Machine Learning Nanodegree
Capstone Project

Arun Dass

February 6 2018

Definition

Project Overview

Cryptocurrencies are quickly becoming a mainstream topic. With the high volatility and massive growth in the cryptocurrency market, there has been countless millionaires created. With this industry being such a new phenomena, it is the optimal time to make financial decisions that could benefit you for years to come.

The way that machine learning works is that a model or algorithm will learn from previous and current data to predict a future object, that could be classification of a object or prediction of a future value. Having the ability to predict a future price of a currency would give a investor an upper hand in his financial goals.

A cryptocurrency, by definition , *is a digital currency in which encryption techniques are used to regulate the generation of units of currency and verify the transfer of funds, operating independently of a central bank.*

A similar project was done by Jakub Aungiers, Aungiers used an LSTM network to predict the next day price of the S&P 500 using the previous 50 days timestep. Aungiers project can be found in the following link, http://www.jakob-aungiers.com/articles/a/LSTM-Neural-Network-for-Time-Series-Prediction.

Also, Nikola Milosevic used a Gated Recurrent Unit to predict the price of Apple and Microsoft. Milosevic's work can be found in the following link, http://inspiratron.org/blog/2018/01/10/predicting-stock-prices-using-recurrent-neural-networks/.

Problem Statement

The objective of this project is to determine if a machine learning approach will provide a positive result in determining the price fluctuations in short term cryptocurrency price change. With the large amount of factors that may contribute to flush fluctuations, it is hard to distinguish certain patterns without the help of scientific models. Machine learning appears to be a great approach to solving this problem as it can find patterns in multidimensional data, that would otherwise be very hard to discover.

The type of problem that I will be solving is a multidimensional regression problem. I will be using two types of RNN models called the LSTM and GRU to solve this problem. The features that will be input into the two RNN models are, the open price, close price, daily high, daily low, percentage change in price, and google trend data.

Preprocessing the data will involve transforming the multidimensional data frame into a array of a time step of 50 days. A time step is the amount of days previous to the prediction.. The output prediction will be 5 days into the future of the last time step.

Both the GRU and the LSTM will be 4 layer networks with neurons of each layer being 50,100,100,100 as well as 1 dense output layer.

Metrics

RMSE is the standard deviation of prediction error and is a measure of how far the data points are from the regression line.

The formula to calculate the RMSE is as follows,

$$RMSE = \sqrt{}(f - o)^2$$

Where:

f = forecasts(expected values or unknown results)

o= observed values (known results)

Root mean squared error is a variance of the mean squared error(MSE). MSE is a statistics tool that measures the squares of the errors of an estimator, Let's back up and define what an estimator is, an estimator is a procedure for estimating an unobserved quantity. Therefore what the MSE does is that it measures the difference between the estimator and what is estimated. The MSE measures the quality of an estimator, values closer to 0 are considered better while values closer to 1 are considered worse.

The difference between RMSE and MSE is that the RMSE surrounds the MSE function with a square root. Taking the square root of the MSE has some interesting implications. Since the errors are squared before they are averaged, the RMSE gives a high weight to large errors. This means that the RMSE is more useful when large errors are undesirable. This is perfectly fit for this application, as large errors are something that should been as very undesirable when predicting future prices.

Analysis

Data Exploration

The data used for this project was taken from Kaggle.com, where i retrieved the Ethereum, Bitcoin and Ripple daily High, Low, Close, Volume and Market Cap values from each currencies inception until Nov 7/2017.

In the figure 1 below, you can see the normalized closing prices for Ethereum, Bitcoin and Ripple. From the plot we can see that major growth in the price of all three currencies occurred around the time of May 2017. Bitcoin is the currency which had the most constant growth, while Ripple experienced the biggest spike in price as well as correction.

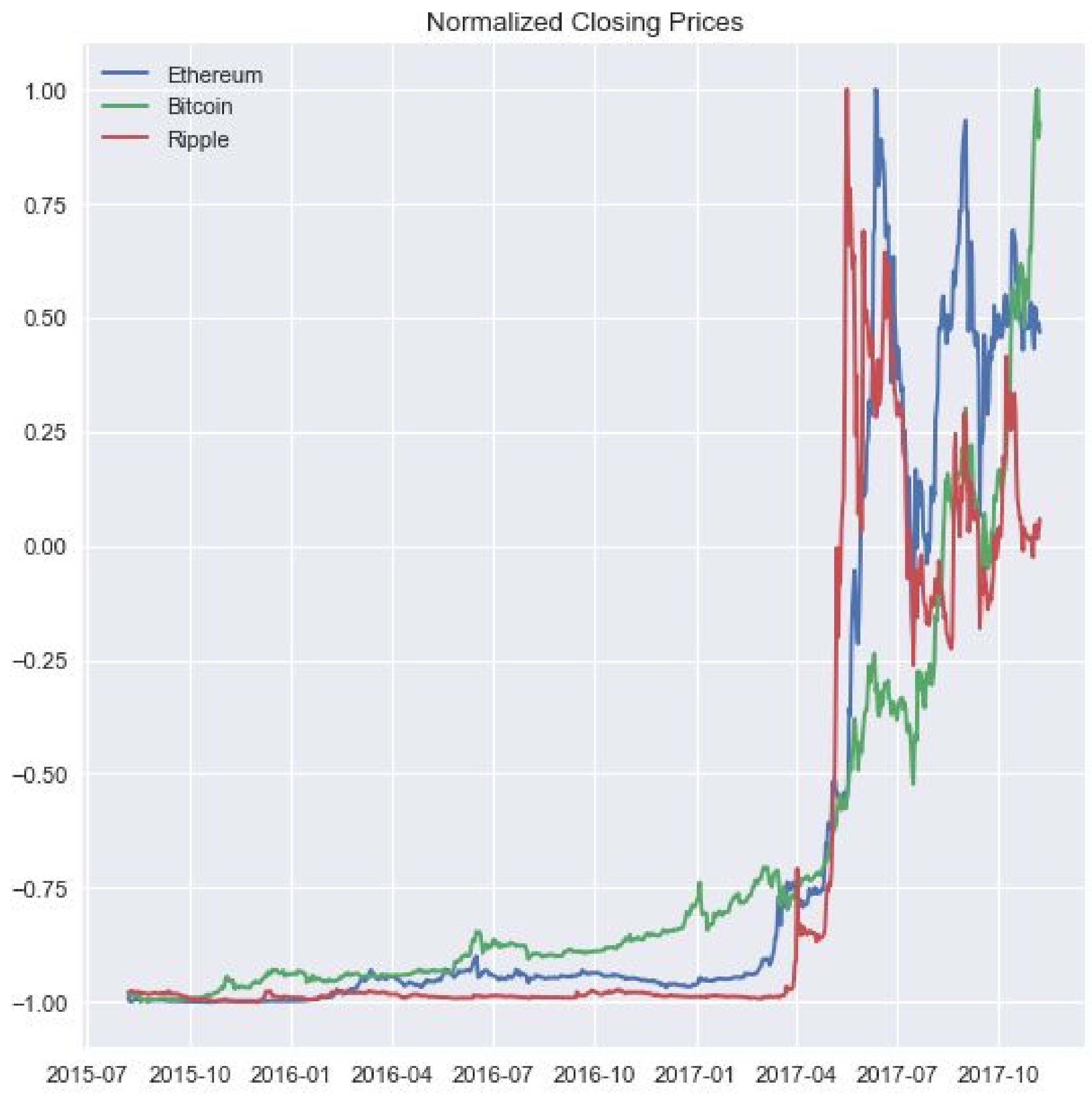


Figure 1: Normalized Close Prices for Ethereum, Bitcoin and Ripple

Figure 2 below shows the percent change in price of the three cryptocurrencies plotted on the same graph. The graph is a good visual representation of the volatility of each cryptocurrency. Overall it appears that Ethereum is constantly the most volatile, while Ripple began to become extremely volatile around April 2017. This can be seen in the close to 200% spike in price around the April 2017, followed by the 50% correction.

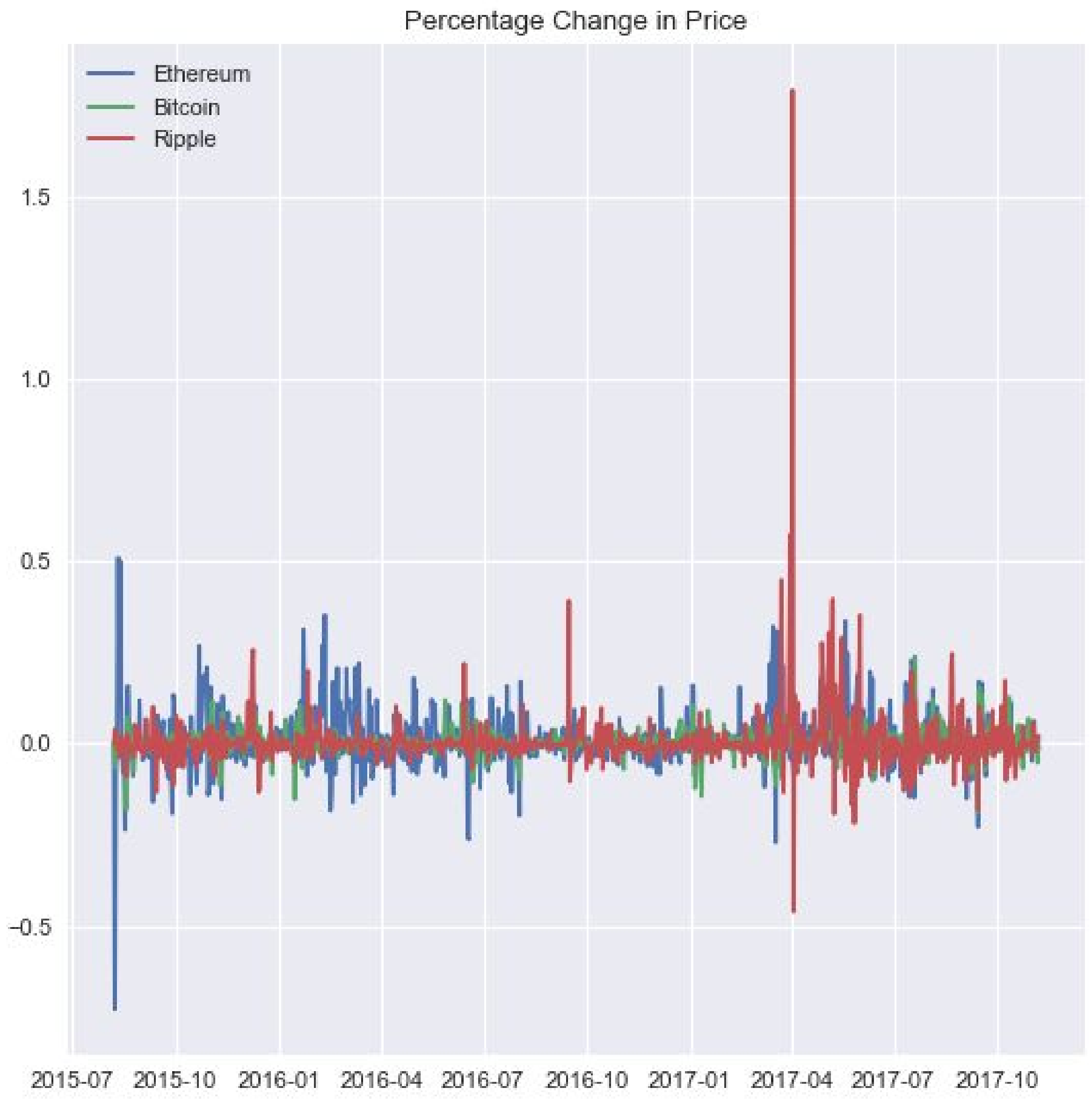


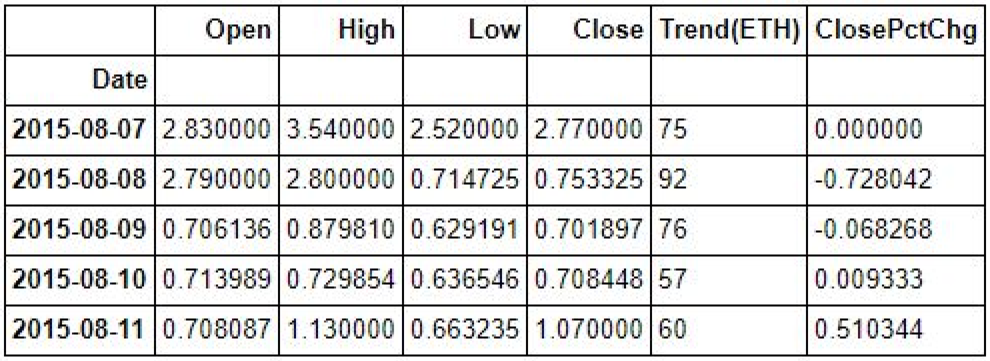
Figure 2: Percentage Change in Price

Along with the Kaggle dataset, I used the Pytrends API to pull google trend data for all currencies to see if there is any correlation between the search trench of the terms Ripple or XRP, Ethereum or Ether and Bitcoin or BTC. Some very interesting correlations appeared from the closing price and google trend data.

Currency	Correlation
Ethereum	0.15
Ripple	-0.45
Bitcoin	0.04

Figure 3a: Correlation of Close Price and Trend Data

As a result, interesting data appeared from checking the correlation between the closing price and google trend data for each respective cryptocurrency. Bitcoin has a very low correlation at 4%, This is unfortunate as I was hoping that the trend data would have some impact on predicting the future price. Ethereum had a much larger correlation than Bitcoin did at 15%, This result is promising and should have a positive impact on my prediction. The most interesting result was for Ripple, which had a -45% correlation. This is very interesting because of 1) the magnitude, which is substantial and 2) the negative sign, which means that when the google trend data was high the price was low and when the trend data was low the price went high. This observation could mean that when there is a lull in interest in Ripple, there is a higher chance that the price could spike.



	Open	High	Low	Close	Trend(ETH)	ClosePctChg
Date						
2015-08-07	2.830000	3.540000	2.520000	2.770000	75	0.000000
2015-08-08	2.790000	2.800000	0.714725	0.753325	92	-0.728042
2015-08-09	0.706136	0.879810	0.629191	0.701897	76	-0.068268
2015-08-10	0.713989	0.729854	0.636546	0.708448	57	0.009333
2015-08-11	0.708087	1.130000	0.663235	1.070000	60	0.510344

Figure 3b: Ethereum RNN Dataset

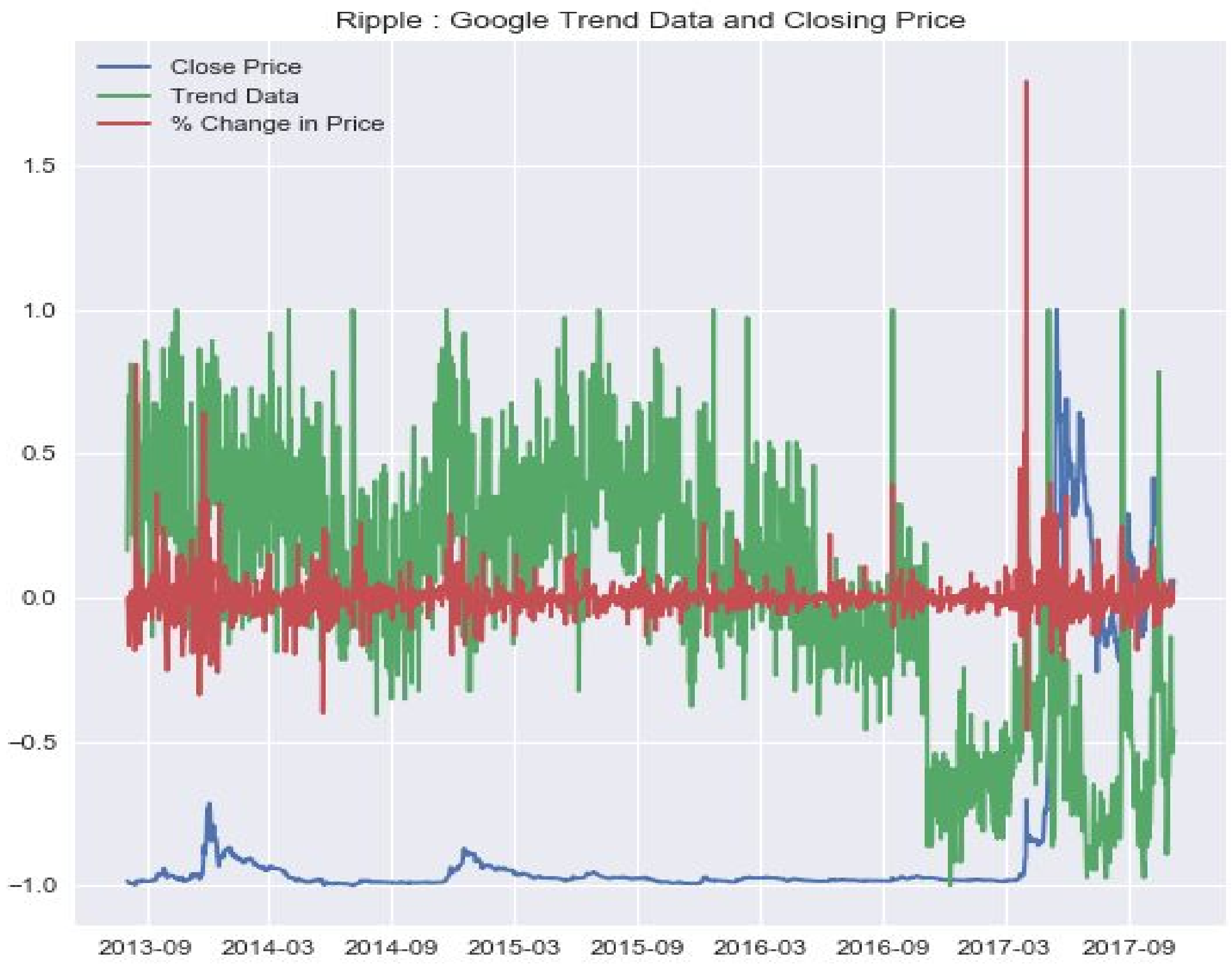


Figure 4: Ripple: Google Trends vs Normalized Price vs Percent Change in Price

See Appendix A for Google trend graphs of Ethereum and Bitcoin

Exploration using FB Prophet

Prophet is a time series forecasting tool created by Facebook's core data science team.

Prophet is an additive regression model with four main components

1) A piecewise linear or logistic growth curve trend
2) A yearly seasonal component modeled using Fourier series
3) A weekly seasonal component using dummy variables
4) A user-provided list of important holidays

The interesting use for Prophet is that it provides future forecast as well as weekly and yearly seasonality. This is very useful in determining what part of the week or year is most optimal to buy based on past price trend.

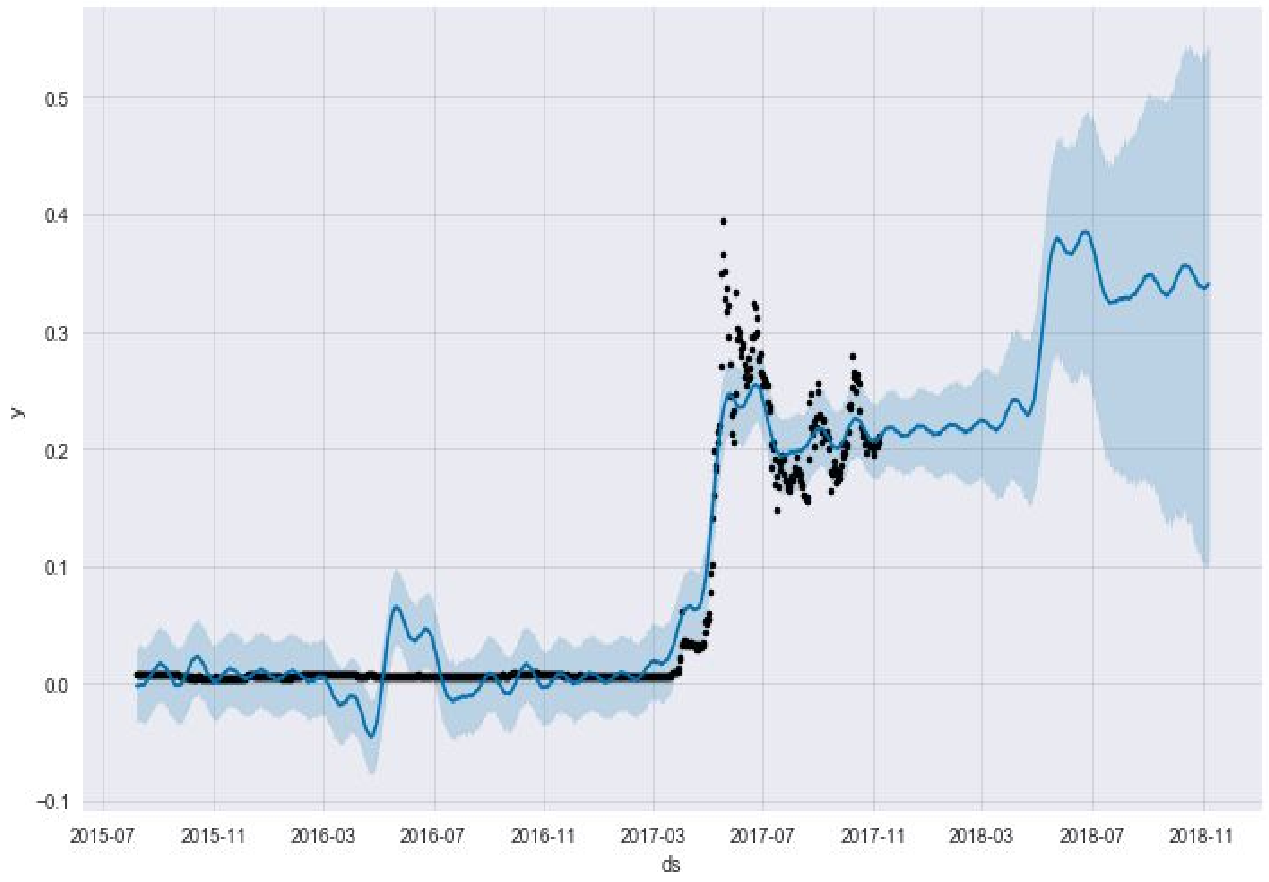


Figure 5a: 1 year forecast of Ripple using FB Prophet

The 1 year forecast for ripple predicts a potential spike in the months of April and May. The farther along the prediction is, the bigger the margin of values will be. This can be seen by the large amount of light blue fog around the 2018-07 to 2018-11 range. The model forecasted over the entire dataset, and interestingly predicted a large price fluctuation in the 2106-05 range, while the ground truth data showed minimal movement.

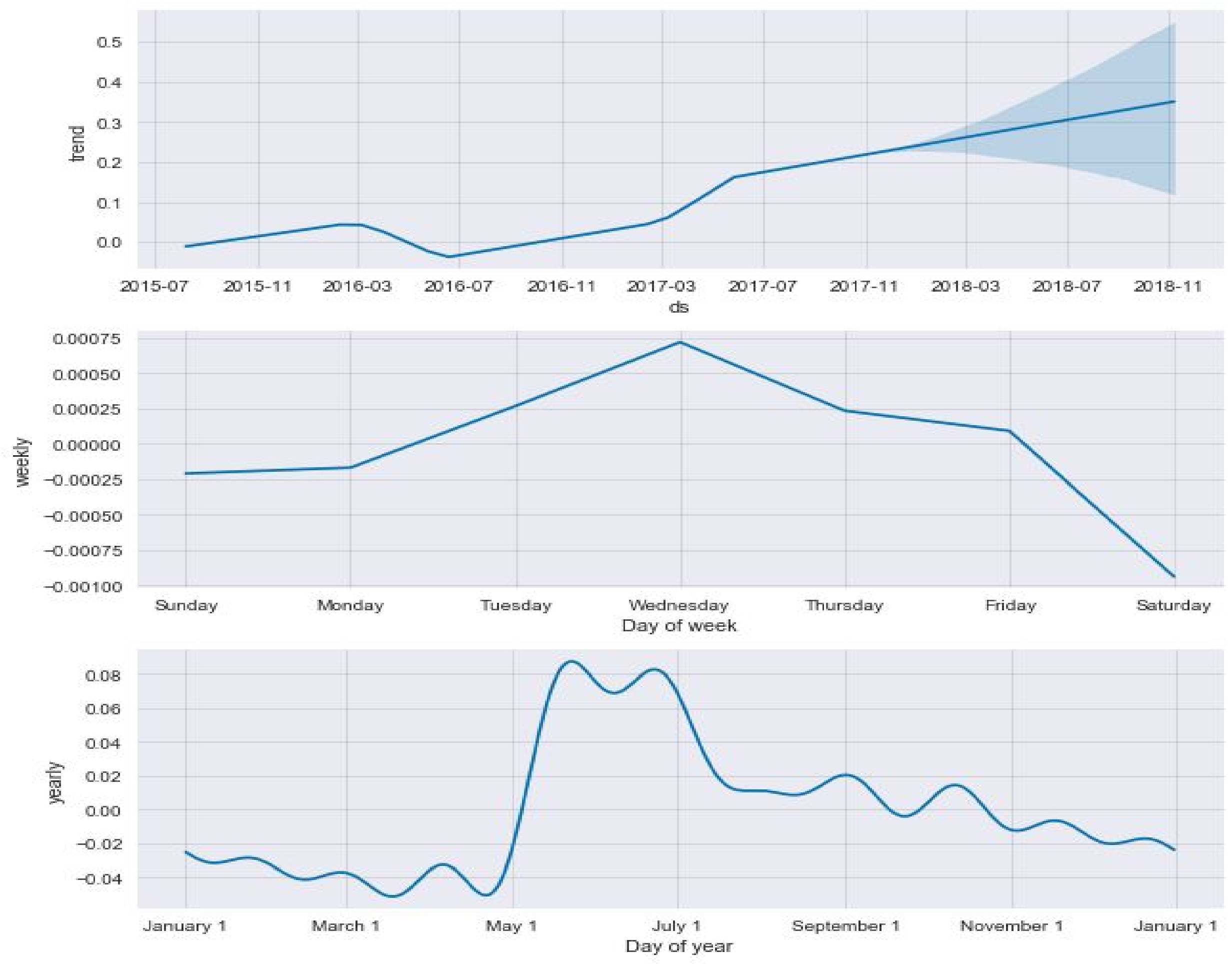


Figure 5b: Seasonality of Ripple using FB Prophet

The graphs above show the overall trend of the price as well as the weekly and yearly seasonality. The overall trend is upward which is positive for anyone who is a long term investor in the currency. The yearly and weekly seasonality graphs produce interesting results. The weekly seasonality shows a steady price progression from Monday to Wednesday. Then is a steep drop until Saturday. This graph provides insight into which days would be best to buy into the currency and which day would be best to sell. The yearly seasonality is also very interesting, it shows that majority of the price increase occurs over a few months starting in May, then a correction all the way to April.

See Appendix B for Prophet forecasting charts for Ethereum and Bitcoin

Algorithms and Techniques

I will be using two time series forecasting models to predict the future closing price of the three cryptocurrencies chosen. Both models are part of the recurrent neural network family. A recurrent neural network is a class of the artificial neural network, where connections between units form a directed cycle. RNNs are especially useful with sequential data as they can use their internal memory to process sequences.

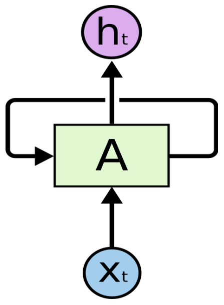
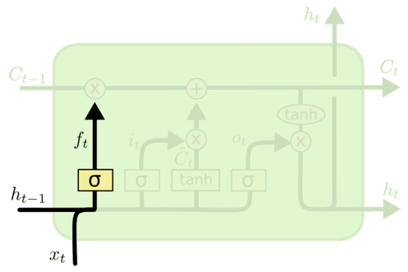


Figure 5a: RNN has loops

The first model is the Long Short Term Memory RNN, we will call it the LSTM for short. The LSTM was invented in 1997 by Hochreiter and Schmidhuber and avoids the vanishing gradient problem. The basic idea behind the LSTM is that it contains information outside the normal flow of the recurrent network in a gated cell. The information can be manipulated much like how data in a computer memory is. It can be stored in, written to and read from the cell. The cell decides what to store and when to allow read and writes via gates that open and close. The gates are analog and are operated by element wise multiplication of sigmoid functions.

Walk Through of an LSTM

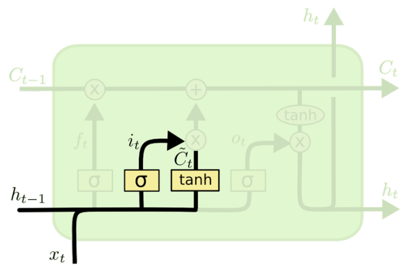
The first step of the LSTM is to decide what information to throw away using the forget gate layer. The forget gate layer looks at xt and ht-1, outputs a number between 1 and 0 for each number in the cell state ct-1. A 0 represents ‘remove this completely’ while a 1 represents ‘completely keep this’. See figure 5b.



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

Figure 5b

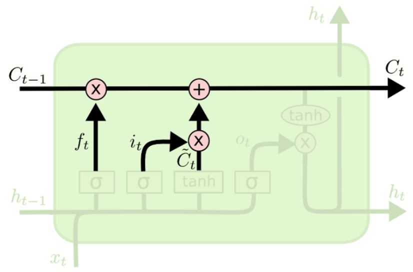
The second step is to figure out what information we are going to store in the cell. This can occur in two parts. First, the input gate layer decides which value it will update. Next a tanh layer creates a vector of new candidate ct values, that could be added to the state.See figure 5c



$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

Figure 5c

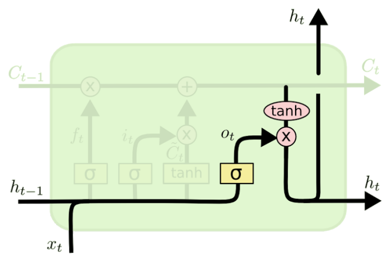
In this next step we update the old cell state ct-1, into the new ct cell state. We multiply the cold state by ft, to forget the things we wanted to in the earlier step. Then we add it*Ct, and the result is the new candidate values. See figure 5d.



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

Figure 5d

Lastly, we decide what we are going to output. The output is based on a filtered version of our cell state. First , we run a sigmoid layer which decides what parts of the cell state we are going to output . Next we put the cell state through a tanh and multiply it by the output of the sigmoid gate to only output the parts we decide. See figure 5e.



$$o_t = \sigma\left(W_o \ [h_{t-1}, x_t] \ + \ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

Figure 5e

LSTM figures sourced from http://colah.github.io/posts/2015-08-Understanding-LSTMs/

The second model used is the Gated Recurrent Unit or GRU. The GRU is a variant of the LSTM and was invented in 2014 by K Cho. It retains the same resistance to the vanishing gradient problem as the LSTM but has a simpler internal structure which allows for faster training.

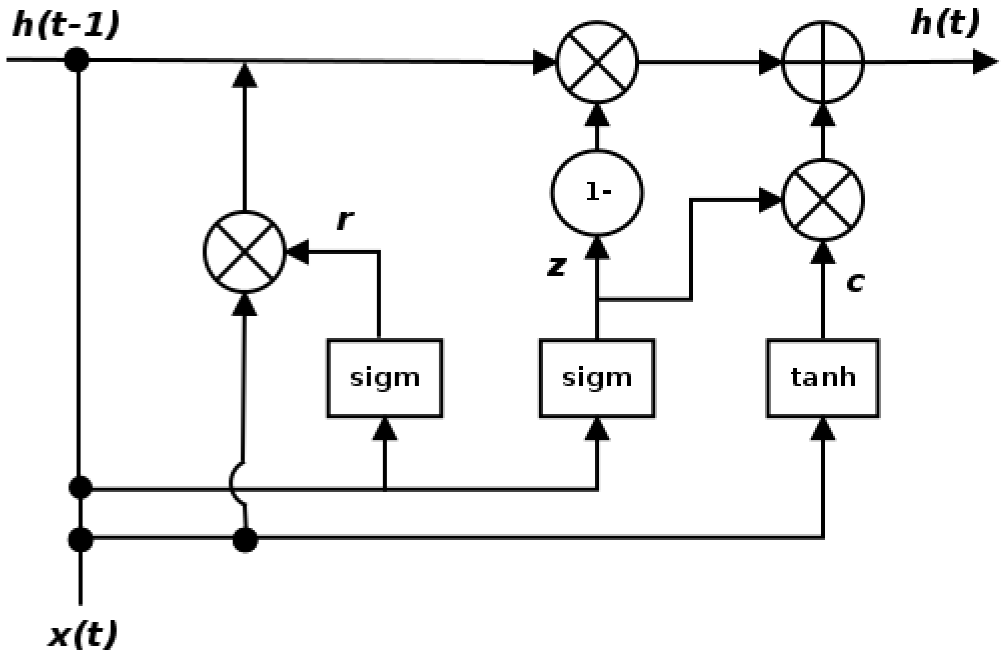


Figure 5d: GRU Diagram

GRU figures sourced from https://www.packtpub.com/mapt/book/big_data_and_business_intelligence/9781787128422/6/ch06lvl1sec44/gated-recurrent-unit--gru

Instead of th input, forget and output gates in the LSTM cell, the GRU has two gates, an update gate and a reset gate. The update gate defines how much previous memory to keep around and the reset gate defines how to combine the new input with the previous memory. The equations used in the GRU are as follows,

$$z = \sigma(x_t U^z + s_{t-1} W^z)$$
$$r = \sigma(x_t U^r + s_{t-1} W^r)$$
$$h = tanh(x_t U^h + (s_{t-1} \circ r) W^h)$$
$$s_t = (1 - z) \circ h + z \circ s_{t-1}$$

Z and R are equations for the gating mechanisms, while h and s(t) represent the model.

Benchmark

I will be comparing my RNN models result to a simple linear regression benchmark. The benchmark will be of the same prediction size as the RNN models at 0.06% percent of the dataset. The values will also be normalized to a range of 0,1 using MinMaxScaler to match the RNN RMSE normalization.

	Ethereum	Ripple	Bitcoin
Benchmark RMSE	0.18	0.21	0.13

Figure 6a: Benchmark Table

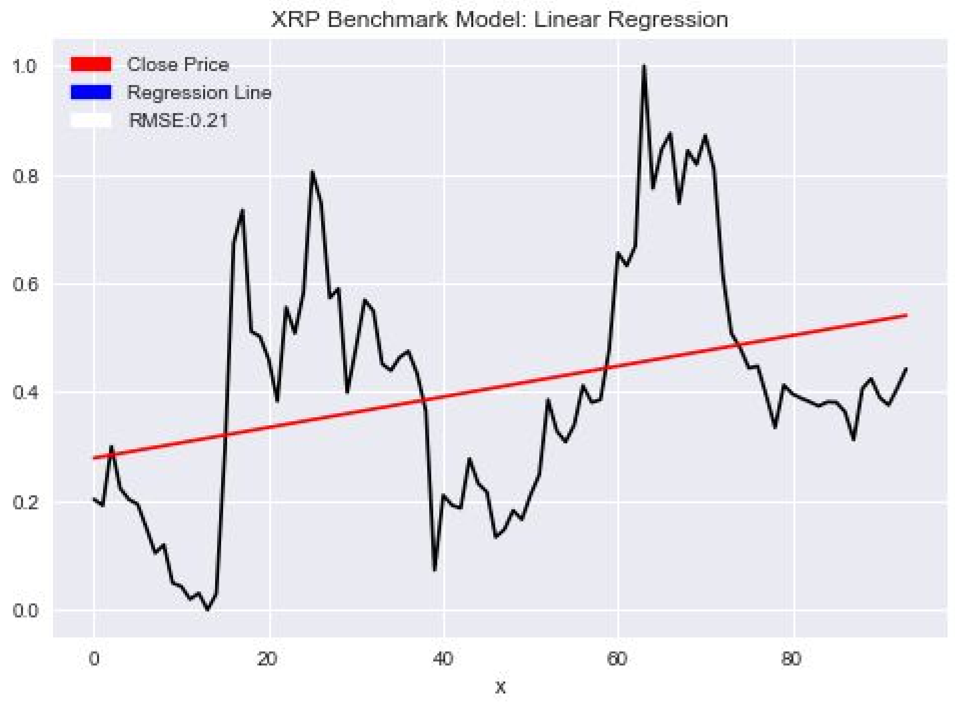


Figure 6b: XRP Benchmark

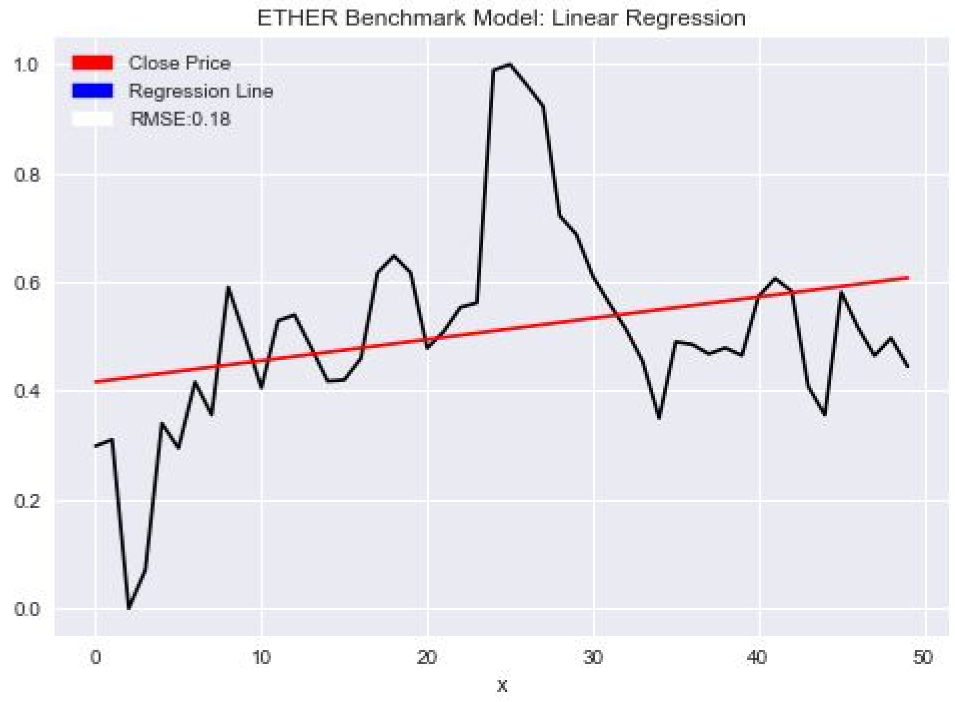


Figure 6c: Ether Benchmark

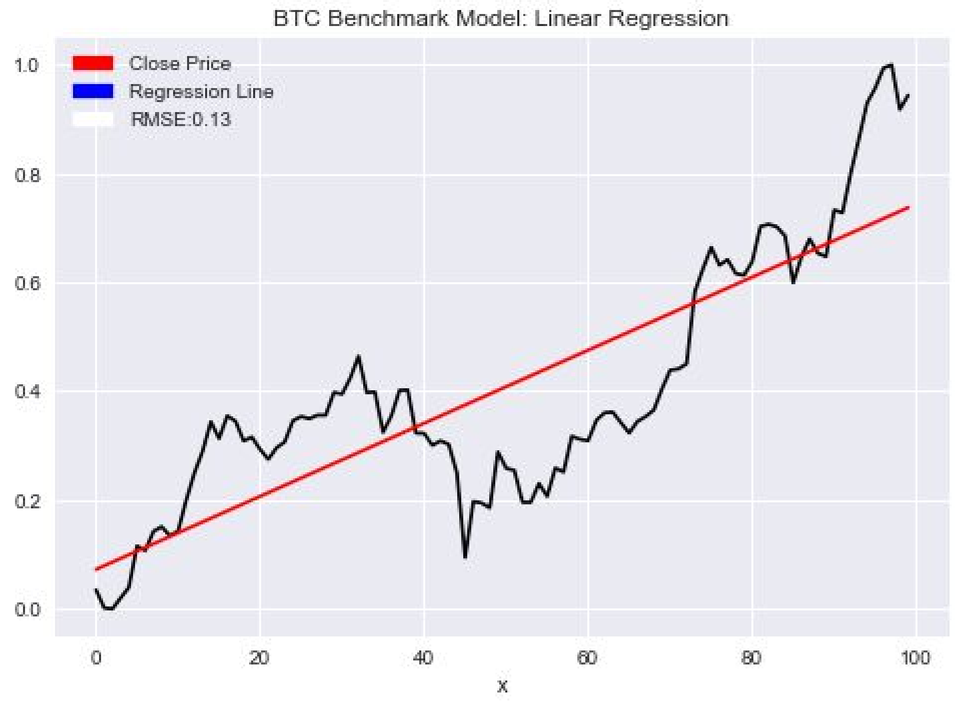


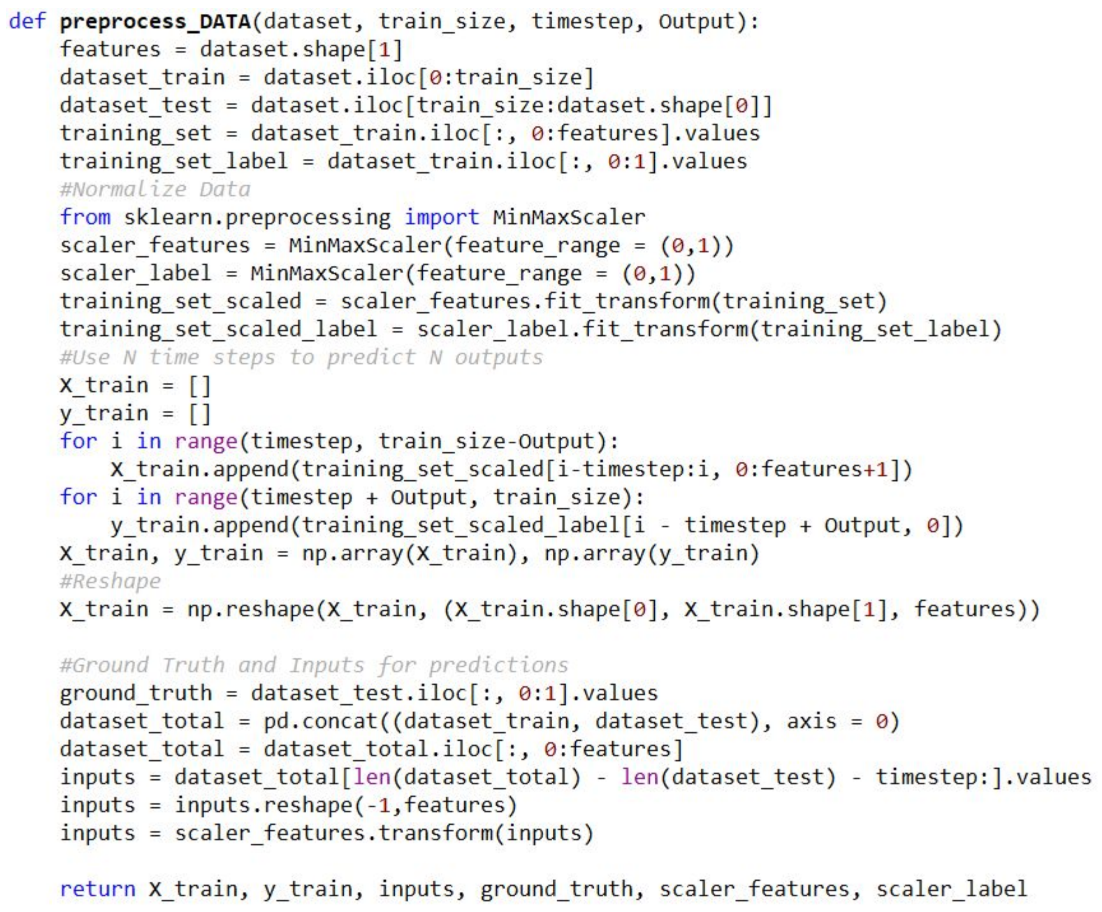
Figure 6d: BTC Benchmark

Data Preprocessing

Facebook's Prophet is a single variable forecaster, which dates in the date labelled as 'ds' and the value you want to forecast in my case the closing price data, as 'y'. From the data I obtained from the Kaggle dataset, I have 6 categories, Open, High, Low , Close, Volume and Market Cap along with the dates. At first, the data preprocessing for this was very simple. It involved using the pandas library and removing all the columns that I did not want to forecast. However, I ran into two issues, The first issue occured when the data was structured so that the newest data was at the top of the pandas dataframe and I required the new data to be at the bottom. This was needed so that the dataset intuitively showed the price change forward through time. This issue was an easy fix which involved reindexing the dataset in reverse order and then setting the date as the index column. This was completed using only the pandas library. The second issue was that the dates would not show up on my Matplotlib graphs. This was due to a formatting issue which required the dates to be shown as datetime values. I used the index_to_pydatetime() function to convert the dates to datetime so that they showed up on the graph.

Using the LSTM/GRU network involved a lot more preprocessing than was needed for the previous Prophet tool. Since the LSTM was used as a multivariable time series predictor, the first step in data preprocessing was accumulation of the data. As discussed above most of the data was obtained from the kaggle dataset, while the remainder was obtained using the Pytrends API. The Pytrends API was used to pull google trend data for each currency that I explored. From the Kaggle dataset, the columns for 'Volume' and 'Market Cap' were incomplete so they were dropped them from the dataframe. Also I manipulated the 'Close ' column to create a percent change column which I thought could be useful.

Feeding the data in the RNN models involved restructuring the data so that an n-feature array of the timestep (in this case 50) is fed into the RNN models. Both the GRU and the LSTM accepted the inputs this way.



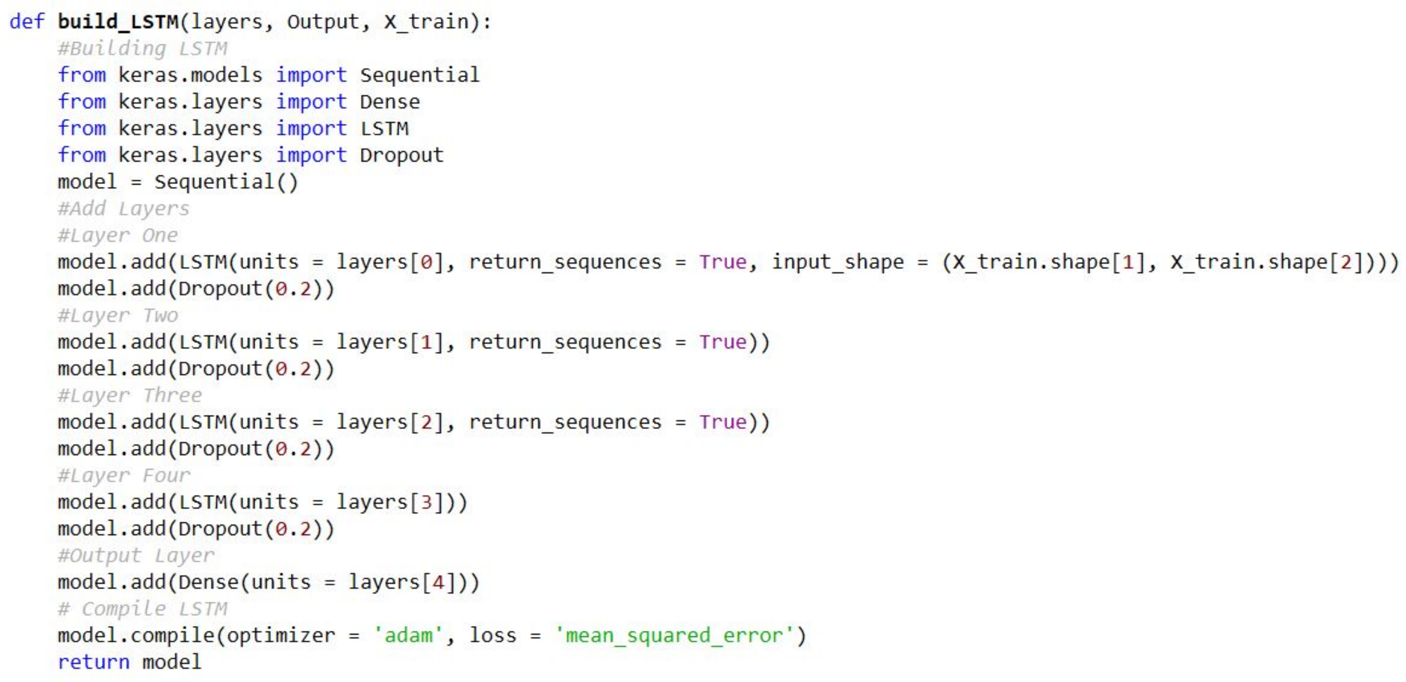
```python
def preprocess_DATA(dataset, train_size, timestep, Output):
    features = dataset.shape[1]
    dataset_train = dataset.iloc[0:train_size]
    dataset_test = dataset.iloc[train_size:dataset.shape[0]]
    training_set = dataset_train.iloc[:, 0:features].values
    training_set_label = dataset_train.iloc[:, 0:1].values
    #Normalize Data
    from sklearn.preprocessing import MinMaxScaler
    scaler_features = MinMaxScaler(feature_range = (0,1))
    scaler_label = MinMaxScaler(feature_range = (0,1))
    training_set_scaled = scaler_features.fit_transform(training_set)
    training_set_scaled_label = scaler_label.fit_transform(training_set_label)
    #Use N time steps to predict N outputs
    X_train = []
    y_train = []
    for i in range(timestep, train_size-Output):
        X_train.append(training_set_scaled[i-timestep:i, 0:features+1])
    for i in range(timestep + Output, train_size):
        y_train.append(training_set_scaled_label[i - timestep + Output, 0])
    X_train, y_train = np.array(X_train), np.array(y_train)
    #Reshape
    X_train = np.reshape(X_train, (X_train.shape[0], X_train.shape[1], features))

    #Ground Truth and Inputs for predictions
    ground_truth = dataset_test.iloc[:, 0:1].values
    dataset_total = pd.concat((dataset_train, dataset_test), axis = 0)
    dataset_total = dataset_total.iloc[:, 0:features]
    inputs = dataset_total[len(dataset_total) - len(dataset_test) - timestep:].values
    inputs = inputs.reshape(-1,features)
    inputs = scaler_features.transform(inputs)

    return X_train, y_train, inputs, ground_truth, scaler_features, scaler_label
```

Implementation

Initially I created the preprocessing function for the LSTM and GRU models. Both models take in inputs with the same characteristics, therefore, I only needed to create one preprocessing function. First the inputs need to be normalized so that they can be understood by the RNN models. I used the MinMaxScaler library to achieve the normalization. Next the inputs need to be reshaped so that they are an N-dimensional array of 50 timesteps.

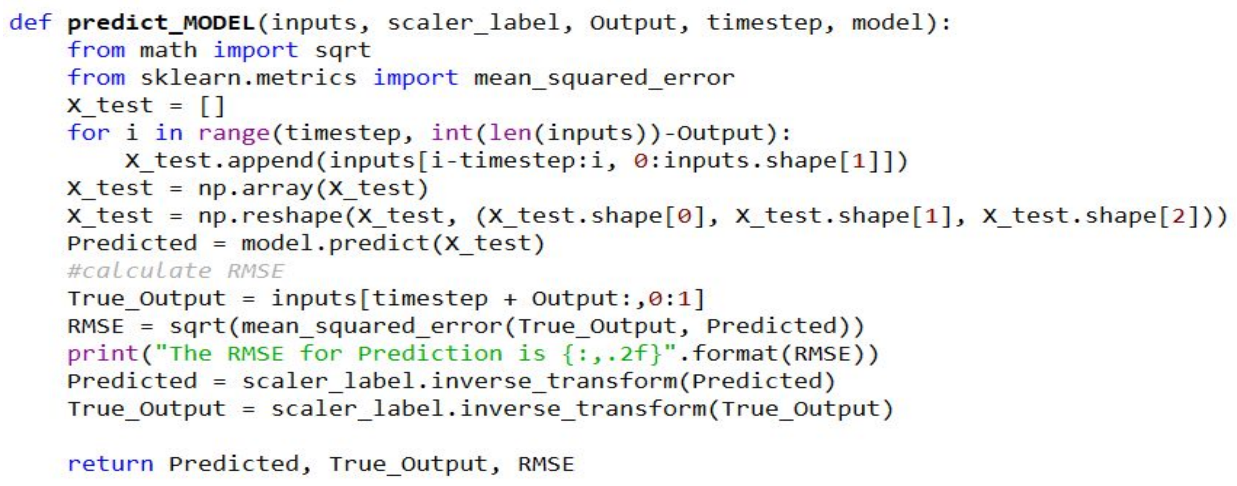
At first my output only predicted one day ahead, but I quickly realized that had limited benefits and I wanted the model to predict N days ahead. The first method I tried to solve this problem was by changing the output of last layer of the RNN models to 5 . The results were terrible, as I would always get 5 day repeating patterns that did not add any value. I eventually achieved the result I wanted by having a buffer between the input timestep and the output prediction label. For example this model used the 50 previous days(t-50) and predicted 5 days ahead(t+5).

After completing the preprocessing function the next function to create the model was very straightforward due to the Keras library. All that was involved was adding the amount of layers and the amount of neurons required for each layer, then finishing off with a 1 output dense layer. The optimizer used was 'Adam' and the loss was 'RMSE', these values were used based on research of what typically provided the best results.



```
def build_LSTM(layers, Output, X_train):
    #Building LSTM
    from keras.models import Sequential
    from keras.layers import Dense
    from keras.layers import LSTM
    from keras.layers import Dropout
    model = Sequential()
    #Add Layers
    #Layer One
    model.add(LSTM(units = layers[0], return_sequences = True, input_shape = (X_train.shape[1], X_train.shape[2])))
    model.add(Dropout(0.2))
    #Layer Two
    model.add(LSTM(units = layers[1], return_sequences = True))
    model.add(Dropout(0.2))
    #Layer Three
    model.add(LSTM(units = layers[2], return_sequences = True))
    model.add(Dropout(0.2))
    #Layer Four
    model.add(LSTM(units = layers[3]))
    model.add(Dropout(0.2))
    #Output Layer
    model.add(Dense(units = layers[4]))
    # Compile LSTM
    model.compile(optimizer = 'adam', loss = 'mean_squared_error')
    return model
```

The prediction function reshaped the test values in the same fashion as the train values. Once reshaped, the values were fed into the the previously trained model and results were compared against the normalized ground truth data using the RMSE. From there both the predictions and the ground truth were de normalized and returned for plotting.

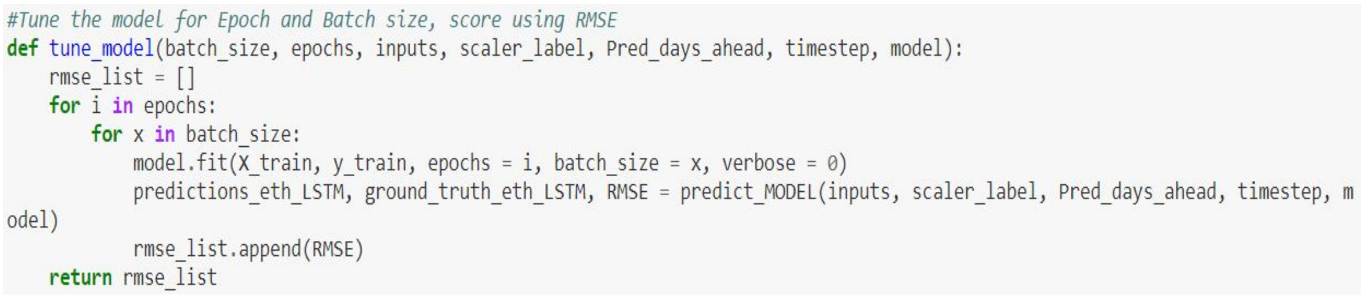


```
def predict_MODEL(inputs, scaler_label, Output, timestep, model):
    from math import sqrt
    from sklearn.metrics import mean_squared_error
    X_test = []
    for i in range(timestep, int(len(inputs))-Output):
        X_test.append(inputs[i-timestep:i, 0:inputs.shape[1]])
    X_test = np.array(X_test)
    X_test = np.reshape(X_test, (X_test.shape[0], X_test.shape[1], X_test.shape[2]))
    Predicted = model.predict(X_test)
    #calculate RMSE
    True_Output = inputs[timestep + Output:,0:1]
    RMSE = sqrt(mean_squared_error(True_Output, Predicted))
    print("The RMSE for Prediction is {:,.2f}".format(RMSE))
    Predicted = scaler_label.inverse_transform(Predicted)
    True_Output = scaler_label.inverse_transform(True_Output)

    return Predicted, True_Output, RMSE
```

Refinement

Refinement of the models posed an issue, first of all training the models is very computationally intensive, and using a form of gridsearch would require a lot more power than my laptop could handle. This led me to use a EC2 instance from Amazon Web services to refine my models. The parameters that I decided to tweak were the 'batch size' and 'epochs'.

The way this was accomplished was by creating two for loops to test out every variation of the parameters and test the model.



```
#Tune the model for Epoch and Batch size, score using RMSE
def tune_model(batch_size, epochs, inputs, scaler_label, Pred_days_ahead, timestep, model):
    rmse_list = []
    for i in epochs:
        for x in batch_size:
            model.fit(X_train, y_train, epochs = i, batch_size = x, verbose = 0)
            predictions_eth_LSTM, ground_truth_eth_LSTM, RMSE = predict_MODEL(inputs, scaler_label, Pred_days_ahead, timestep, m
odel)
            rmse_list.append(RMSE)
    return rmse_list
```

This took quite a bit of time for 6 models(2 for each cryptocurrency) and the results were pretty hit or miss.

The results were as follow,

	Ethereum		Ripple		Bitcoin	
	LSTM	GRU	LSTM	GRU	LSTM	GRU
Epoch:5 Batch: 56	0.0455	0.322	0.108	0.200	0.965	0.853
Epoch:5 Batch: 186	0.0468	0.189	0.144	0.177	0.767	1.01
Epoch:5 Batch: 396	0.0529	0.227	0.148	0.142	0.734	1.01
Epoch:25 Batch: 56	0.0557	0.060	0.166	0.169	0.790	1.05
Epoch:25 Batch: 186	0.0841	0.0639	0.179	0.307	0.861	0.889
Epoch:25 Batch: 396	0.0849	0.0443	0.181	0.279	0.792	0.867
Epoch:50	0.0919	0.0402	0.166	0.210	0.832	0.786

Batch: 56						
Epoch:50 Batch: 186	0.0892	0.0411	0.231	0.216	0.795	0.807
Epoch:50 Batch: 396	0.0899	0.0427	0.211	0.190	0.793	0.807

Figure 6: RMSE Table

Results

Model Evaluation and Validation

The hyper parameters that were tuned in this project were the epochs and batch size. An epoch is defined as a single pass through of a training model and in our case we defined three options 5, 25 and 50. The batch size refers to the number of training examples utilized in one iteration. The three batch sizes used were 56, 186, 396. In the refinement portion of the project there were times when the lower epoch produced a good RMSE result, but I believe this was due to luck as more training time is optimal for a model unless it is overfit. Overall an epoch of 25/50 along with a batch size of 56 produced the best result.

	Ethereum		Ripple		Bitcoin	
	LSTM	GRU	LSTM	GRU	LSTM	GRU
Epoch: Batch:	25 56	50 56	50 56	25 56	5 396	50 56
Training TIme	72 secs	145 secs	289 secs	145 secs	5.122 secs	301 secs

Figure 7a: Hyper Parameters and Training time

There are other hyper parameters used in the project that were not tuned, they included the number of neurons per layer, the number of layers per model, optimizer, loss, the timestep range and the output length. Tuning all these parameters could have had a positive effect on the results.

The biggest issue I ran into with this project was not being able to recreate the results from training. Using a random seed with the Numpy and Tensorflow libraries did not provide any value. This is a problem that I will need to spend more time getting to the root of.

To validate my models I ran each model three times and calculated the mean result in the table below.

	Ethereum		Ripple		Bitcoin	
	LSTM	GRU	LSTM	GRU	LSTM	GRU
Trial 1	0.06	0.08	0.17	0.11	0.75	0.91
Trial 2	0.08	0.08	0.17	0.16	0.84	0.93
Trial 3	0.05	0.04	0.21	0.22	0.83	0.76
Mean	0.06	0.07	0.18	0.16	0.81	0.87
Variance	0.00025	0.00037	0.00037	0.002	0.0016	0.0058

Figure 7b: Model Validation Table

All models displayed a variance in price, while the GRU model for Bitcoin showed the highest variance with 0.0058.

Justification

The final results of the three models can be shown in Figure 8. The results showed a large difference for the three cryptocurrencies. Bitcoin had an RMSE in the 0.8 region for both models which is not significant enough to have solved the problem. The Ripple models produced mean RMSE of 0.16 for the LSTM and 0.18 for the GRU. These values are good, however, I am not confident enough to use these models in cryptocurrency trading. The only model to predict a significant result was for Ethereum. The LSTM for Ethereum had a mean RMSE of 0.06 and the GRU had a RMSE of 0.07. These models produce fantastic results.

	Ethereum		Ripple		Bitcoin	
RMSE	LSTM	GRU	LSTM	GRU	LSTM	GRU
Prediction Mean	0.06	0.07	0.18	0.16	0.81	0.87
Benchmark	0.18		0.21		0.13	

Figure 8: Prediction Results

Comparing the RNN model's results to the benchmark linear regressions shows that the RNN's performed better for both Ethereum and Ripple. While the RNN's significantly underperformed for Bitcoin compared to a simple linear regression.

Conclusion

Free-Form Visualization

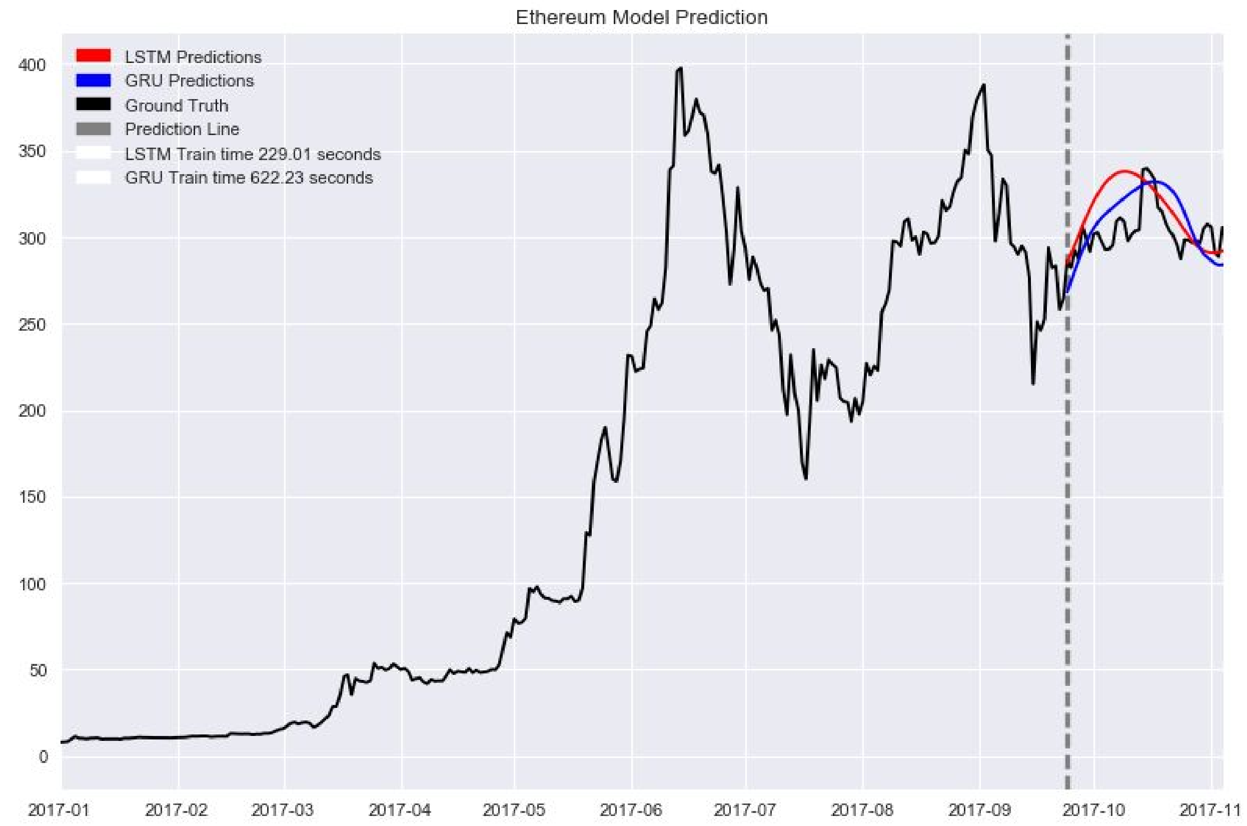


Figure 9a: Ethereum Model Prediction

The Ethereum models produced the best results out of the three cryptocurrencies chosen. I am actually quite surprised by the result. Figure 9a shows the ground truth in black, a prediction line as a dashed grey line and the predictions in red and blue. The blue prediction is the GRU which used a batch size of 56 and epoch of 50. The training time of the GRU took close to 622 seconds, this large training time is due to the larger epoch number of 50. The LSTM took approximately half the time which is expected as it has trained on half the number of epochs.

Both models produced extremely good results with the GRU having a mean variance of 0.07 while the LSTM predicted a little bit better with a mean RMSE of 0.06.

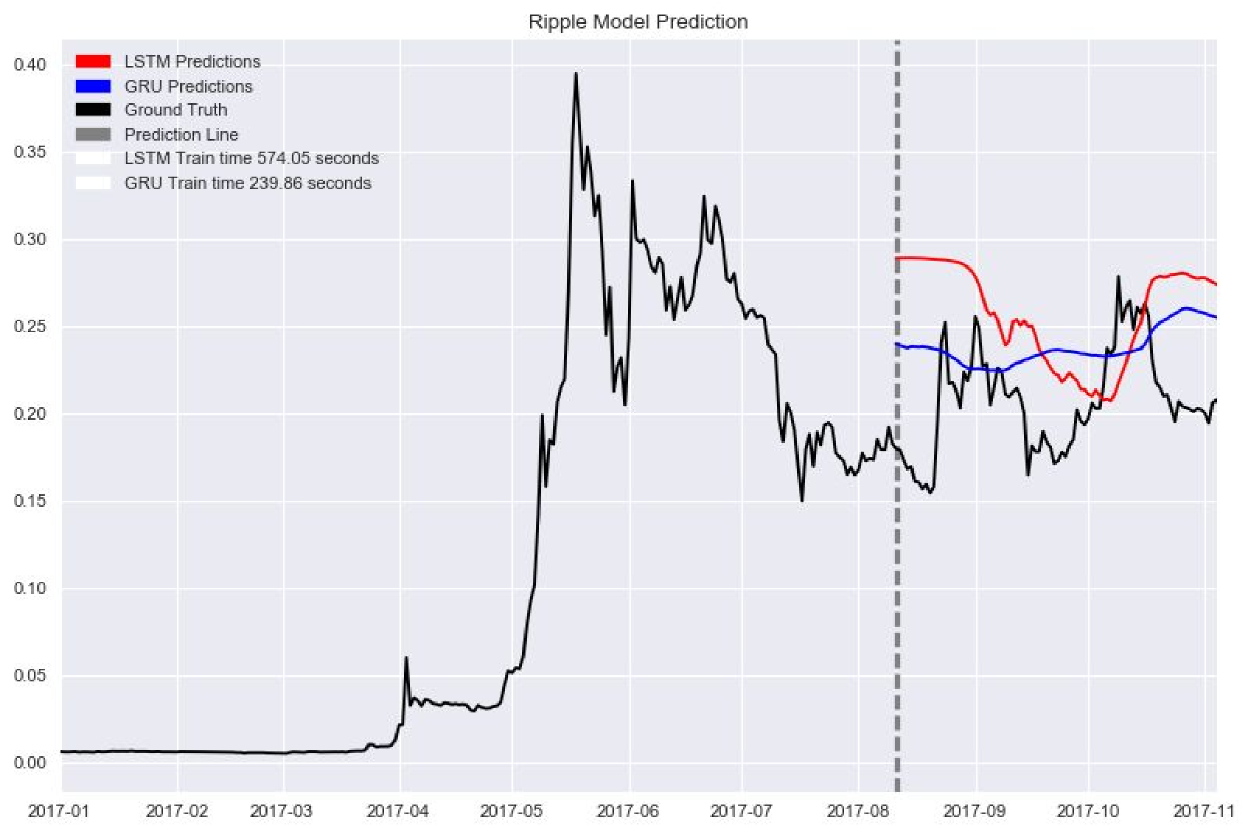


Figure 9b: Ripple Model Prediction

The Ripple models produced the second best results out of the three cryptocurrencies, however, it did not produce a useful result. The LSTM took 574 seconds to train compared to the GRU which took 240 seconds. This was due to the LSTM using 50 epochs compared the GRU which used 25. The batch sizes for both models were the same at 56. One of the mysteries of both predictions was that they started a lot higher in price compared to the ground truth. The LSTM did a better job predicted the dips and spikes while the GRU did not predict as much volatility. The mean RMSE for the LSTM was 0.18 while the GRU produced a better mean RMSE with 0.16.

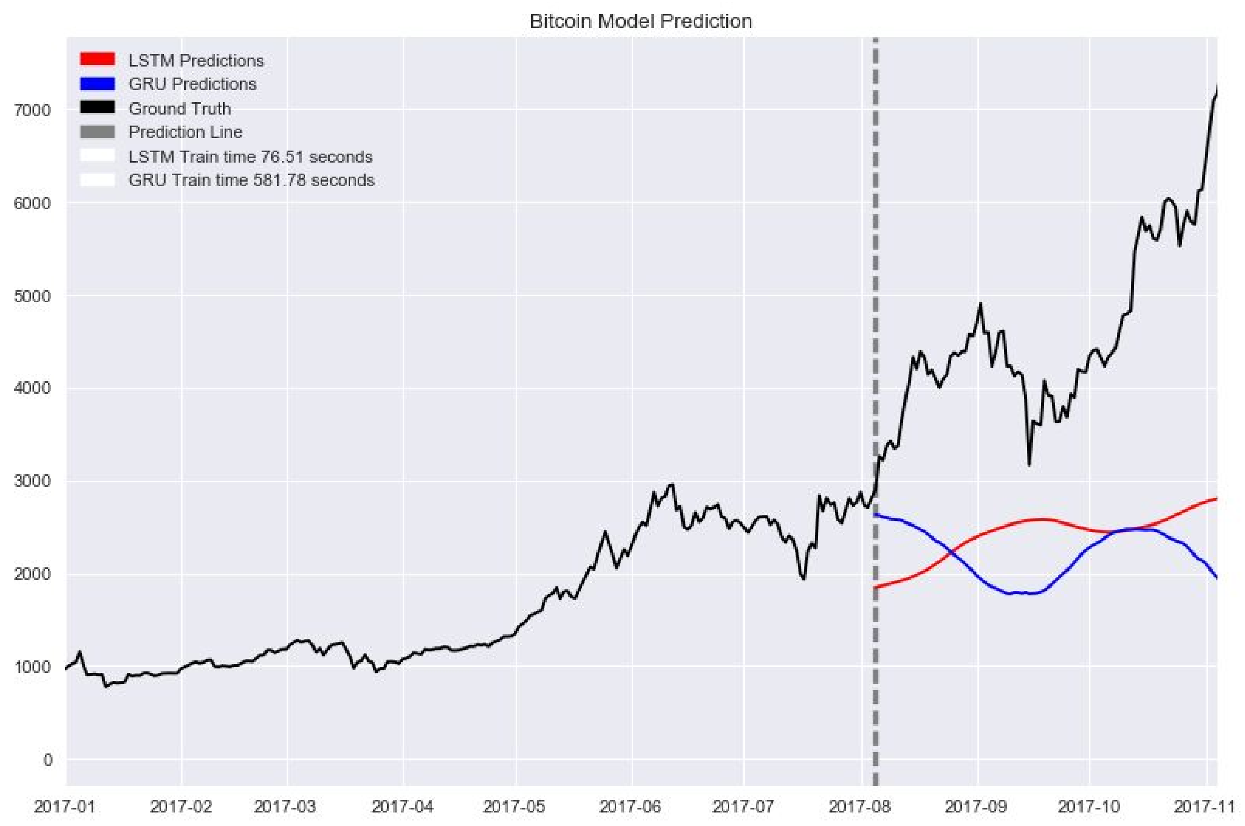


Figure 9b: Bitcoin Model Prediction

In my opinion, the Bitcoin models were considered a total failure. Both the LSTM and GRU produced an RMSE in the 0.8 range. The LSTM produced a similar terrible result as the GRU but only did so in 76 seconds of training time while having an epoch of 5 and batch size of 396. The GRU had an epoch of 50, batch size of 56, and took 580 seconds train. I believe the unfavourable results of the Bitcoin models were due to the google trend data not providing any valuable insight as well as having large spikes in price that were not captured in the training set. The mean RMSE for the LSTM is 0.81, while the GRU had a mean RMSE of 0.87.

Reflection

The main goal of this project is to be able to accurately predict the future price of the three cryptocurrencies: Ethereum, Ripple, and Bitcoin using two types of RNN models.

Based on the results, the models varied in performance. Ethereum performed very well, while Ripple had mediocre results, and Bitcoin performed terribly. The outcomes of the variance in results could be attributed to a few reasons. The first reason would be the quality of the data used to train the model. Based on the results, the google trend data had a very positive impact on the training of the Ethereum model, while it might have been detrimental to the Bitcoin result. Google trend data provides a glimpse into what the mass population is searching for. One area that google trend data may have fallen short would be in China. China has their own search system called Baidu, accessing the Baidu trends may have been helpful in data quality.

The second reason that the models may have fallen short is due to the drastic jumps in price. The Bitcoin training data included large increases in price but nothing close to the increases found October/Nov 2017, which were not captured in the training set.

Going into this project, I assumed that predicting future prices of anything would be a very difficult task. If it was easy, than there would be alot more very wealthy people in the world. Although I do find the results for the Ethereum model to be fantastic, the project has the potential for drastic improvement.

Improvement

Machine learning is a field that is experiencing very quick growth with new techniques being tested every few months. Looking into the future I can confidently assume there will be techniques available to provide much better models than currently available.

One area that could provide the most benefit to this project is in the data used to train and test the model. Currently, I have only used the previous price information and Google trend data. If other data sources were readily available that could drastically improve the models performance. Social media sources like Facebook, Twitter, and Instagram as well as news sources like Reuters, WSJ and Bloomberg, would all be useful.

The second area that could provide a benefit to this project is more effort into hyper parameter tuning. The only parameters tuned were the epochs and batch size, while many others could have also been tuned including the number of neurons per layer, the number of layers per model, optimizer, loss, the timestep range and the output length.

Appendix A: Google Trend Graphs-of Ethereum and Bitcoin

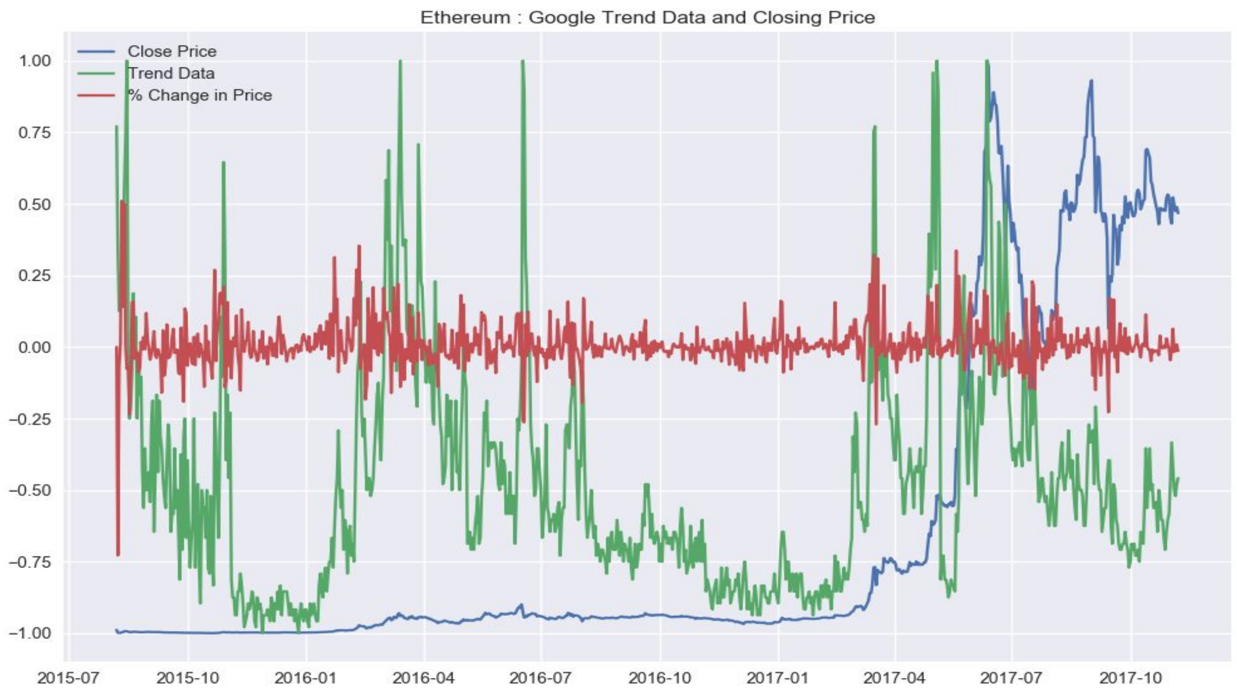


Figure A1: Ethereum: Google Trends vs Normalized Price vs Percent Change in Price

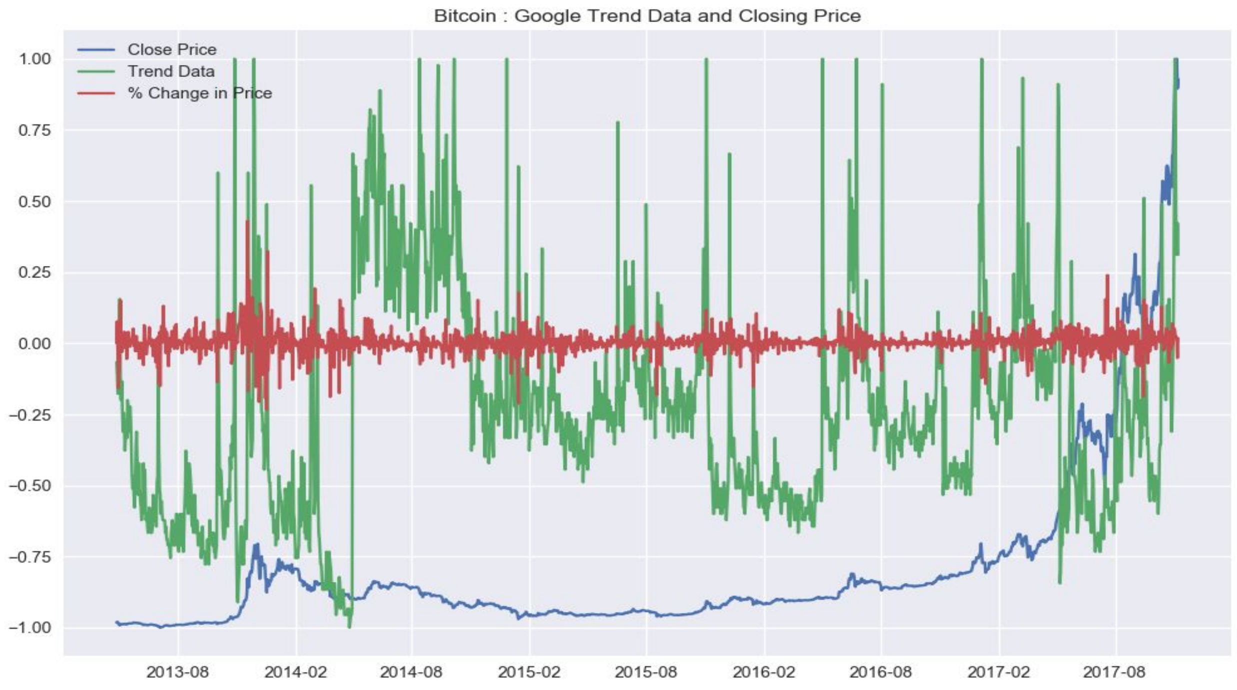


Figure A2: Bitcoin: Google Trends vs Normalized Price vs Percent Change in Price

Appendix B: Facebook Prophet Forecasts Ethereum and Bitcoin

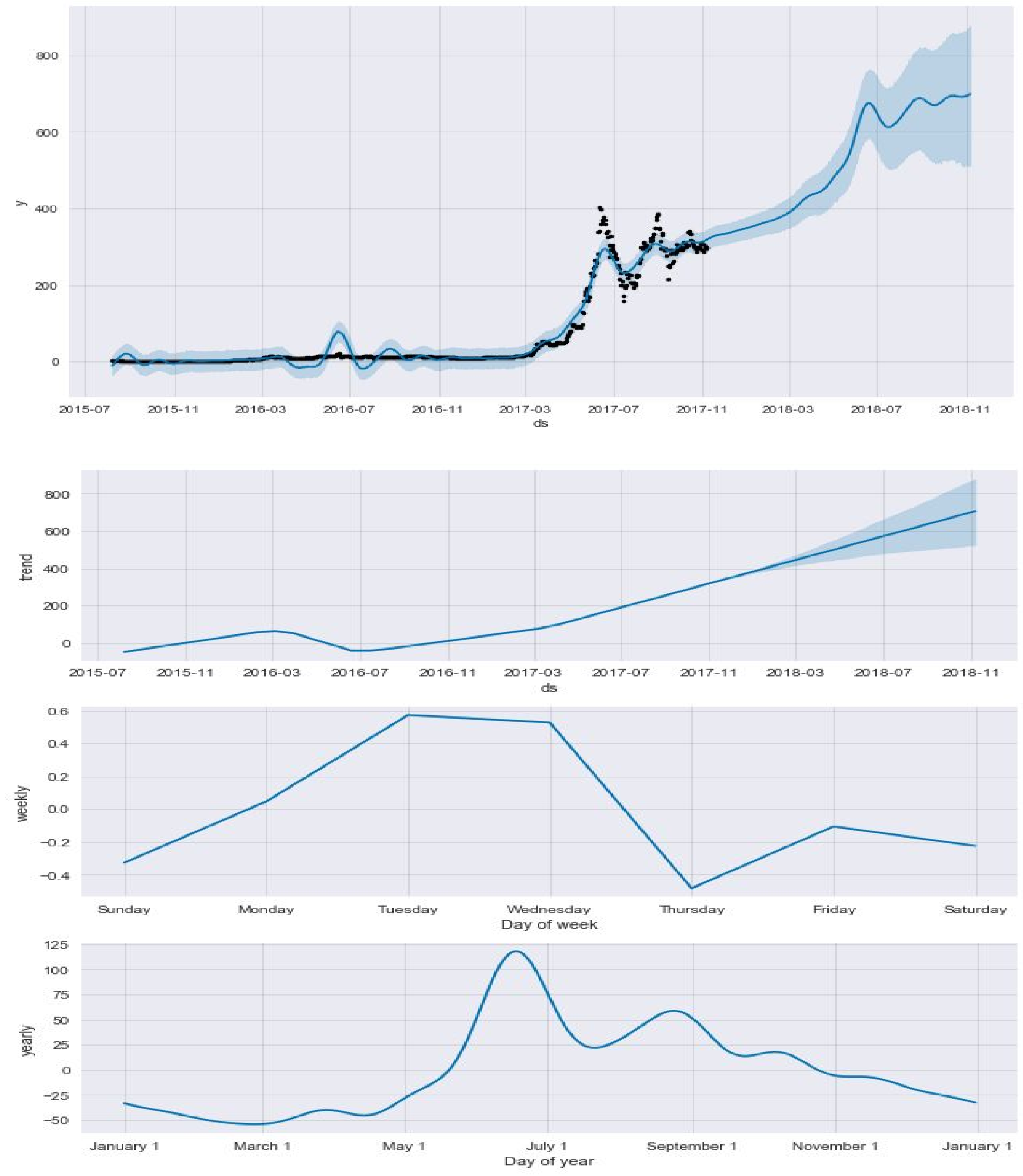


Figure B1: Ethereum FB Prophet Forecasts and Seasonality

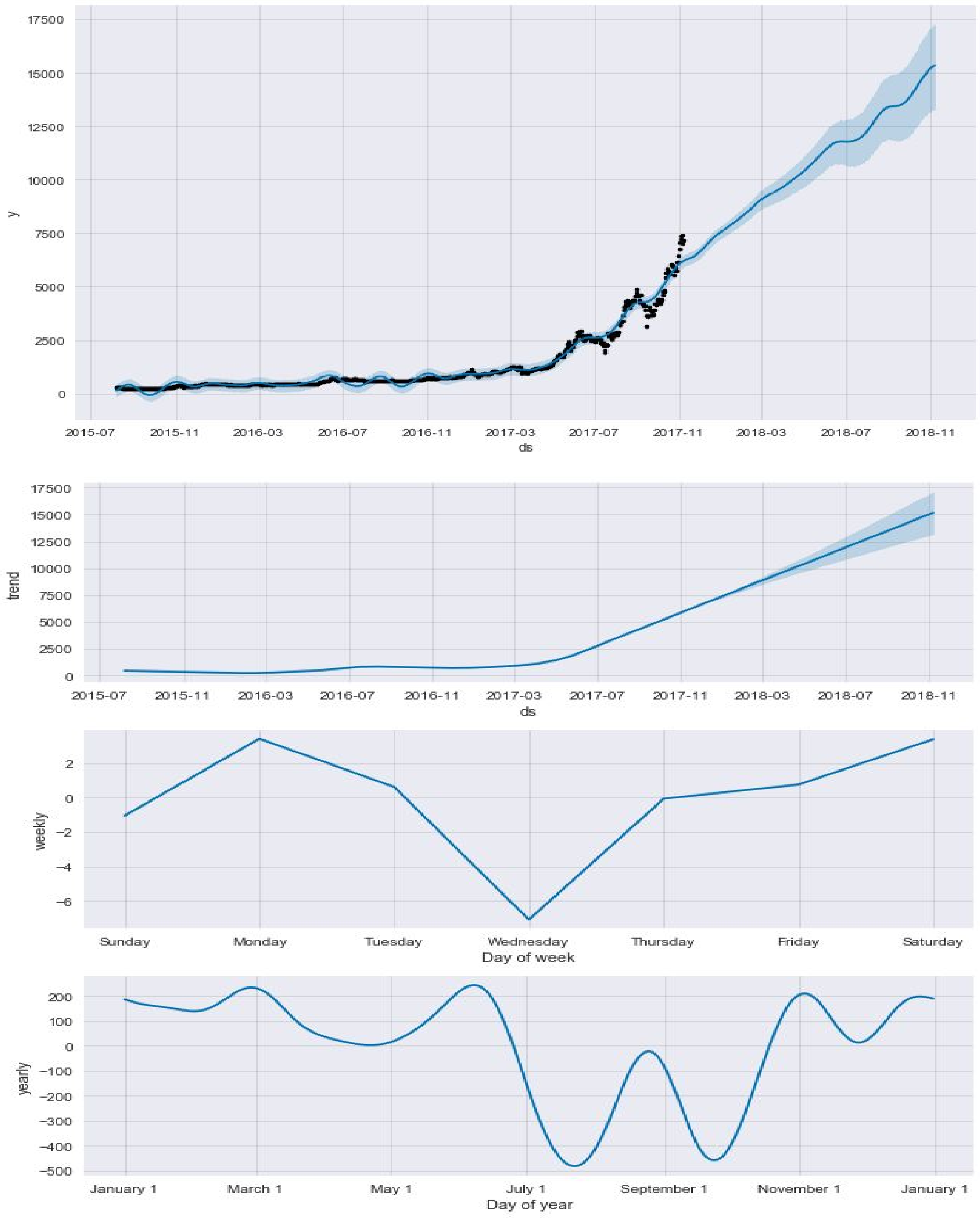


Figure B2: Bitcoin FB Prophet Forecasts and Seasonality