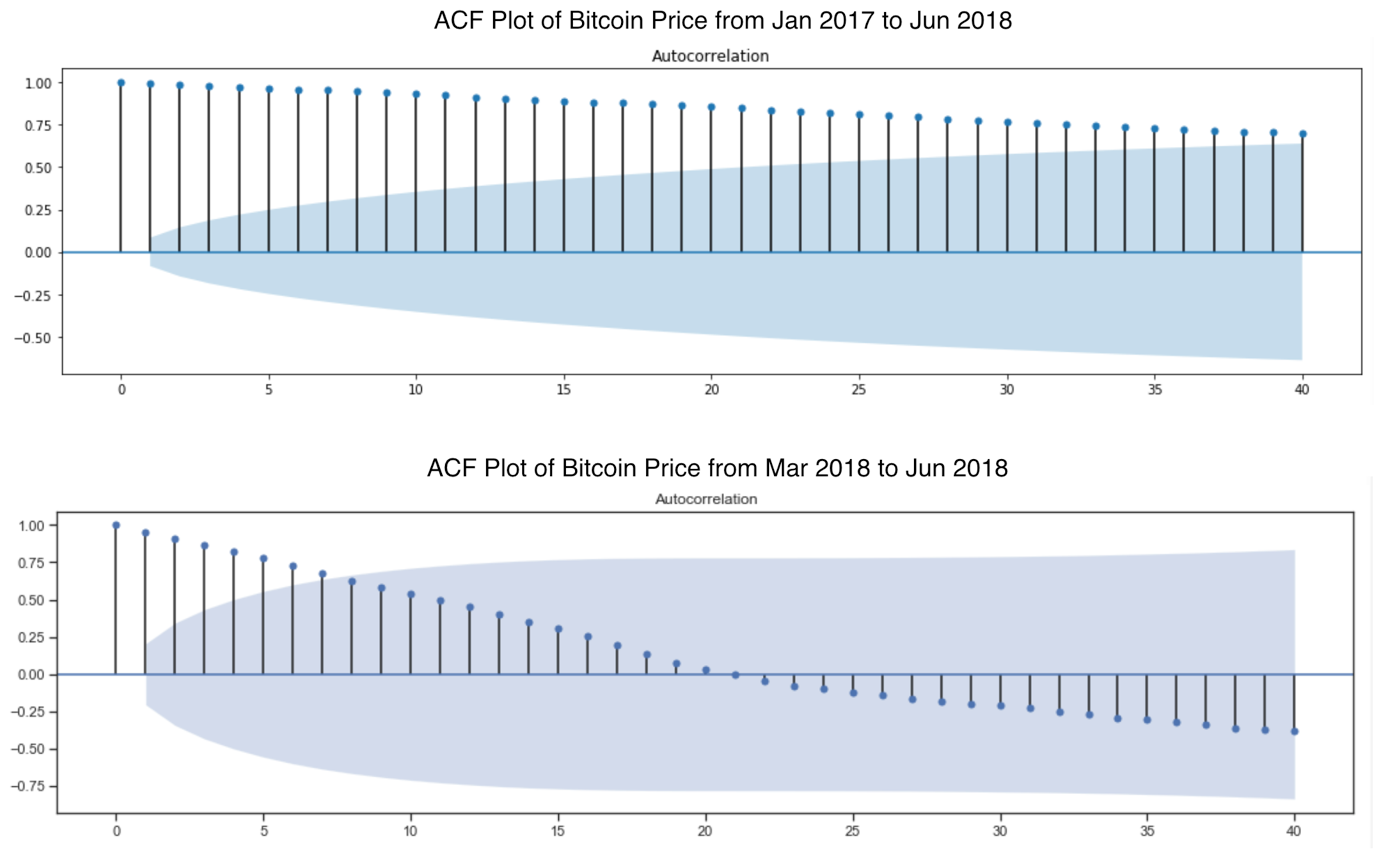
We collected historical daily data of 45 variables from 04 January 2011 to 30 September 2018. The variables, divided into seven main categories, are described in Table 1. Bitcoin close price (row 5) is chosen as our response variable to be predicted and other columns are treated as independent variables to be used as input features to train our models. We collected data from multiple sources and combined them into the following CSV datasets which can be found on our Github repository:

* **bitcoin\_train.csv** (546 rows, 46 columns) is our training set which contains data from 04 January 2011 to 30 June 2018.
* **bitcoin\_test.csv** (92 rows, 46 columns), is our test set which contains data from 01 July 2018 to 30 September 2018

We use Python package *cryptory,* available from *github.com/dashee87/cryptory,* which has built-in functions to collect Bitcoin trading data from *coinmarketcap.com*, google trends from *trends.google.com*, and commodity prices, currency exchange rates, stock indexes from *finance.yahoo.com*. Blockchain technical data can be downloaded from *www.quandl.com*. Bitcoin price technical indicators are generated using Python *TA-lib* library, available from *github.com/mrjbq7/ta-lib*.

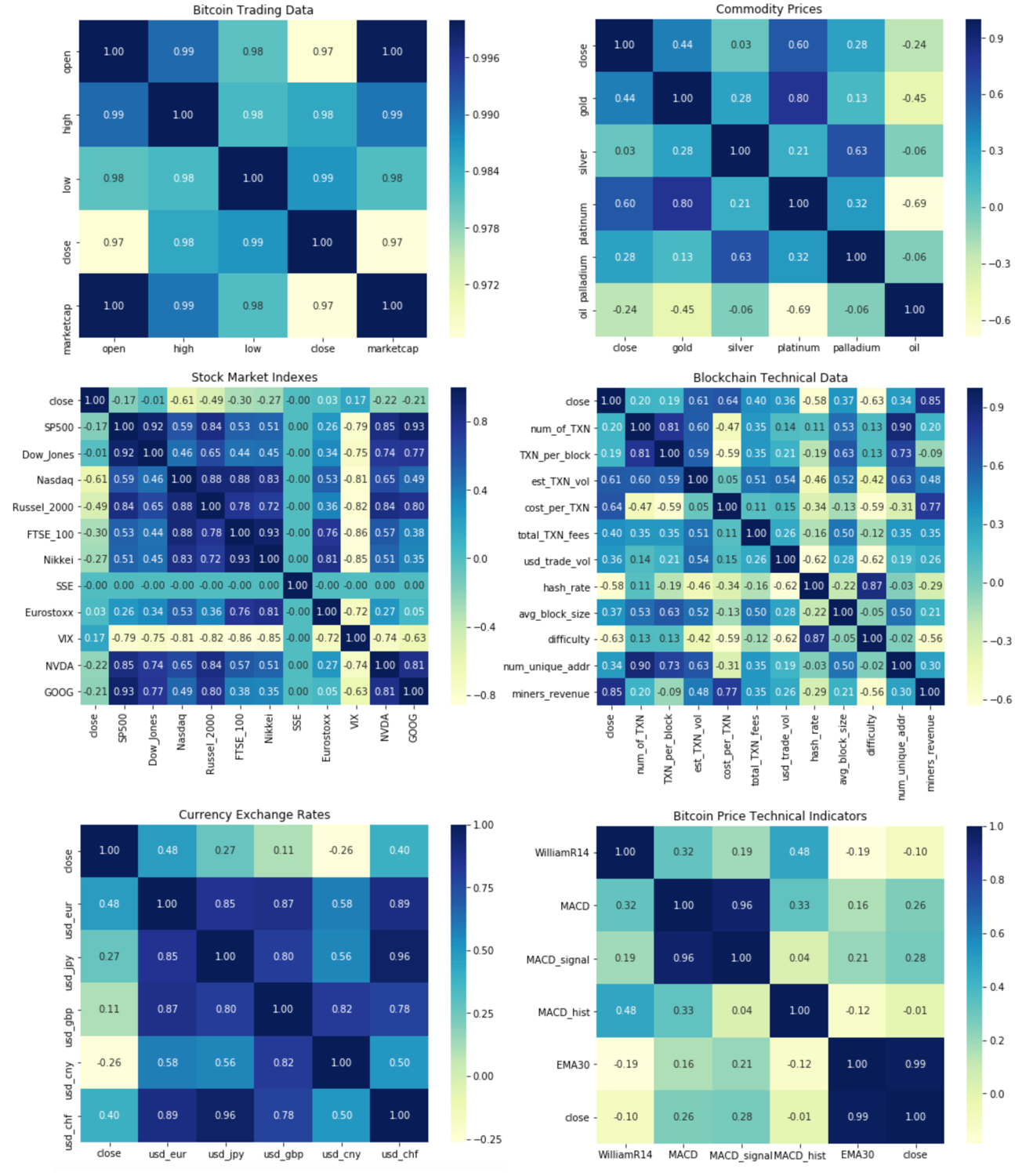
# Experiments & Results

## Exploratory Data Analysis



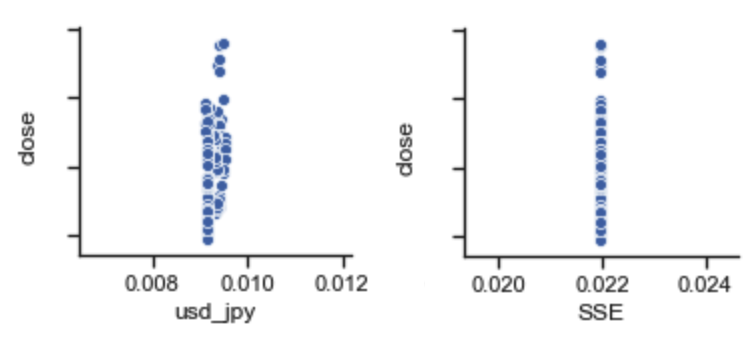
**Figure 2.** ACF Plots of Bitcoin Price

Since we collected data for our dataset manually using *Cryptory* package and web scraping, our dataset does not have any missing or null values. A stationary time-series is defined as having constant properties such as mean, variance and autocorrelation over time (Nau, 2018). It is necessary for time-series data to be stationary in order for predictive and forecasting models to learn these properties for future prediction. For instance, if the time-series is non-stationary and has increasing mean and variance over time, a forecasting model will underestimate these properties of future periods. To check the stationarity of a time-series, Autocorrelation Function (ACF) plot can be used. Using the training set containing data from January 2011 to June 2018, we perform stationarity analysis to check the stationarity of Bitcoin price with different starting points. Figure 2 shows the ACF plots of Bitcoin price data from January 2017 to June 2018 and from March 2018 to June 2018. The left plot shows a very slow decaying of the ACF which is staying well above the significant range represented by the shaded area. This indicates the time-series data is non-stationary. On the other hand, the right plot shows a faster decaying of the ACF and that it falls below the significant range quickly indicating some stationarity does exist in the data. Ideally, the ACF should fall below the significant range and also to zero very quickly and remains so. We discover that Bitcoin price has no stationarity due to the heavy fluctuations in late 2017, and that excluding 2017 data and only using data from March 2018 the stationarity slightly improves. Since without stationarity predictive models cannot learn the time-series data to make future predictions, we decide to use only the data from March 2018 to June 2018 as training data for our models.



**Figure 3.** Correlation matrices of features with Bitcoin close price

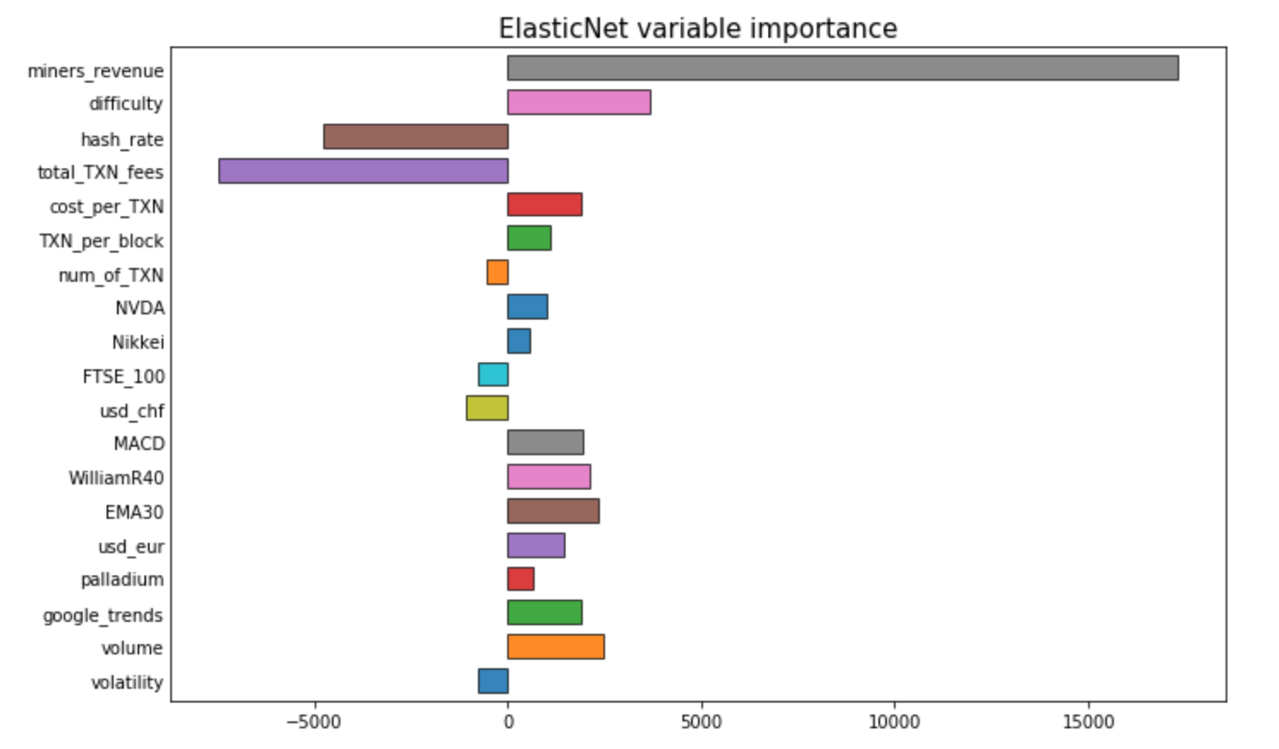
Next, we analyse the correlation of each feature with each other and with Bitcoin price (indicated as *close* on the matrix) using correlation matrices as shown in Figure 3. The correlation matrix of Bitcoin trading data group indicates that high, low, open prices, marketcap have almost perfect correlation with close price. Since we are predicting close price, we decide to remove these features to prevent multicollinearity. For commodity prices, platinum price has the highest correlation while silver price has almost no correlation with Bitcoin price. For currency exchange rates, all five selected currencies have relatively low correlation, below 0.5, with Bitcoin price. For stock market prices, Nasdaq is most correlated with Bitcoin price followed by Russell 2000. Overall, miners revenue has highest correlation with Bitcoin price (0.85), among not only internal Blockchain technical factors but among all features.



**Figure 4.** Scatter Plots of *‘SSE’* and ‘*usd\_jpy’* with Bitcoin close price

We use scatter plots to visualise the relationship between each feature with Bitcoin price and observe that features *SSE* and *usd\_jpy* have almost no relationship with price as shown in Figure 4. Hence, these two features are removed from our dataset. Lastly, since our features have vastly different scales, we use *MinMaxScaler* to scale all features into the same range of 0 to 1.

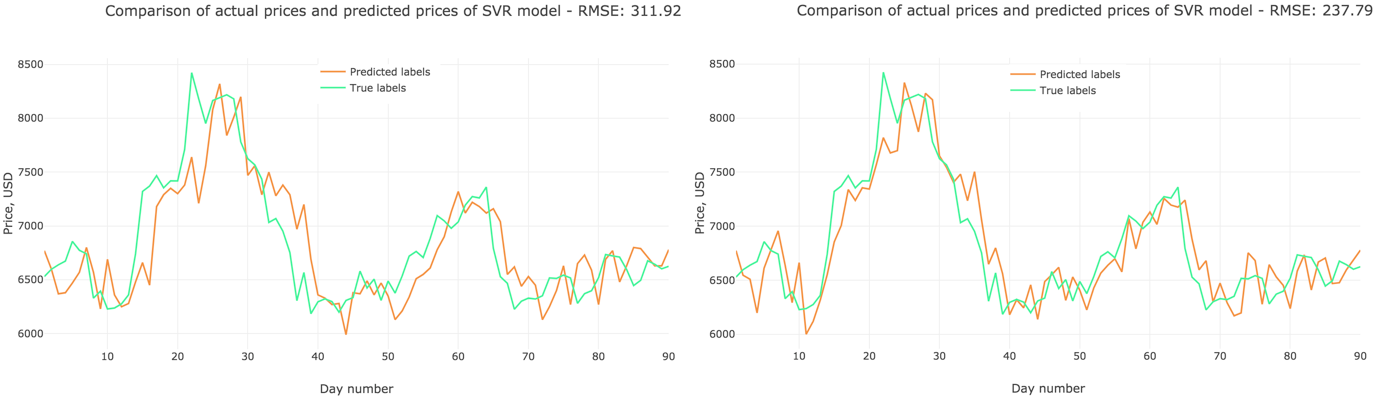
## Feature Engineering



**Figure 5.** Selected Features by Elastic Net

We use Elastic Net to perform filter feature selection on our dataset with the aim to reduce the number of features by half. Figure 5 shows the ranking of 19 features selected from the original batch of 40 features by Elastic Net on our training set. The advantage of using Elastic Net for feature selection is that it combines the regularisation terms from Lasso and Ridge for optimal performance, and also shows the direction of features importance by the sign of their coefficients. Miners revenue, total transaction fees (USD) and hash rate are shown to be the three most significant features. This result is similar to our correlation analysis as miners revenue is the most correlated feature with Bitcoin price. The positive coefficient of Miners revenue indicates it has a positive relationship with Bitcoin price while it is the opposite case for total transaction fees and hash rate. These 20 features are then applied to train our models.

## Lagging Predictions

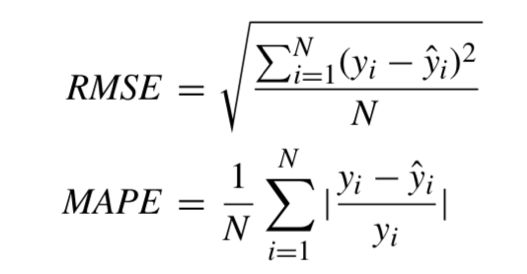


**Figure 6.** Lagging and Non-lagging predictions

Initially, we discover that our models give lagging predictions which means that for every day, the predicted price is highly similar to the actual price of the previous dat. This is demonstrated in the left plot of Figure 6 which shows the lagging predictions made by the SVR model. The predicted price line (in orange) looks like a lagged version of the actual price line (in green). This indicates that the model cannot learn any pattern from the training data and thus is unable to make future predictions. After performing stationarity and correlation analysis, we find that by removing multicollinear features from our dataset and excluding 2017 data to improve the stationarity of the training data, the lagging prediction can be significantly reduced for several models. The right plot of Figure 6 shows the predicted prices of our final SVR model. The predicted price line now matches the actual price line more closely compared to that of the left plot. We are able to reduce this issue for most models except for Extra Trees, KNN and the neural networks.

## Model Evaluation

Two metrics are used to measure the predictive accuracy of our models: Root Mean Square Error (RMSE) and Mean Absolute Error (MAE) which are expressed in the same unit as the response variable (USD). The definitions of these two metrics are as follows:



**Figure 7**. Root Mean Square Error and Mean Absolute Error (Jang & Lee, 2018)

where *N* is the number of samples, *y* is the actual value and *ŷ* is the predicted value. Since the RMSE squares the errors, it penalises large errors more heavily while the MAE treats all errors equally by using their absolute values.

**6.4.1. Traditional Machine Learning Models**

1. **Parametric Models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Models** | **Training RMSE** | **Test RMSE** | **Training MAE** | **Test MAE** |
| OLS | 387.76 | 396.5 | 269.44 | 286.1 |
| Ridge | 346.22 | **284.3** | 215.41 | **215.7** |
| Lasso | 365.5 | 309.8 | 225.97 | 244.5 |
| ElasticNet | 346.22 | **284.3** | 215.41 | **215.7** |
| Bayesian Ridge | 367.26 | 304.1 | 240.35 | 225.4 |

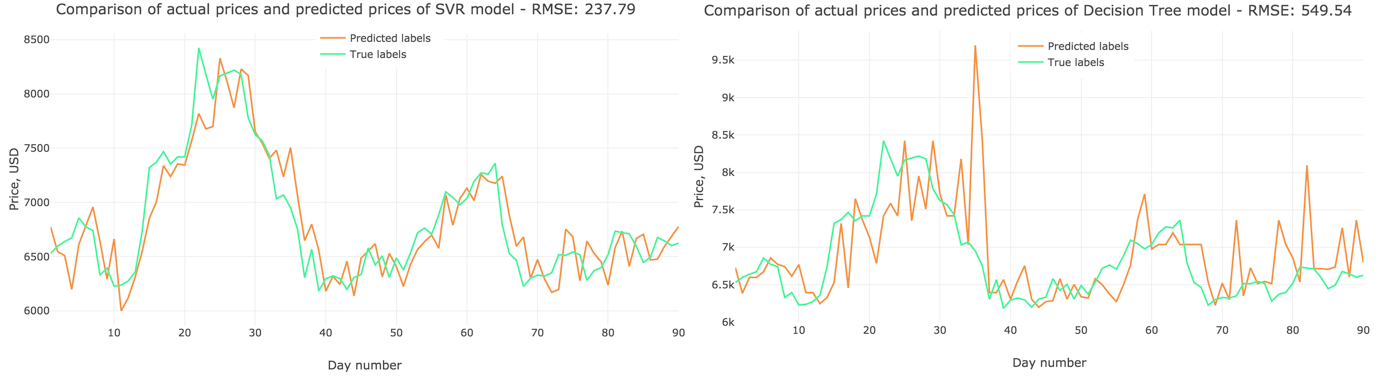
**Table 2.** Predictive Accuracy of Parametric Models

Table 2 summarises the training and test errors of our parametric linear regression models. Overall, the test errors of our models are lower than the training errors indicating that the models are not overfitting. Our Elastic Net and Ridge models have the lowest test errors for both metrics. The reason both models having the same errors is that our Elastic Net model selects only Ridge’s regularisation term, hence it is basically a Ridge model. Since Ridge does not eliminates irrelevant features like Lasso but only shrinks their coefficients, this indicates all features in our dataset have some degrees of importance with relation to Bitcoin price. As expected, our simple OLS model performs worse than regularised models.

1. **Non-parametric Models**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Models** | **Training RMSE** | **Test RMSE** | **Training MAE** | **Test MAE** |
| KNN | 382.21 | 340.6 | 254.25 | 251.8 |
| SVR | 376.28 | **237.8** | 211.71 | **190.4** |
| Decision Tree | 562.07 | 549.5 | 356.01 | 361.5 |
| XGBoost | 423.03 | 337.9 | 246.55 | 251.4 |
| Extra Trees | 398.67 | 288.1 | 220.9 | 222.7 |

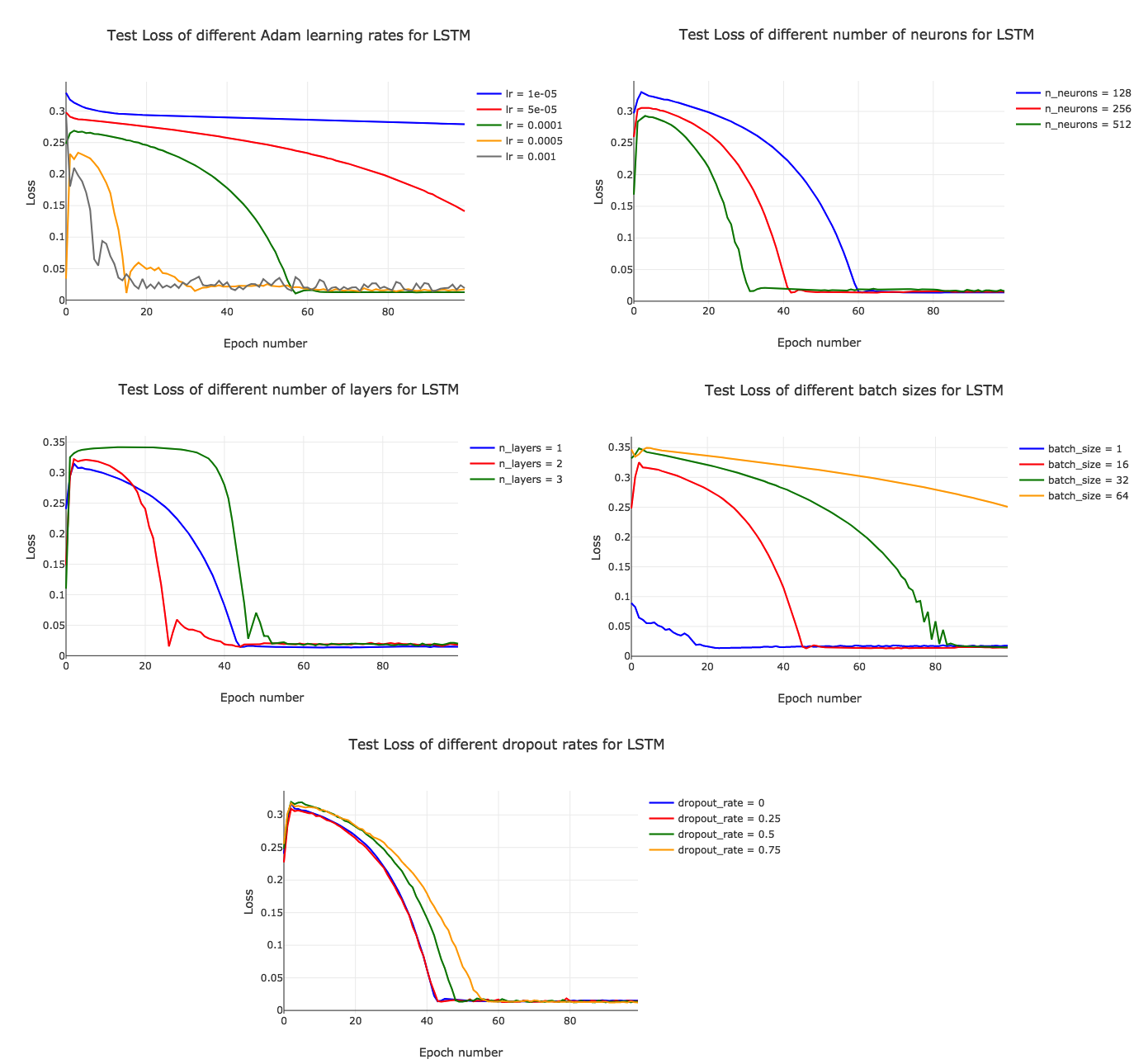
**Table 3.** Predictive Accuracy of Non-parametric Models



**Figure 8**. Prediction graphs of SVR and Decision Tree

For our non-parametric models, the results are summarised in Table 3. Our Support Vector Regression achieves the lowest test error for both metrics among our traditional machine learning models. Ensemble model like XGBoost and Extra Trees perform much better than the single Decision Tree model. Figure 8 shows the predictions made by our SVR and Decision Tree model. While SVR’s predictions match the actual prices closely, the predictions made by Decision Tree have several outliers contributing to its higher test errors.

**6.4.2. Deep Learning Models**



**Figure 9**. Parameters Tuning Results of LSTM

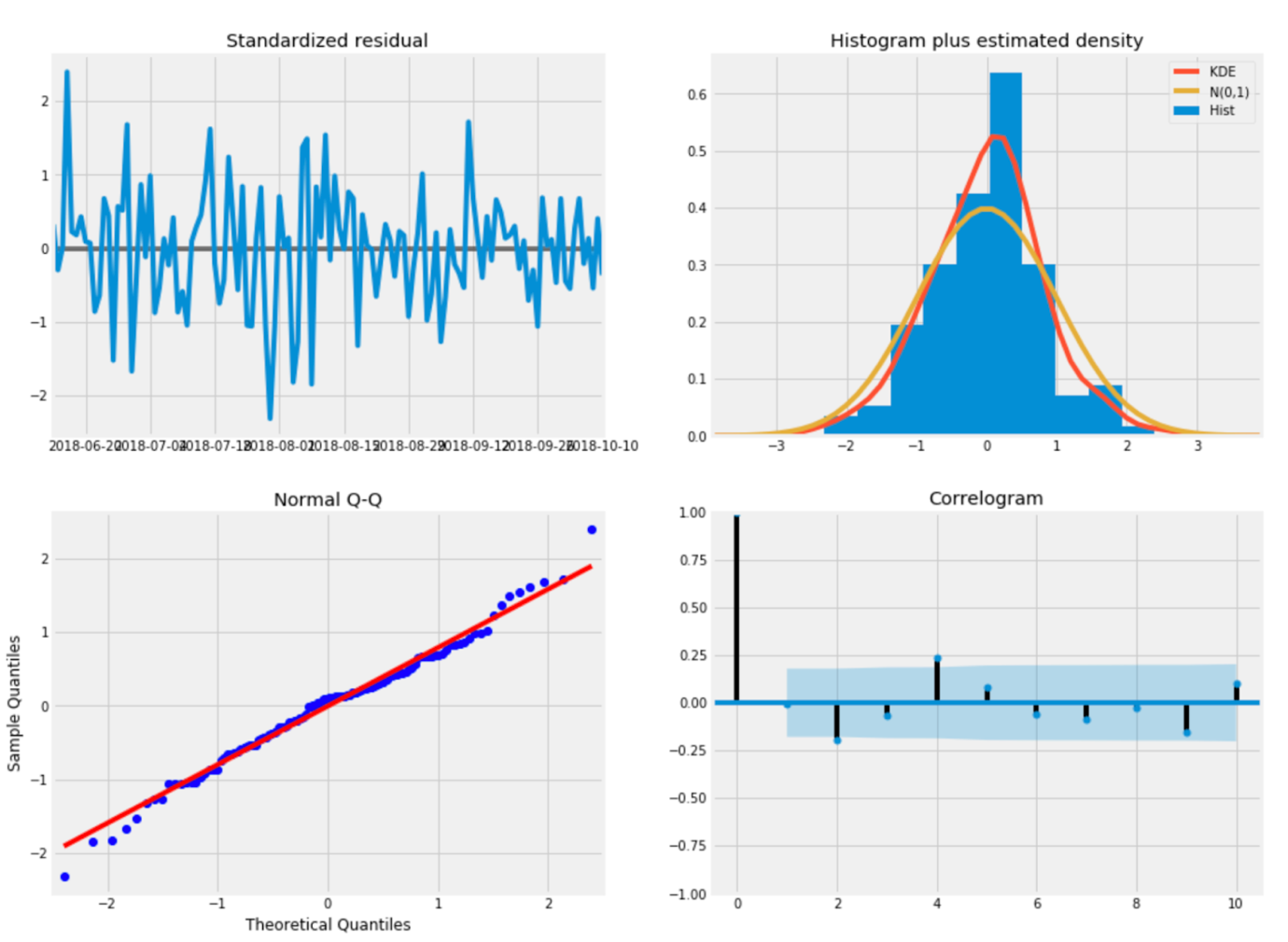
For our three neural networks, we fine-tune the following hyper-parameters progressively in which the best value for previous parameters are used to fine-tune the remaining ones: learning rate, optimizer function (Adam, RMSprop, SGD), number of neurons (128, 256, 512), number of layers (1, 2, 3), batch size (1, 16, 32, 64) and drop out rate (0, 0.25, 0.5, 0.75). Overall, Adam is the optimal optimizer function for our LSTM model while RMSprop performs better for GRU and MLP models. Figure 9 shows the validation loss of different parameters for our LSTM model. The loss in these plots does not have the same unit with the final error as it was plotted before *MinMaxScaler’s* reverse transformation. For learning rate, 0.0001 got the lowest loss converging after around 60 epochs while higher rates are unstable and lower rates take too long to converge. 256 is the best number of neurons and 1 is the optimal number of layer as more layers lead to unstable losses. While batch size 1 converges faster, batch size 16 is more stable and can reach lower losses in the long run after 50 epochs. For dropout, 0.75 is the optimal rate indicating that regularisation is important the model to prevent overfitting.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Models** | **Training RMSE** | **Test RMSE** | **Training MAE** | **Test MAE** |
| MLP | 500.17 | 337.0 | 268.96 | 265.3 |
| RNN-LSTM | 415.1 | **293.9** | 230.86 | **229.2** |
| RNN-GRU | 426.22 | 301.9 | 249.85 | 230.4 |

**Table 4.** Predictive Accuracy of Deep Learning Models

The training and test errors of three models are presented in Table 4. Our LSTM model got the lowest test errors, followed closely by GRU model and both outperform the simple MLP model. Notably, the training errors of the all three models are much higher than the test errors. This could be explained by the training set having too few samples compared to combined set with the test set and neural networks are unable to learn with not enough training samples.

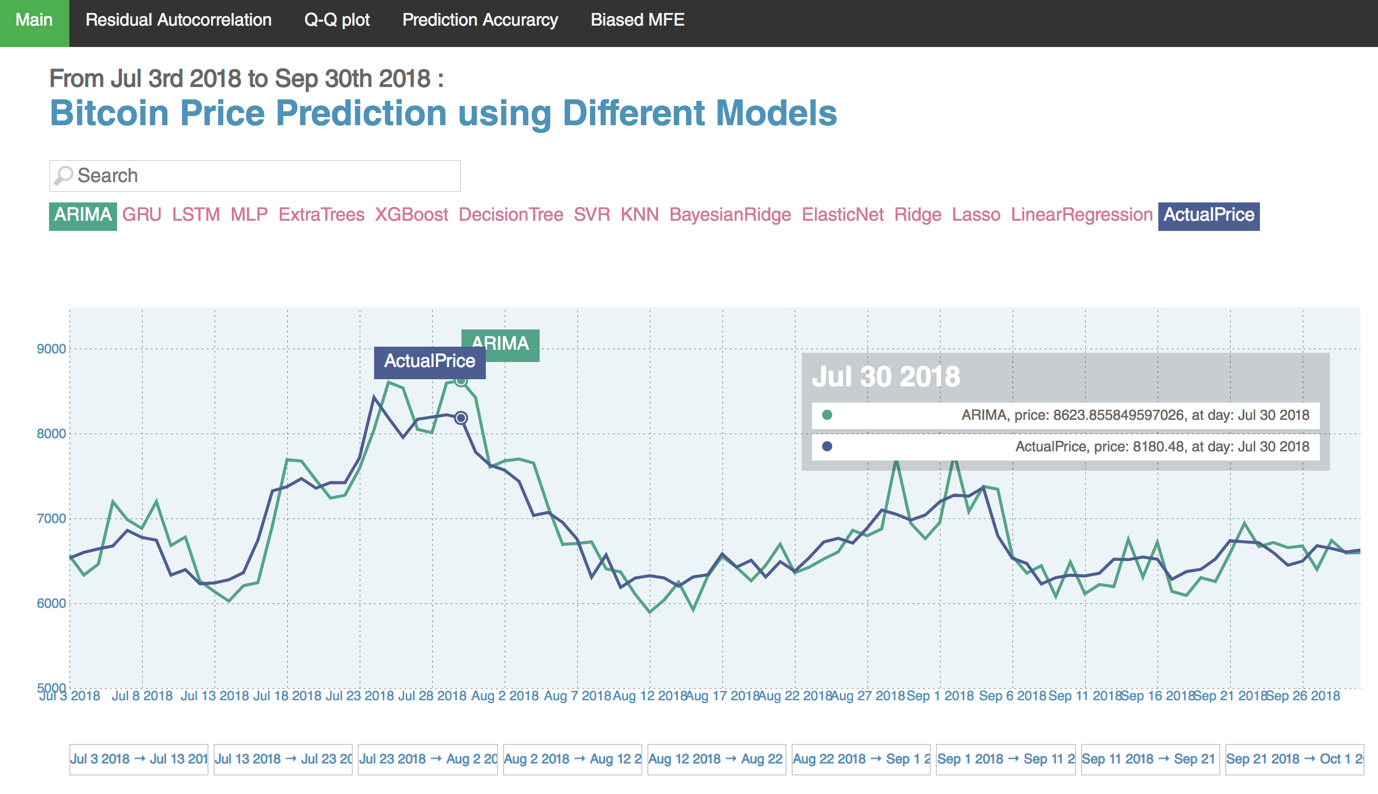
**6.4.3. Time-Series Models**



**Figure 10**: Residual Plots of ARIMA

After parameter tuning, we find the optimal (p,d,q) order of the model – the number of AR parameters, differences, and MA parameters – as: p=6, d=2, q=2. Where the seasonal components of order are pretty similar (p=6, d=2, q=1 for monthly data), we also plot the standardised residual and normal Q-Q diagrams for reference, as shown in Figure 10. The standard residual is a measure of the strength of the difference between the observed and expected values; normal Q-Q is to make the sample quantiles and theoretical quantiles identical; histogram pdf shows the seeds of error assessment between empirical density estimate and theoretical density estimate; correlogram is similar to the autocorrelation in the previous graph. The final ARIMA model achieved a test RMSE of 270.7 and MAE of 219.9.

**6.4.4. Visualisation Webpage**



**Figure 11**: Visualisation Main Page

Our visualisation webpage can be accessed through this URL [*https://s3-ap-southeast-2.amazonaws.com/capstone-bitcoin/prices/index.html*](https://s3-ap-southeast-2.amazonaws.com/capstone-bitcoin/prices/index.html). Figure 11 shows the layout of our main page. It displays an interactive visualisation of Bitcoin Price from 1 July 2018 to 30 September 2019 predicted by our models. This visualisation is built using *Javascript’s D3.js*. From the main page, four subpages for model comparison can be navigated to through the menu.

This is built using *Python’s plotly*:

* Residual Autocorrelation: Displays interactive visualisation of autocorrelation plots of all models’ residuals.
* Q-Q Plot: Displays Quantile-Quantile plots of all models’ residuals.

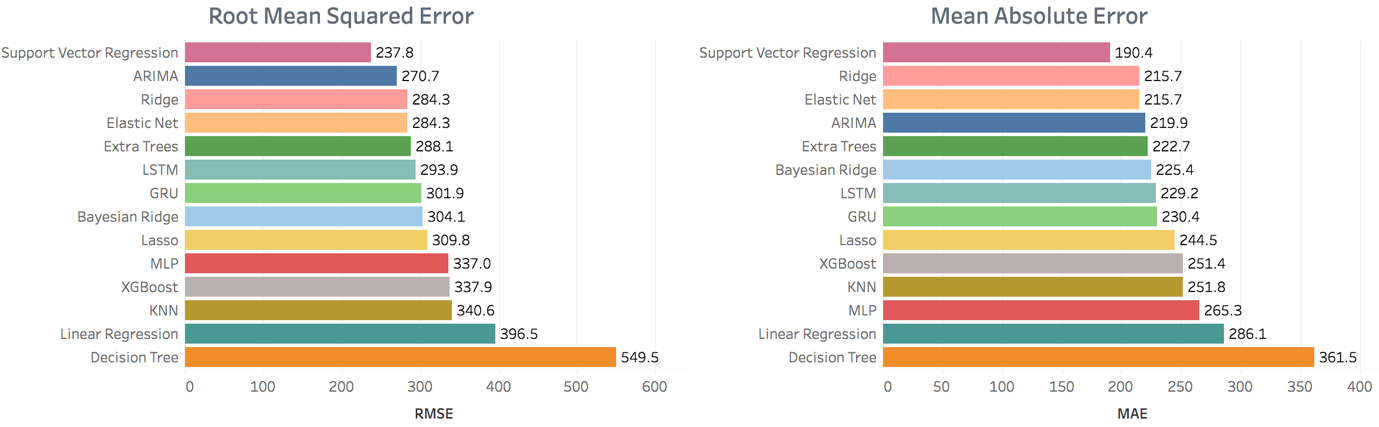
These visualisations are built using *Tableau*:

* Prediction Accuracy: Displays bar charts comparing the RMSE and MAE of 14 models.
* Biased MFE: Displays bar charts comparing the Mean Forecast Error of 14 models.

# Discussion

We compare our models based on four different criteria. The most important criteria is predictive accuracy measured by RMSE and MAE. We also examine if the model’s prediction is biased using Mean Forecast Error. For residual analysis, we plot the residual autocorrelation to check if the model gives lagging predictions and use Quantile-Quantile plot to check of the residual has a normal distribution.

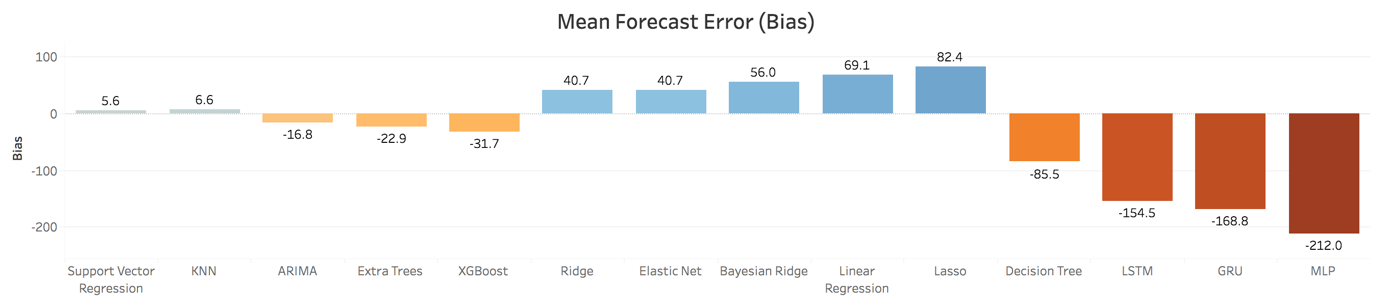
## Predictive Accuracy



**Figure 12**. RMSE and MAE

Figure 12 compares the predictive performance of our models using RMSE and MAE. Our SVR model achieves the lowest test errors for both metrics, followed by ARIMA, Ridge and Elastic Net in the top four. On the other hand, Linear Regression and Decision Tree got the highest errors while the latter performed significantly worse than other models which was contributed by several outliers in its predictions. Overall, we observe that both ensemble models, XGBoost and Extra Trees, perform better than the single Decision Tree as ensemble methods are less sensitive to outliers. For linear models, regularised regression models such as Elastic Net have better accuracy the simple OLS model indicating it is necessary for our models to either reduce the number of features as in Lasso or to shrink the coefficients of irrelevant features as in Ridge. For neural networks, our RNN models with LSTM and GRU perform better than the simple MLP model.

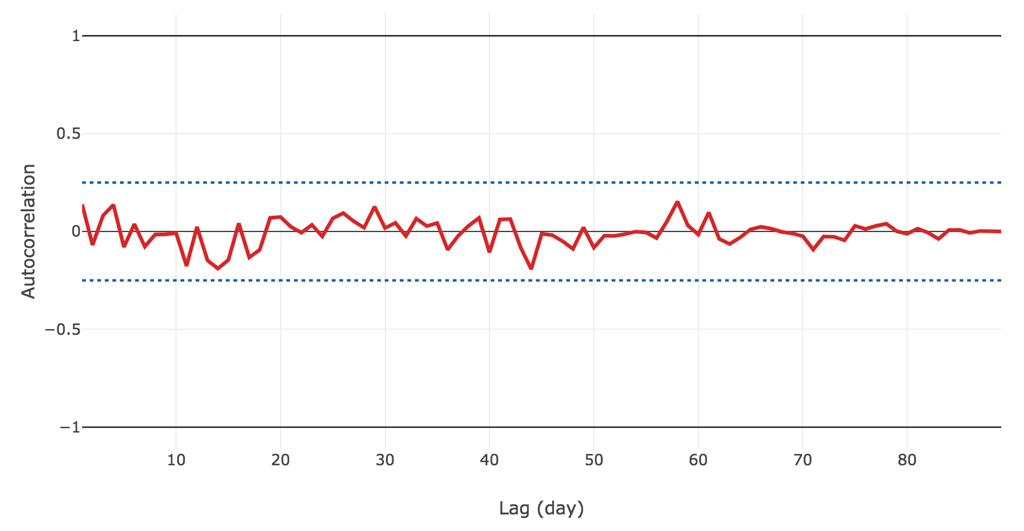
## Forecast Bias

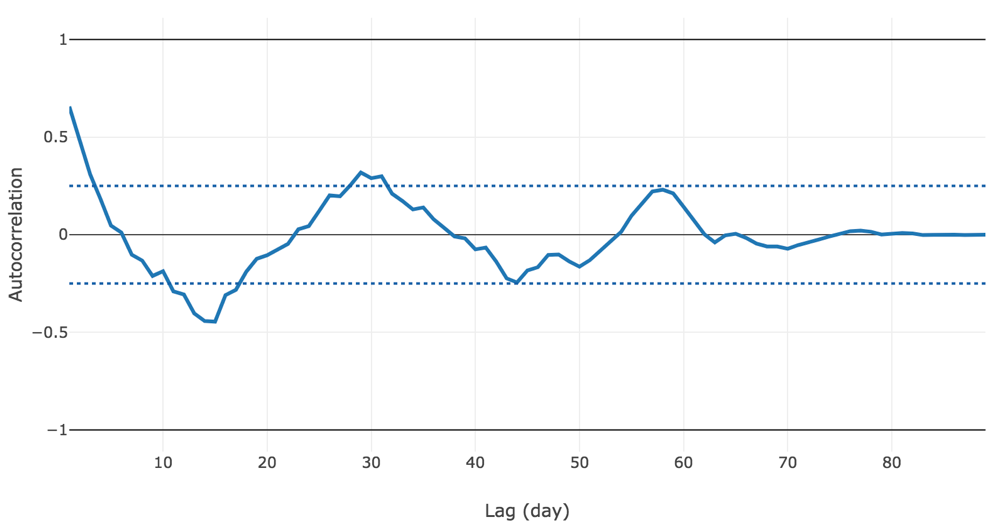


**Figure 13**. Mean Forecast Error (Bias)

The second criteria we chose to evaluate our model is Mean Forecast Error (bias), which is the average of all residual errors made by the models. A positive bias value indicates that the model tends to over-predict as it has more positive errors, while a negative bias indicates the model tends to under-predict due to a greater number of negative errors. Ideally, the bias value should be zero or very close to it which means the model’s prediction errors are not biased towards any direction. This is important especially in our case as severely over-predicting or under-predicting Bitcoin price can negatively affect investment returns. Figure 13 above compares the bias values of our models. Our SVR model got the best bias value that is very close to zero, followed closely by our KNN model. In contrast, our MLP model got a large negative bias which indicates it under-predicts the price most of the time.

## Residual Autocorrelation



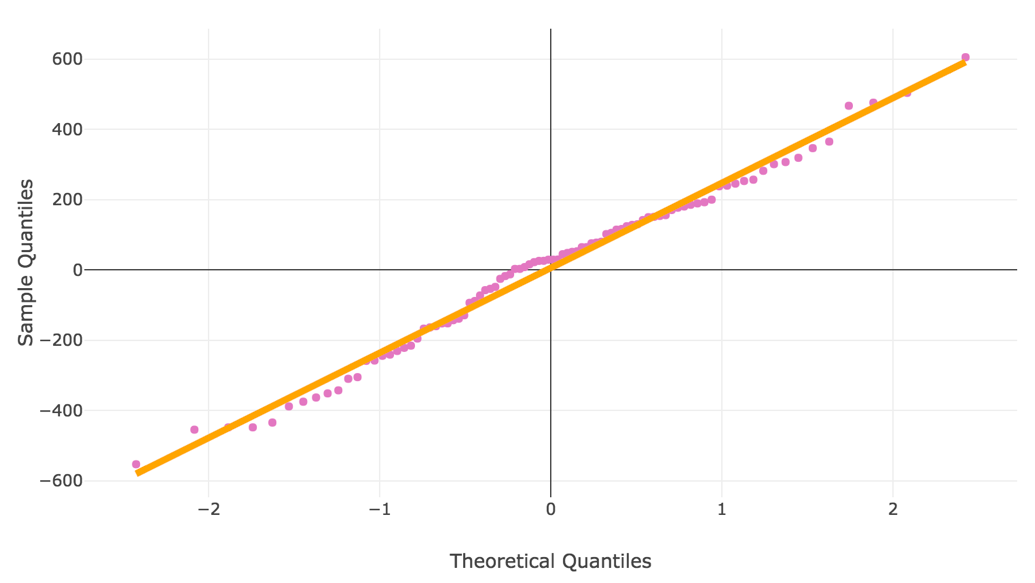


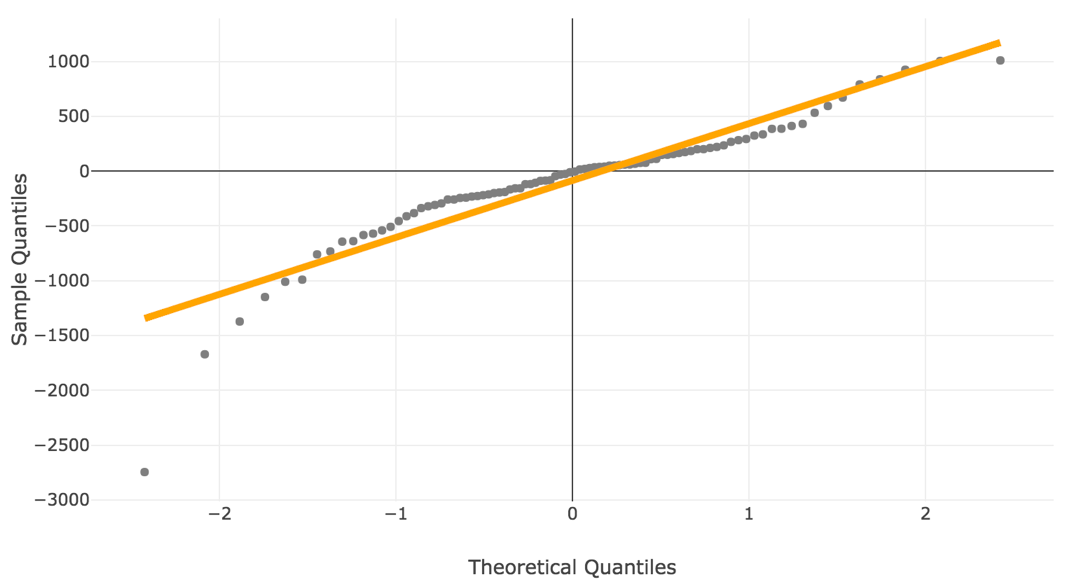
**Figure 14**. Residual Autocorrelation Plots

The residuals of a predictive model for time-series data should not have any correlation with each other as having it indicates the model is not learning. This can be shown by plotting autocorrelation of the residuals over time as shown in Figure 14 below. The autocorrelation plot for our ARIMA model shows that its autocorrelation scores, represented by the red line, for the whole time period of the data are within the significance thresholds (< 0.25 and > -0.25 represented by the two dashed lines), and that it has no clear pattern. On the other hand, the residual autocorrelation plot of our MLP model shows a cyclic pattern and that several autocorrelation points are outside the thresholds. This indicates that the model can be improved by including lagged observations – observation at a previous time step – as input features. Overall, most of our models have residual autocorrelation scores within the thresholds except for KNN, Extra Trees and the three neural networks.

## Residual Distribution

Ideally, the residuals should have a normal distribution which can be visually examined by a Quantile-Quantile plot. The scatter plot in Figure 15 compares the distribution of residuals made by our models (sample quantile) with a standard Gaussian distribution (theoretical quantile). The fitted line shows the ideal match between the two distribution and all points should be on or very close to it indicating the residuals are normally distributed. The goodness of fit is measured by a R-squared value with higher values meaning better fits. Our SVR model has an almost perfect R-squared of 0.99 indicating very good fit and that its residuals have a normal distribution. In contrast, our Decision Tree model has the lowest R-square of 0.88 among our models and a few outliers departing from the fitted line. Overall, models with better predictive accuracy have residuals distributed more closely to a Gaussian distribution.





**Figure 15**. Residual Quantile-Quantile Plots

## Conclusion

We discovered that the time-series of Bitcoin price is non-stationarity due to the fluctuations in price in late 2017. Without any stationarity, our models cannot learn any pattern from the data which results in lagging predictions. By not including data from 2017 and using data from March 2018 as training data, the data has some stationarity which enables our models to learn the patterns and make non-lagging predictions and achieve better accuracy. Moreover, removing multicollinearity and features that have very high correlation with close price (open, high, low prices and marketcap) does reduce the effect of lagging prediction. That said, some models, KNN and neural networks, still have few lagging predicted points. This is due to the low level of stationarity in Bitcoin price time-series and its unstable pattern which make it hard for predictive models to learn and make future predictions. Our deep learning models did not perform as well as regularised linear regression and time-series models, potentially due to the lack of training samples in our dataset. Our time-series ARIMA model has the least lagging predictions as it incorporates lagged observations as input features. Overall, we select SVR as the optimal model for Bitcoin price prediction as it achieves the best accuracy (RMSE: 237.8, MAE: 190.4) and has a bias value closest to zero (5.6) indicating it is a highly unbiased model.

# Limitations and Future Works

## Limitations

Our project has several limitations due to time constraints. First, we only collected and used daily data to train our models. The reason is that even though Bitcoin trading data is available at second, minute or hourly level, we want to combine it with other features which are only available in daily data. Therefore, we only used daily data of Bitcoin price to combine it with other features. To reduce the lagging predictions and improve accuracy, we only used data from March 2018 to June 2018 as training data because data before this time point had no stationarity. Hence, there are not enough training samples for our deep learning models as neural networks requires a large number of samples for training. As a result, our LSTM and GRU models did not perform as well as expected. Even though they achieved reasonably good accuracy and outperformed simple MLP and OLS models, their predictions are still lagging. Another limitation is that our models are currently trained to make only one-day-ahead predictions.

## Future Works

Due to the limitations, this project can be improved in future works. In terms of the limitation regarding number of samples, methods that can match hourly or minute data of Bitcoin price with other features need to be explored. Another option is to use only features such as technical indicators which can be generated from hourly or minute Bitcoin price. This can potentially improve the performance of our LSTM and GRU models. Moreover, with more samples, we can train our models to make several-day-ahead predictions. That said, using our current dataset, we can improve the predictive accuracy of our models by combining the predictions of multiple models using model stacking or Bayesian model averaging methods. In addition, additive boosting can enhance the accuracy of linear regression models by fitting a gradient boosting model to their residuals.

# Project Resources & Outcomes

## Resources

**Hardware**

We use our own computers for data analysis, model training, data visualisation and webpage deployment. Specifications of our computers: Mac OS Operating System, Processor 3.5 GHz Intel Core i7, Memory 16 GB 2133 MHz LPDDR3.

**Software**

The following software are used:

* Anaconda, Python 3.6
* XGBoost library, Keras library
* Amazon Web Services (AWS) & Simple Storage Service (S3),
* Python external libraries: *sklearn, pandas, numpy, statsmodels, scipy, spark-sklearn, matplotlib, cryptory, TA-lib.*
* Visualisation frameworks: D3.js JavaScript library, Python’s plotly, Tableau

## Roles & Responsibilities

|  |  |  |
| --- | --- | --- |
| **Members** | **Roles** | **Responsibilities** |
| Quang Trung Nguyen | + Machine Learning Modeler  + Trouble Shooter  + Group Leader | * Data Collection * Machine Learning Model Development & Testing * Build residual autocorrelation and Quantile-Quantile plots using Python’s plotly * Format and proofread reports * Identify and manage project risks |
| Mingxuan Li | + Time-Series Modeler  + Front-end Programmer | * Time-Series Models Development & Testing * Develop predicted price movements visualization using d3.js * Deploy d3.js codes and other visualisations into Amazon Web Services |
| Bin Liu | + Organizer  + Tester | * Machine Learning Models Comparison * Schedule group meetings * Monitor progress of the project |
| Jun Xiong | + Deep Learning Modeler  + Data Analyst | * Deep Learning Models Development & Testing * Exploratory Data Analysis * Build predictive accuracy and mean forecast error bar charts using Tableau |
| Jiaming Wei | + Deep Learning Modeler  + Tester | * Deep Learning Models Development & Testing * Test functionality and usability of the webpage |

## Project Deliverables

|  |  |  |  |
| --- | --- | --- | --- |
| **No** | **Deliverables** | **Completion Date** | **Notes** |
| 1 | Predictive Models | Week 8  19/09/2018 | Python codes implementing 14 models for Bitcoin Price prediction *(available on our Github)* |
| 2 | Data Visualizations | Week 9  26/10/2018 | D3.js, Python’s plotly codes and Tableau notebooks visualizing (available on our Github):- Predicted Bitcoin Price movements of 14 models- Predictive accuracy bar chart of 14 models - Mean Forecast Error (Bias) bar chart of 14 models  - Residual Quantile-Quantile plots of 14 models  - Residual Autocorrelation plots of 14 models |
| 3 | Static AWS Webpage | Week 10  10/10/2018 | The AWS hosted static webpage displays the visualizations in deliverable 2 |
| 4 | Group Proposal Report | Week 5  31/08/2018 | None |
| 5 | Group Progress Report | Week 9  05/10/2018 | None |
| 6 | Group Presentation | Week 12  26/10/2018 | *None* |
| 7 | Group Final Report | Week 13  02/11/2018 | *None* |

# Milestones & Risk Assessment

## Milestones

|  |  |  |  |
| --- | --- | --- | --- |
| **Milestone** | **Tasks** | **Reporting** | **Date** |
| Week-1 | **Project start:** 1.Kick-off Session  2. Familiarize with project requirements & group members’ background  3. Set up communication channels via Slack, WeChat, Google Drive and Github | Meeting with tutor and group members to review the project | 08/08/2018 |
| Week-2 | **Analysis stage:**  1. Define project scope  2. Design work plan and tasks allocation  3. Literature review about project topic | *None* | 15/08/2018 |
| Week-3 | **Analysis stage:** 1. Literature review about suitable methodologies & data  2. Data collection | Meeting with tutor to review the work plan | 22/08/2018 |
| Week-4 | **Analysis stage:** 1. Finalise datasets and upload to Github  2. Finalise set of machine learning models to implement and the resources needed  3. Proposal report writing | Group meeting to discuss and write proposal report | 29/08/2018 |
| Week-5 | **Proposal Report Due** | Meeting with the tutor to review the proposal report | 31/08/2018 |
| Week-6 | **Development stage:**  1. Perform Exploratory Data Analysis on the dataset  2. Perform Feature Engineering to select the best subset of features for model training | *None* | 05/09/2018 |
| Week-7 | **Development stage:**  1. Collected a new dataset with more samples  2. Re-performed EDA and Feature Engineering  3. Created a model training workflow based on Day-Forward-Chaining algorithm | Group meeting to collect new dataset as there was not enough samples | 12/09/2018 |
| Week-8 | **Development stage:**  1. Generated technical indicators for price using TA-lib library and added them to the dataset  2. Machine Learning model training  3. Time-Series model training  4. Deep Learning model training | *None* | 19/09/2018 |
| Week-9 | 1. Initial Models Evaluation  3. Progress Report Writing  **Progress Report Due** | Meeting with tutor to review the progress report | 05/10/2018 |
| Week-10 | **Deployment & Testing stage:**  1. Built data predicted price movements visualizations using D3.js  2. Built residual autocorrelation and Quantile-Quantile plots for all models using Python’s plotly  3. Built predictive accuracy and mean forecast error bar charts for all models using Tableau | *None* | 10/10/2018 |
| Week-11 | **Deployment & Testing stage:**  1. Deploy the visualisations onto the AWS hosted webpage  2. Unit Testing on each visualization  3. Integration Testing on the entire webpage  4. Model comparison using 4 criteria and the visualisations | *None* | 17/10/2018 |
| Week-12 | **Final Documentation stage:**  1. Prepared PowerPoint slides for the presentation  2. Improved the layout of the visualisations on the webpage to ensure correct display on large screen | Group meeting to design the PowerPoint slides | 26/10/2018 |
| Week-13 | **Final Documentation stage:**  1. Finalised PowerPoint slides for the presentation  2. Rehearse the presentation  3. Designed the outline for the Final Report | Group meeting to rehearse the presentation | 02/11/2018 |
| Week-14 | **Final Documentation stage:**  1. Worked on experiment results, discussion, conclusion and future works sections of the Final Report  **Final Presentation** | Group meeting to write the Final Report | 09/11/2018 |
| Week-15 | 1. Finalised the content and format of the Final Report  **Final Report (thesis)** | Group meeting to write the Final Report | 16/11/2018 |

## Risk Assessment

We present our Risk Register Plan in Table 1. which identifies and categorises the major risks that our project faced. Our Risk Mitigation Plan in Table 2. describe responses that we undertook to handle these risks.

**Table 5. Risk Register Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk No** | **Category** | **Risk Description** | **Probability of Occurring** | **Impact** |
| 1 | Data Collection | Python package *cryotory* becomes unavailable to collect data for test dataset | Medium | Low |
| 2 | Deployment | Non-indicative visualisations on the webpage | Low | Medium |
| 3 | Development | Python external libraries become unavailable | Low | Medium |
| 4 | Project Planning | Over-engineering - Implement changes that increase the project scope significantly | Low | High |
| 5 | Project Planning | Failure to achieve planned milestones due to time overrun | Low | High |
| 6 | Data Collection | The collected datasets contain irrelevant/ incorrect features that can negatively impact machine learning model’s performance | High | Low |

**Table 6. Risk Mitigation Plan**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Risk No** | **Planned**  **Response Type** | **Response** | **Responsible Persons** | **Status** |
| 1 | Mitigation | Manually collect and combine data from multiple sources  Store the existing dataset into github repository | Quang Trung Nguyen  Mingxuan Li | Avoided |
| 2 | Avoidance | Carefully evaluate different visualisation methods from d3.js library | Mingxuan Li | Avoided |
| 3 | Mitigation | - Implement the models algorithms manually in Python  - Use R or other programming languages to implement the models | All members | Avoided |
| 4 | Avoidance | - Follow the planned milestones and methodologies precisely  - Hold frequent group meetings to discuss any possible changes made to the project scope | All members | Mitigated |
| 5 | Avoidance/  Mitigation | - Monitor project progress frequently and identify any issues early  - Reduce the scope of the project, complete all baseline models first before adding complexities and extra features | All members | Mitigated |
| 6 | Avoidance | Perform Exploratory Data Analysis and Feature Engineering to eliminate irrelevant features | Quang Trung Nguyen  Jun Xiong | Mitigated |

# References

Alessandrettia, L., ElBahrawya, A., Aielloc, L., M., & Baronchellia, A. (2018). Machine Learning the Cryptocurrency Market. *Cornell University Library*.

Ali, J., Khan, R., Ahmad, N., & Maqsood, I. (2012). Random Forests and Decision Trees. *IJCSI International Journal of Computer Science Issues*, 9, 272-278.

A beginner’s Guide to LSTMs. (n.d.). Retrieved from https://skymind.ai/wiki/lstm#long.

Amazon EC2. (2015). Amazon Web Services. Retrieved from https://aws.amazon.com/es/ec2/

Asteriou, D., Hall, S., G. (2011). ARIMA Models and the Box–Jenkins Methodology. *Applied Econometrics*, 265–286.

Bobriakov, I. (2018, March 6). Bitcoin price forecasting with deep learning algorithm. Retrieved from https://medium.com/activewizards-machine-learning-company/bitcoin-price-forecasting-with-deep-learning-algorithms-eb578a2387a3.

Chohan, U., W. (2017). Cryptocurrencies: A Brief Thematic Review. *Economics of Networks Journal.*

Chen, T., & He, T. (2015). xgboost: eXtreme Gradient Boosting.

Chen, T., & Guestrin, C. (2016). [XGBoost: A Scalable Tree Boosting System](http://delivery.acm.org/10.1145/2940000/2939785/p785-chen.pdf?ip=203.87.78.153&id=2939785&acc=CHORUS&key=4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E4D4702B0C3E38B35%2E6D218144511F3437&__acm__=1528046673_2ef90f1d96bb22964c882ae638af125c). *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining.*

Chung, J., Gulcehre, C., Cho, K., & Bengio, Y. (2014). Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling. *Cornell University Library*.

Geurts, P., Ernst, D., & Wehenkel, L. (2006). Extremely randomized trees. *Machine Learning*, 63, 1, 3-42.

Guo, T., & Antulov-Fantulin, N. (2018). An experimental study of Bitcoin fluctuation using

machine learning methods. *Proceedings of ACM Conference* (Conference’18).

How XGBoost Works. Amazon Web Services. Retrieved from https://docs.aws.amazon.com/sagemaker/latest/dg/xgboost-HowItWorks.html.

Jain, A. (2014). Data visualization with the D3.JS Javascript library. *Journal of Computing Sciences in Colleges*, 30, 2, 139-141.

James, G., Witten, D., Hastie, T., & Tibshirani, R. (2013). *An Introduction to Statistical Learning - with Applications in R*. New York, NY: Springer-Verlag.

Jang, H., & Lee, J. (2018). An Empirical Study on Modeling and Prediction of Bitcoin Prices With Bayesian Neural Networks Based on Blockchain Information. *Institute of Electrical and Electronics Engineers,* 6, 5427-5437

Makridakis, S., Hibon, M. (1997). ARMA Models and the Box–Jenkins Methodology. Journal of Forecasting, 16, 3, 147-163.

Nair, L., & Shetty, S. (2016). Interactive visual analytics on Big Data: Tableau vs D3.js. *Journal of e-Learning and Knowledge Society*, 12, 4.

Nakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Bitcoin. Retrieved from https://bitcoin.org/bitcoin.pdf.

Nau, R. (2018). Stationarity and Differencing. Duke University. Retrieved from https://people.duke.edu/~rnau/411diff.htm.

Olah, C. (2015, August 27), Understanding LSTM Networks [Blog post]. Retrieved from http://colah.github.io/posts/2015-08-Understanding-LSTMs/.

Schut, M. (2017). Bitcoin analysis from an investor’s perspective. *Erasmus University Rotterdam, Erasmus School of Economics*.

Shah, D., & Zhang, K. (2014). Bayesian regression and Bitcoin. *Institute of Electrical and Electronics Engineers*, 409-414

Zhu, N., Q. (2013). *Data Visualization with D3.js Cookbook*. Birmingham, England: Packt Publishing.

# Appendix 1 – Project Gannt Chart

