PRICE FORECASTING – CRYPTOCURRENCY

A final project on

CS5661- Advanced Data Science

Group:

Vishwa Pankajbhai Doshi (401329471)

Jatin Raheja (401329770)

Nikhil Katariya (401975324)

Rohit Mehta (401324986)

Akshat Jain (401971879)

Contents

1. Introduction

Cryptocurrency is a new type of asset that has emerged because of financial technology advancement, and it has created a significant opportunity for researchers. Cryptocurrency is a virtual or digital currency used in financial systems. It is secured by cryptography which makes it impossible to be faked or double-spent. As of March 2022, there are more than eighteen thousand cryptocurrencies in the market. Among those, the most famous and outstanding one is Bitcoin. Bitcoin uses a peer-to-peer network, relying on Blockchain technology, which is extremely complex and aims to store the data that makes it difficult or impossible to alter, hack, or trick the system, to verify the transaction instead of the central authorities and it is also decentralized. Due to price volatility and dynamism, forecasting cryptocurrency prices is difficult. This report proposes three types of recurrent neural network (RNN) algorithms for predicting the price of Bitcoin (BTC). When we start using large amounts of data, we notice a limitation of RNNs. For example, data was gathered from roughly 5 years of Bitcoin prices. This means that it had over 2,500 timesteps of input. As a result, whenever the model is updated, derivatives on each of those inputs are calculated. This can result in the weights dropping close to zero (vanishing gradient) or exploding (exploding gradient), indicating that the model is slow to learn. RNNs are said to have short-term memory because they have difficulty learning early inputs on large datasets. To address this issue, specialized RNNs were created. Long Short Term Memory units (LSTMs), Bidirectional LSTMs (BiLSTMs), and Gated Recurrent Units (GRUs) are all types of RNNs. These models can control how information flows through the network by utilizing internal mechanisms known as gates. The results of these models show that the gated recurrent unit (GRU) outperformed the long short-term memory (LSTM) and bidirectional LSTM (bi-LSTM) models in prediction for all types of cryptocurrencies. Lastly, the performance of each of these models is measured by Mean Square Error(MSE), Root Mean Square Error(RMSE), and Mean Absolute Percentage Error (MAPE).

2. Goal

The main goal is to forecast cryptocurrency price movements. We hope to learn, develop, and optimize learning models using machine learning algorithms in order to predict test data, forecast price, and compare the MSE, RMSE, and MAPE of different machine learning methods. The primary aim of this experiment is to show the following: For cryptocurrency forecasting, the AI algorithm is dependable and appropriate, GRU outperforms LSTM and bi-LSTM at forecasting cryptocurrency values.

3. Summary and Result

In this project, we intend to predict the closing price of the well-known cryptocurrency Bitcoin. Neural Network: As it is a sequential dataset, we used a Recurrent Neural Network (RNN) Algorithms/Methods: Long Short-Term Memory units (LSTMs), Bidirectional LSTMs (BiLSTMs), and Gated Recurrent Units (GRUs)

Steps in this project:

- Read the dataset
- Plot a graph for the dataset
- Split the data for training and testing
- Plot the training and testing data
- Data preprocessing, reshaping, and scaling
- Generating and reshaping the train and test data
- Fit the data in the LSTM model
- Predict the price using LSTM model
- Fit the data in the BiLSTM model
- Predict the price using BiLSTM model
- Fit the data in the GRU model
- Predict the price using GRUmodel
- Calculate the performance metrics such as Mean Square Error, Root Mean Square Error, and Mean Absolute Percentage Error for calculating the accuracy of the three models
- Plot the combined prediction of each model for comparison

By performing all the steps above we get the final result as the GRU model is the best overall model among the three models to predict the closing price of bitcoin. And, BiLSTM is the best model to look for trends in cryptocurrency.

4. Tools and Technology

Programming Language: Python

IDE: Google Colab

Python libraries: pandas, numpy, matplotlib, tensorflow, keras, and math

Neural Network: RNN (Recurrent Neural Network)

Neural Network Models: Long Short-Term Memory (LSTM), Bidirectional Long Short-Term Memory (BiLSTM), and Gated Recurrent Unit (GRU)

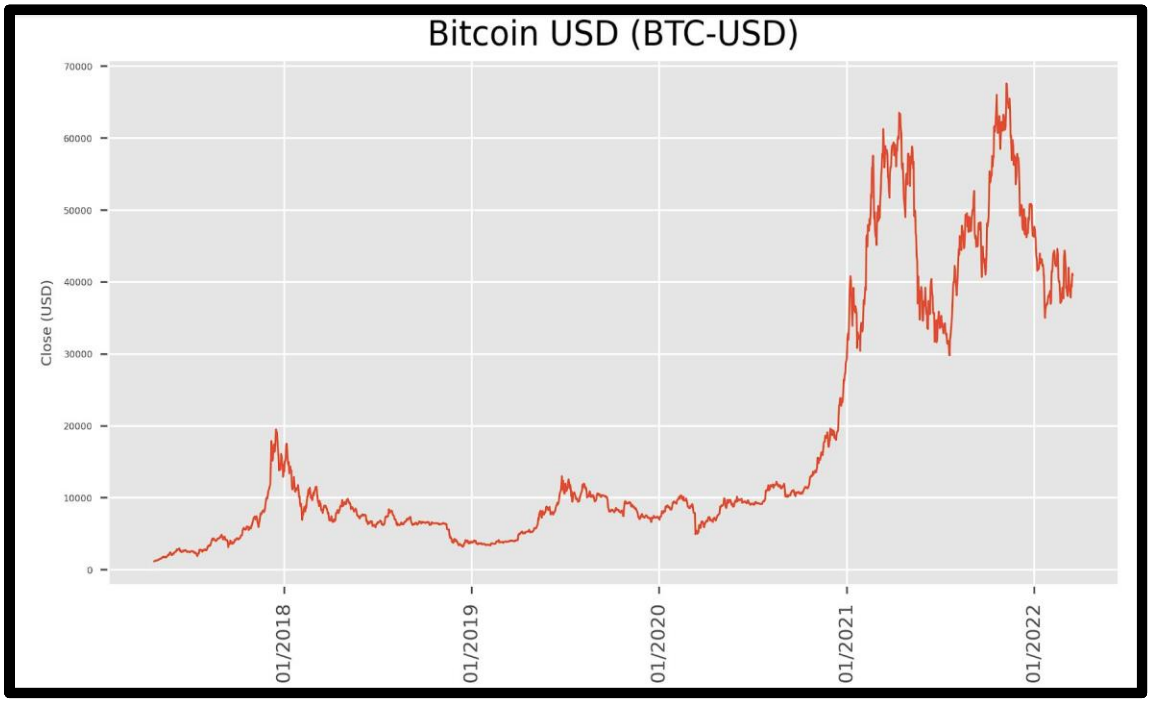
Performance Metrics: Mean Square Error (MSE), Root Mean Square Error (RMSE), and Mean Absolute Percentage Error (MAPE)

5. Future Works

We have predicted only the closing value of the cryptocurrency bitcoin. As future works, we can take maybe the trends in the market as an additional input and further train the models to improve the accuracy of the prediction. We can also take into consideration the real world market values, economic situation, competition cryptocurrency price, and supply and demand, and then incorporate these parameters into different suitable neural networks and further predict the value of cryptocurrency.

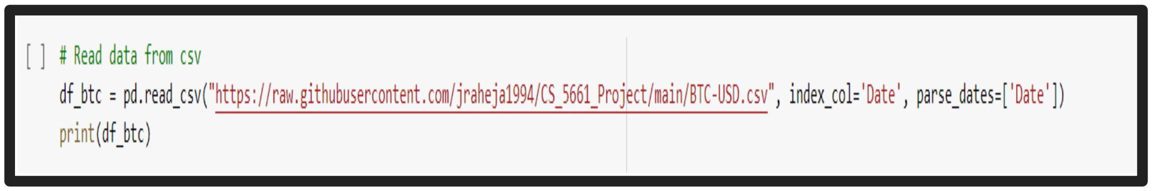
6. Data

Let's start with a look at our data. The price of Bitcoin has risen and fallen throughout time, as shown in the following figure.

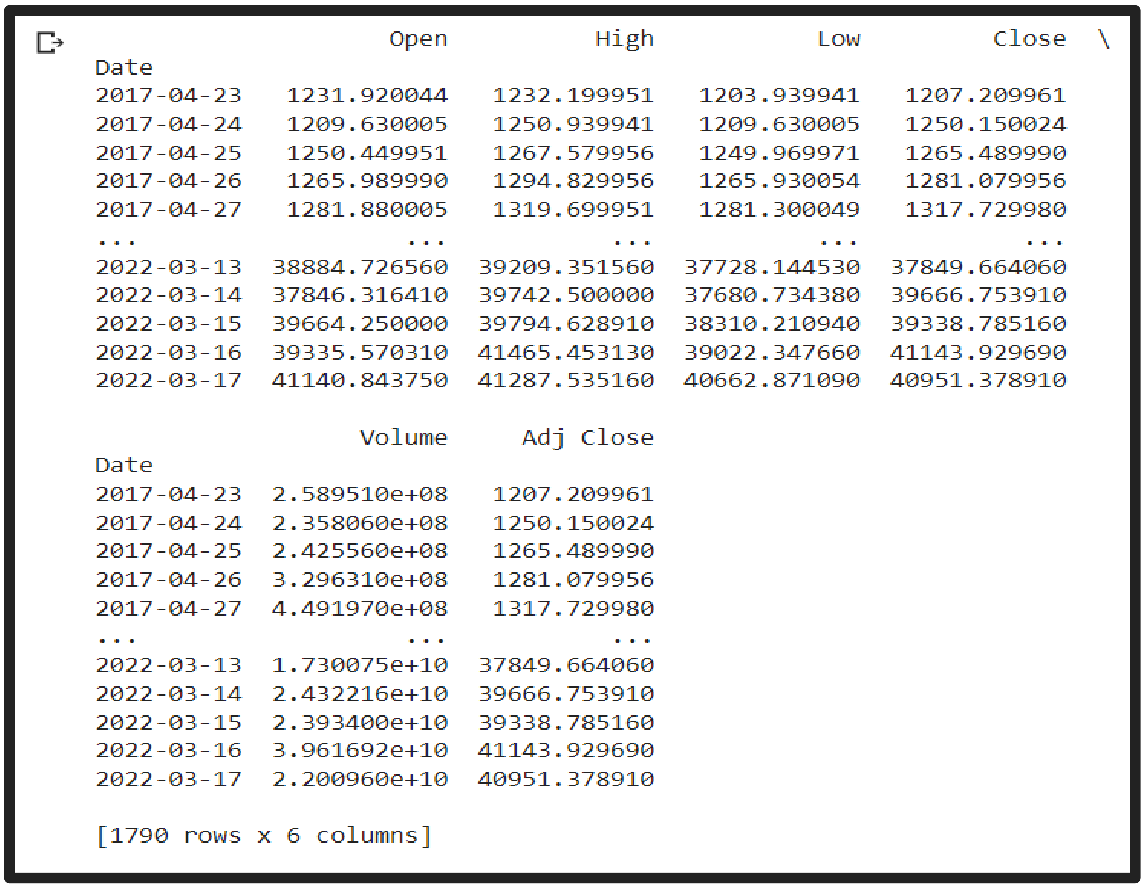


In this project, we are going to use a dataset of Bitcoin USD (BTC-USD) from yahoo finances.

This dataset contains all the numeric values such as Date, Open Value, High Value, Low Value, Close Value, Volume, and Adj Close.



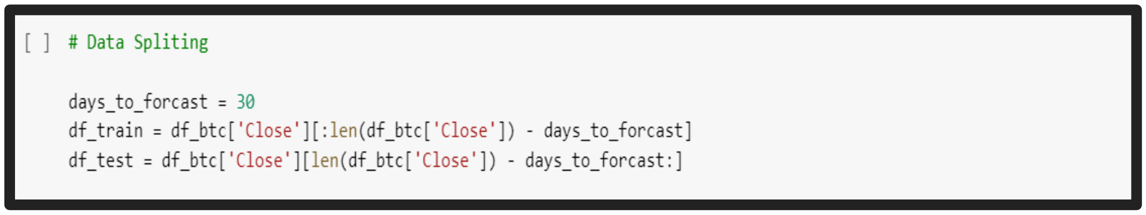
```
[ ] # Read data from csv
    df_btc = pd.read_csv("https://raw.githubusercontent.com/jraheja1994/CS_5661_Project/main/BTC-USD.csv", index_col='Date', parse_dates=['Date'])
    print(df_btc)
```



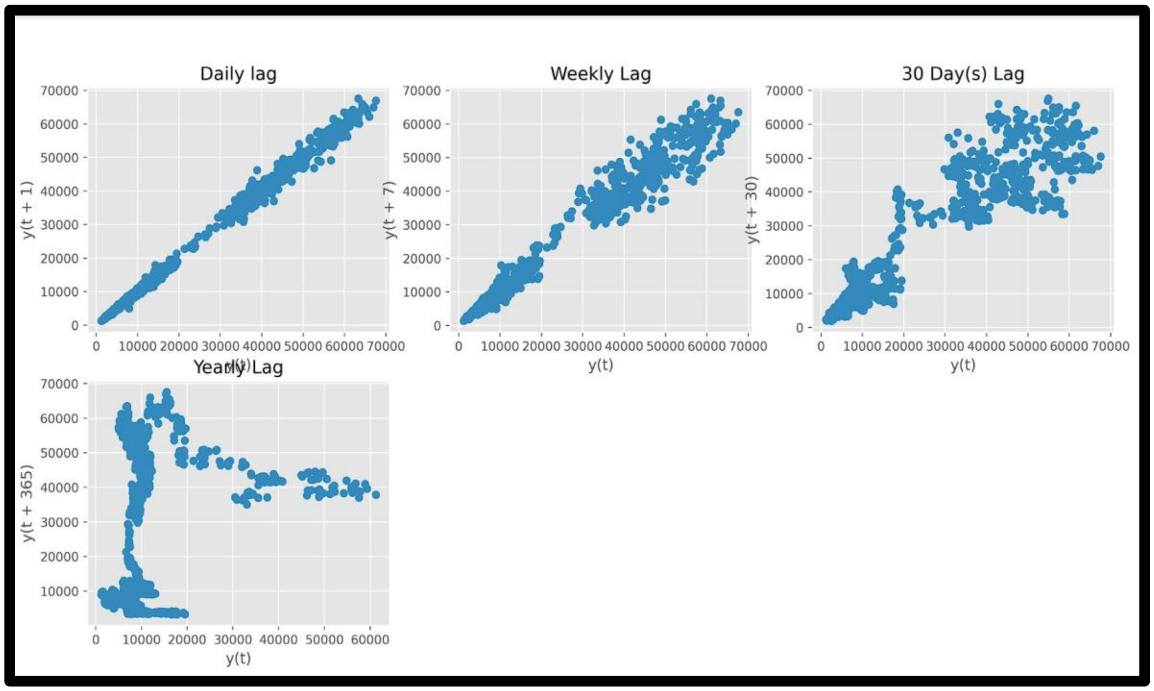
```
                 Open          High          Low           Close     \
Date
2017-04-23    1231.920044   1232.199951   1203.939941   1207.209961
2017-04-24    1209.630005   1250.939941   1209.630005   1250.150024
2017-04-25    1250.449951   1267.579956   1249.969971   1265.489990
2017-04-26    1265.989990   1294.829956   1265.930054   1281.079956
2017-04-27    1281.880005   1319.699951   1281.300049   1317.729980
...                   ...           ...           ...           ...
2022-03-13   38884.726560  39209.351560  37728.144530  37849.664060
2022-03-14   37846.316410  39742.500000  37680.734380  39666.753910
2022-03-15   39664.250000  39794.628910  38310.210940  39338.785160
2022-03-16   39335.570310  41465.453130  39022.347660  41143.929690
2022-03-17   41140.843750  41287.535160  40662.871090  40951.378910

                   Volume      Adj Close
Date
2017-04-23   2.589510e+08    1207.209961
2017-04-24   2.358060e+08    1250.150024
2017-04-25   2.425560e+08    1265.489990
2017-04-26   3.296310e+08    1281.079956
2017-04-27   4.491970e+08    1317.729980
...                   ...            ...
2022-03-13   1.730075e+10   37849.664060
2022-03-14   2.432216e+10   39666.753910
2022-03-15   2.393400e+10   39338.785160
2022-03-16   3.961692e+10   41143.929690
2022-03-17   2.200960e+10   40951.378910

[1790 rows x 6 columns]
```

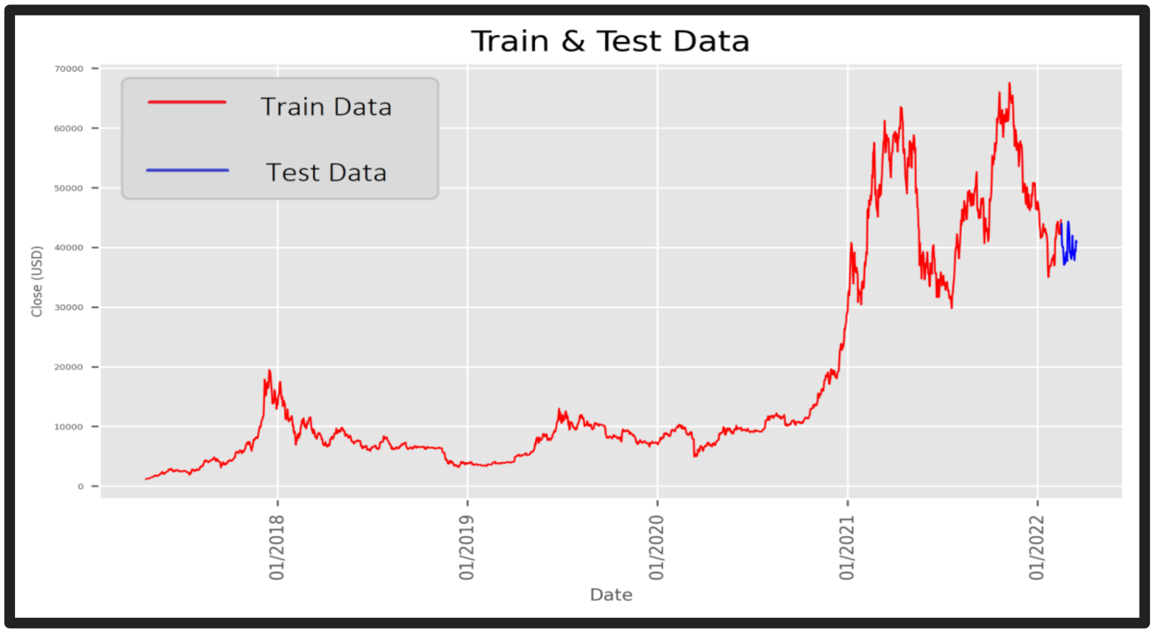
6.1 **Training & Testing**

A lag plot is a sort of scatter plot in which the two variables (X,Y) have been "lagged". One set of observations in a time series is plotted (lagged) against a second, later batch of data. The kth lag is the period of time that occurred "k" time points before the current time. There are in total 1790 rows meaning 1790 days of data. We are splitting the training data as the 1760 rows meaning 1760 days of bitcoin closing value, and testing data as the last 30 rows meaning 30 days of the closing value. And we use this training and testing data to predict the closing value of the cryptocurrency bitcoin for the last 30 days.



```
[ ]  # Data Spliting

     days_to_forcast = 30
     df_train = df_btc['Close'][:len(df_btc['Close']) - days_to_forcast]
     df_test = df_btc['Close'][len(df_btc['Close']) - days_to_forcast:]
```


Daily lag
Weekly Lag
30 Day(s) Lag
Yearly Lag
y(t + 1)
y(t + 7)
y(t + 30)
y(t + 365)
y(t)


Train & Test Data
Train Data
Test Data
Close (USD)
Date
01/2018
01/2019
01/2020
01/2021
01/2022

7. Models

The forecasting model that this project uses is Time Series Regression Model. Time series regression is a statistical method for predicting future responses based on response history and dynamics transferred from pertinent factors. To solve the Vanishing-Exploding gradients problem often encountered during the operation of a basic Recurrent Neural Network, many variations were developed such as following models.

7.1 RNN

Neural network is one of the most popular Machine learning algorithms and it almost outperform all other algorithms in both speed and accuracy. Basically, a neural network is interconnected layers having functionality like human brains. There are various types of neural networks available based on the type of work need to be done. For example, we use feed-back neural network for general regression of classification problems, convolutional neural network for object detection or image classification, deep belief network used in health sectors for cancer detection, and recurrent neural networks for speech/voice recognition and prediction.

RNN is a type of neural network used to develop speech recognition, time series prediction and natural language processing models since it retains a memory of what it has already process and so it can learn from previous states while in training. It uses a feedback loop which helps to process the sequential data. This loop allows the data to be shared to various nodes and predict according to the data available.

7.1.1 Why RNN

As RNN can handle sequential data, accepting the current input data, and previously received inputs. RNNs can memorize previous inputs due to their internal memory.

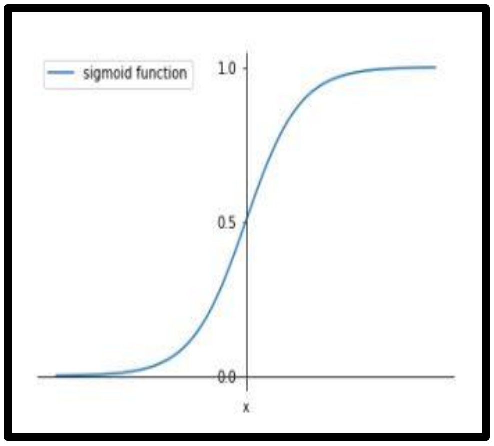
7.2 Long Short-Term Memory (LSTM)

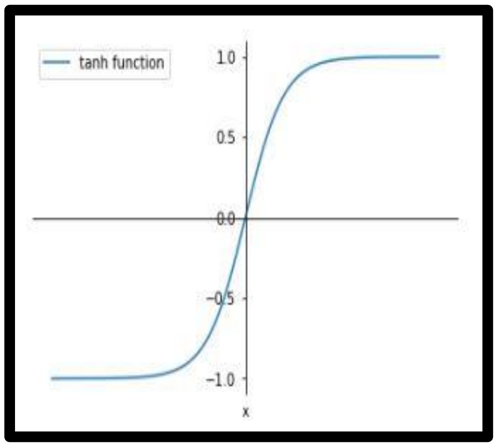
One of the most emerging as an effective and scalable approach for a variety of learning problems involving sequential data in RNN (Recurrent neural networks) is long short-term memory (LSTM). Hochreiter and Schmidhuber introduced the LSTM network as a popular deep learning method in time series forecasting in 1997. The LSTM employs a Recurrent neural network (RNN) architecture, which employs a loop to transfer information from one network step to the next. The LSTM is known for the capability to learn from sequence dependency while solving the traditional RNN’s memory problem as it is generic and effective. The LSTM is an RNN-style architecture with gates that regulate information flow between cells. The input and forget gate structures can alter data. The RNN struggles to retrieve relevant information, so the LSTM was introduced to address this issue. The LSTM is an RNN variant with more layers and a similar recurrent or chain-like structure. The LSTM's key component is the memory cell Ct, as well as three gates (input gate, output gate, and forget gate) that control the information added to or removed from the cell. In LSTM there are two activation functions that plays essential role, the first is sigmoid function which regulates how much information to be passed through these gates, it is defined as:

$$\boldsymbol{\sigma(x) = 1/(1 + e^{-x}) = e^{x}\,/(1\ +\ e^{x})}$$

The output of the function ranges between 0 and 1. A value 0 represents "don't let anything through" while 1 represents "let everything through". The second activation function is hyperbolic function or tanh function whose output ranges between -1 and 1 which is defined as:

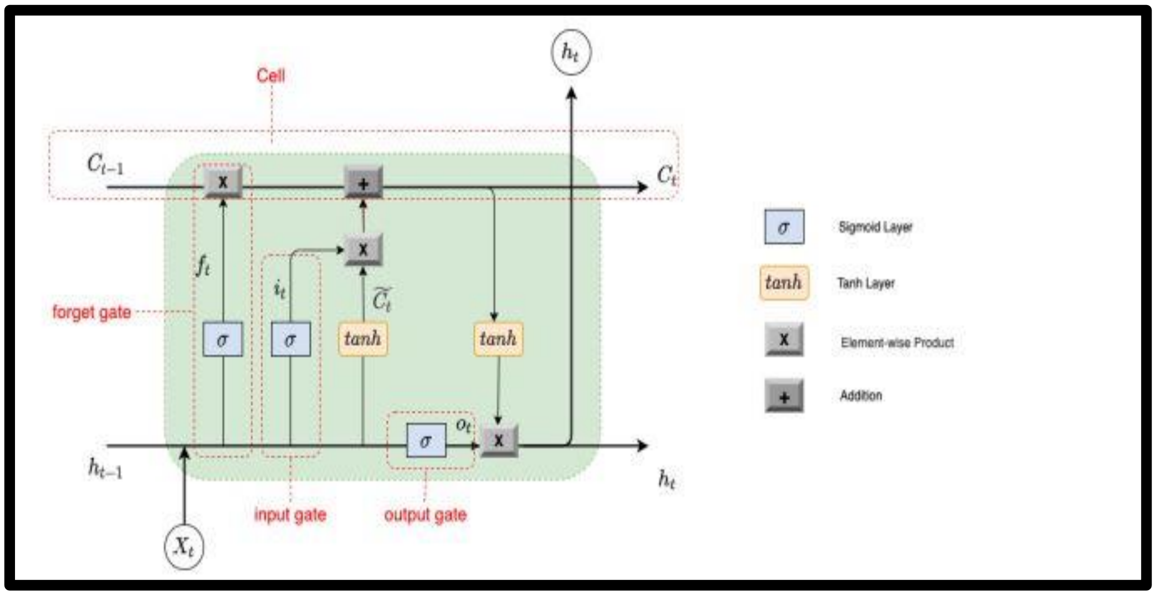
$$\boldsymbol{tanh(x)} = (\boldsymbol{e^{x}} - \boldsymbol{e^{-x}})/(\boldsymbol{e^{x}} + \boldsymbol{e^{-x}})$$





The LSTM is made up of mainly four parts which are:

1. Forget Gate: It determines which previous information is discarded from the cell.
2. Input Gate: It generates new data by utilizing an input gate layer and a tanh layer.
3. Cell State: The cell state then modifies it by combining the new information from the previous two parts.
4. Output Gate: It generates the output for the current state.



Forget Gate:

This is also known as the “Remember Vector” which is discarding the details in the cell. So, to accomplish this the previous hidden state and the new input data are fed into a neural network. So, with the monitoring of the sigmoid activation function this network creates a vector with each element in the range [0,1]. This network in the forget gate is trained to output near 0 when a component of the input is deemed irrelevant and near 1 when it is deemed relevant. This means by multiplying 0 to a place in the matrix, the forget gate's output notifies the cell state which information to forget. If the forget gate's output is 1, the information is stored in the cell state. The sigmoid function is applied to the weighted input/observation and the prior hidden state using the equation.

$$\boldsymbol{f_t} = \sigma(W_f x_t + U_f h_{t-1} + b_f)$$

Input Gate:

This is also known as the “Save Vector” which is the entry of new information using input gate layer and a tanh layer. Here the sigmoid function receives the current state x_t and the previously hidden state $h_{(t-1)}$. The values are changed from 0 (important) to 1 (not important). The tanh function will then be used to pass the identical information from the hidden state and current state. The tanh operator will construct a vector (C_t) with all the possible values between -1 and 1 to regulate the network. The activation functions generate output values that are ready for point-by-point multiplication.

$$\boldsymbol{i_t} = \sigma (W_i x_t + U_i h_{t-1} + b_i)$$

$$\boldsymbol{C_t} = \tanh (W_c x_t + U_c h_{t-1} + b_c)$$

Cell State:

The former cell is then modified by combining the new information from the previous two parts. The forget gate and input gate have provided enough information to the network. The information from the new state must then be decided and stored in the cell state. The forget vector $f_{(t)}$ is multiplied by the prior cell state $C_{(t-1)}$. If the result is 0, the values in the cell state will be dropped. The network then executes point-by-point addition on the output value of the input vector $i_{(t)}$, updating the cell state and giving the network a new cell state $C_{(t)}$.

$$\boldsymbol{C_t} = f_t \circ C_{t-1} + i_t \circ C_t$$

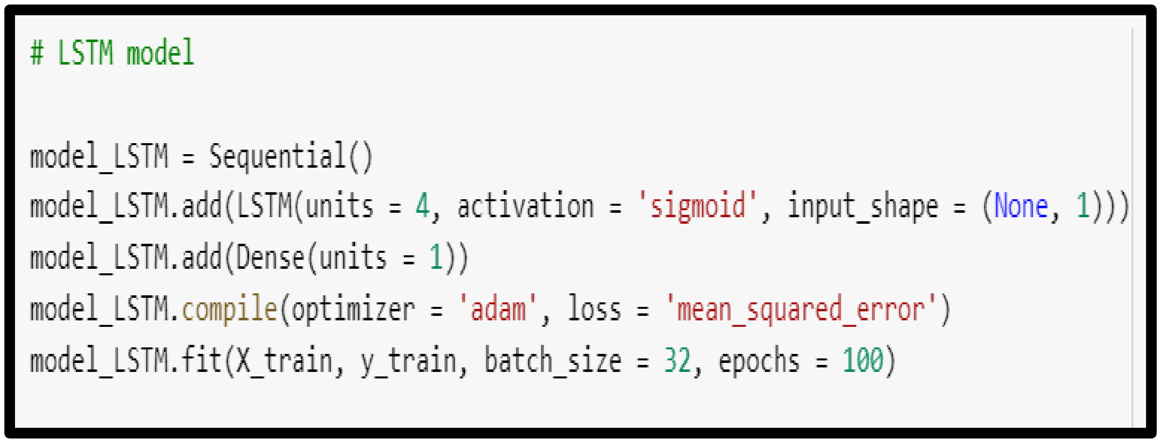
Output Gate:

The final step is to generate the output for the current state. It has an output gate with the output vector o_t and a tanh layer with the current cell's input C_t, which determines what data from the cell should be included in the output. Multiplying each element in these vectors yields a new hidden vector h_t as the state's final output.

$$\boldsymbol{o_t} = \sigma\,(W_o\,x_t + U_o\,h_{t-1} + b_o)$$

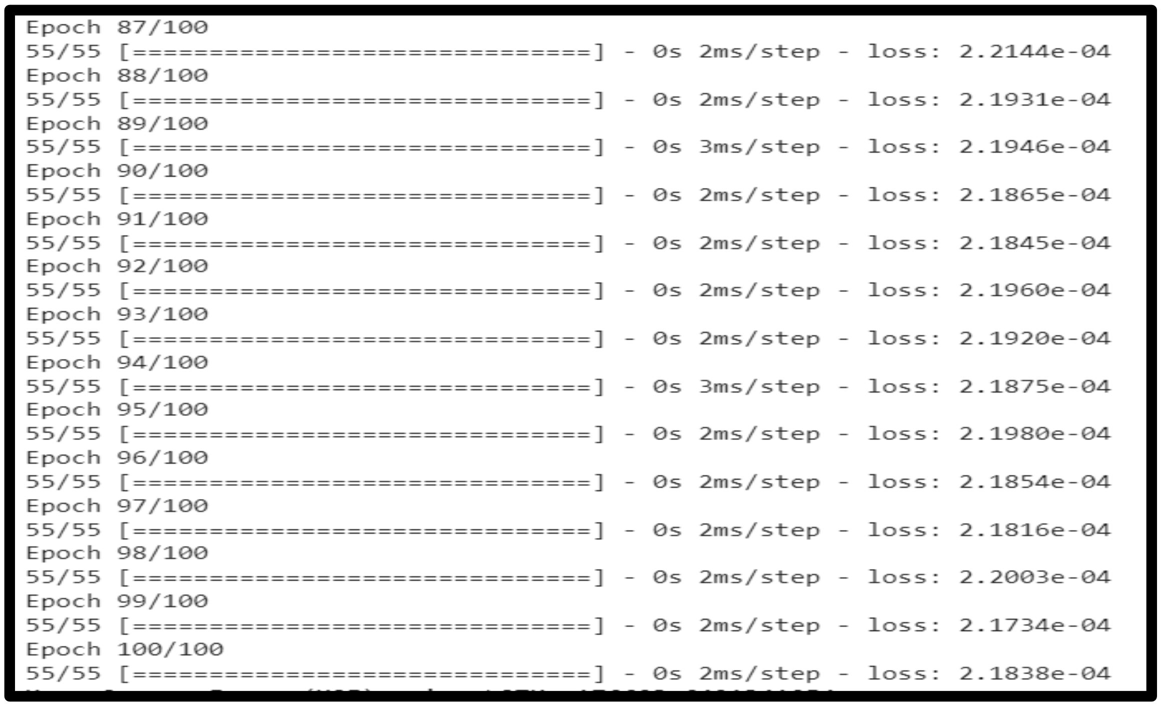
$$\boldsymbol{h_t} = o_t \circ tanh\,(C_t)$$

Below is the snippet for LSTM model using sigmoid function:



```
# LSTM model


model_LSTM = Sequential()
model_LSTM.add(LSTM(units = 4, activation = 'sigmoid', input_shape = (None, 1)))
model_LSTM.add(Dense(units = 1))
model_LSTM.compile(optimizer = 'adam', loss = 'mean_squared_error')
model_LSTM.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

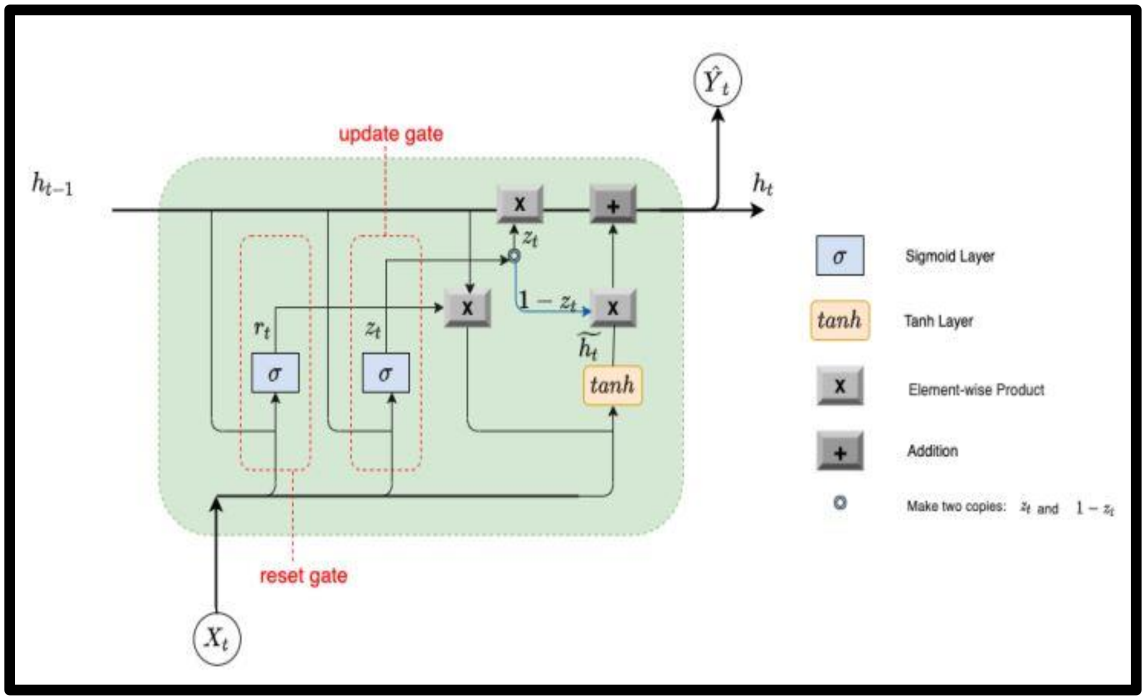
Output:



```
Epoch 87/100
55/55 [==============================] - 0s 2ms/step - loss: 2.2144e-04
Epoch 88/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1931e-04
Epoch 89/100
55/55 [==============================] - 0s 3ms/step - loss: 2.1946e-04
Epoch 90/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1865e-04
Epoch 91/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1845e-04
Epoch 92/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1960e-04
Epoch 93/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1920e-04
Epoch 94/100
55/55 [==============================] - 0s 3ms/step - loss: 2.1875e-04
Epoch 95/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1980e-04
Epoch 96/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1854e-04
Epoch 97/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1816e-04
Epoch 98/100
55/55 [==============================] - 0s 2ms/step - loss: 2.2003e-04
Epoch 99/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1734e-04
Epoch 100/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1838e-04
```

7.3 Gated Recurrent Unit (GRU)

The GRU is a newer type of recurrent neural network that looks a lot like an LSTM. The GRUs abandoned the cell state in favor of using the hidden state to transfer data. There are only two gates on it: a reset gate and an update gate. Since this GRU has fewer parameters but a similar design to an LSTM, it can operate better with fewer datasets. As a result, it works better for our bitcoin price dataset. Another intriguing feature of GRU is that, unlike LSTM, it lacks a separate cell state (Ct). It only has one state: hidden (Ht). GRUs are better to train because of their simpler architecture.



It takes an input X_t and the hidden state H_{t-1} from the previous timestamp t-1 at each timestamp t. It then returns a new hidden state Ht, which is then passed to the next timestamp. A GRU cell now has primarily two gates as opposed to three gates in an LSTM cell.

- **Reset Gate:** The Reset Gate is responsible for the short-term memory of the network i.e the hidden state (Ht). The Reset gate's equation is shown below.

$$\mathbf{r_t} = \boldsymbol{\sigma}(x_t * u_r + H_{t-1} * W_r)$$

  Because of the sigmoid function, the value of r_t will range from 0 to 1. U_r and W_r are the reset gate's weight matrices.

- **Update Gate:** Similarly, for long-term memory, we have an Update gate, and its equation is shown below. The only distinction is between weight metrics, i.e. U_u and W_u.

$$\mathbf{u_t} = \boldsymbol{\sigma}(x_t * u_u + H_{t-1} * W_u)$$

Let's take a look at how these gates work. It takes two steps to find the Hidden state H_t in GRU. The first step is to generate the candidate hidden state. It takes the input and hidden state from the previous timestamp, t-1, and multiplies them by the reset gate output rt. The candidate's concealed state is the outcome of passing this information to the tanh function.

$$\mathbf{H_t'} = \boldsymbol{tanh}(x_t * U_g + (r_t \circ H_{t-1}) * W_g)$$

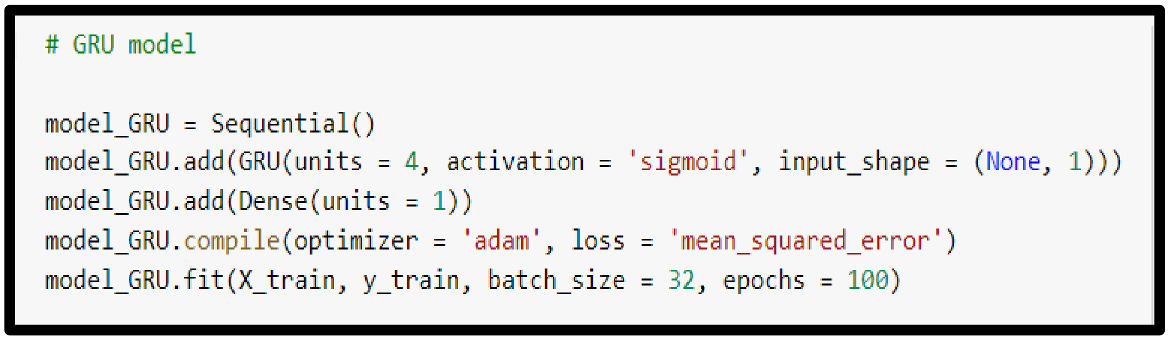
The most crucial element of this equation is how we use the reset gate's value to determine how much the previous concealed state can impact the candidate state. When the value of r_t equals 1, the full information from the previous concealed state H_{t-1} is taken into account. Similarly, if the value of r_t is 0, the information from the preceding hidden state will be completely ignored.

The current hidden state Ht is generated using the candidate state. It's at this point when the Update gate comes into play. Instead of utilizing a separate gate, as in LSTM, we utilize a single update gate in GRU to regulate both the historical information (H_{t-1}) and the fresh information (from the candidate state).

$$\boldsymbol{H_t} = u_t \circ H_{t-1} + (1-u_t) \circ H_t'$$

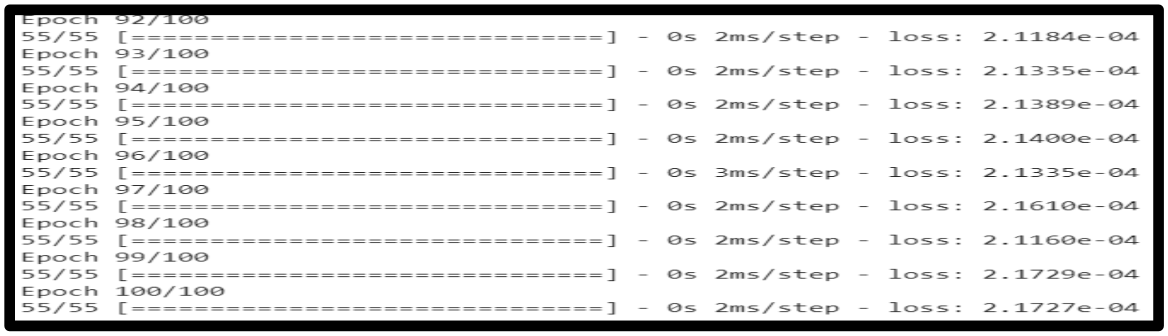
If u_t is close to 0, the first term in the equation will disappear, implying that the new hidden state will not have much information from the prior hidden state. The second portion, on the other hand, merges into one, implying that the hidden state at the present timestamp will only contain information from the candidate state. Similarly, if the value of ut is on the second term, the current hidden state will be fully dependent on the first term, that is, the information from the hidden state at timestamp t-1.

Below is the snippet for GRU using sigmoid function:



```
# GRU model

model_GRU = Sequential()
model_GRU.add(GRU(units = 4, activation = 'sigmoid', input_shape = (None, 1)))
model_GRU.add(Dense(units = 1))
model_GRU.compile(optimizer = 'adam', loss = 'mean_squared_error')
model_GRU.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

Output:



```
Epoch 92/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1184e-04
Epoch 93/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1335e-04
Epoch 94/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1389e-04
Epoch 95/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1400e-04
Epoch 96/100
55/55 [==============================] - 0s 3ms/step - loss: 2.1335e-04
Epoch 97/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1610e-04
Epoch 98/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1160e-04
Epoch 99/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1729e-04
Epoch 100/100
55/55 [==============================] - 0s 2ms/step - loss: 2.1727e-04
```

7.4 Bi-directional Long Short Term Memory (Bi-LSTM)

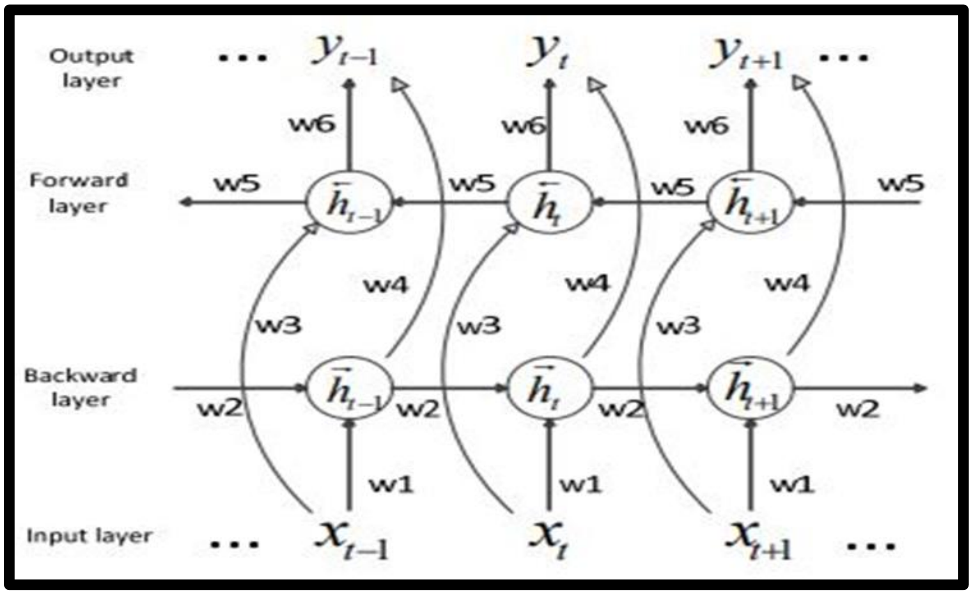
Bi-LSTM was developed by Schuster and Paliwal to train a network using past and future input data sequences. Two linked layers are used to process the input data. Basically Bi-LSTM is the process of allowing any neural network to store sequence information in both directions, either backwards (future to past) or forwards (past to future). Our input flows in two directions in bidirectional, distinguishing a bi-lstm from a regular LSTM. We can make input flow in only one direction with a regular LSTM, either backwards or forwards. However, with bi-directional, we can make the input flow in both directions, preserving both the future and the past. So, one is taking the information in a forward manner, while the other is taking it backwards. Bi-LSTMs effectively increase the quantity of data available to the network, giving the algorithm better context.

The forward and backward hidden state updates are as follows:

$$h_t = f\,(w_1 x_t + w_2 h_{t-1})$$

$$h_t' = f\,(w_3 x_t + w_5 h_{t+1})'$$

$$o_t = g\,(w_4 h_4 + w_6 h_t')$$



7.4.1 Advantages of Bi-LSTM

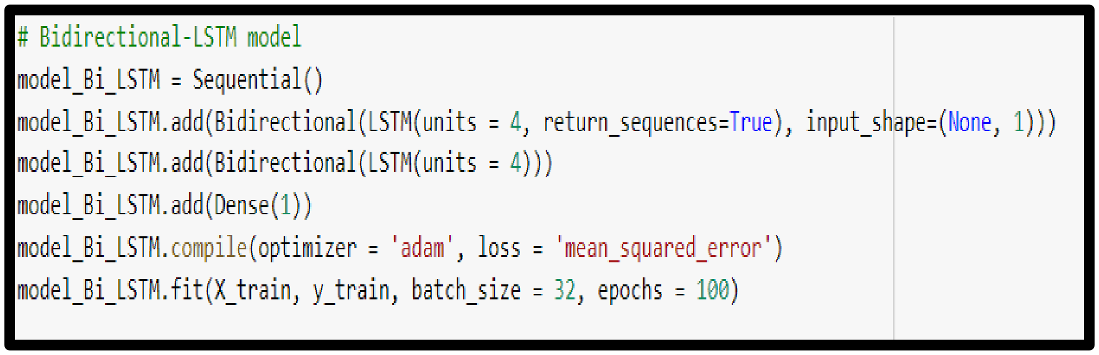
- Since every component of an input sequence contains information from both the past and the present, Bi-LSTM can produce a more meaningful output by merging LSTM layers from both directions, resulting in a more meaningful output.
- Every component (word) of the sequence will have a different Bi-LSTM output (sentence). As a result, the Bi-LSTM model can help with phrase categorization, translation, and entity recognition.
- By learning future time steps in a bidirectional manner, bidirectional LSTM (BLSTM) allows for a greater comprehension of context. In addition, GRU uses reset and update

gates in the hidden layer, which is more computationally efficient than a traditional LSTM.

7.4.2 Disadvantages of Bi-LSTM

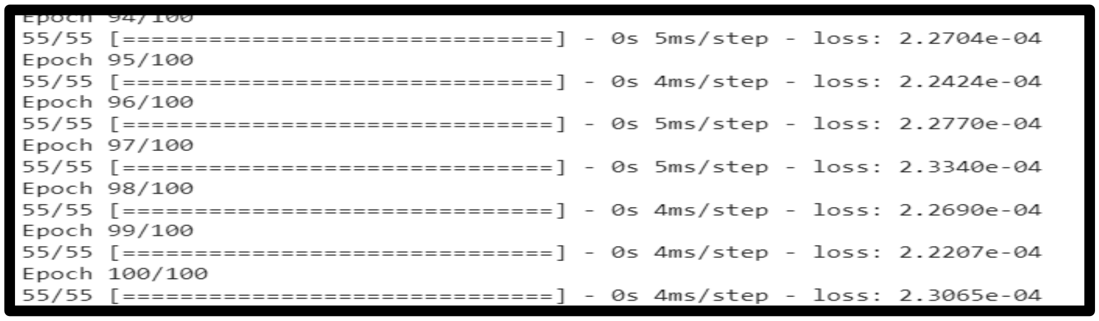
- Bi-LSTM is a much slower model as compared to LSTM because it requires more time to train the model.
- As Bi-LSTM has double LSTM in the model so it is quite costly in comparison with LSTM or GRU.

Below is the snippet for Bi-LSTM model:



```
# Bidirectional-LSTM model
model_Bi_LSTM = Sequential()
model_Bi_LSTM.add(Bidirectional(LSTM(units = 4, return_sequences=True), input_shape=(None, 1)))
model_Bi_LSTM.add(Bidirectional(LSTM(units = 4)))
model_Bi_LSTM.add(Dense(1))
model_Bi_LSTM.compile(optimizer = 'adam', loss = 'mean_squared_error')
model_Bi_LSTM.fit(X_train, y_train, batch_size = 32, epochs = 100)
```

Output:

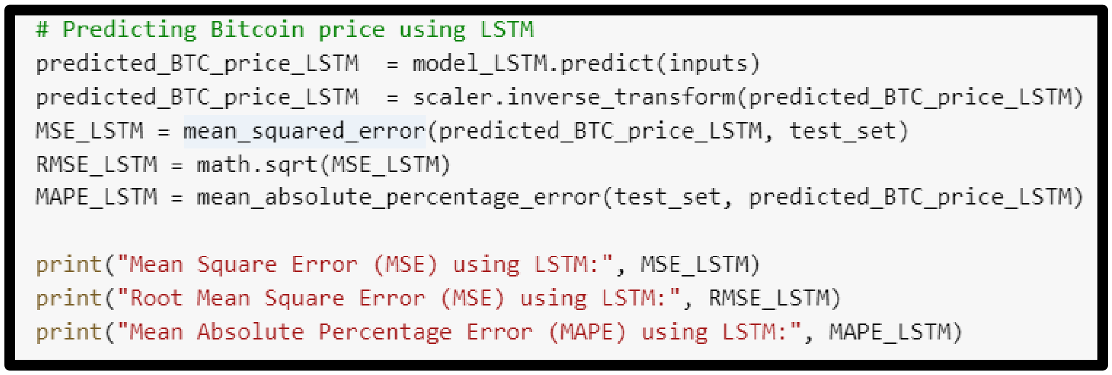


```
Epoch 94/100
55/55 [==============================] - 0s 5ms/step - loss: 2.2704e-04
Epoch 95/100
55/55 [==============================] - 0s 4ms/step - loss: 2.2424e-04
Epoch 96/100
55/55 [==============================] - 0s 5ms/step - loss: 2.2770e-04
Epoch 97/100
55/55 [==============================] - 0s 5ms/step - loss: 2.3340e-04
Epoch 98/100
55/55 [==============================] - 0s 4ms/step - loss: 2.2690e-04
Epoch 99/100
55/55 [==============================] - 0s 4ms/step - loss: 2.2207e-04
Epoch 100/100
55/55 [==============================] - 0s 4ms/step - loss: 2.3065e-04
```

8. Analysis and Prediction of Models

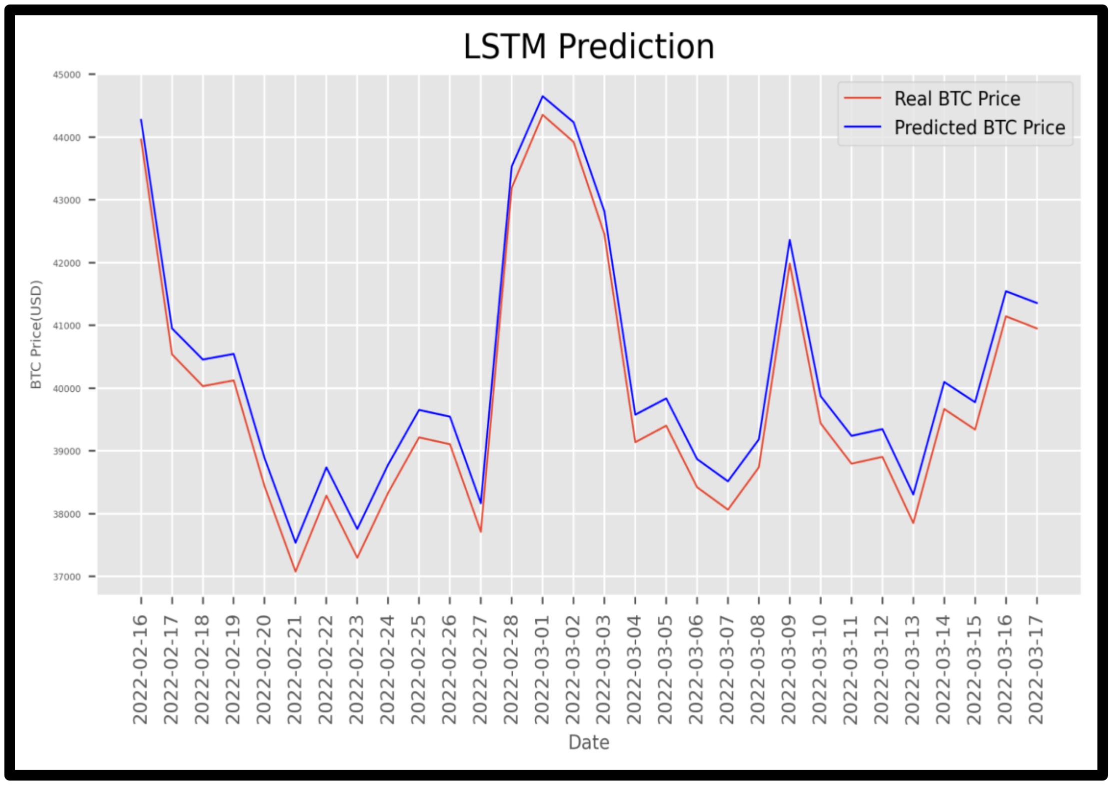
8.1 LSTM

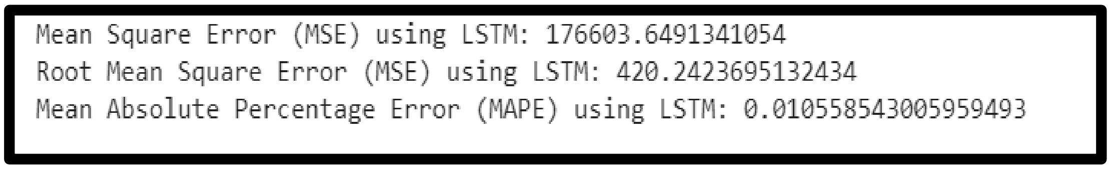
LSTM networks are well-suited to categorizing, processing, and making predictions based on time series data. Because there might be lags of undetermined duration between critical occurrences.



```python
# Predicting Bitcoin price using LSTM
predicted_BTC_price_LSTM  = model_LSTM.predict(inputs)
predicted_BTC_price_LSTM  = scaler.inverse_transform(predicted_BTC_price_LSTM)
MSE_LSTM = mean_squared_error(predicted_BTC_price_LSTM, test_set)
RMSE_LSTM = math.sqrt(MSE_LSTM)
MAPE_LSTM = mean_absolute_percentage_error(test_set, predicted_BTC_price_LSTM)

print("Mean Square Error (MSE) using LSTM:", MSE_LSTM)
print("Root Mean Square Error (MSE) using LSTM:", RMSE_LSTM)
print("Mean Absolute Percentage Error (MAPE) using LSTM:", MAPE_LSTM)
```

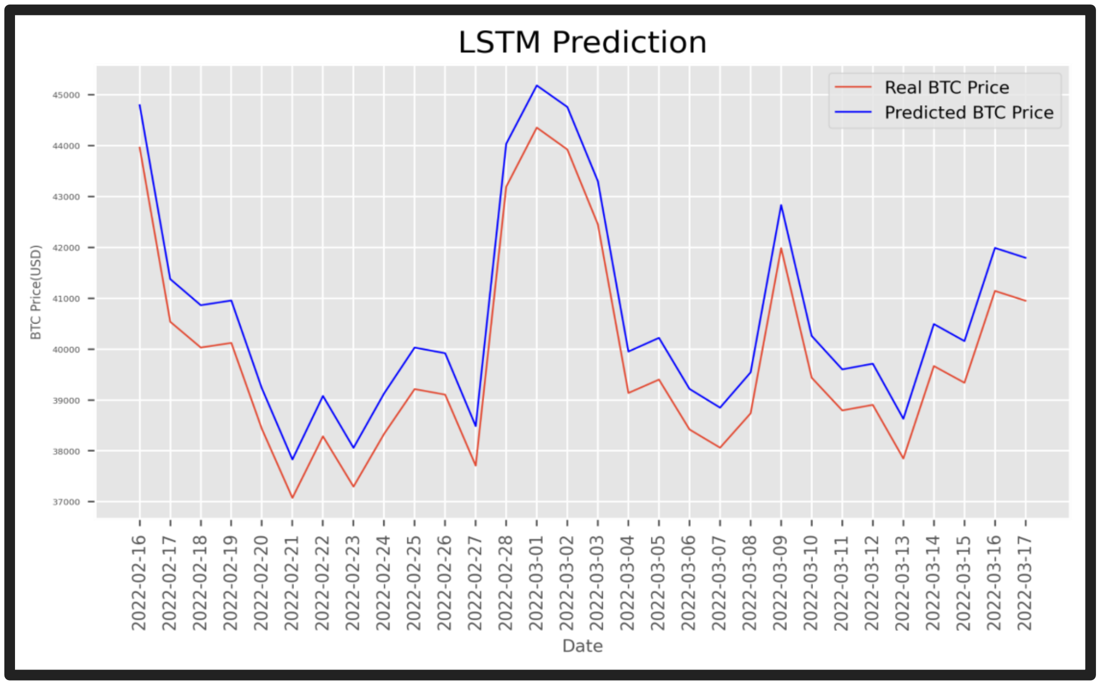
For sigmoid activation function:

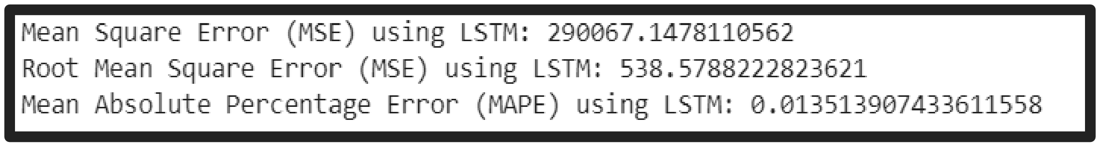




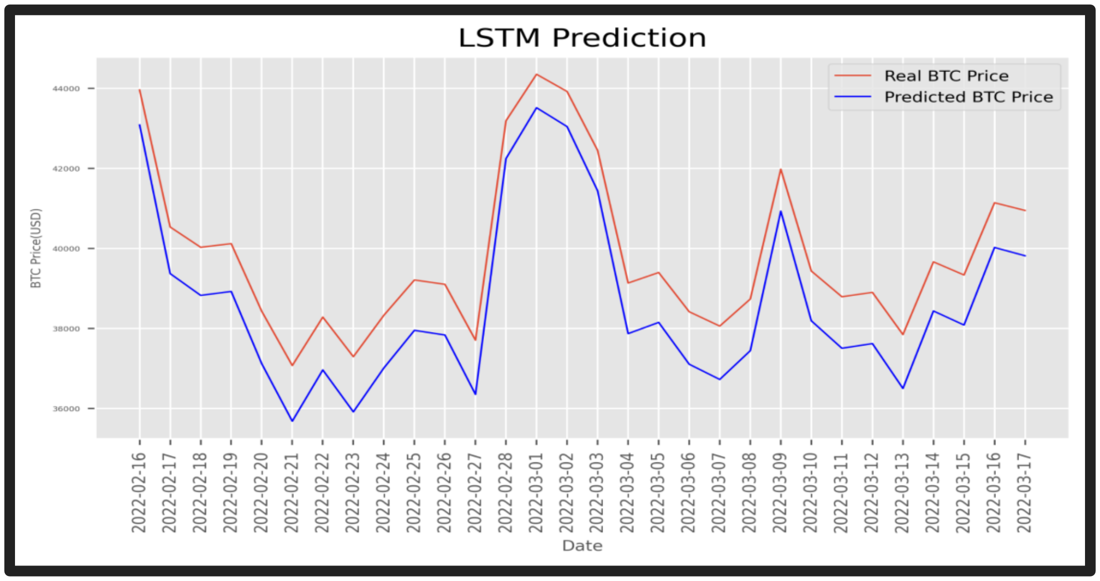
```
Mean Square Error (MSE) using LSTM: 176603.6491341054
Root Mean Square Error (MSE) using LSTM: 420.2423695132434
Mean Absolute Percentage Error (MAPE) using LSTM: 0.010558543005959493
```

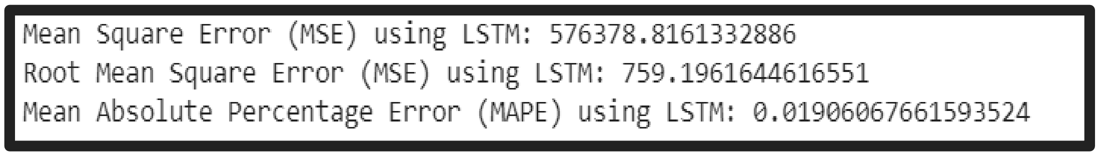
For tanh activation function:




```
Mean Square Error (MSE) using LSTM: 290067.1478110562
Root Mean Square Error (MSE) using LSTM: 538.5788222823621
Mean Absolute Percentage Error (MAPE) using LSTM: 0.013513907433611558
```

For ReLU activation function:

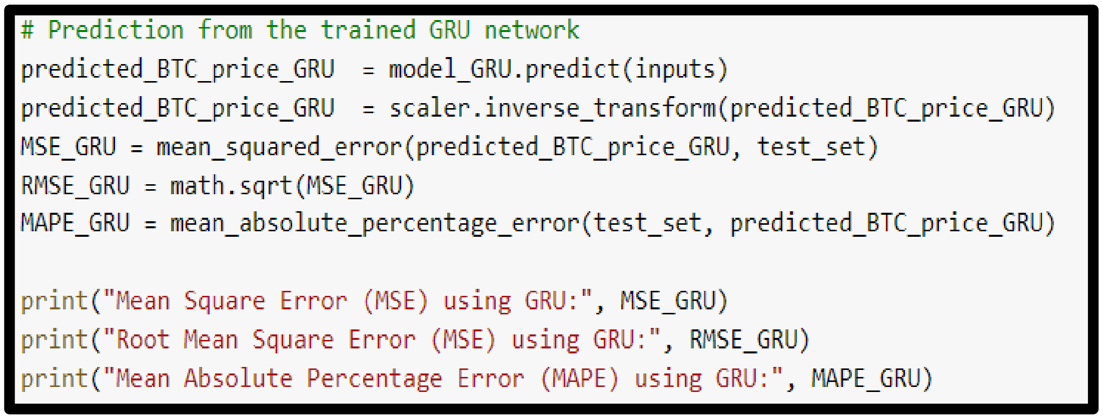




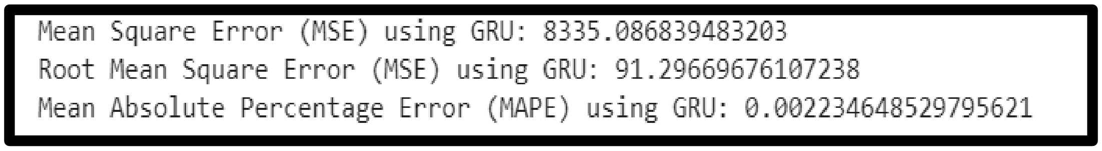
```
Mean Square Error (MSE) using LSTM: 576378.8161332886
Root Mean Square Error (MSE) using LSTM: 759.1961644616551
Mean Absolute Percentage Error (MAPE) using LSTM: 0.01906067661593524
```

8.2 GRU

On some smaller and less frequent datasets, GRUs have been found to perform better.

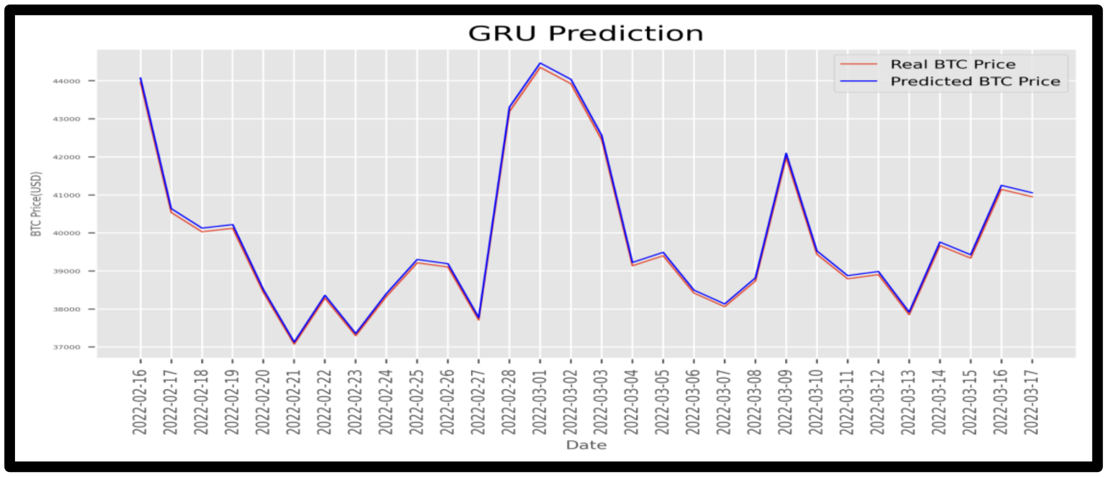


```
# Prediction from the trained GRU network
predicted_BTC_price_GRU  = model_GRU.predict(inputs)
predicted_BTC_price_GRU  = scaler.inverse_transform(predicted_BTC_price_GRU)
MSE_GRU = mean_squared_error(predicted_BTC_price_GRU, test_set)
RMSE_GRU = math.sqrt(MSE_GRU)
MAPE_GRU = mean_absolute_percentage_error(test_set, predicted_BTC_price_GRU)

print("Mean Square Error (MSE) using GRU:", MSE_GRU)
print("Root Mean Square Error (MSE) using GRU:", RMSE_GRU)
print("Mean Absolute Percentage Error (MAPE) using GRU:", MAPE_GRU)
```

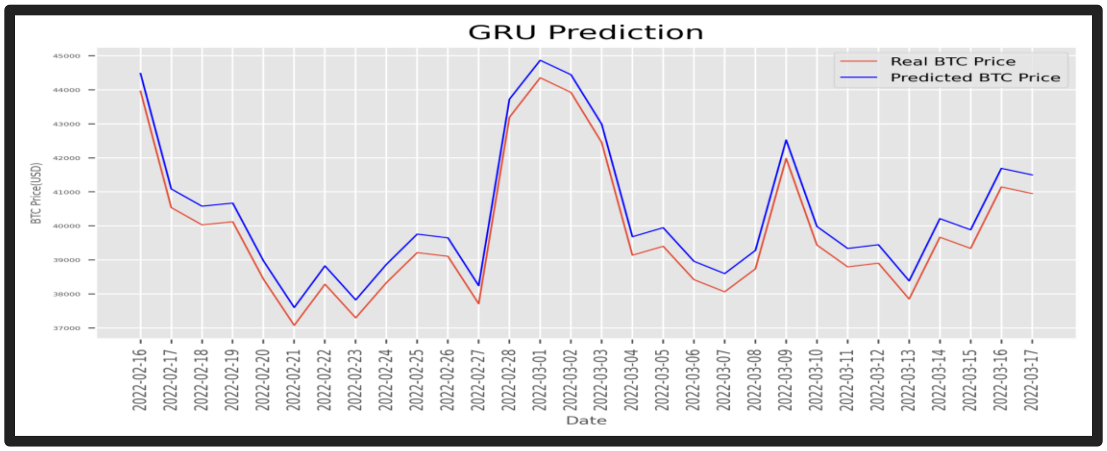


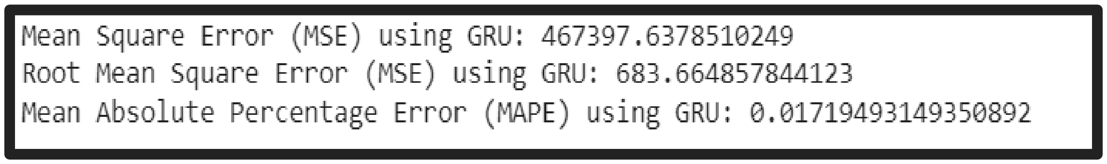
```
Mean Square Error (MSE) using GRU: 8335.086839483203
Root Mean Square Error (MSE) using GRU: 91.29669676107238
Mean Absolute Percentage Error (MAPE) using GRU: 0.002234648529795621
```

For sigmoid activation function:



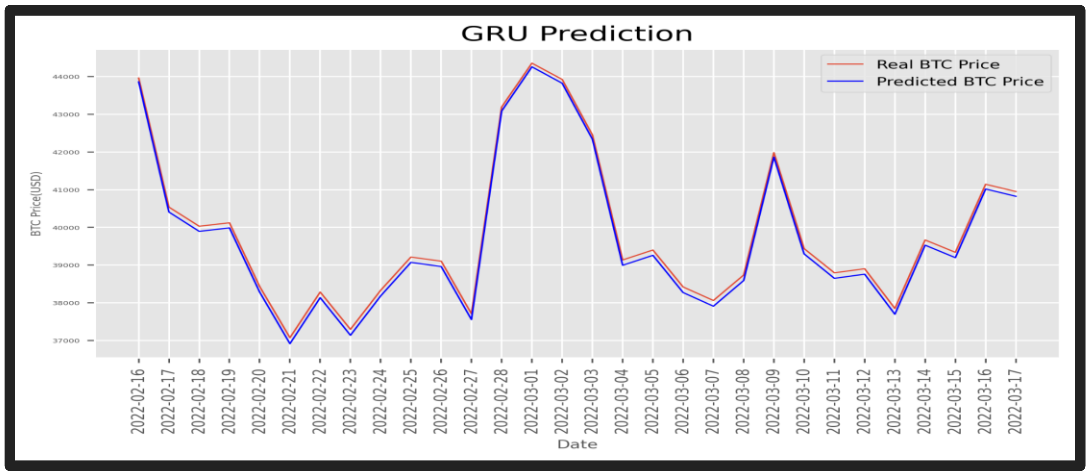
For tanh activation function:

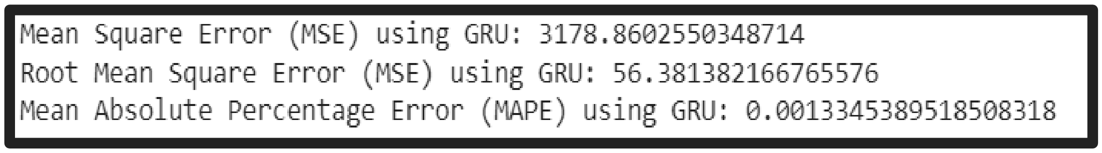




```
Mean Square Error (MSE) using GRU: 467397.6378510249
Root Mean Square Error (MSE) using GRU: 683.664857844123
Mean Absolute Percentage Error (MAPE) using GRU: 0.017194931493508092
```

For ReLU activation function:

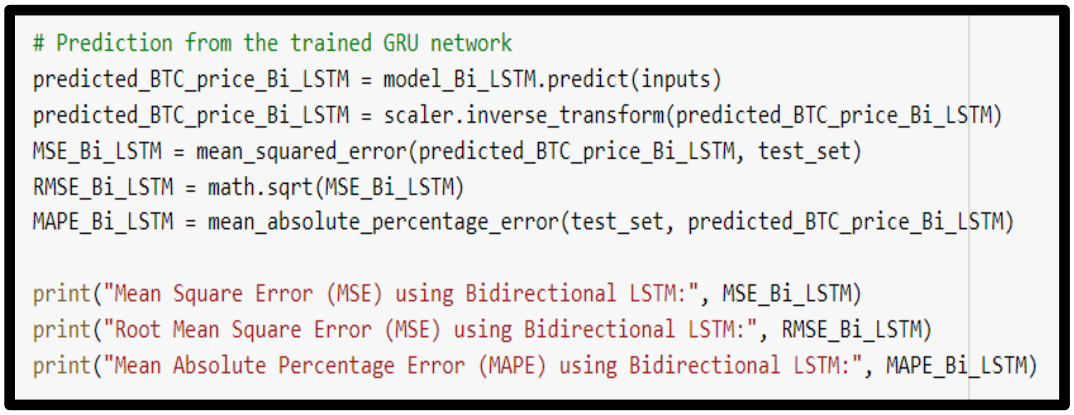




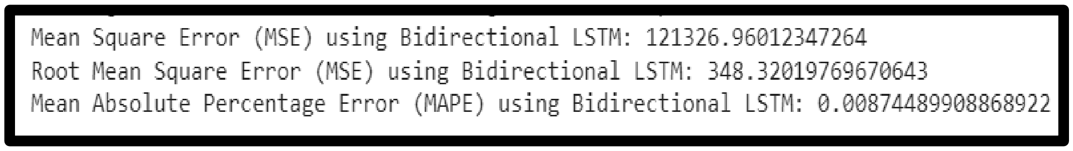
```
Mean Square Error (MSE) using GRU: 3178.8602550348714
Root Mean Square Error (MSE) using GRU: 56.381382166765576
Mean Absolute Percentage Error (MAPE) using GRU: 0.0013345389518508318
```

8.3 Bi-LSTM

Bidirectional long-short term memory (bi-lstm) is the process of constructing a neural network that can store sequence information in both ways, backwards (future to past) and forwards (past to future).

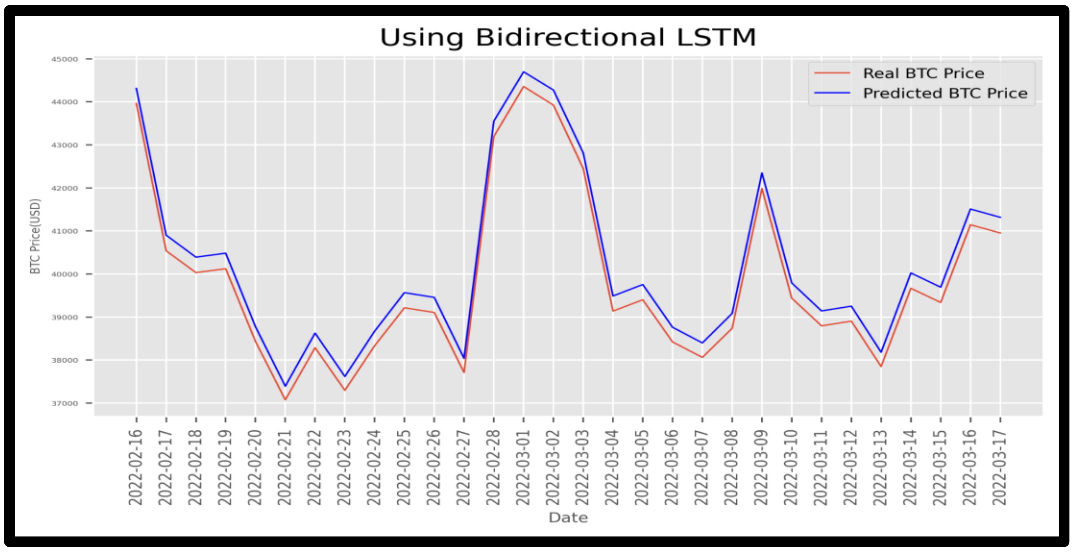


```
# Prediction from the trained GRU network
predicted_BTC_price_Bi_LSTM = model_Bi_LSTM.predict(inputs)
predicted_BTC_price_Bi_LSTM = scaler.inverse_transform(predicted_BTC_price_Bi_LSTM)
MSE_Bi_LSTM = mean_squared_error(predicted_BTC_price_Bi_LSTM, test_set)
RMSE_Bi_LSTM = math.sqrt(MSE_Bi_LSTM)
MAPE_Bi_LSTM = mean_absolute_percentage_error(test_set, predicted_BTC_price_Bi_LSTM)

print("Mean Square Error (MSE) using Bidirectional LSTM:", MSE_Bi_LSTM)
print("Root Mean Square Error (MSE) using Bidirectional LSTM:", RMSE_Bi_LSTM)
print("Mean Absolute Percentage Error (MAPE) using Bidirectional LSTM:", MAPE_Bi_LSTM)
```

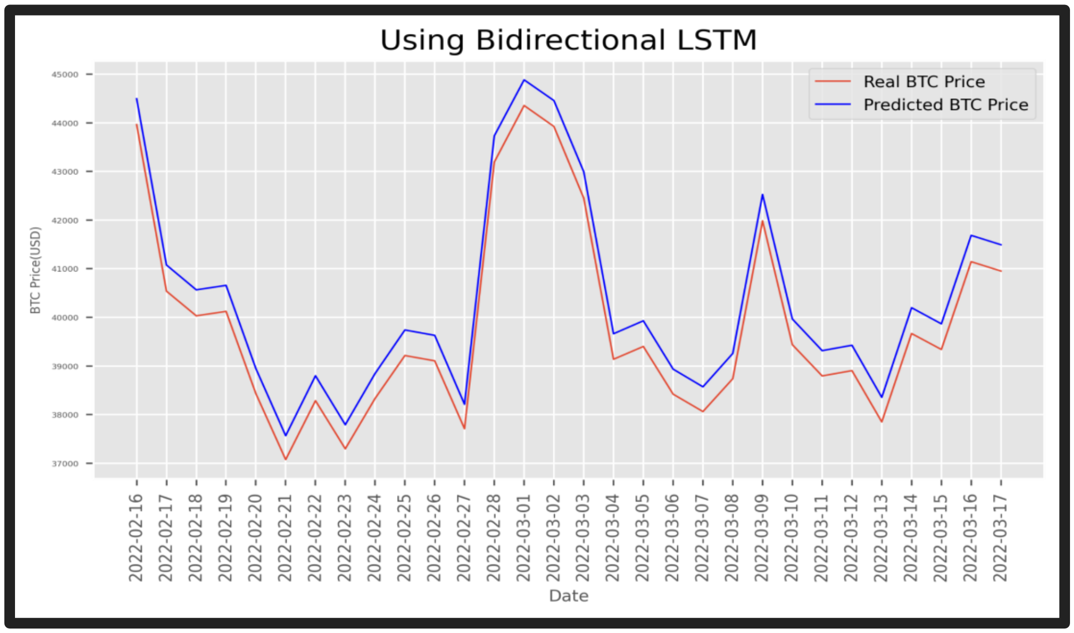


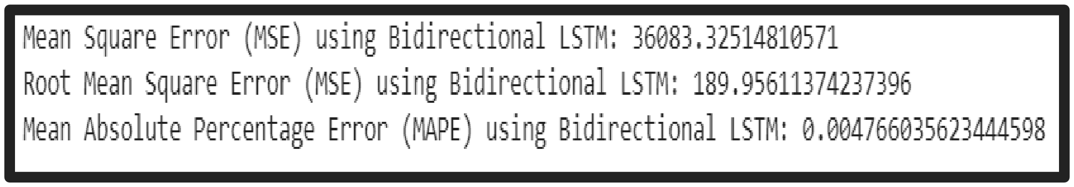
```
Mean Square Error (MSE) using Bidirectional LSTM: 121326.96012347264
Root Mean Square Error (MSE) using Bidirectional LSTM: 348.32019769670643
Mean Absolute Percentage Error (MAPE) using Bidirectional LSTM: 0.00874489908868922
```

For sigmoid activation function:



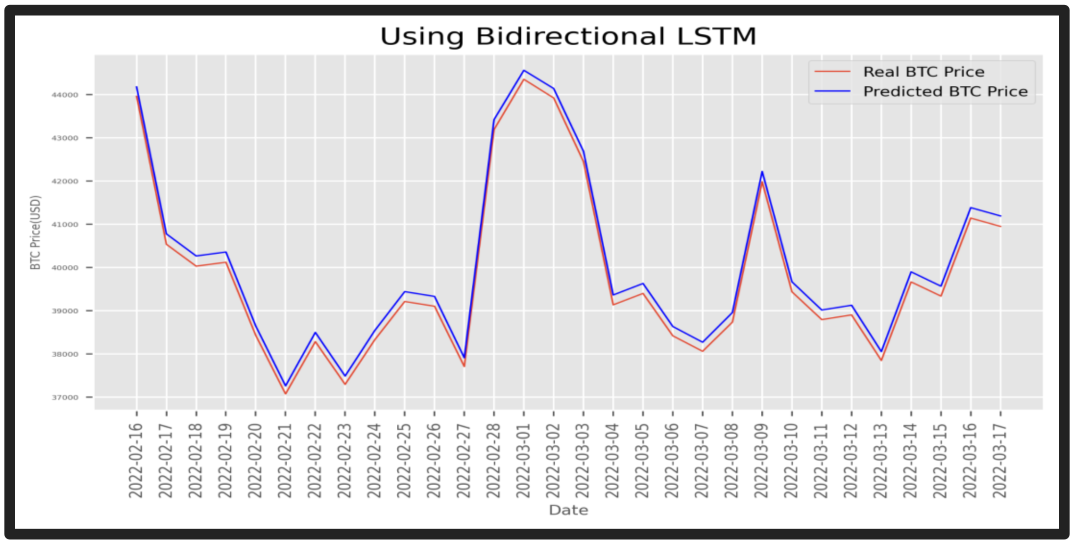
For tanh activation function:

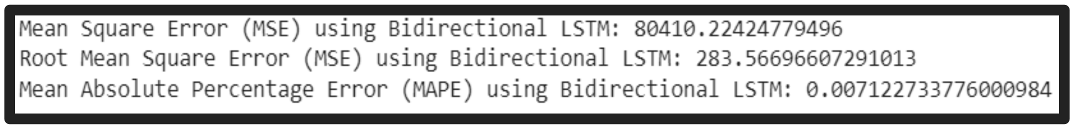




```
Mean Square Error (MSE) using Bidirectional LSTM: 36083.32514810571
Root Mean Square Error (MSE) using Bidirectional LSTM: 189.95611374237396
Mean Absolute Percentage Error (MAPE) using Bidirectional LSTM: 0.004766035623444598
```

For ReLU activation function:



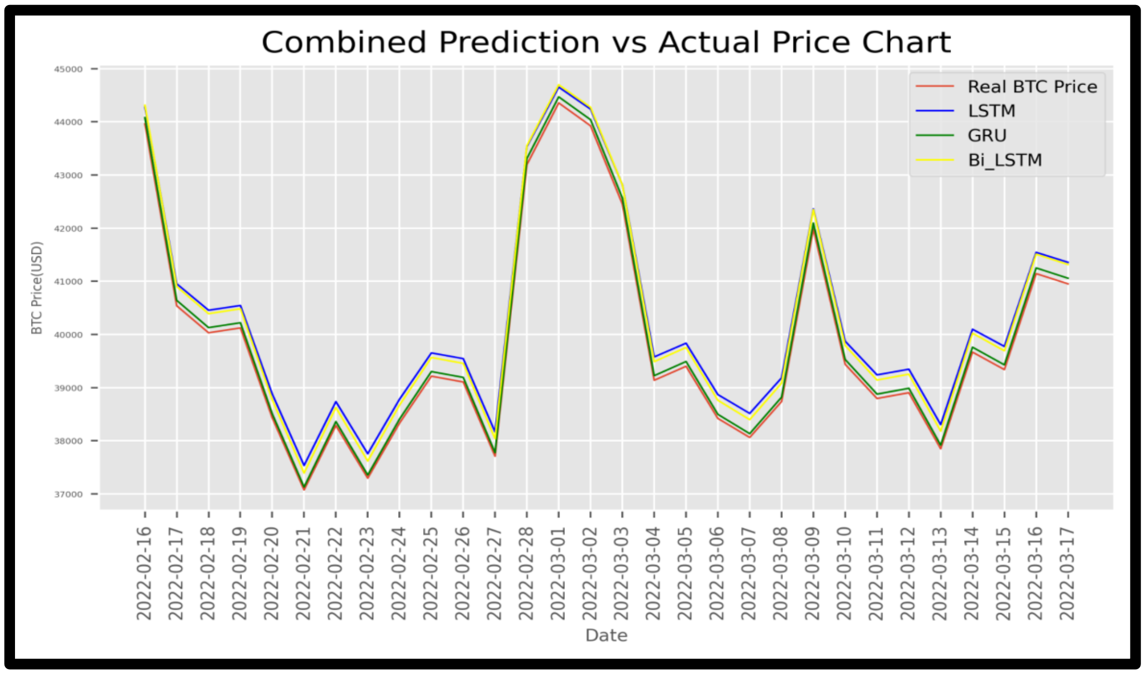


```
Mean Square Error (MSE) using Bidirectional LSTM: 80410.22424779496
Root Mean Square Error (MSE) using Bidirectional LSTM: 283.56696607291013
Mean Absolute Percentage Error (MAPE) using Bidirectional LSTM: 0.007122733776000984
```

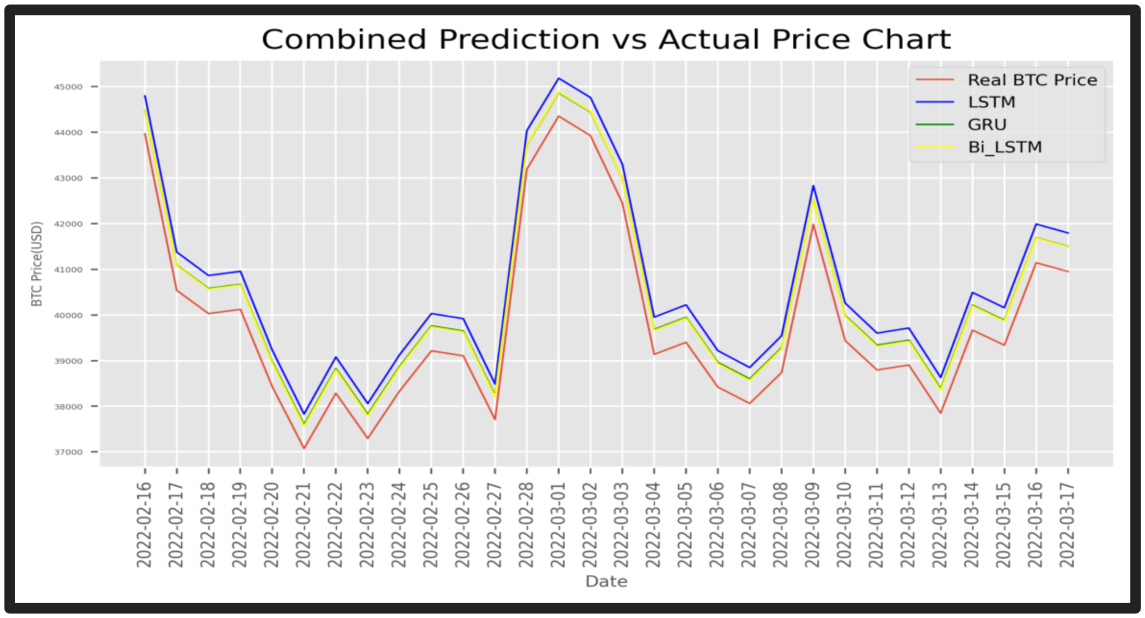
9. Best Results

Following figure shows the comparison between Combined Prediction and Actual Price Chart.

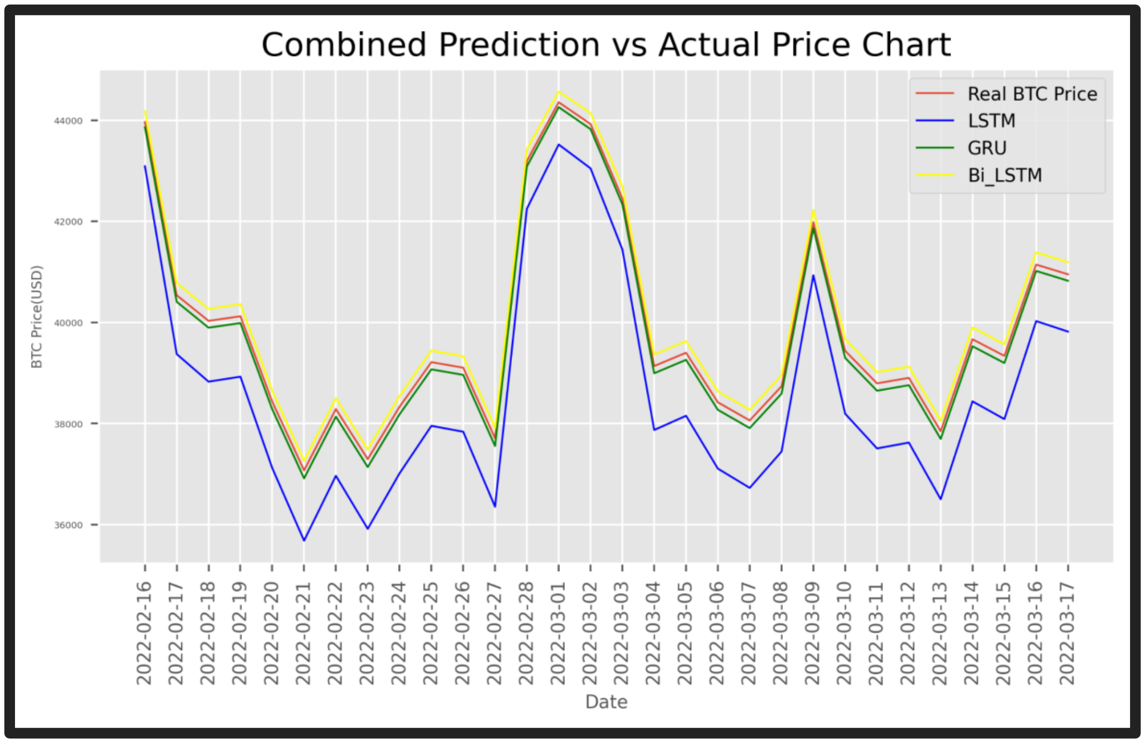
For Sigmoid activation function:



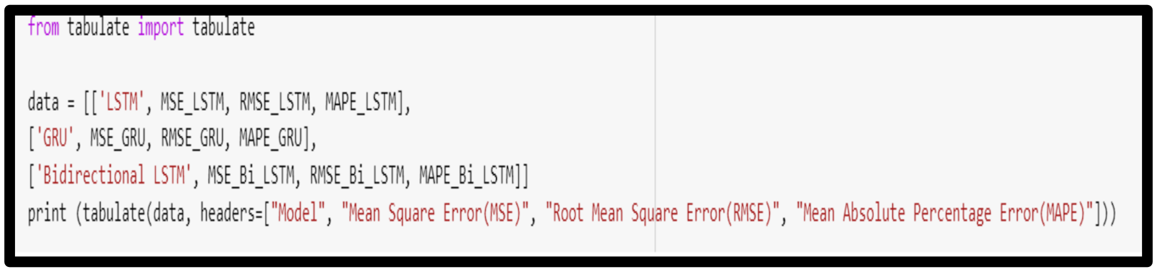
For tanh activation function:



For ReLU activation function:

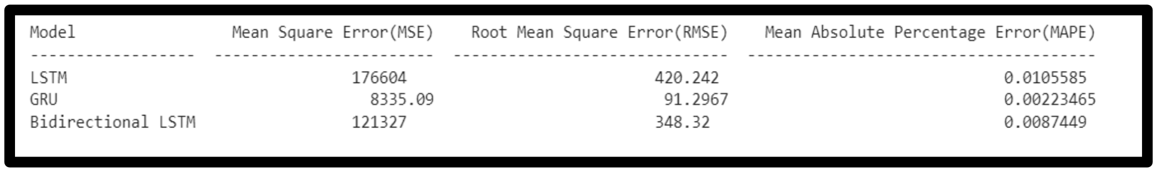


Here, in the final step we are going to analyze and compare the results of each model we have used and build a model in this project to find out which algorithm can be used best for the price forecasting of Bitcoin.



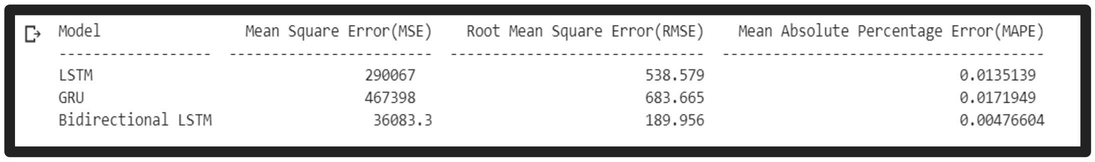
```
from tabulate import tabulate

data = [['LSTM', MSE_LSTM, RMSE_LSTM, MAPE_LSTM],
['GRU', MSE_GRU, RMSE_GRU, MAPE_GRU],
['Bidirectional LSTM', MSE_Bi_LSTM, RMSE_Bi_LSTM, MAPE_Bi_LSTM]]
print (tabulate(data, headers=["Model", "Mean Square Error(MSE)", "Root Mean Square Error(RMSE)", "Mean Absolute Percentage Error(MAPE)"]))
```

For sigmoid activation function:



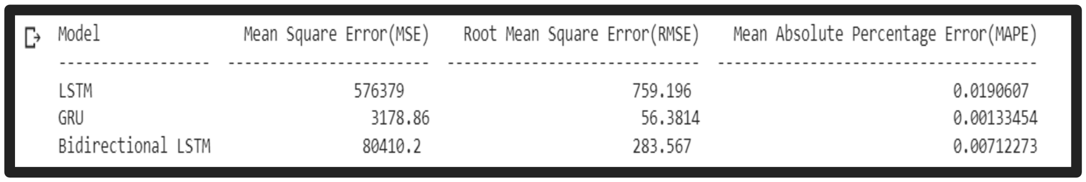
```
Model                 Mean Square Error(MSE)    Root Mean Square Error(RMSE)    Mean Absolute Percentage Error(MAPE)
------------------  ------------------------  ------------------------------  --------------------------------------
LSTM                               176604                          420.242                              0.0105585
GRU                                  8335.09                        91.2967                             0.00223465
Bidirectional LSTM                 121327                          348.32                               0.0087449
```

For tanh activation function:



```
Model                Mean Square Error(MSE)    Root Mean Square Error(RMSE)    Mean Absolute Percentage Error(MAPE)
-----------------  -----------------------  -----------------------------  -------------------------------------
LSTM                                290067                        538.579                               0.0135139
GRU                                 467398                        683.665                               0.0171949
Bidirectional LSTM                 36083.3                        189.956                               0.00476604
```

For ReLU activation function:



```
Model                Mean Square Error(MSE)    Root Mean Square Error(RMSE)    Mean Absolute Percentage Error(MAPE)
-----------------  -----------------------  -----------------------------  -------------------------------------
LSTM                                576379                        759.196                               0.0190607
GRU                                3178.86                        56.3814                               0.00133454
Bidirectional LSTM                 80410.2                        283.567                               0.00712273
```

From the above results, we find that GRU is the best overall model among the three models, LSTM, BiLSTM, and GRU, for predicting/forecasting the closing price of the cryptocurrency Bitcoin. But for the tanh activation function, the LSTM model performed a little bit better than GRU.

10. Conclusion

Since Bitcoin and Blockchain technology was introduced in 2008, it has taken a predominant place in the cryptocurrency domain. There are millions of users around the world, especially in the United States. Predicting the price of cryptocurrencies has been a popular topic, from which we can take advantage of the latest technology for investment. In this project, we used yahoo finance from the Kaggle website and took 100 epochs for each model, so in total 300 epochs of information as we implemented three models, to forecast the price of the cryptocurrency. Three models were implemented: Long Short-term Memory Model (LSTM), Gated Recurrent Units Model (GRU), and Bidirectional Long Short-term Memory Model (BiLSTM).

Performance measures were conducted to test the prediction as an output of different models. We also tested the efficiency and accuracy of LSTM, BiLSTM, and GRU model in predicting Bitcoin prices. Then, we compared the actual and predicted prices. Each performed rather well, but the Bi-Directional LSTM model was found to be a better indicator of trend, which would be more relevant from a trading perspective. When the Bi-LSTM model was used to forecast values, it performed quite well. Although the prices were not predicted exactly, the trend was on point! For the overall prediction, however, GRU is the best according to the dataset we used, and it had the best matched predicted value of bitcoin.

11. Responsibility of Team Members

Nikhil Katariya

Collecting the dataset
Implementation of LSTM model
Report
Presentation Slides

Vishwa Pankajbhai Doshi

Implementation of GRU model
Calculating performance metrics and visualizing the combined forecasts
Report
Presentation Slides

Jatin Raheja

Code Analysis
Implementation of GRU model
Report
Presentation Slides

Akshat Jain

Code for plotting graphs using matplotlib
Implementation of LSTM model
Report
Presentation Slides

Rohit Mehta

Code Analysis
Implementation of Bi-LSTM model
Report
Presentation Slides

12. Code

https://github.com/jraheja1994/CS_5661_Project.git

13. Reference

1. Xu, Yike.; *Bitcoin Price Forecast Using LSTM and GRU Recurrent networks, and Hidden Markov Mode, PhD thesis: UCLA, 2020*
2. Mohammad J. Hamayel, Amani Yousef Owda.; *A Novel Cryptocurrency Price Prediction Model Using GRU, LSTM and bi-LSTM Machine Learning Algorithms*
3. Huynh, T.L.D*.; Nasir, M.A.; Vo, X.V.; Nguyen, T.T. "Small things matter most": The spillover effects in the cryptocurrency market and gold as a silver bullet. North Am. J. Econ. Financ. 2020.*
4. ChristopherOlah;*"UnderstandingLSTMNetworks.", 2015*