Springboard - DSC
Capstone Project Two

Bitcoin (BTC) Price - A Predictive Analysis

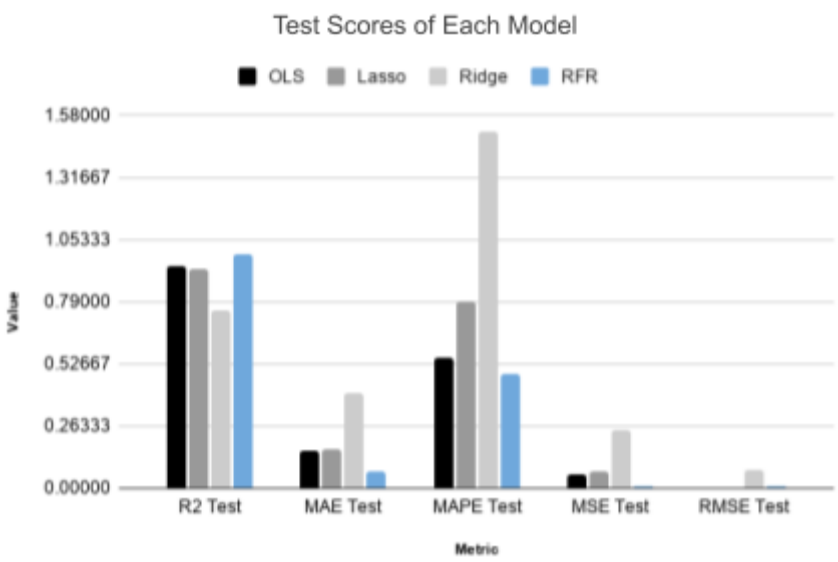
By John Arancio

January 2021

Table of Contents

Abstract

While the validity of cryptocurrency as an actual asset class remains in dispute, Bitcoin has grasped the attention of many participants within the financial markets. However, across all the classes of participants, understanding its price action has remained quite ambiguous. As of 2021, Grayscale Trust now owns 3.9% of the BTC in circulation equating to $31,678,972,086USD in AUM and holds over 70% of the total BTC held by publicly traded companies. JP Morgan's strategists estimate the OTC trust is expanding their AUM by $1 billion USD per month. Capital floating between accredited investors in quantities like that is revealing concerning the amount of potential for profit. This explosive expansion and volatility seen from Bitcoin over time has developed into a prime factor for speculators, who are just now recognizing that cryptocurrency isn't going away anytime soon. Now more than ever it is clear the market is in great demand for an accurate forecasting model to sustain profit and reduce risk.



Using the Binance cryptocurrency exchange API client, the required data was obtained and examined, culminating in the construction of five distinct model architectures. Three classified as simple linear regression; Ordinary Least Squares (OLS), Lasso, and Ridge. One using a decision tree ensemble method as Random Forest Regression (RFR). Then last, one using recurrent neural network (RNN) in sequential form as Long Short Term Memory (LSTM). All model features were computed as financial market indicators, which can to be regarded as "inferential statistics"

Through the entire predictive analysis, we identified the strongest performing model[s] out of the five to be the RFR or the LSTM. The RFR scored higher using the R2 metric (0.988), however; the LSTM scored less in various other error metrics, like MAPE (0.024). Through our EDA, we discovered that the "close" values had the greatest correlation with the other variables (0.98) which was later confirmed while interpreting the models applying various methods like "feature importance" and "LIME". Interpretability results proved "close" to have an overwhelming influence in predicted values. Even though RFR had the highest R2 score, we determined that further testing on different test sets called for to verify true model accuracy. The extra testing is not essential, however, because of the valid out-of-bag (OOB) score produced (0.913). That being said, the LSTM model might vary in suggested practice from the RFR. The RFR possesses the most potential to be exploited as an automated (HF) trading strategy, while the LSTM might be better fit for a signal generator of sorts, as a means of confluence, to confirm an already completed analysis.

Keywords: lstm, rfr, ols, lasso, ridge, bitcoin, cryptocurrency, investing, modeling, deep learning, neural network, regression

(1) Introduction

This project intends to clarify the volatile and erratic nature of Bitcoins price. Predictive analytics are remarkably resourceful when seeking to interpret the irregular pattern of market movement and can be a viable solution for maximizing profitability. Research and discoveries in the project prove helpful to any individual involved in the financial markets, yet more precisely, those taking part in the cryptocurrency market. These parties are retail investors and various institutions who wish to generate accurate signals or improve overall speculation, therefore encouraging a more rigorous price action analysis. Traditional market players may still find the analysis provided in this project useful, as Bitcoins price action seems to operate similarly to alternative asset classes, such as equities.

Four separate Jupyter notebooks contain the implementation of all methods used. Each notebook represents the specific stages performed to arrive at a conclusion from respective predicted values (data acquisition/wrangling, exploratory data analysis and inferential statistics, baseline modeling/preprocessing, and extended modeling). All notebooks and analyses can be found in its designated GitHub repository: (https://github.com/jra333/BTC-Price-Prediction).

From the initial dataset ("btc_df.csv") we use 6 specific columns to generate our predictor features. The columns used were, "closeTime", "open", "high", "low", "close", and "volume". From those 6 columns we then calculated our indicators providing us with additional features and should be considered as a form of inferential statistics. These indicators are, "RSI", "EMA_Long", "EMA_Short", "ATR", "ADX", and "OBV". We include a supplementary column as an indicator/feature containing the mean polarity for the gained tweet sentiment regarding what users discussed in a tweet about BTC for each day, labeled as, "tweet_sentiment". All together, ending in a final dataframe of 441 rows and 13 columns spanning a dated range from 09/02/2019 to 11/15/2020.

(2) Approach

(2.1) Data Acquisition and Wrangling

In order to acquire the data called for, we employed a Python wrapper of Binances REST APIv3 as a client. The client provided selection of any length of time, and permitted the capacity to draw any of the data measures (i.e., volume, open, high, close) for the various time decided. That being said, we based the date range chosen on the initial idea of using Binances "future" prices as a feature or variable. Although, since Binances "future contracts' ' didn't go live until 09/02/2019, that was the decisive starting date and ultimately the starting position of the entire time series dataset. Out of the data collected for both BTCUSD and BTCFUT, market measures were positioned on a millisecond basis. Each was classified respectively: "'openTime", "open", "high", "low", "close", "volume", "closeTime", "quoteAssetVolume", "numTrade", "takerBuyBaseAssetVolume", "takerByQuoteAssetVolume", and "ignore". Because the cryptocurrency market is "open'' 24/7, the "closeTime'' could be anytime during the day, as opposed to the more conventional "NASDAQ Exchange'' for example, where that market is only open 5 days a week and the closing time is the same for each day. However, because Binance stores their data in milliseconds, and since the goal was to forecast daily prices, there is minimal error regarding potential skewed results like a potential timeline bias or error due to lack of accurate data or causing. That being said, the same time (end of day) every 24 hours, was used as the "closeTime" (approx. 11:59 PM).

Once the data was organized, data cleaning began with column removal first. The following columns were dropped from the dataset; "quoteAssetVolume", "numTrade", "takerBuyBaseAssetVolume", "takerByQuoteAssetVolume", and "ignore". We eliminated those columns because those types of data measures are not widely available for most people, and it varies from exchange to exchange and market to market. The intention here was to make the approach as replicable as possible, and somewhat applicable for investors who are not strictly concerned with cryptocurrency. After the removal of specified columns, the ones left over were; "open", "high", "low",

"close", "volume" and "closeTime", leaving us with more than enough data to calculate the indicators. To determine the indicators, we used BTALib, and derived: "RSI", "EMA_Long", "EMA_Short", "ATR", "ADX", and "OBV".

Calculations for each indicator:

```
RSI (Relative Strength Index):
Period - 14
       up = upday(data)  # max(close - close(-1), 0.0)
       down = downday(data)  # abs( min(close - close(-1), 0.0) )
       maup = movingaverage(up, period)
       madown = movingaverage(down, period)
       rs = maup / madown
       rsi = 100 - 100 / (1 + rs)

EMA_LONG/SHORT (Exponential Moving Average):
Long period - 50
Short period - 20
       Exponential Smotthing factor: alpha = 2 / (1 + period)
       prev = mean(data, period)
       movav = prev * (1.0 - alpha) + newdata * alpha
       (or alternatively #  movav = prev + alpha(new - prev))

ATR (Average True Range):
Period - 14
       truerange = max(high, close(-1)) - min(low, close(-1))
       atr = SmoothedMovingAverage(truerange, period)

ADX (Average Directional Index):
Period - 14
         upmove = high - high(-1)
         downmove = low(-1) - low
         dm = upmove if upmove > downmove and upmove > 0 else 0
         dm = downmove if downmove > upmove and downmove > 0 else 0
         di = 100 * SmoothedAccum(+dm, period) / SmoothAccum(truerange, period)
         di = 100 * SmoothedAccum(-dm, period) / SmoothAccum(truerange, period)
         dx = 100 * abs(+di - -di) / (+di + -di)
         adx = MovingAverage(dx, period)

OBV (On Balance Volume):
       close > close(-1) => vol_pressure = +volume
       close < close(-1) => vol_pressure = -volume
       close == close(-1) => vol_pressure = 0

       obv = obv(-1) + vol_pressure
```

The last part of the data acquisition was to assemble the tweets and calculate the sentiment for each day in terms of "BTC". To accomplish this, we used an open source package called "Twint", a historical tweet scraper. Because of Twitter's API limitations, this approach provided the best workaround and most accessibility to a considerable amount of data. We performed the scrape by searching for any tweet containing the phrase, "BTCUSD" and only filtered the minimum "likes". This allowed me to pull tweets that had 33 "likes" or more. This tactic was believed to improve the overall data reliability, as tweets with more likes should equal higher accuracy sentiment wise. Once we had the tweets for the selected trading time range, we then performed sentiment analysis using "TextBlob", an API driven text processing package for NLP (Natural Language Processing) built on top of "NLTK". In order to start the sentiment analysis, cleaning the tweets was a priority (i.e., removal of special characters and emojis). Once that finished, we used mean polarity to spawn a very general sentiment from each tweet for that given day. A positive tweet receives a 1, a neutral tweet receives a 0, and a negative tweet receives a -1. Each day contained anywhere from 1 to 10 tweets, producing an average from the individual tweet sentiment. The reason for so few tweets for each day was likely because of the high "like" filtering. Quality over quantity is more practical in this case, as the cryptocurrency "scene"

can become polluted. After we assigned each tweet a sentiment rating and averaged for the day, we compiled all its data into a new dataframe and later merged with the master dataframe to create another form of "indicator".

The comprehensive process resulted in 441 rows ranging from dates 09/02/2019 to 11/15/2020 and 14 columns. Containing zero "NaN" values, and everything required to begin the next phase of investigation.

(2.2) Storytelling and Inferential Statistics [EDA]

During this stage, through exploratory data analysis, we identify what features or variables drive price movement by analyzing their relationship and correlations. Using graphical representations of data distribution, we can understand the relationships better and extrapolate that information further to predict price more accurately using our uniquely compiled architectures.

The analysis begins with a "pair plot" in order to view the relationship between each variable. Noting the structure of correlation that's visually visible.

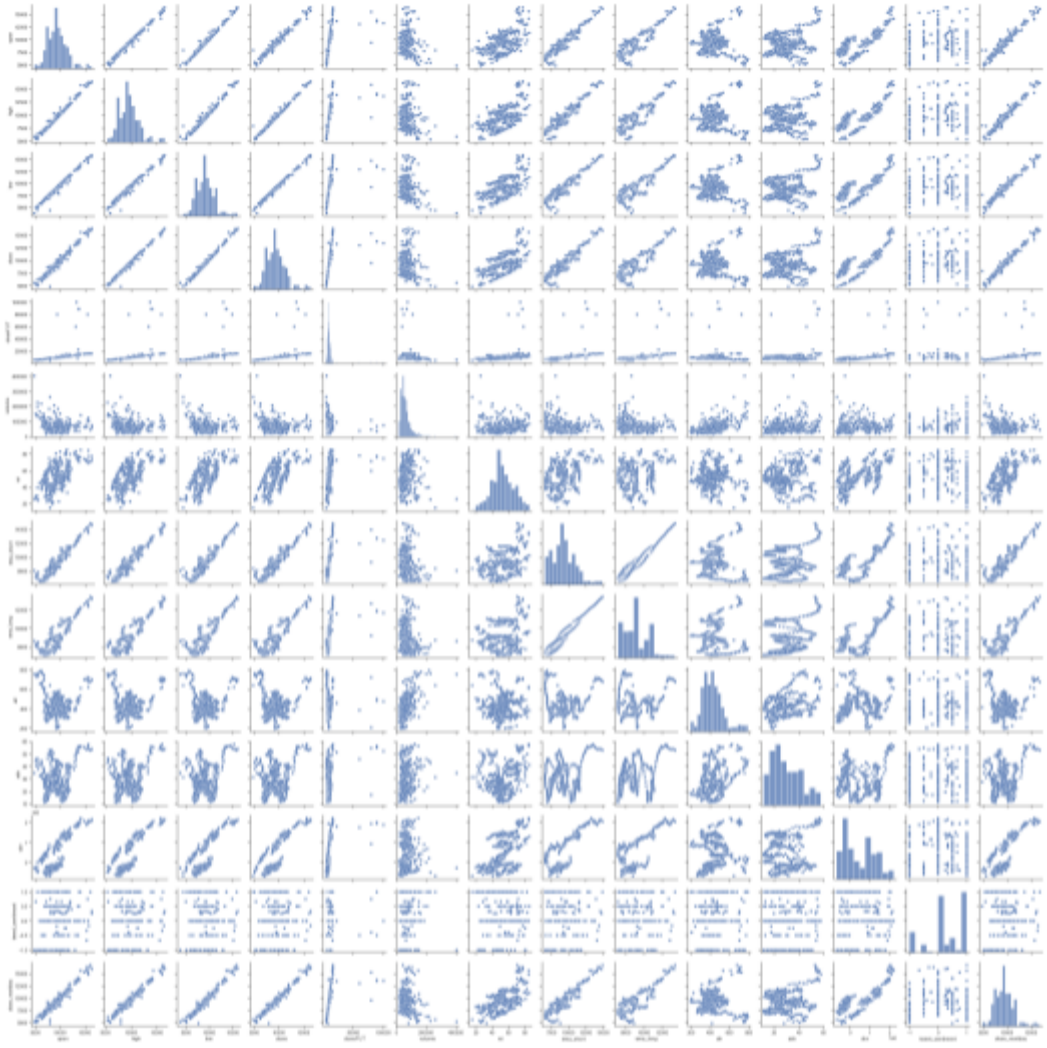


Figure 1 (pair plot, variable corr.)

As we can see, there are some clear, direct linear relationships between variables shown. Notably, the relationship between "open", "high", "low", and "close". This makes sense, as it is uncommon for the "high" of the day to deviate far from the "low" of the day, unless there is an anomaly in price action. This relationship between "open", "close" and "high", "low" creates a range in price for the day, which is where the ATR indicator we calculated

previously provides some aid in measuring price volatility. The volatile nature and relationships within BTCs price become more visible by the “pair plot”, as the price range becomes more sporadic, you can see a thinning of distribution at higher measures, implying high volatility. Even though the relationship between the ATR and “open”, “high”, “low”, “close” isn't directly related linearly, the distributions tell a unique story. It is very clear what variables or indicators have a direct relationship with our “prediction” target of “closeNextday”. You can see a very obvious linear relationship with the “rsi”, “ema_long”, “ema_short”, “atr”, and “obv”. Noticing this, we strive to validate the inference with the use of a correlation matrix heat map *(Figure 2)*.

The heat map provides a numerical threshold we can gauge variable correlation with. The identified threshold is:

- Low-correlation = 0.2 - 0.4
- Mid-correlation = 0.4 - 0.6
- High-correlation = 0.6 - 0.8

The higher the number, the more likely the given two variables are correlated with each other.

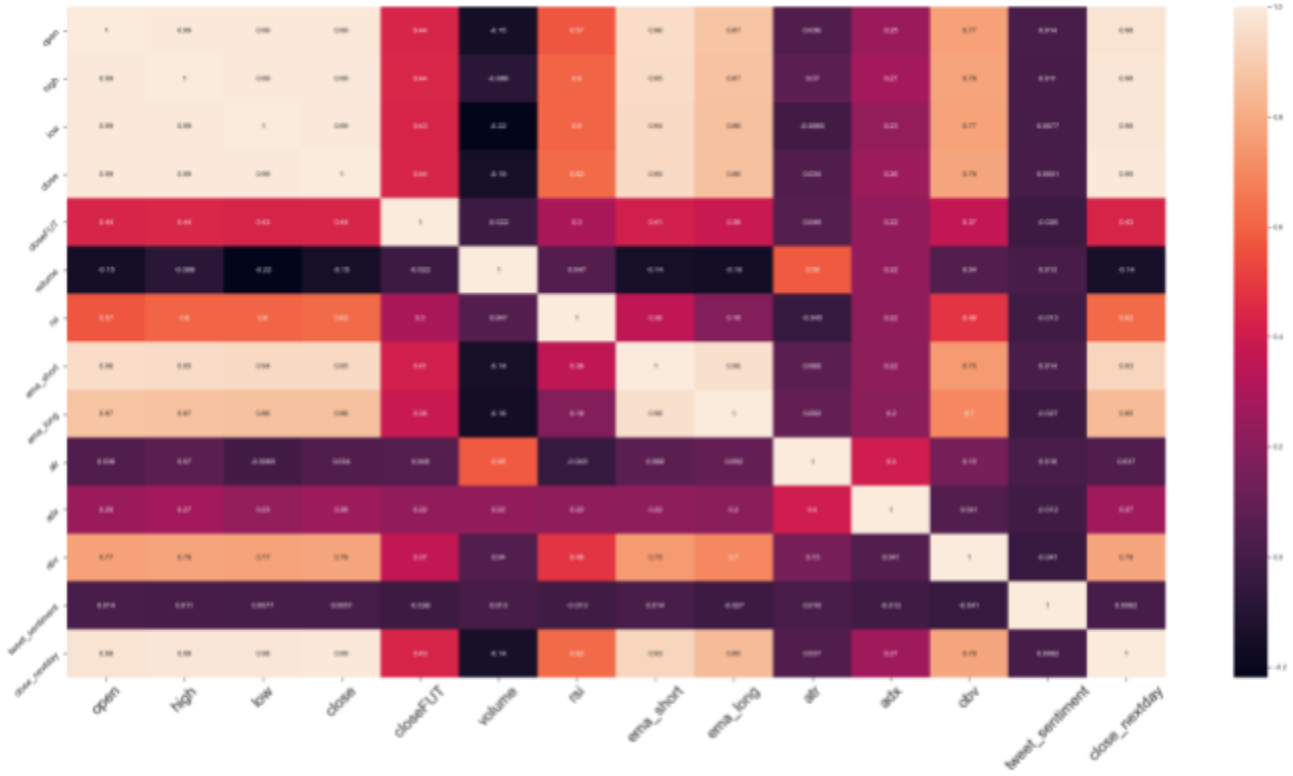


Figure 2 (heatmap, variable corr.)

By examining the correlation of our target with the other variables, it's clear that we were accurate in our assumptions regarding the relationship between “close”, “open’, “low” and “high" to the “rsi”, “ema_short”, “ema_long”, “atr”, and “obv”. Those indicators sit rather high on our threshold scale, unlike “volume” or “atr”. However, even though those variables seem to have no correlation to our target, they have a rather strong correlation to each other, with a score of 0.58. With a closer look, we can see they even have a slightly linear relationship.

Since we rank them “mid” correlated, it might be useful to include those two variables for future analysis, as combined they will most likely produce interesting results. Furthermore, “obv” has a high correlation regarding “rsi”, “ema_short” and “ema_long”. Similarly, those three variables also have a fairly high correlation with price, and more specifically the “closing price”.

On the opposite side of the spectrum when looking at the uncorrelated variables we can see that “obv” and “tweet_sentiment” have a very low score of -0.041. Although, upon further investigation and in knowing that “tweet_sentiment” does not have any linearity, we can see that when the data is plotted that there is actually a substantial relationship between the two *(Figure 4/5)*. Similarly, the “ATR”, a range of volatility, has a surprising relationship with “tweet_sentiment” as well.

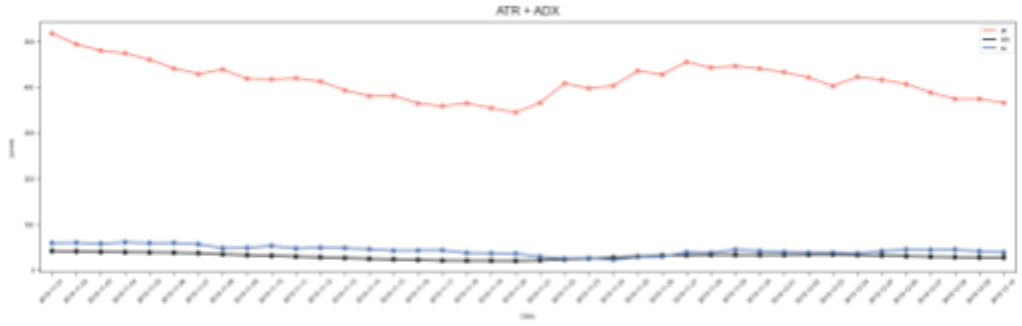


Figure 3 (ATR/ADX line plot, sample from dataframe)

Figure 4 (OBV line plot, sample from dataframe)

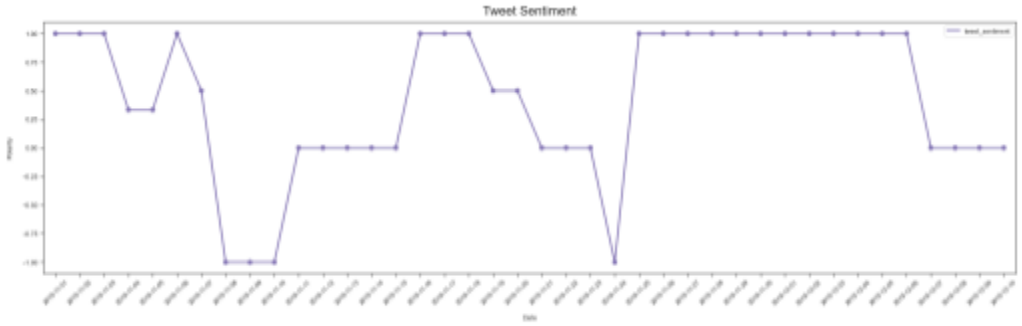


Figure 5 (tweet_sentiment line plot, sample from dataframe)

Both of the variables “obv” and “tweet_sentiment” even seem to correspond to “close”, “ema_long” and “ema_short”. As you can see in (*Figure 6)* below, “close” demonstrates the micro movements of the price action, while both EMAs, since they are rolling averages, operate as broad or obtuse “trend lines”.

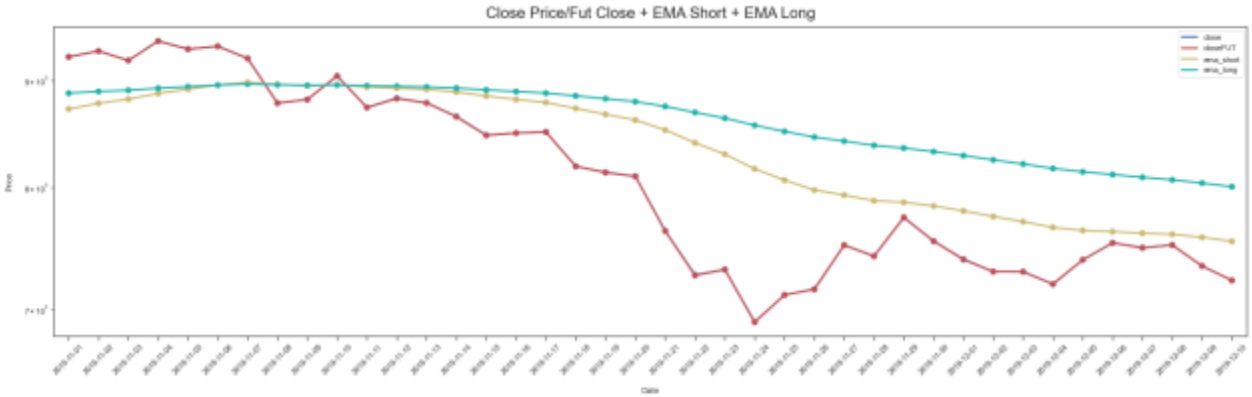


Figure 6 (close/EMAs line plot, sample from dataframe)

Still we can see how each of those variables have a distinct connection to one another. Here we can see the linear relationship of "obv" and "close" clearly *(Figure 7)*.

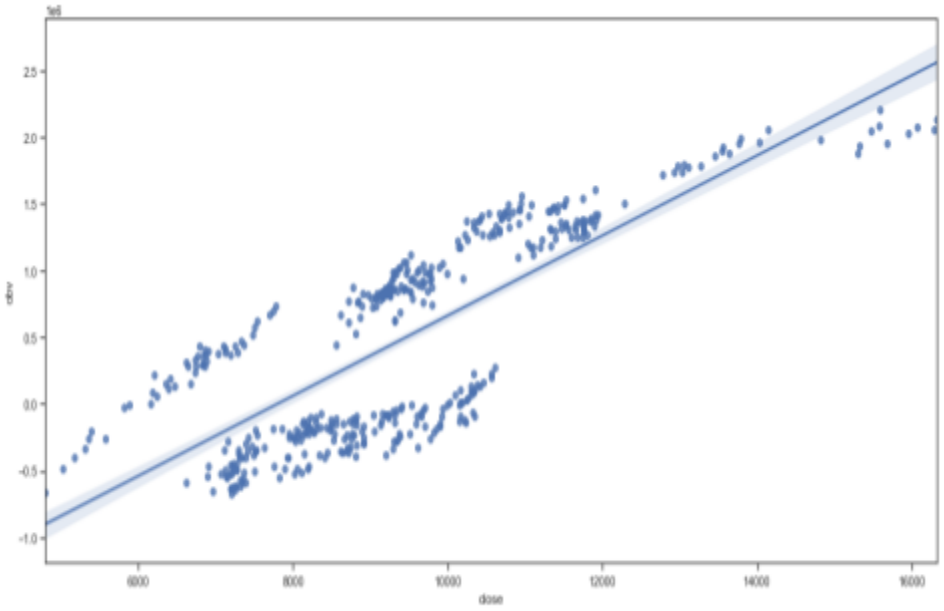


Figure 7 (OBV/close scatter plot, linear relationship)

To explore the possible linear relationship and distributions between the target and X variables further a KDE plot *(Figure 8)* proved to be useful. You can see that even though "obv" is a volume based metric where volume usually has no linear relationship to price, the "obv" is correlated by linearity. As for distribution, you might notice how when price is stable, the "obv" or on balance volume is large, whereas when price increases exponentially the "obv" distribution skews and becomes less linear.

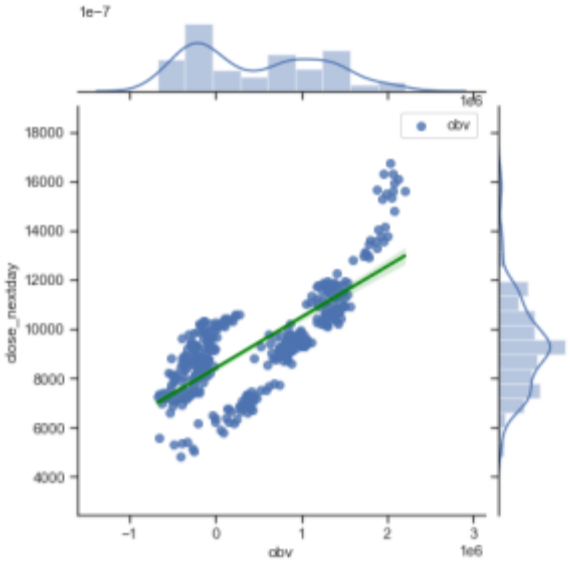


Figure 8 (close_nextday/OBV KDE plot, variable linearity)

Overall, the exploratory analysis of the provided variables revealed some interesting information, that of which has proved useful during the modeling and prediction stage.

As a summation, we can conclude that:

- Tweet sentiment has no linear relationship to target
- OBV has a linear relationship to target and close price
- Both ema_short and ema_long have linear relationship to target
- RSI has mild to zero linear relationship to target
- Close price is directly correlated to target

Because of this information revealed during our EDA, we could identify the most probable and useful variables to perform modeling with. The final dataframe for the same time range now contains 441 rows and 8 columns. Those 8 columns being, “close”, “volume”, “ema_short”, “ema_long”, “atr”, “obv”, “tweet_sentiment” and then the target “close_nextday”. Those variables were chosen because they represent the highest correlated variables with our target, and each other. Trimming the dataframe to only contain the highly correlated variables is useful for generating higher accuracy with forecasting.

(2.3) Baseline Modeling

To start the modeling phase, we needed to form some sort of baseline model to compare the rest of the methods when modeling. For this baseline comparison, we used a linear regression method as OLS (Ordinary Least Squares). This framework was selected because of the datas overall linearity and the simple fact that it is an ordered time series.

First, using “TimeSeriesSplit()” we split the X, y variables in ordered fashion, where the train sets contain 70% of the data (2019-09-02 : 2020-07-28 [331 rows]) and the test sets contain the rest of the data which is 30% (2020-07-29 : 2020-11-15 [110 rows]). The X variable contains the features selected from EDA, so, “close”, “volume”, “ema_short”, “ema_long”, “atr”, “obv”, and “tweet_sentiment”. Then our y variable contains the target prediction, which is “close_nextday”. We then split X into a train/test set respectively, and we performed the same for y. Next we scaled the datasets (“X_train”, “y_train”, “X_test”, “y_test”) using “StandardScalar()” to normalize the data. This function allows us to normalize the features of each column(X) individually, so that each column/feature/variable will have a mean of 0 and standard deviation of 1. From there, once finished with preprocessing, we use the OLS regressor to train on both “X_train” and “y_train”, and then test on “X_test” and “y_test”.

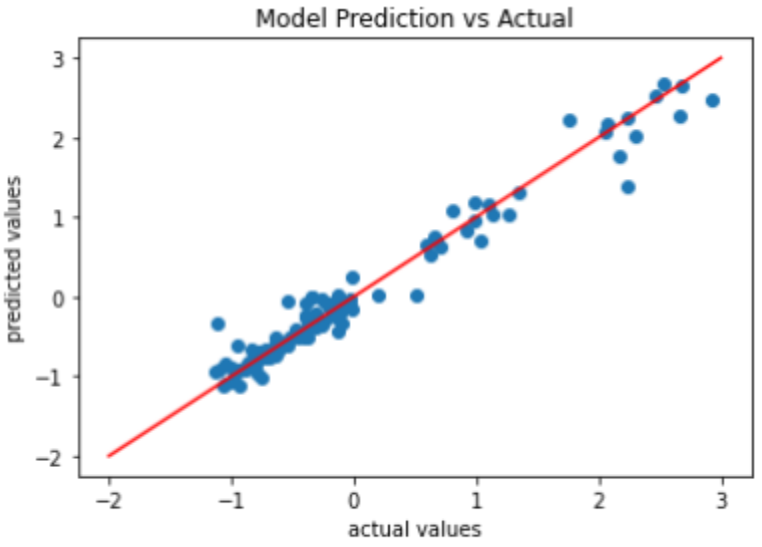


Figure 9 (OLS baseline performance, pred/actual)

With the OLS regression technique, we performed two “tests”. One iteration of our linear regression model, for X contained all available features, aside from the target (y). While the other one for X contained all features except the target and "close", with the target (y) remaining the same. We then calculated the *MAPE*, *MAE*, and *MSE* for each. By comparison, the second linear regression model performed worse *(Figure 10.1)* as the mean absolute percent error was roughly 354%, meaning the errors are "much greater" than the actual values. Whereas the first linear regression model gave us an *MAPE* of roughly 304%, meaning our errors of that model were "slightly greater" than the actual values *(Figure 9)*. This difference in *MAPE* can be seen as well when comparing *RMSE* results to *MAE* results for each model. We can see the first model gave an *RMSE* output of approx. 0.1999 and an *MAE* output of approx. 0.1333. Since the two measurements are relatively close to each other, we can imply it that the model makes many mistakes, but the mistakes are "small". We can say the same for the other model, but opposite when comparing metrics. The figures below are formed with the results of our first and second iteration of the OLS model, respectively. The first test received an R2 score of 0.930 *(Figure 10)* while the second received an R2 score of 0.790 *(Figure 10.1)*.

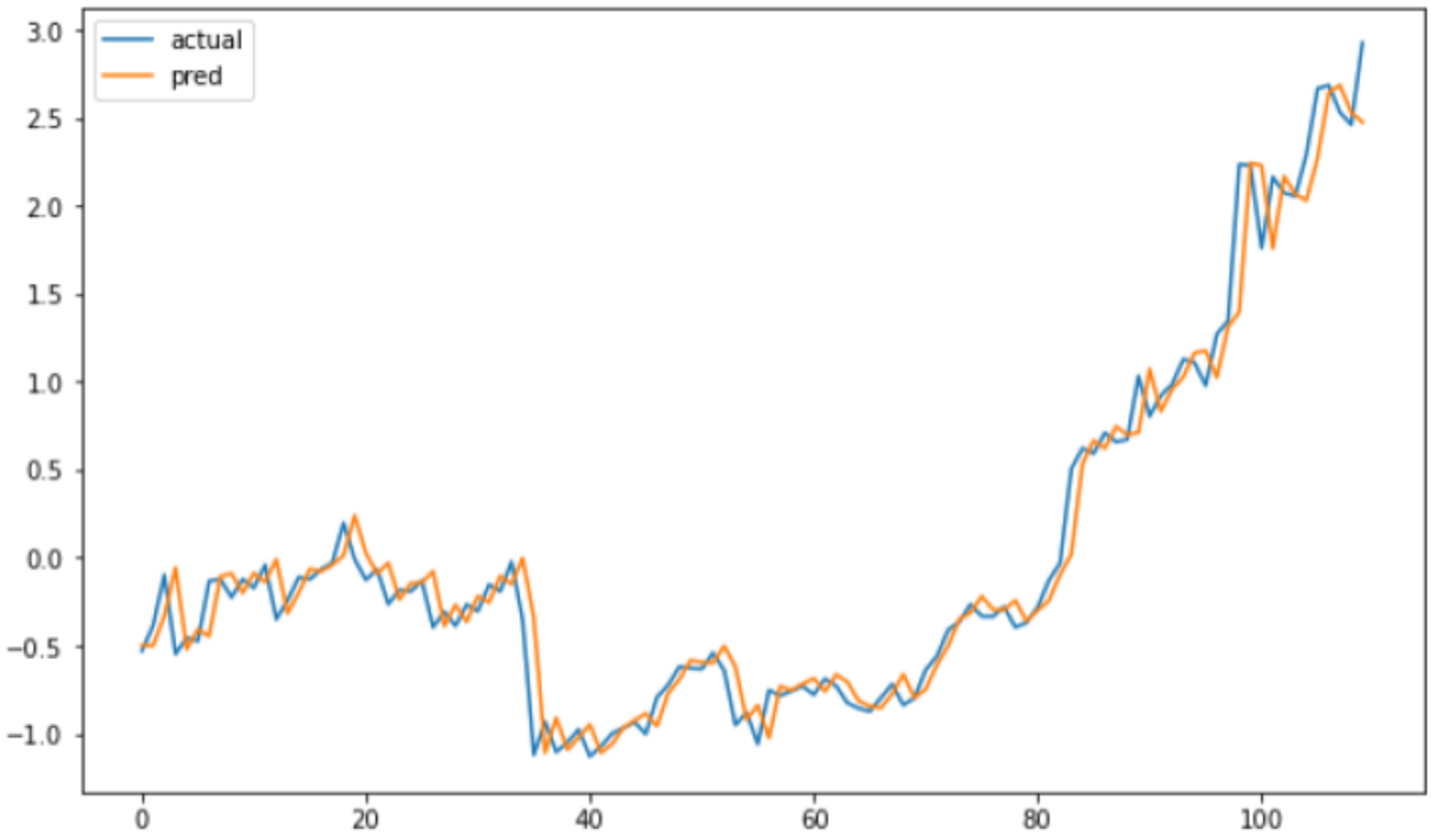


Figure 10 (OLS baseline test performance, pred/actual)

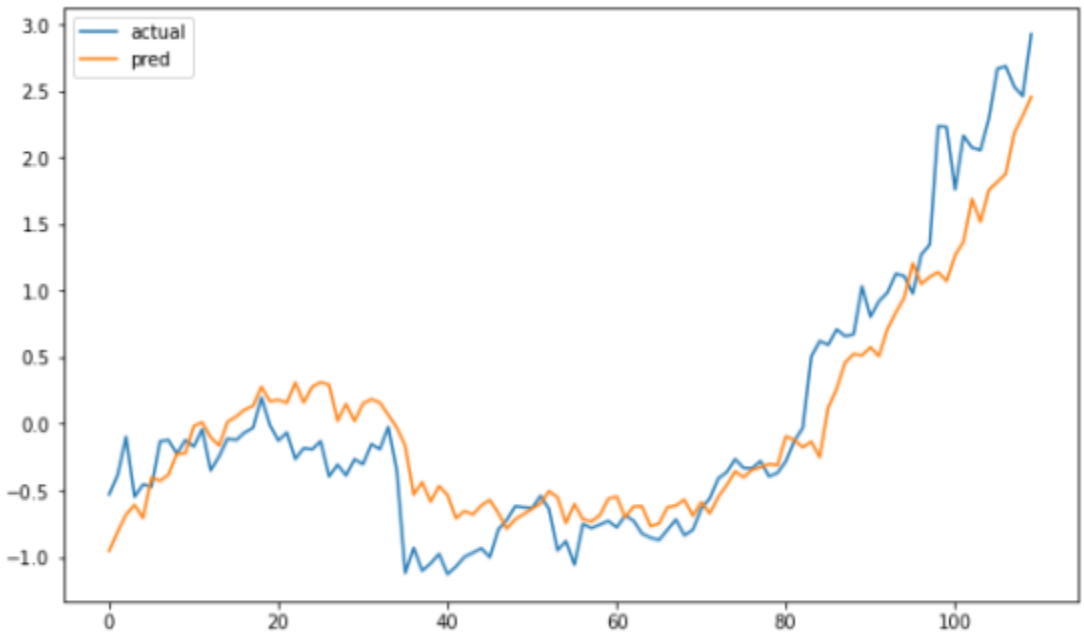


Figure 10.1 (OLS baseline test performance, w/o "close")

That being said, we can see that the feature "close" is fairly important when being used in a regression model. When it was removed, the performance of the model decreased significantly *(Figure 10.1)*. However, by including the "close" price, one might assume that it causes the model to be overfit. That is not the case though, because the "close" price is widely available before the next day or session begins. Thus, leakage does not seem like a possibility.

Because of these brief findings, we have further confirmed the original "hypothesis" formed during the EDA stage in exploring the high correlation between "close_nextday" and "close".

(2.4) Extended Modeling

This next stage contains we build a more in-depth approach to modeling that is based on the original baseline model. The models used as far as linear regression goes are OLS (Ordinary Least Squares), Lasso, and Ridge. Further experimentation warranted a tree based approach using the ensemble method of RFR (Random Forest Regressor). Then last, we applied a deep learning approach to our dataset using an RNN (Recurrent Neural Network) for time series forecasting using an LSTM (Long Short Term Memory) framework.

All simple linear regression based models (OLS, Lasso and Ridge) are built using this equation: $y_i = \alpha + \beta x_i + \varepsilon_i$

However, each individual model, respectively, provides a different variant or strategy of that equation which optimizes the regression model to minimize errors as best as it can. More specifically, OLS as an optimization strategy minimizes bias of OLS function with the "Best Linear Unbiased Estimator" (BLUE). Its goal is to obtain a straight line from the model, making it as close as possible to the data points. Its properties as a popular regression strategy are more than applicable for this predictive analysis. Providing an output of unbiased estimators, of the actual values alpha (**α**) and beta (**β**).

$$\mathscr{L}_{OLS} = \|Y - X^T\beta\|^2$$

Figure 11 (OLS equation)

We also have Lasso, similar to the OLS strategy in its minimization goal, however it takes a novel approach using “shrinkage”. The Lasso model will try to shrink its data values down to a central point, like the mean. This strategy is useful for sparse models and high multicollinearity. Since it performs “L1 Regularization” it adds a penalty equal to the absolute value of the magnitude of coefficients. This type of regularization can cause sparse models with few coefficients. Some coefficients can become zero and eliminated from the model.

$$\mathscr{L}_{LASSO} = \|Y - X^T\beta\|^2 + \lambda\|\beta\|$$

Figure 11.1 (OLS - Lasso regularization equation)

Similar to Lasso, Ridge performs L2 regularization where it doesn’t result in elimination of coefficients or sparse models. The three different simple linear regression models are well suited for a time series prediction, especially in this case since the data is restricted quantitatively.

$$\mathscr{L}_{Ridge} = \|Y - X^T\beta\|^2 + \lambda\|\beta\|^2$$

Figure 11.2 (OLS - Ridge regularization equation)

Both linear and random forest regression use GridSearchCV() to tune hyperparameters. Using grid search through the model pipeline, we added a tuner as PolynomialFeatures(). Polynomial features usually improve linear regression models significantly by extrapolating the square values of data to produce additional features. For the regression specific portion of testing we compute the R2, MAE, MSE, RMSE score for each model and additionally the OOB (Out of Bag) score for the RFR.

As for the time series forecasting using LSTM, we applied a Keras wrapper of Hyperopt called Hyperas in order to tune the LSTM hyperparameters (i.e. Dropout, Activation, Dense, Optimizer, Batch size and Epochs). Hyperas is a more streamlined version of Hyperopt, providing a more versatile way to alter parameters and tune layers. The loss metric used was the MSE in order to minimize loss between test and validation datasets. Since the overall goal of the project is a regression "problem", accuracy itself is not applicable. Accuracy as a measure only applies for classification (discrete outputs), comparing for equality. With regression, we are predicting a continuous value (price), so if the target is 0.99, and the network predicts 1.0, then it would be considered as misclassified.

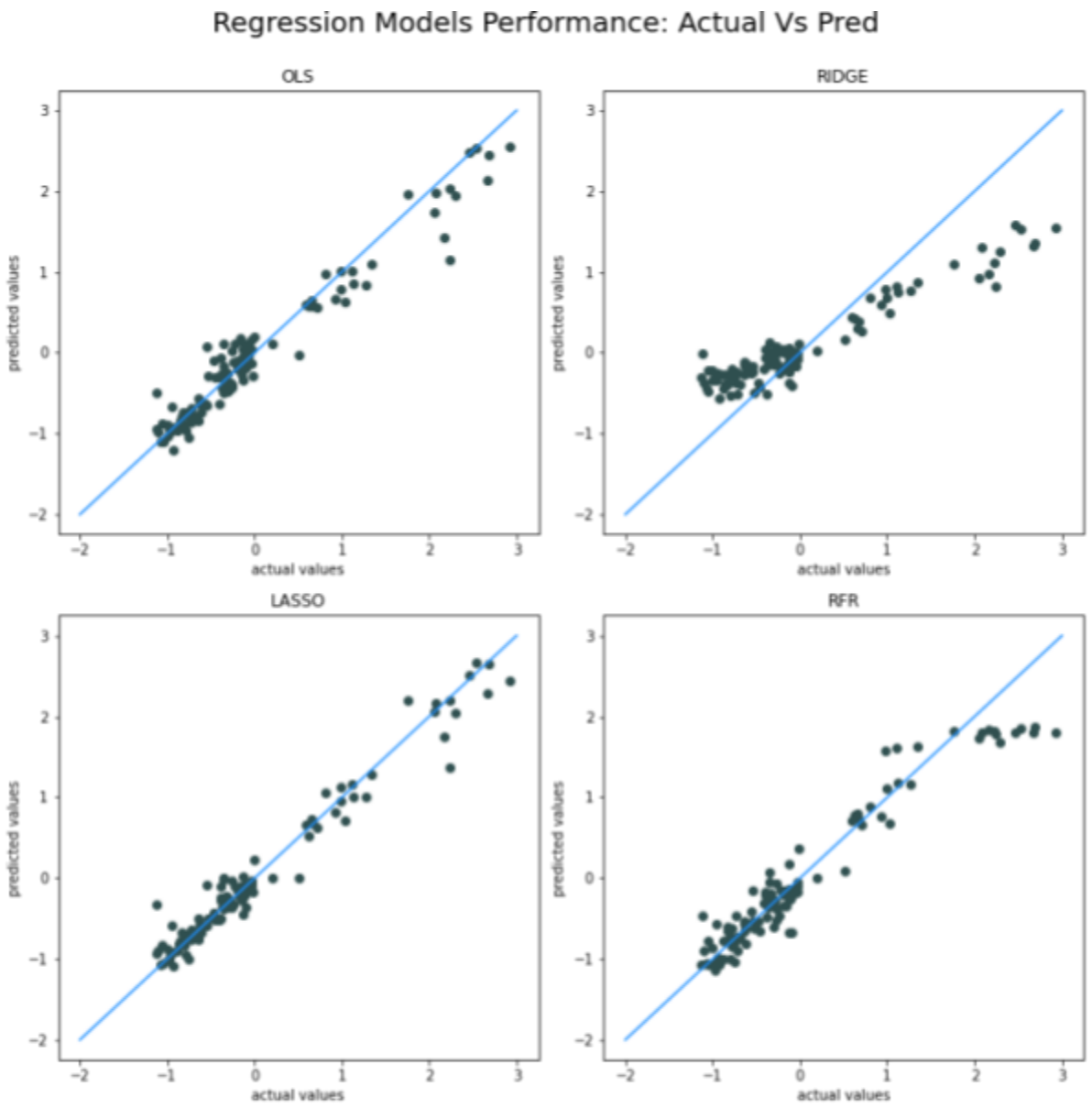


Figure 12 (regression models performance - test set)

As you can see, the two that performed the best as shown in *Figure 12* above, were the OLS and Lasso. The RFR seemed to perform fairly well until the price increased exponentially, and then the predicted values began to taper off. Similarly, the Ridge operated along the same lines as the RFR.

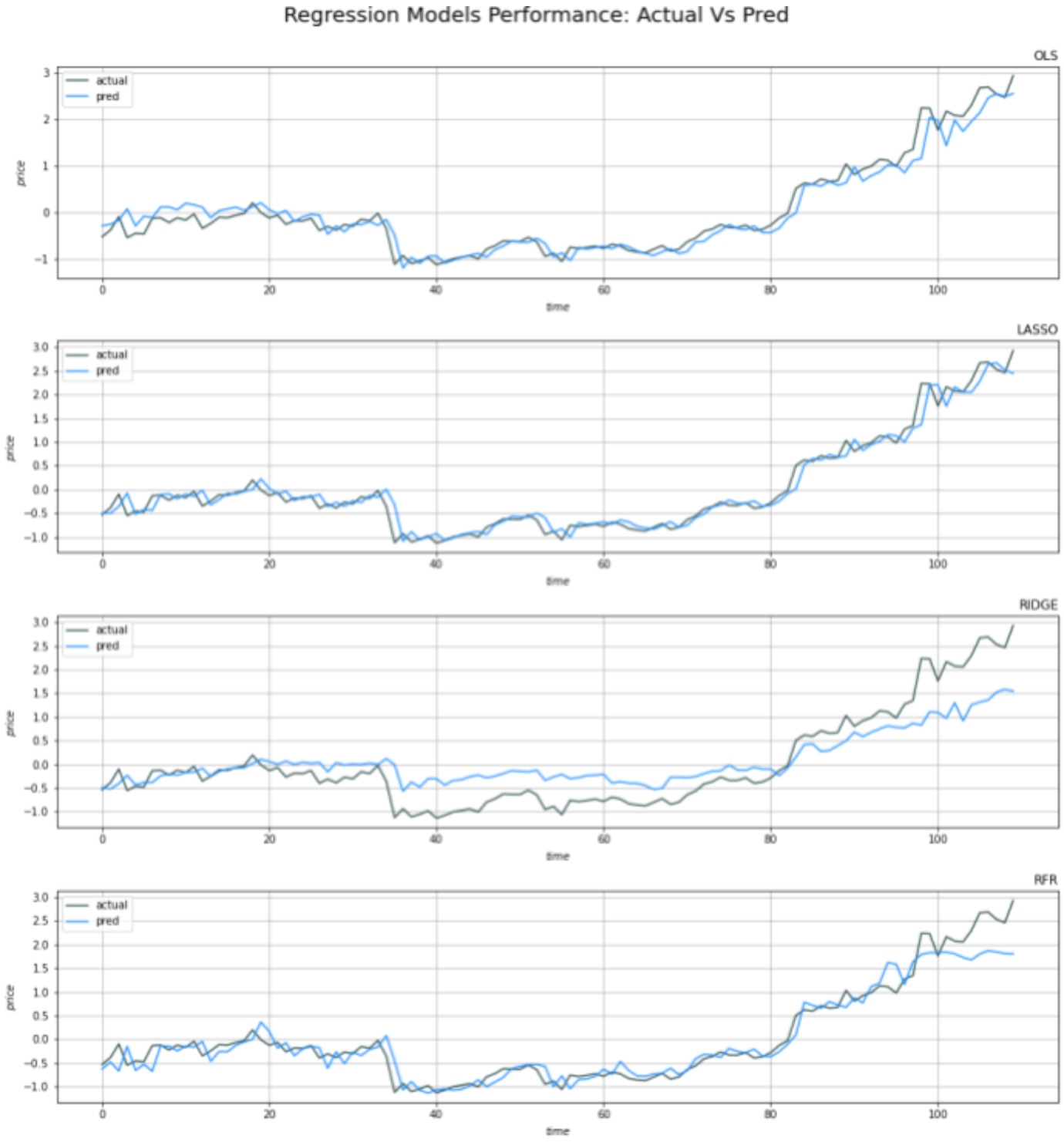
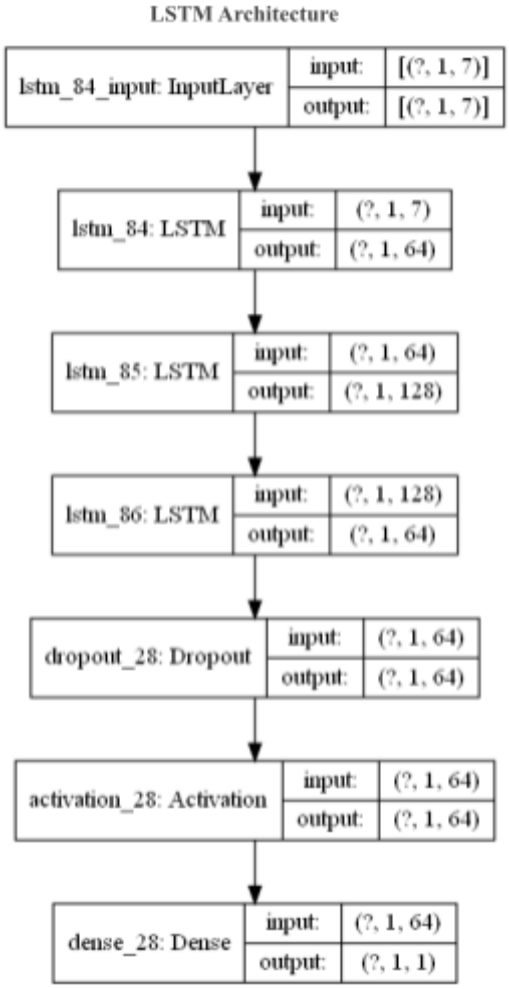


Figure 12.1 (regression models performance - test set)

(2.4.1) LSTM

As for the deep learning portion of our modeling, the following framework was used to produce an applicable model. We used the same exact dataframe that was originally compiled after EDA.



- **Process:**
  - Normalization using “MinMaxScaler()”
  - Add "predictor", timesteps of "1" day from "close"
  - Train, test split - 80% train (352 values) and 20% split (80 values)
  - Create LSTM model
  - Optimize through Hyperas
  - Return output of best performing model from given parameters
- **Sequential model layers:**
  - LSTM_1 (choice: 32, 64, 86, 128)
  - LSTM_2 (128) w/ L2 activity regularization
  - LSTM_3 (64)
  - Dropout (choice: range of 0 to 1)
  - Activation (choice: 'relu', 'sigmoid', 'linear', 'softmax', 'elu')
  - Dense (1)
- **Fitting params:**
  - Epochs (choice: 2, 4, 16, 32, 64, 72, 128, 1000, 1100, 1200, 1300, 1400, 1500, 1600, 1700, 1800, 2000, 2100, 2200, 2300, 2400)
  - Batch size (choice: 2, 4, 16, 32, 64, 72, 128)
  - Validation (X_test, y_test)
  - Callbacks (monitors "val_loss")
- **Loss metrics:**
  - MSE (mean squared error)
- **Optimizer:**
  - Choice (adamax, rmsprop, adam, sgd, nadam)

We used Hyperas to find the optimal parameters of the LSTM, however, some tuning was done manually. The parameter evaluation runtime was approx. 30-40 min. Hyperas then produced an output of the best model and best parameters that produced said model over the 20 evaluations. The compiled model with the best parameters achieved prediction values on our test set that had an R2 score of 0.9395 and an MAPE of 0.0246.

Best performing parameters:

“ `{'Activation': 1, 'LSTM': 1, 'batch_size': 3, 'epochs': 14, 'optimizer': 2, 'rate': 0.2257197809643675}` “

The output is available as a dictionary where the keys are names of the parameters, and values are the indexes that can retrieve the optimal values. Hyperas selected the activator at index “1” which was “sigmoid”. The next parameter is the LSTM layer's “hidden units” having an index of “1” which is 64 units. The units of the layer determined the output of the dimensions, meaning the best model had a vector of 64 dimensions. Next, “batch_size” follows with an index of “3”, which is an actual size of 32. Meaning the number of sequences the model trained at a time was 32. Referring to the time step of 1 day, since each batch moving forward is a “sequel” of the previous batch, this means that for every 32 batches, the last step was used to learn for the next batch. This is very useful for price prediction because markets move in a linear ordered fashion, and occasionally in pattern.

As for the number of “epochs”, at index “14”, Hyperas selected 1700 epochs to be optimal. Over the course of our 20 evaluations, when the model used 1700 epochs, it seemed to learn the best and produce the smoothest loss

improvement between validation and train sets. For our optimizer, the choice of index "2" was "adam". Adam is an algorithm derived from "adaptive moment estimation" and makes use of an exponentially decaying average of past gradients. It combines the best properties of the AdaGrad and RMSProp algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. As its learning rate is adaptive, it is relatively insensitive to hyperparameters, which is useful in this specific case.

Last, we have "dropout", Hyperas was given a choice to uniformly try all fractional numbers in between 0 and 1 and found the optimal dropout rate to be "0.22571978", meaning that the model was regularized by that amount where input and recurrent connections to LSTM units are probabilistically excluded from activation and weight updates while training. This layer prevents over-fitting while improving accuracy. Not shown in the Hyperas output as an addition to the second LSTM layer with 128 units was an activity regularizer using L2 regularization. The activity regularizer was useful as it penalizes the weights of the respective layers output. During the early stages of trials, we frequently ran into stagnation and gradient vanishing, as my weights were large. The regularizer helped maintain the layers output and minimize sporadic large weights.

We identified the parameters of the LSTM using the MSE (mean squared error) as our loss metric. After testing other metrics like MAPE or RMSE, for example, the MSE proved to have the most utility in accurately minimizing loss consistently. That being said, for each epoch of validation, we used the MSE on both the "X_test" and "y_test" set to see how the model performed from its respective training through "X_train" and "y_train".

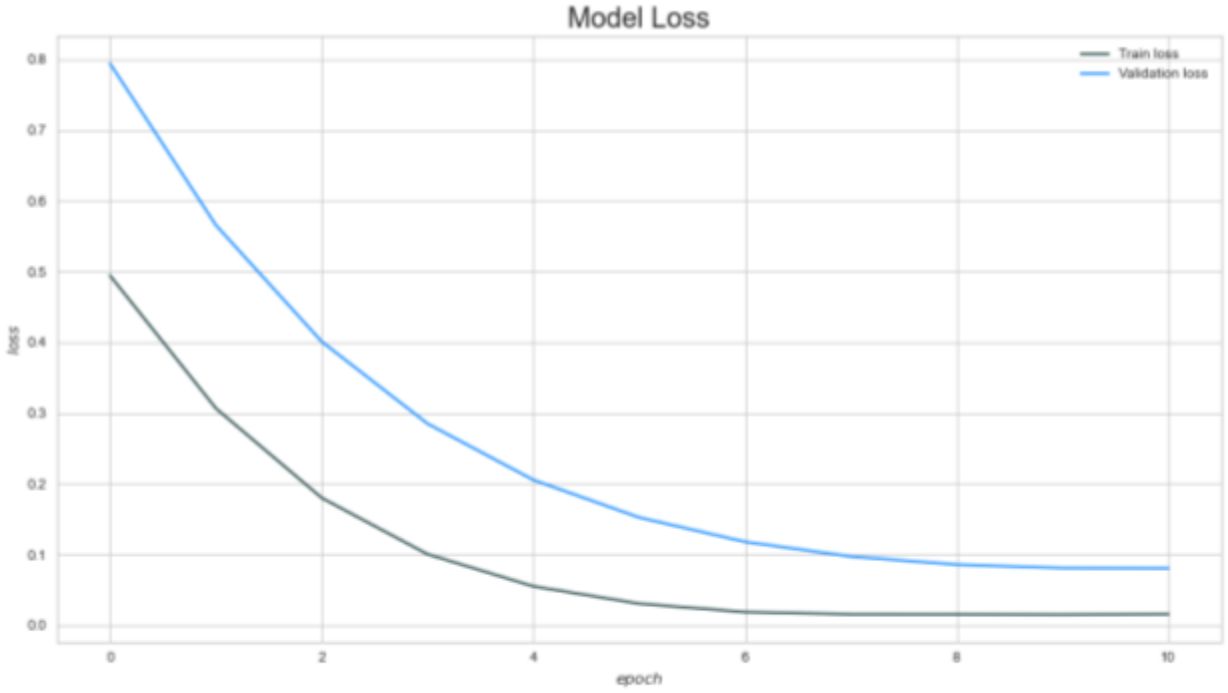


Figure 13 (LSTM loss plot, train vs validation)

As you can see in *Figure 13*, the best "validation loss" produced decreases as the "train loss" does. The larger error spread with validation is expected. The loss plot shows minimal signs of over-fitting. Over-fitting is not prominent, regarding our model, the loss stays relatively consistent, which means the model is true in "accuracy".

The next figure below *(Figure 14)* is a visualization of the LSTM models performance on the test set. The plot shows how accurate the prediction was and also shows minimal signs of over-fitting. If the model was extremely overfit, the prediction line would be almost a replica of the actual value line and would lack a realistic error spread.

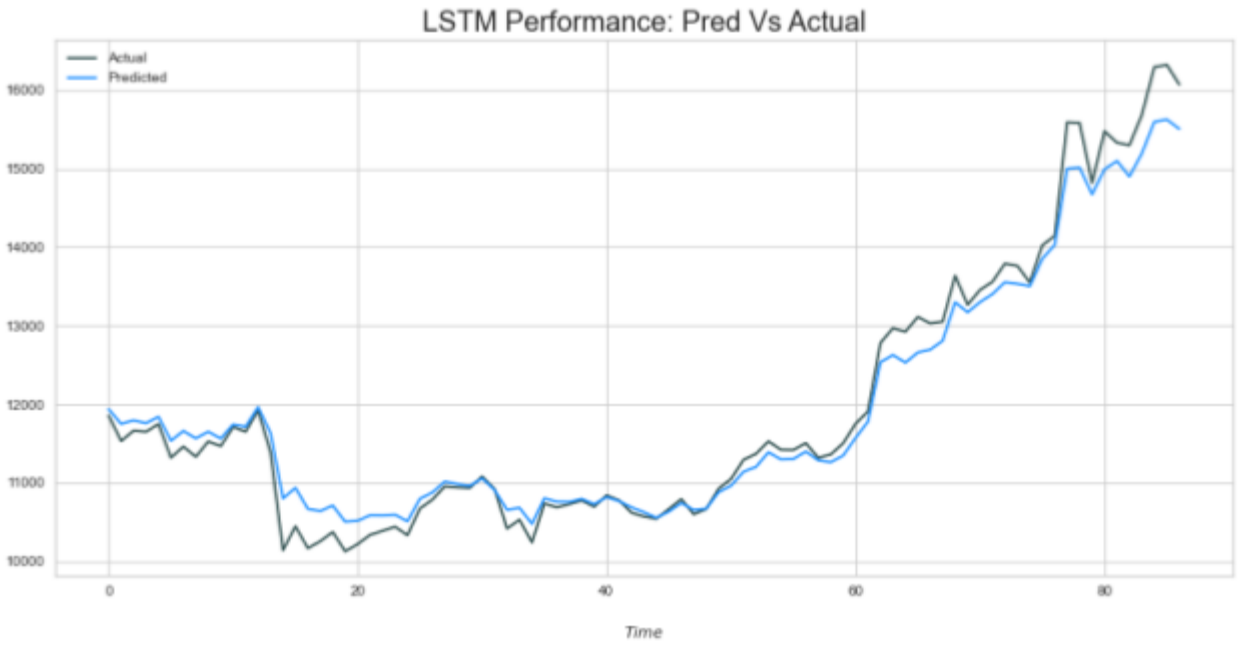


Figure 14 (LSTM model performance - test set)

The plot below in *Figure 15* visualizes the test set performance while showing the values the model was trained on. Looking at the legend the red line is the train values which was 80% of the original dataset, and the test values are visualized by the black/grey line which is 20% of the original dataset. Then the blue line is the predictions made by the LSTM same as what's shown in *Figure 14*.

Figure 14.1 (line plot visual of train set w/ test set and pred)

(3) Findings

(3.1) Comparative Analysis

Initially, when performing simple linear regression, we found the RFR model produced the most accurate predictions regarding BTCs closing price. However, the R2 score of the training set for the RFR was substantially lower (0.9172). This is a concern of the model's true accuracy, but upon further investigation we believe that this is due to the size difference of training/test sets. For all simple linear regression models we used a 75% train, 25% test split and since we separated it in a linear time series fashion, the RFR model clearly has more data values as inputs during training than testing. This notion was confirmed by computing its OOB (out-of-bag) score, which was 0.9136. Comparing that score to the train R2 (0.9172) we can see how similar they are. Being that OOB is an unbiased estimate of models performance on a test set the same size as the training, we can confirm that the model produces relatively true predictions, solidifying its test R2 score of 0.98845. Furthermore, by looking at the table below *(Figure 15)* the MAE, MAPE, MSE and RMSE were also computed. All of which were lower in value compared to the other simple regression models, with the MAE showing the most drastic difference in comparison. The MAE produced (0.06853) is realistic in that regard when comparing it to its MAPE (0.48569), showing minimal difference in value.

		R2 Test ▼	MAE Test	MAPE Test	MSE Test	RMSE Test	OOB Train
1	RFR	0.98845	0.06853	0.48569	0.01176	0.00686	0.91361
2	LSTM	0.9395	304.9840	0.0247	180,906.31	425.3308	N/A
3	OLS	0.93821	0.15942	0.55165	0.06179	0.00335	N/A
4	Lasso	0.93038	0.16465	0.79153	0.06962	0.00163	N/A
5	Ridge	0.75385	0.40099	1.51219	0.24615	0.08010	N/A

Figure 15 (model error scores - test set)

Out of the simple regression models, the OLS and Lasso frameworks produced nearly identical results. Their respective R2 testing scores were 0.93821 and 0.93038, differing by only 0.00783 points. However, when looking at their MAPEs we can see that the OLS received a much better score than Lasso by 0.23988 meaning the OLS model produced more accurate predictions compared to the actual values. The MAPE was more useful here than the rest of the metrics because of how similar the models were and how little the others differed in comparison.

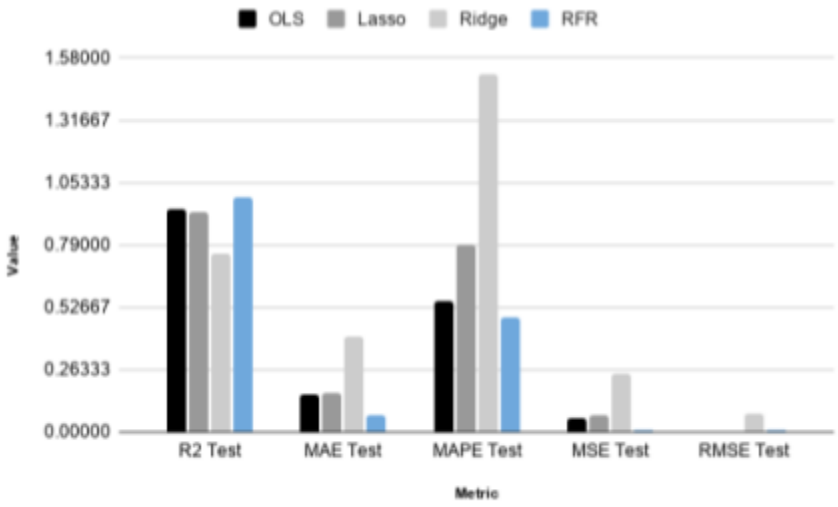


Figure 16 (simple linear regression model metrics)

The Ridge model performed the worst across the board, only receiving an R2 score of 0.75385 and a terrible MAPE in comparison to 1.51219. Meaning that this model's predictions differed from the actual values during testing by over 1% even with the data normalized and scaled. See *Figure 16* for a visualization of side-by-side model comparison of metric “size”, where Ridges poor performance is obviously prominent. Further, the RMSE is only a valid metric in this case to show the worst performing model because all derived values are significantly smaller than the Ridge.

As for the deep learning portion of the modeling, the LSTM was the second best in performance according to R2 scores (0.9395). Since we scaled the training and test sets back to their original values, the error outputs are much larger than the simple linear regression models. That being said, the MAPE metric was one that could be compared to the rest of the models, receiving a 0.0247, which is the lowest out of the five. As for the MAE (304.98), we can conclude that the average prediction by the LSTM was off from BTCs actual price by $305 rounded. We can justify its MAE by looking at the square root of the average of squared differences between its prediction and actual observation (RMSE, 425.33) since it is only approx. $120 in difference. Moreover, one might argue that the LSTM is more accurate than the top performing model (RFR) because of its MAE+RMSE and MAPE relative to actual price.

(3.2) Model Interpretability

All models except the LSTM could be interpreted well considering the heavy use of automation regarding tuning hyperparameters. The residuals, coefficients, and LIME explanations were calculated for each simple linear regression model. The same was calculated for the RFR, aside from coefficients, since "feature importance" is better suited for decision tree ensemble methods.

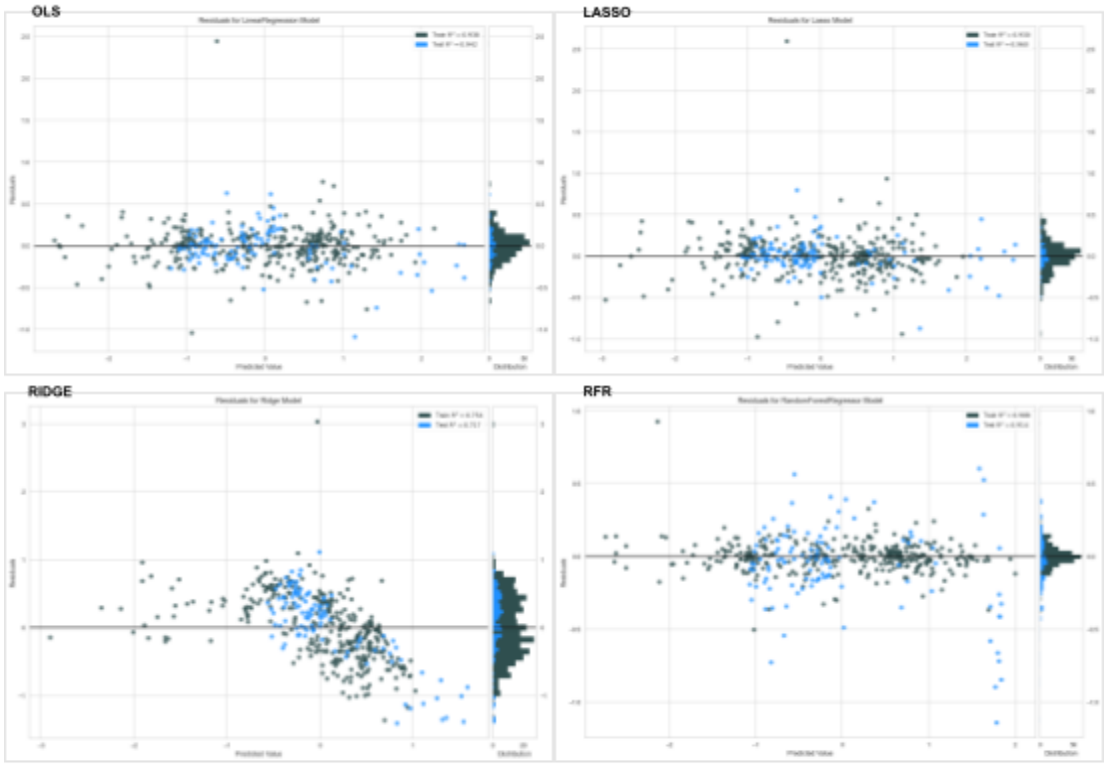


Figure 17 (residual plots, train/test)

Based on what's shown in *Figure 17*, the distribution of the residuals of errors is normal for both OLS, Lasso, and RFR, while Ridge has a slight right skewed distribution. The distribution of the RFR errors is not uniform with the test set in relation to the train set because of the disproportional size of the data split, however, during training the model maintained a relatively normal distribution. Looking at the Ridge models plot, we can see that it failed to minimize its errors during training by noting the decreasing residuals moving further from train residuals.

Moreover, careful examination of coefficients and LIME explanations, we could find the dominant feature amongst all models. Each result showed a different route to its predictions, but the single constant through each was the utilization of the feature "close". In *Figure 18*, we can see how the LIME explainer interpreted the respective models' predictions.

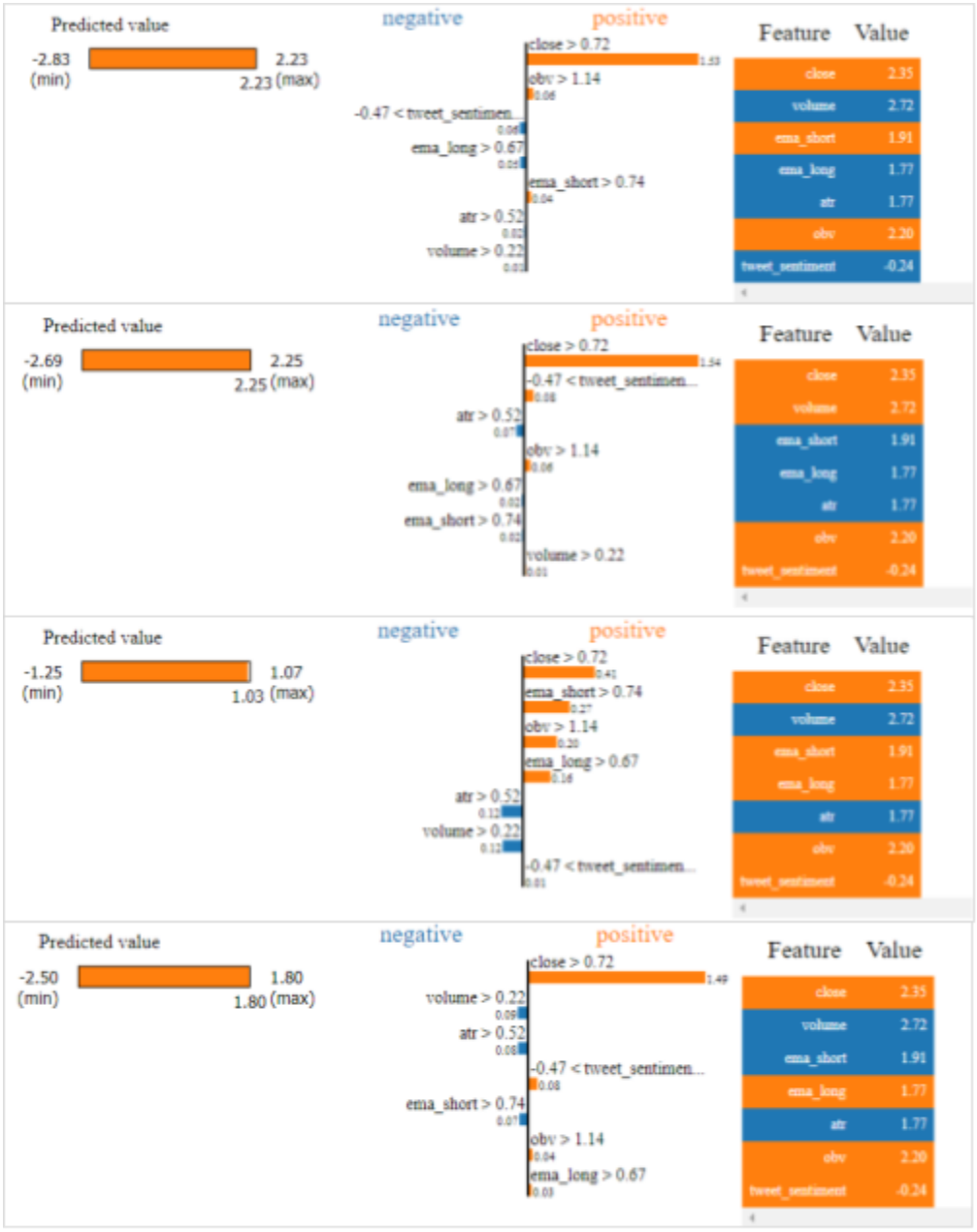


Figure 18 (LIME explanations, regression models)

In *Figure 18*, you can see that each explainer produced a different output regarding feature importance, however, "close" is the biggest factor in each model's prediction. That being said, it is interesting to see how certain variables that did not have any linear correlation (i.e. "OBV") during our EDA, yet influence the model. The LIME explainer produced nearly identical results compared to the OLS, Lasso, and Ridge models, all showing "close" with the largest absolute value, even when passed through "PolynomialFeatures()".

As for the RFR, "feature importance" revealed the same thing the LIME explainer did, showing us that "close" is once again the most important factor in the model's predictions. To the right, *Figure 20* contains each feature's value. To reiterate, "close" played such a large role in our modeling and actually confirmed the

		Importance
1	close	0.924208
2	atr	0.028656
3	obv	0.013587
4	ema_long	0.013533
5	volume	0.011165
6	ema_short	0.00675

Figure 20 (RFR "feature importance")

findings during our EDA regarding its correlation with other variables. It also relates to the market itself, revealing the importance of BTCs previous closing price and that one should not put as much weight into alternative forms of indicators like “ATR " or “OBV " when making trading or investing decisions. That being said, it is important to see how the models handled volatile price action. The volatile movement in price is usually the most unpredictable aspect within a given market, and the top models handled them well, on one of the most erratic assets nonetheless. For example, a large spike in volatility happened around time period 15 on the x-axis of Figure 12.1 (RFR and OLS) and Figure 14 (LSTM). RFR and OLS failed to predict the price completely during that sell-off, but their predictions were spot on just before the drop happened. With the LSTM, the same thing took place, its price prediction holding more steady than the other two models but interestingly enough the predicted values just before are nearly 100% accurate. By identifying patterns like this within the models, a trader or investor can make plans or plays ahead of the volatile spike, presenting many opportunities for maximizing profitability. Likewise, if added to a previous analysis, one can generate confluence, signaling a potential successful play with minimal risk.

(4) Conclusions

Overall, based on our findings, the models proved to be a success with using linear regression, random forest regression, and neural network models to predict each day's future price of BTC. In fact, when analyzing the performance reports of the top three models (RFR: R2 - 0.988, OLS: R2 - 0.938, and LSTM: R2 - 0.939), not only is there potential in predicting the price for the next day, but they could also speculate further and improve overall fundamental or technical analysis. The business problem regarding how to improve analysis by predicting price is more than achievable. We can present them as versatile components to every trader/investor's arsenal when it comes to maximizing net profit. We can see that the LSTM model produced an average error in predictions vs actual values of roughly $300 (minimal loss depending on capital), however, by looking at its line plot we can see nearly identical predictions before each large move up or down presenting virtually unlimited speculative plays.

The potential in these methods for predicting price does not stop with the results. It should be noted that the top models were trained and tested on only 440 data points. This is an insignificant number in comparison to the multitude of data available for BTC and even traditional financial assets or products. BTCs historical data goes back to 2010, and since we now know that the “close” price has the most predictive value we can now try different combinations of other highly correlated variables or indicators. Further investigation of other variables parallel to the possibility of training these models on larger amounts of data means the success rate can only increase in the future. Traders, investors and even institutions can certainly find use of the analysis as a whole.

(4.1) Future Work

- Perform deeper analysis into Twitter tweet sentiment
- Train/test models on more data
- Fine tune the hyperparameters of models
- Perform EDA with other indicators
- Implement signal generator
    - Back test (compute prospective Sharpe value)
- Test for replicable results on different assets (equities, ETFs, etc.)

(5) Client Recommendations

- Use top performing models as a signal generator in conjunction with own due diligence
    - Minimizes risk
    - Provides access to “blind spots” or unseen patterns or trends
    - Can confirm previous ideas regarding price action
- Implement automated strategy to run LSTM as its own “advisor” or “HF trader”
    - Passive approach to generate profit
    - High risk, less human decision/influence
- Use findings to improve previous analysis or speculation for identifying potential volatile spikes
    - Minimizes risk
    - Ability to create confluence of factors

(6) Consulted Resources

- Bitcointreasuries.org. 2021. *Bitcoin Treasuries in Publicly Traded and Private Companies - List of large holders*. [online] Available at: <https://bitcointreasuries.org>.
- Brownlee, J., 2021. *Machine Learning Mastery*. [online] Machine Learning Mastery. Available at: <https://machinelearningmastery.com>.
- Ferrando, P., 2021. *Understanding how LIME explains predictions*. [online] Medium. Available at: <https://towardsdatascience.com/understanding-how-lime-explains-predictions-d404e5d1829c>.
- GeeksforGeeks. 2021. *GeeksforGeeks | A computer science portal for geeks*. [online] Available at: <https://www.geeksforgeeks.org>.
- GitHub. 2021. *Twint*. [online] Available at: <https://github.com/twintproject/twint>.
- GitHub. n.d. *Lime: Explaining the predictions of any machine learning classifier*. [online] Available at: <https://github.com/marcotcr/lime>.
- Hyndman, R., 2021. *Hyndsight Blog*. [online] Robjhyndman.com. Available at: <https://robjhyndman.com/hyndsight/>.
- Ipython.readthedocs.io. 2021. *IPython Documentation — IPython 7.21.0 documentation*. [online] Available at: <https://ipython.readthedocs.io/en/stable/>.
- Matplotlib.org. 2021. *Matplotlib: Python plotting — Matplotlib 3.3.4 documentation*. [online] Available at: <https://matplotlib.org/stable/>.
- Molnar, C., 2021. *Interpretable Machine Learning*. [online] Christophm.github.io. Available at: <https://christophm.github.io/interpretable-ml-book/>.
- Numpy.org. 2021. *NumPy*. [online] Available at: <https://numpy.org>.
- Pandas.pydata.org. 2021. *pandas - Python Data Analysis Library*. [online] Available at: <https://pandas.pydata.org>.
- Pumperla, M., 2021. *Hyperas: Keras + Hyperopt: A very simple wrapper for convenient hyperparameter optimization*. [online] Maxpumperla.com. Available at: <http://maxpumperla.com/hyperas/>.
- Saabas, A., 2021. *Diving into data | A blog on machine learning, data mining and visualization*. [online] Blog.datadive.net. Available at: <http://blog.datadive.net>.
- Saini, B., 2021. *Hyperparameter Tuning of Decision Tree Classifier Using GridSearchCV*. [online] Artifical Intelligence in Plain English. Available at: <https://ai.plainenglish.io/hyperparameter-tuning-of-decision-tree-classifier-using-gridsearchcv-2a6ebcaffeda>.
- Scikit-yb.org. 2021. *Yellowbrick: Machine Learning Visualization — Yellowbrick v1.3.post1 documentation*. [online] Available at: <https://www.scikit-yb.org/en/latest/>.
- Sklearn.org. 2021. *scikit-learn: machine learning in Python — scikit-learn 0.19.1 documentation*. [online] Available at: <https://sklearn.org/index.html>.
- Stack Overflow. 2021. *Stack Overflow - Where Developers Learn, Share, & Build Careers*. [online] Available at: <https://stackoverflow.com>.
- StackExchange. 2021. *Cross Validated*. [online] Available at: <https://stats.stackexchange.com>.
- Statistics How To. 2021. *Statistics How To: Elementary Statistics for the rest of us!*. [online] Available at: <https://www.statisticshowto.com>.
- Statsmodels.org. 2021. *Introduction — statsmodels*. [online] Available at: <https://www.statsmodels.org/devel/>.
- Team, K., 2021. *Keras: the Python deep learning API*. [online] Keras.io. Available at: <https://keras.io>.
- TensorFlow. 2021. *TensorFlow*. [online] Available at: <https://www.tensorflow.org>.
- Textblob.readthedocs.io. 2021. *TextBlob: Simplified Text Processing*. [online] Available at: <https://textblob.readthedocs.io/>.