DEEP LEARNING

CRYPTO CURRENCY FUTURE PRICE PREDICTION USING STACK LSTM

AHMED DIDER RAHAT

s40183@bht-berlin.de

Introduction: Crypto currency is digital currency in modern finance system. It's an end-to-end system that can enable anyone anywhere to send and receive payments. Instead of being physical money carried around and exchanged in the real world, crypto currency payments exist purely as digital entries in the cyber space. As a result, the interest on crypto investment increase dramatically. In this project I try to implement stack **LSTM** to predict the future price of two most widely used crypto currency **BITCOIN** and **ETHEREUM**.

LSTM: Long short-term memory (LSTM) is an artificial neural network used in the field deep learning. LSTM has feedback connections. It is a recurrent neural network that can process not only single data points, but also entire sequences of data. LSTM networks are well-suited to classifying, processing and making predictions based on **time series data**.

Time series data: A time series is a series of data points indexed in time order. Most commonly, a time series is a sequence taken at successive equally spaced points in time. Thus it is a sequence of discrete-time data.

Stacked-LSTM: The Stacked LSTM is an extension to this model that has multiple hidden LSTM layers where each layer contains multiple memory cells.
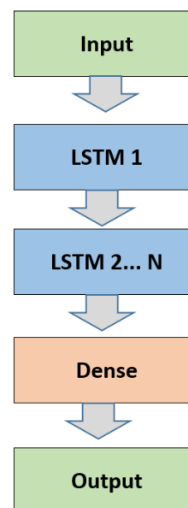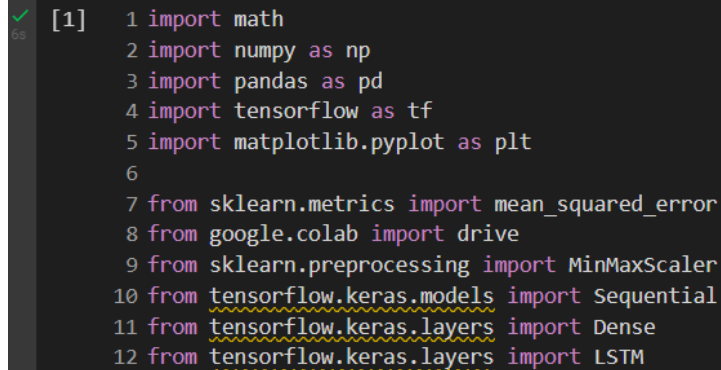

Figure 01: Stacked LSTM with N-Hidden LSTM Layers

My Dataset: To train and test my implementation I used two type of crypto currency data (Bitcoin and Ethereum). All the data are downloaded from finance.yahoo.com.

➢ **Bitcoin data:** There are total 2,829 data points and each of them has 7 columns (Date, Open, High, Low, Close, Adj. Close, Volume). These data are available from 9/17/2014 to 6/15/2022.

➢ **Ethereum data:** Total number of 1,680 data points and also has same columns as Bitcoin and the data points are distributed from 2017-11-09 to 2022-06-15.

Implementation:

1. **IDE:** I used google colab for this project as my IDE.

2. **Import necessary libraries:** I used several libraries in my implementation. So, at first import them in my code base.

```
[1]    1 import math
       2 import numpy as np
       3 import pandas as pd
       4 import tensorflow as tf
       5 import matplotlib.pyplot as plt
       6
       7 from sklearn.metrics import mean_squared_error
       8 from google.colab import drive
       9 from sklearn.preprocessing import MinMaxScaler
      10 from tensorflow.keras.models import Sequential
      11 from tensorflow.keras.layers import Dense
      12 from tensorflow.keras.layers import LSTM
```

Figure 02: Import libraries in IDE

3. **Connect and Read data file:** As I use data file from google drive. So, I used mount to connect the notebook with google drive.

```
[2]  1 drive.mount('/content/drive')

Mounted at /content/drive

[3]  1 all_data = pd.read_csv('/content/drive/MyDrive/datasets/ETH-USD.csv')
     2 print(f'Total number of rows/data points: {len(all_data)}')

Total number of rows/data points: 1680
```

Figure 03: Connect with drive and read input file

4. **Print head and trail of the data:** First and last few data of the input data file:

```
[4]  1 all_data.head()
```

	Date	Open	High	Low	Close	Adj Close	Volume
0	2017-11-09	308.644989	329.451996	307.056000	320.884003	320.884003	893249984
1	2017-11-10	320.670990	324.717987	294.541992	299.252991	299.252991	885985984
2	2017-11-11	298.585999	319.453003	298.191986	314.681000	314.681000	842300992
3	2017-11-12	314.690002	319.153015	298.513000	307.907990	307.907990	1613479936
4	2017-11-13	307.024994	328.415009	307.024994	316.716003	316.716003	1041889984

```
 1 all_data.tail()
```

	Date	Open	High	Low	Close	Adj Close	Volume
1675	2022-06-11	1665.217896	1679.314209	1507.038940	1529.663452	1529.663452	21127089064
1676	2022-06-12	1530.189697	1539.705078	1436.183960	1445.216553	1445.216553	23465074882
1677	2022-06-13	1443.835449	1448.738037	1181.948242	1204.582764	1204.582764	45162788786
1678	2022-06-14	1204.555298	1252.471802	1094.701904	1211.662842	1211.662842	33327826525
1679	2022-06-15	1204.709473	1219.879272	1025.767456	1110.117432	1110.117432	28916948992

Figure 04: head and tail of the data

5. Remove unnecessary data: As I am concern with the Close data and want to use time series. So, I only took the values of Close columns.

```
[6]   1 df = all_data.reset_index()['Close']
      2 df.head()

0    320.884003
1    299.252991
2    314.681000
3    307.907990
4    316.716003
Name: Close, dtype: float64
```

Figure 05: only took the Close data for implementation

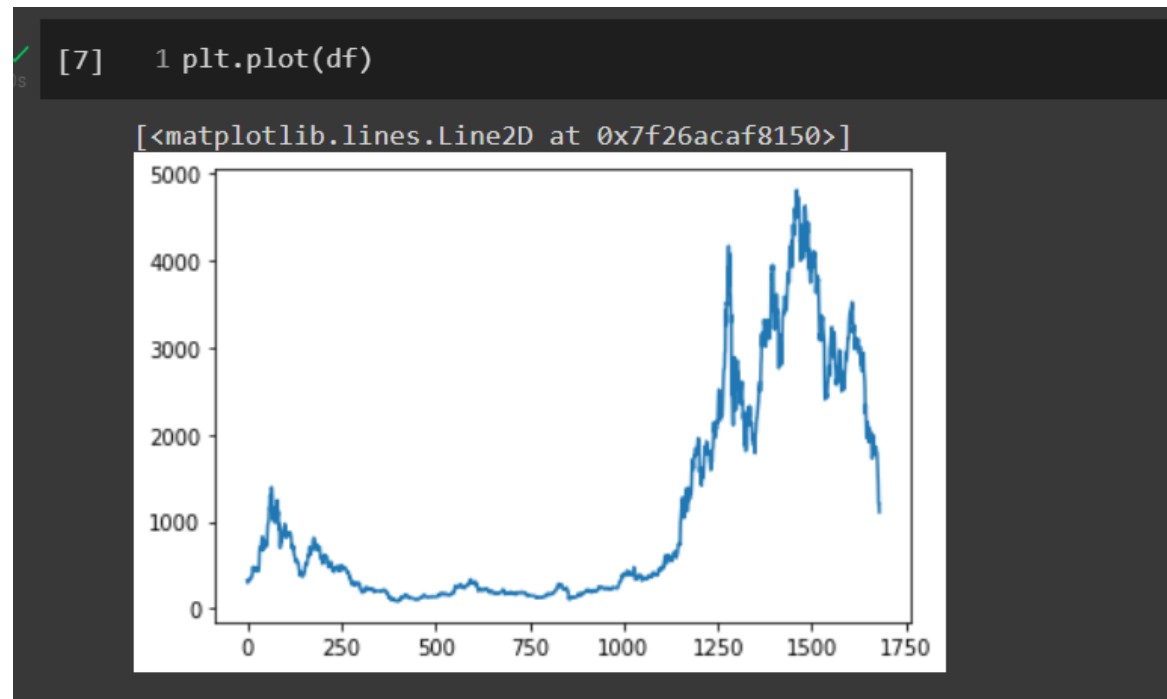**6. Input data plotting:** Plot the close data graphically.

Figure 06: Plot close data

7. Scaling: As LSTM is highly sensitive to scale. So, I have to scale the data into [0, 1] interval.

```
[8]  1 scaler = MinMaxScaler(feature_range=(0,1))
     2 scaled_df = scaler.fit_transform(np.array(df).reshape(-1,1))
     3 print(scaled_df)

[[0.0500395 ]
 [0.0454642 ]
 [0.04872747]
 ...
 [0.23695576]
 [0.2384533 ]
 [0.21697484]]
```

Figure 07: Scale data from 0 to 1

8. Data Split: I split my data into 3 categories train 60%, validation 20%, and test 20%.

```
[9]  1 train_size = int(len(scaled_df)*.60)
     2 validation_size = int(len(scaled_df)*.20)
     3 test_size = len(df) - (train_size + validation_size)
     4
     5 print(f'Total train data: {train_size}, Validation size: {validation_size} and total test data: {test_size}')

Total train data: 1008, Validation size: 336 and total test data: 336


[20] 1 train_data, validation_data, test_data = scaled_df[0: train_size], scaled_df[train_size:(train_size + validation_size)], \
     2   scaled_df[(train_size + validation_size): len(scaled_df)]
```

Figure 08: Split the input data

9. Time series creation: This function create time series from the input data set.

```
1  def create_time_series(dataset, time_step=1):
2      dataX, dataY=[], []
3
4      for i in range(len(dataset)-time_step-1):
5          a = dataset[i: (i+time_step), 0]
6          dataX.append(a)
7          dataY.append(dataset[i+time_step, 0])
8
9      return np.array(dataX), np.array(dataY)
```

Figure 09: time series function

```
1  time_step=7
2  X_train, Y_train = create_time_series(train_data, time_step)
3  X_validation, Y_validation = create_time_series(validation_data, time_step)
4  X_test, Y_test = create_time_series(test_data, time_step)
```

Figure 10: time calling

10. Re-shape time series: In this part I am reshaping the input sets into 3-d array as per the LSTM requirement.

```
[25]  1  # reshape for LSTM
      2  X_train = X_train.reshape(X_train.shape[0], X_train.shape[1], 1)
      3  X_vlidation = X_validation.reshape(X_validation.shape[0], X_validation.shape[1], 1)
      4  X_test = X_test.reshape(X_test.shape[0], X_train.shape[1], 1)
```

Figure 11: re-shape time calling

11. Model Initialization: In my code base I used 3 layers of LSTM. Though I used stacked LSTM the first 2 return sequence are true. I also used 'adam' as the optimizer and 'mean_squared_error' as loss function.

```
1   # init model
2   model = Sequential()
3   model.add(LSTM(64, return_sequences=True, input_shape=(X_train.shape[1],1)))
4   model.add(LSTM(64, return_sequences=True))
5   model.add(LSTM(64))
6   model.add(Dense(1))
7   model.compile(loss='mean_squared_error', optimizer='adam')
```

Figure 12: Init. LSTM model

```
[27]  1   print(model.summary())

Model: "sequential"

_____
 Layer (type)                Output Shape              Param #
=================================================================
 lstm (LSTM)                 (None, 7, 64)             16896

 lstm_1 (LSTM)               (None, 7, 64)             33024

 lstm_2 (LSTM)               (None, 64)                33024

 dense (Dense)               (None, 1)                 65

=================================================================
Total params: 83,009
Trainable params: 83,009
Non-trainable params: 0
_____
None
```

Figure 13: Summary of LSTM model

12. Model Training: In this portion I trained my model. The method fit has 6 parameters. I used the epochs and batch size as hyper parameter with some other parameter mention in the result analysis portion.

```
1   history = model.fit(X_train, Y_train, validation_data=(X_validation, Y_validation), epochs=10, batch_size=16, verbose=1)

Epoch 1/10
63/63 [==============================] - 2s 29ms/step - loss: 4.7491e-05 - val_loss: 8.2049e-04
Epoch 2/10
63/63 [==============================] - 2s 31ms/step - loss: 5.1332e-05 - val_loss: 0.0012
Epoch 3/10
63/63 [==============================] - 2s 31ms/step - loss: 4.6914e-05 - val_loss: 0.0010
Epoch 4/10
63/63 [==============================] - 2s 29ms/step - loss: 3.9028e-05 - val_loss: 0.0023
Epoch 5/10
63/63 [==============================] - 2s 28ms/step - loss: 4.8572e-05 - val_loss: 0.0022
Epoch 6/10
63/63 [==============================] - 1s 19ms/step - loss: 4.8763e-05 - val_loss: 0.0011
Epoch 7/10
63/63 [==============================] - 1s 19ms/step - loss: 3.8023e-05 - val_loss: 8.9513e-04
Epoch 8/10
63/63 [==============================] - 1s 20ms/step - loss: 4.2405e-05 - val_loss: 6.0495e-04
Epoch 9/10
63/63 [==============================] - 1s 19ms/step - loss: 4.2995e-05 - val_loss: 0.0013
Epoch 10/10
63/63 [==============================] - 1s 19ms/step - loss: 3.7280e-05 - val_loss: 6.8445e-04
```

Figure 14: Fit the model

13. Calculate RMSE values: To calculate the RMSE value I need to re-scale the prediction and actual data and then used mean_square_error function to calculate the value.

```
[36]  1   train_predict = model.predict(X_train)
      2
      3   #re-scale into previous scale
      4   train_predict_original = scaler.inverse_transform(train_predict)
      5   Y_train_original = scaler.inverse_transform(Y_train.reshape(-1,1))
      6
      7   train_rsme = round(math.sqrt(mean_squared_error(Y_train_original, train_predict_original)), 4)
      8   print(f'Total train RSME= {train_rsme}')

Total train RSME= 26.011
```

Figure 15: calculate RMSE from training data

```
[37]    1    validation_predict = model.predict(X_validation)
        2
        3    #re-scale into previous scale
        4    validation_predict_original = scaler.inverse_transform(validation_predict)
        5    Y_validation_original = scaler.inverse_transform(Y_validation.reshape(-1,1))
        6
        7    validation_rsme = round(math.sqrt(mean_squared_error(Y_validation_original, validation_predict_original)), 4)
        8    print(f'Total validation RSME= {validation_rsme}')

    Total validation RSME= 123.6879
```

Figure 16: calculate RMSE from validation data

```
[39]    1    test_predict = model.predict(X_test)
        2
        3    #re-scale into previous scale
        4    test_predict_original = scaler.inverse_transform(test_predict)
        5    Y_test_original = scaler.inverse_transform(Y_test.reshape(-1,1))
        6
        7    test_rsme = round(math.sqrt(mean_squared_error(Y_test_original, test_predict_original)), 4)
        8    print(f'Total test RSME= {test_rsme}')

    Total test RSME= 198.2239
```

Figure 17: calculate RMSE from test data

14. Plot the original vs. predication value: I used pyplot function to plot the prediction value and predicted value.

```
[40]  1  new_train_predict = np.empty_like(df)
      2  new_train_predict[:]   = np.NaN
      3  new_train_predict[time_step:train_size-1] = train_predict_original.reshape(1,-1)
      4
      5  new_validation_predict = np.empty_like(df)
      6  new_validation_predict[:]   = np.NaN
      7  new_validation_predict[(train_size+time_step) : (train_size+validation_size-1)] = validation_predict_original.reshape(1,-1)
      8
      9  new_test_predict = np.empty_like(df)
     10  new_test_predict[:]   = np.NaN
     11  new_test_predict[(train_size+validation_size+time_step) : len(df)-1] = test_predict_original.reshape(1,-1)
     12
     13
     14  plt.plot(df)
     15  plt.plot(new_train_predict)
     16  plt.plot(new_validation_predict)
     17  plt.plot(new_test_predict)
     18  plt.show()
```

Figure 18: code for drawing the actual and prediction data
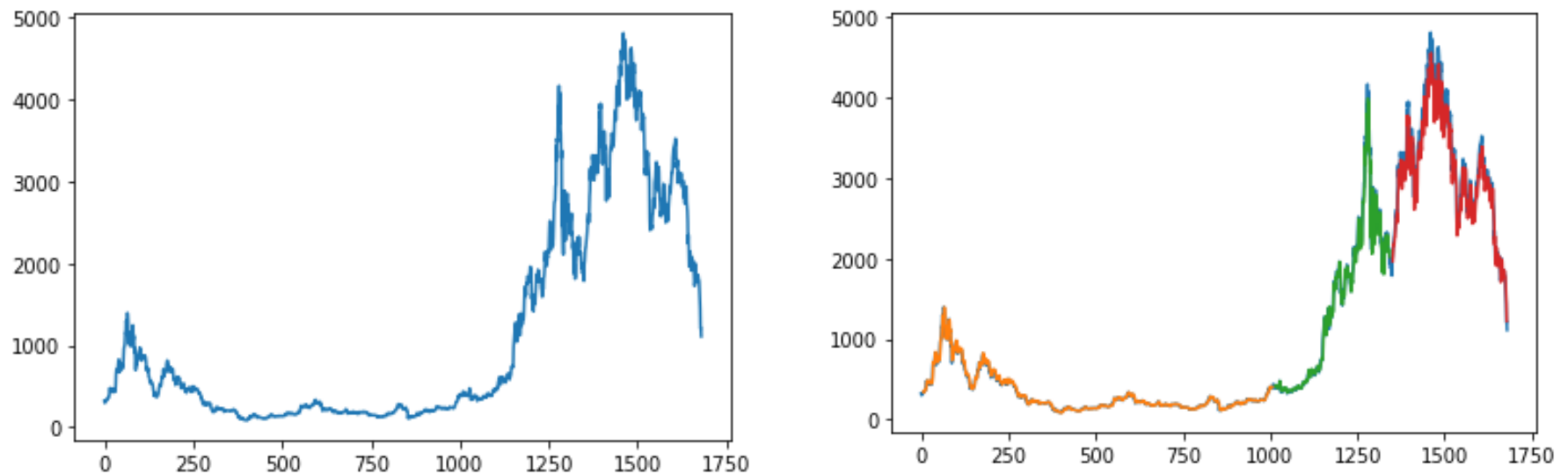
Figure 19: visualization of actual and prediction data

Result: In this project I used 4 hyper parameters. They are time_step, lstm_unit, epocs, batch_size.

➢ time_step: This parameter for time series number of previous day for prediction. The values are 7, 14, and 28 days.

➢ lstm_unit: Define the LSTM units in each LSTM layers and the values are 64, and 128.

➢ epochs: The number of epochs is the number of complete passes through the training dataset. Values are 100, and 500.

➢ batch_size: The batch size is a number of samples processed before the model is updated. Values are 16, and 64.

As there are total 3, 2, 2, 2 numbers of parameter. So, I have to run total (3 x 2 x 2 x 2) = 24 times for each data set. And I have two dataset. So, I need to run my algorithm 48 times and the all possible values are given below:

SN	Data Set	size	Time step	LSTM unit	Epoch	batch size	RMSE Value			graph
							train	validation	test	
1	ETHEREUM	1680	7	64	100	16	32.7397	135.7386	179.9898	ETH_01.png
2	ETHEREUM	1680	7	64	100	64	27.991	157.5079	346.8778	ETH_02.png
3	ETHEREUM	1680	7	64	500	16	24.1677	1142.4956	3589.8872	ETH_03.png
4	ETHEREUM	1680	7	64	500	64	27.9442	180.9189	349.129	ETH_04.png
5	ETHEREUM	1680	7	128	100	16	32.2787	241.2323	535.1814	ETH_05.png
6	ETHEREUM	1680	7	128	100	64	29.4337	130.1892	210.2508	ETH_06.png
7	ETHEREUM	1680	7	128	500	16	25.7114	673.5333	1763.0242	ETH_07.png
8	ETHEREUM	1680	7	128	500	64	24.5529	154.3715	322.0906	ETH_08.png
9	ETHEREUM	1680	14	64	100	16	25.7608	167.3771	306.2787	ETH_09.png
10	ETHEREUM	1680	14	64	100	64	27.0288	166.5529	368.1227	ETH_10.png

11	ETHEREUM	1680	14	64	500	16	24.0209	352.0767	950.0765	ETH_11.png
12	ETHEREUM	1680	14	64	500	64	25.2772	171.015	320.046	ETH_12.png
13	ETHEREUM	1680	14	128	100	16	31.1989	181.9564	383.8307	ETH_13.png
14	ETHEREUM	1680	14	128	100	64	26.3979	167.5377	327.0955	ETH_14.png
15	ETHEREUM	1680	14	128	500	16	25.4947	575.5442	1918.1468	ETH_15.png
16	ETHEREUM	1680	14	128	500	64	25.2096	158.4983	354.5904	ETH_16.png
17	ETHEREUM	1680	28	64	100	16	25.3652	219.656	440.0884	ETH_17.png
18	ETHEREUM	1680	28	64	100	64	28.4892	203.3473	455.5522	ETH_18.png
19	ETHEREUM	1680	28	64	500	16	21.902	845.7621	2059.503	ETH_19.png
20	ETHEREUM	1680	28	64	500	64	24.269	190.8735	419.5623	ETH_20.png
21	ETHEREUM	1680	28	128	100	16	25.7966	150.3631	297.699	ETH_21.png
22	ETHEREUM	1680	28	128	100	64	26.3979	149.5504	278.656	ETH_22.png
23	ETHEREUM	1680	28	128	500	16	21.1161	651.9912	1181.6978	ETH_23.png
24	ETHEREUM	1680	28	128	500	64	27.0435	185.1505	415.8409	ETH_24.png
For BIT-COIN data:										
25	BITCOIN	2829	7	64	100	16	260.1122	362.4712	4031.6923	BTC_01.png
26	BITCOIN	2829	7	64	100	64	322.7874	409.0278	4510.9961	BTC_02.png
27	BITCOIN	2829	7	64	500	16	253.065	369.4141	3086.9748	BTC_03.png
28	BITCOIN	2829	7	64	500	64	261.1643	372.9401	4166.797	BTC_04.png
29	BITCOIN	2829	7	128	100	16	261.5801	364.0091	3393.2677	BTC_05.png

30	BITCOIN	2829	7	128	100	64	291.1744	483.1328	3646.7425	BTC_06.png
31	BITCOIN	2829	7	128	500	16	277.5947	383.6192	5101.0248	BTC_07.png
32	BITCOIN	2829	7	128	500	64	330.3293	377.9306	8309.7074	BTC_08.png
33	BITCOIN	2829	14	64	100	16	298.4092	454.4378	2969.9365	BTC_09.png
34	BITCOIN	2829	14	64	100	64	282.0142	367.1674	4227.4353	BTC_10.png
35	BITCOIN	2829	14	64	500	16	287.4418	401.0623	11047.1719	BTC_11.png
36	BITCOIN	2829	14	64	500	64	269.5564	370.1261	7332.2935	BTC_12.png
37	BITCOIN	2829	14	128	100	16	268.4907	378.7115	4515.9218	BTC_13.png
38	BITCOIN	2829	14	128	100	64	286.0174	385.6968	2392.9758	BTC_14.png
39	BITCOIN	2829	14	128	500	16	237.2501	403.4383	22797.9305	BTC_15.png
40	BITCOIN	2829	14	128	500	64	298.1996	497.6476	6523.0263	BTC_16.png
41	BITCOIN	2829	28	64	100	16	334.0174	512.9509	6483.3487	BTC_17.png
42	BITCOIN	2829	28	64	100	64	302.7308	498.1252	4850.3347	BTC_18.png
43	BITCOIN	2829	28	64	500	16	270.0163	389.4572	41867.1592	BTC_19.png
44	BITCOIN	2829	28	64	500	64	271.0288	387.5522	4667.3347	BTC_20.png
45	BITCOIN	2829	28	128	100	16	363.8542	522.943	4832.1972	BTC_21.png
46	BITCOIN	2829	28	128	100	64	283.0244	428.8339	3599.4624	BTC_22.png
47	BITCOIN	2829	28	128	500	16	257.7507	396.5924	39996.7215	BTC_23.png
48	BITCOIN	2829	28	128	500	64	271.4777	422.1988	5872.7336	BTC_24.png
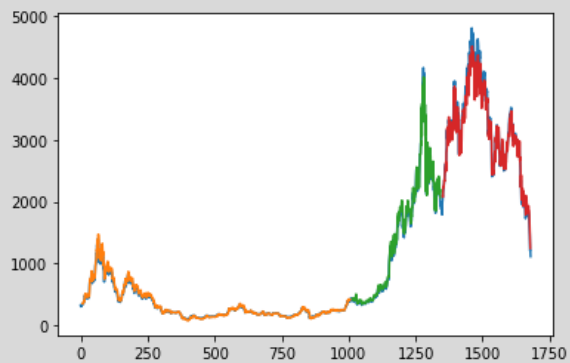
Figure-20: ETH_1.png
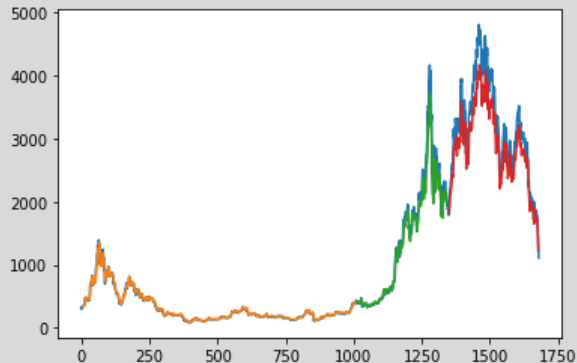
Figure-21: ETH_02.png
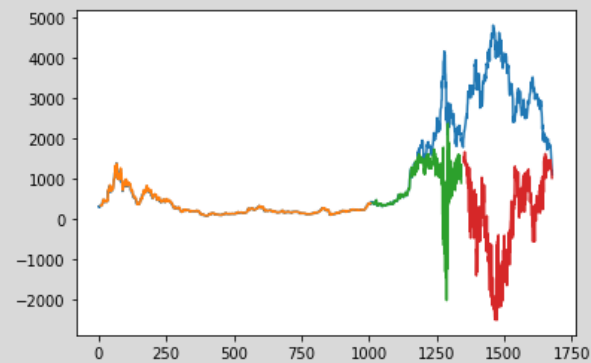
Figure-22: ETH_03.png
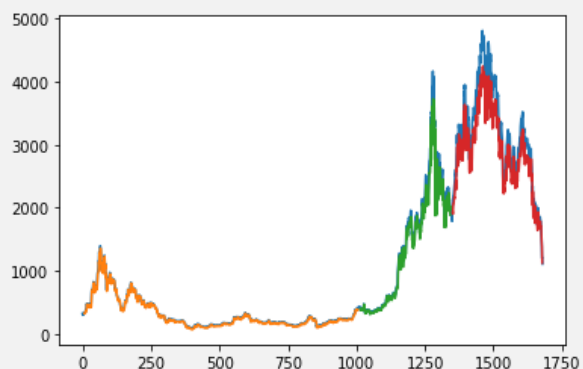
Figure-23: ETH_04.png
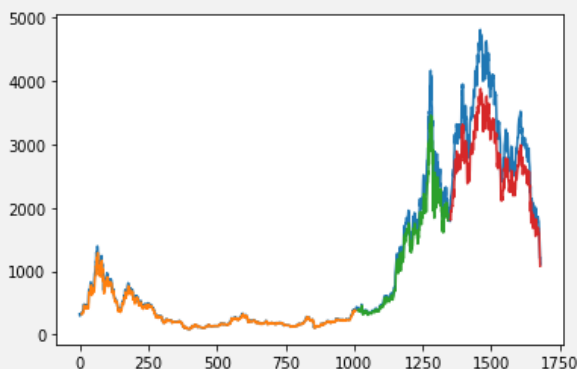
Figure-24: ETH_05.png
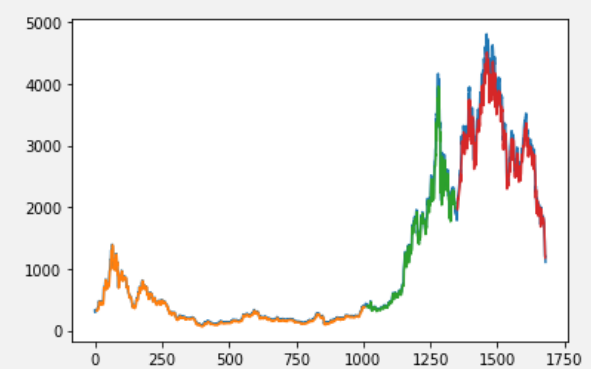
Figure-25: ETH_06.png
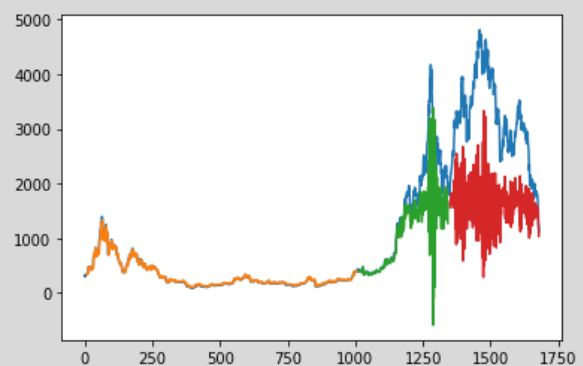
Figure-26: ETH_07.png
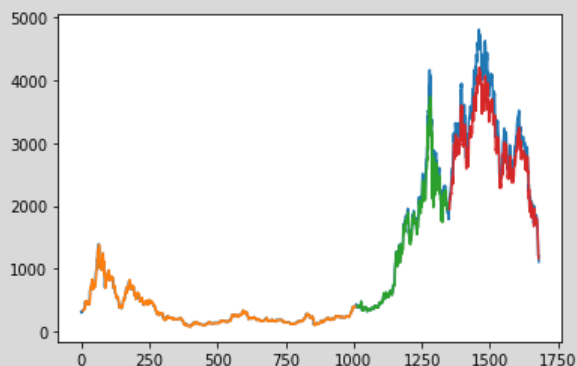
Figure-27: ETH_08.png
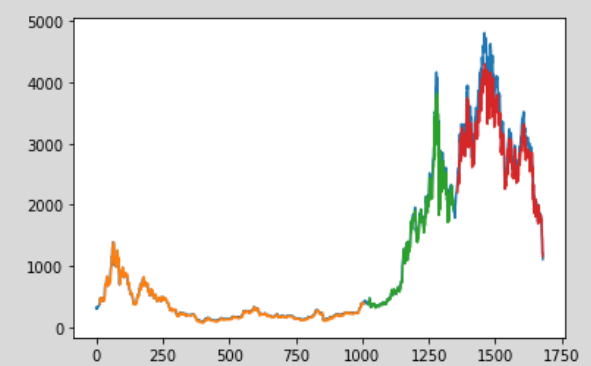
Figure-28: ETH_09.png
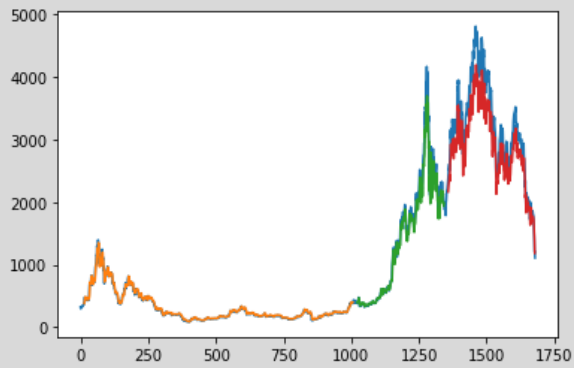
Figure-29: ETH_10.png
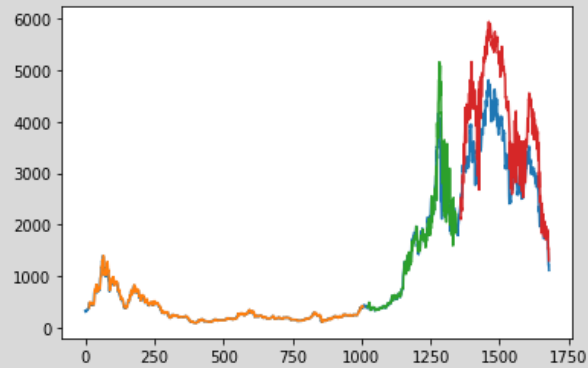
Figure-30: ETH_11.png
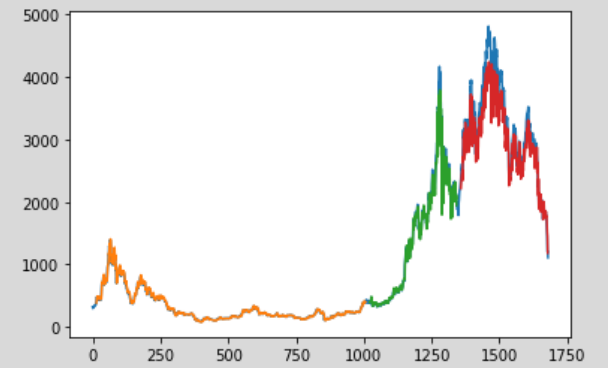
Figure-31: ETH_12.png
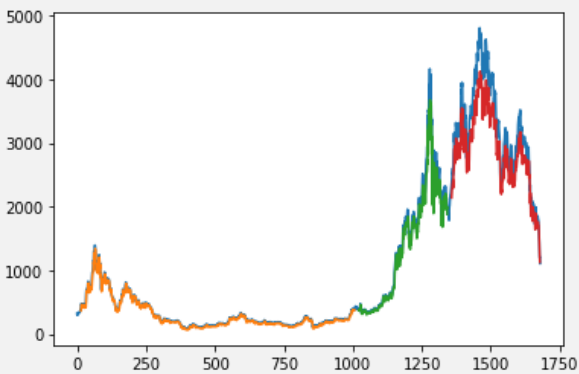
Figure-32: ETH_13.png
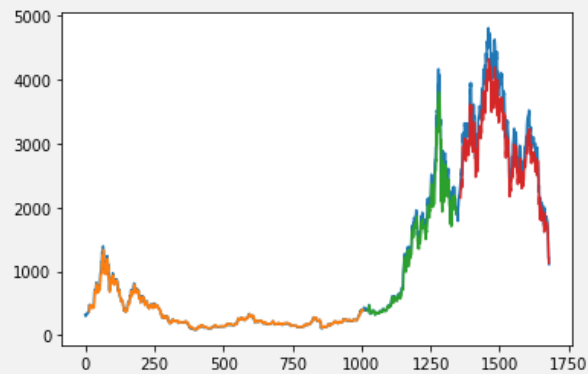
Figure-33: ETH_14.png
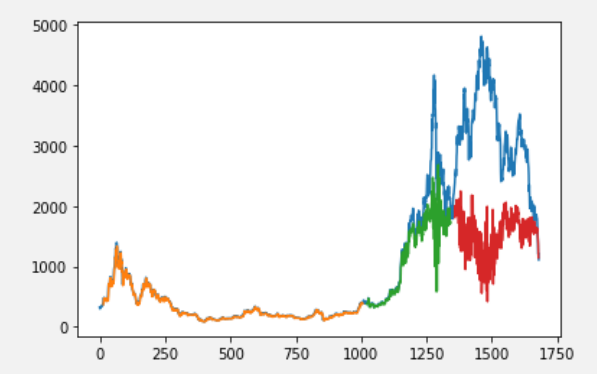
Figure-34: ETH_15.png
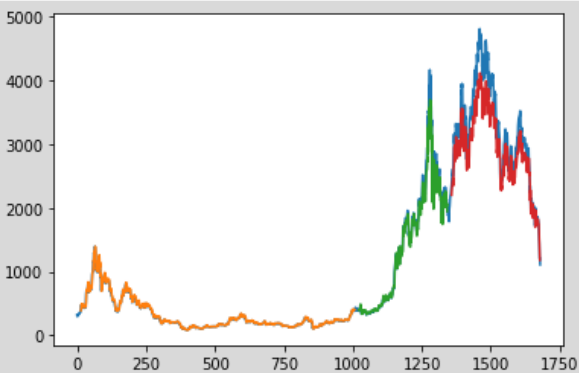
Figure-35: ETH_16.png
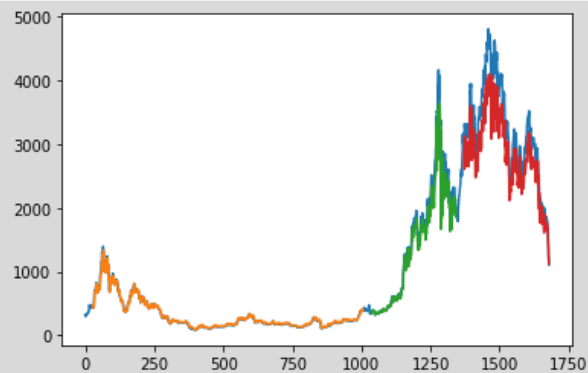
Figure-36: ETH_17.png
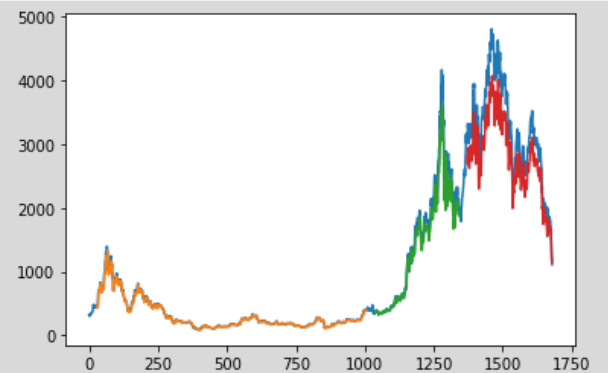
Figure-37: ETH_18.png
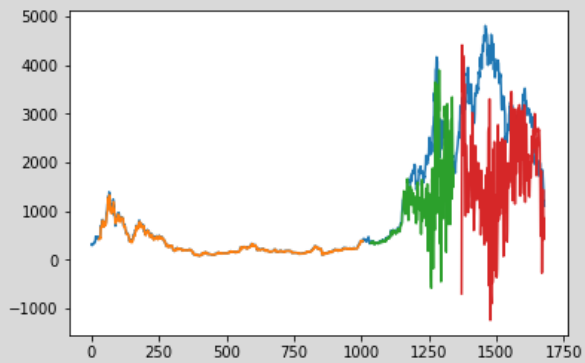
Figure-38: ETH_19.png
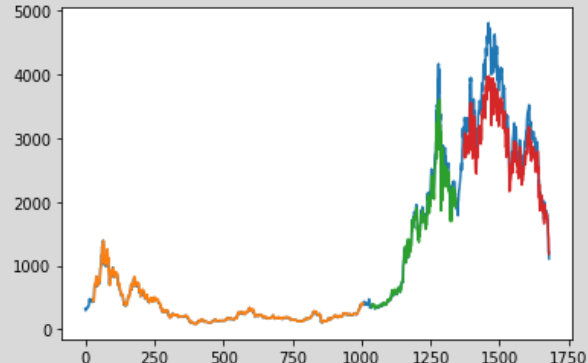
Figure-39: ETH_20.png
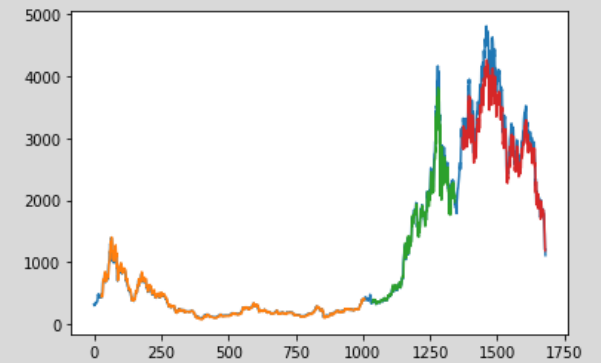
Figure-40: ETH_21.png
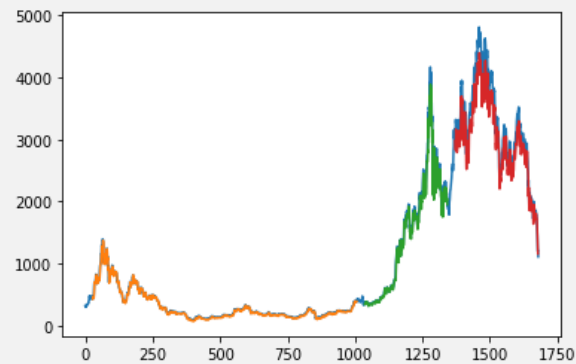
Figure-41: ETH_22.png
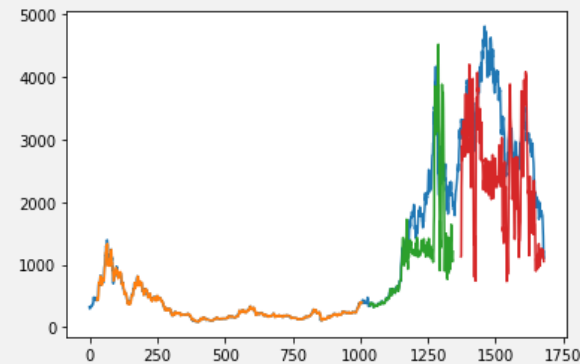
Figure-42: ETH_23.png
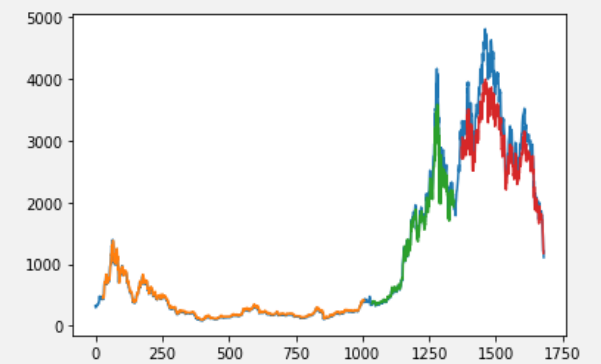
Figure-43: ETH_24.png
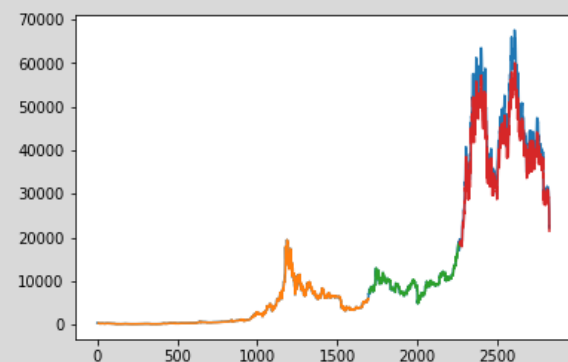
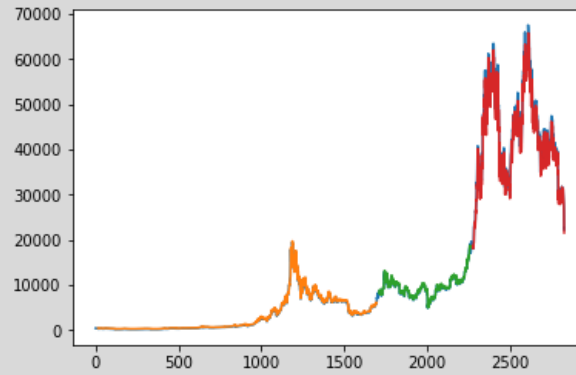Figure-44: BTC_01.png
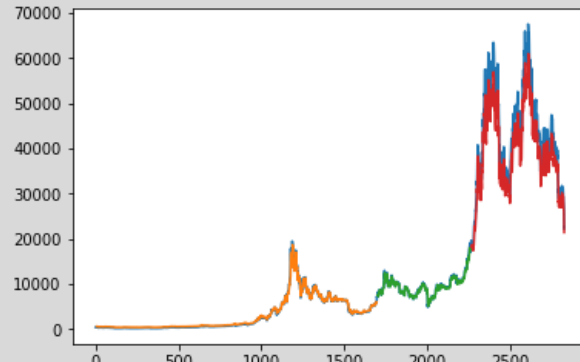
Figure-45: BTC_02.png
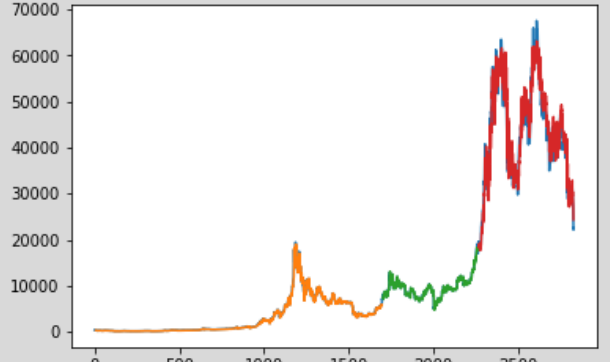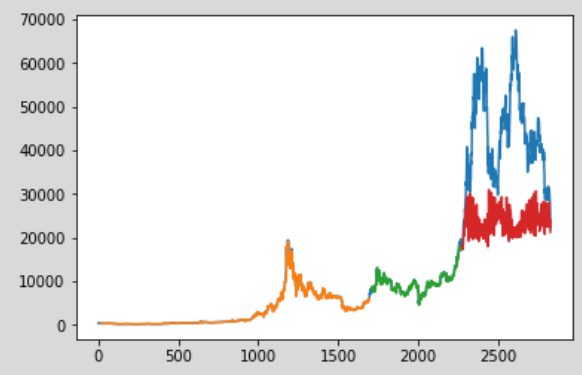

Figure-46: BTC_03.png
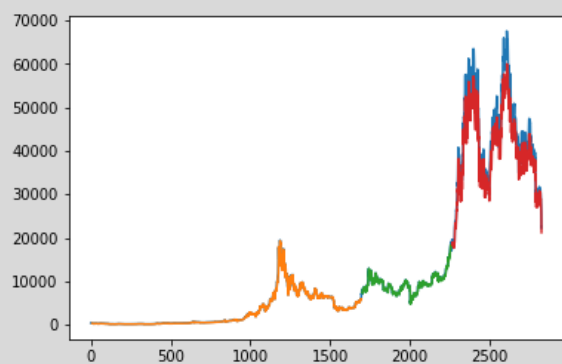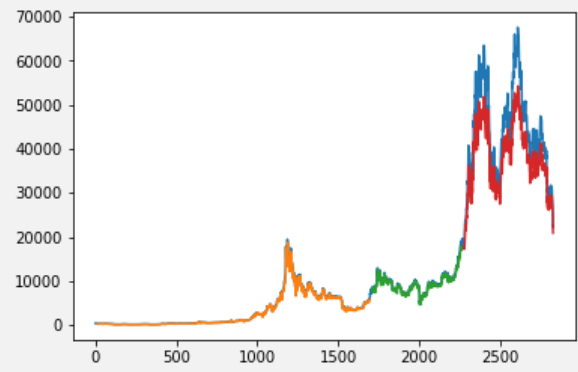
Figure-47: BTC_04.png
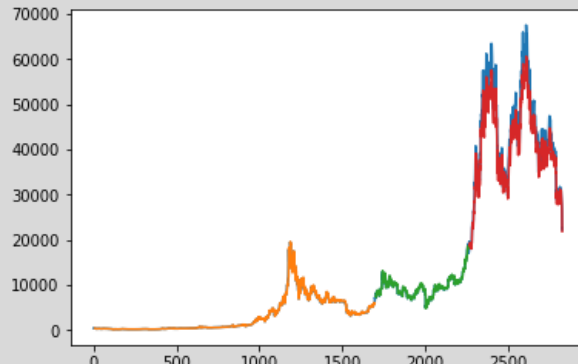
Figure-48: BTC_05.png
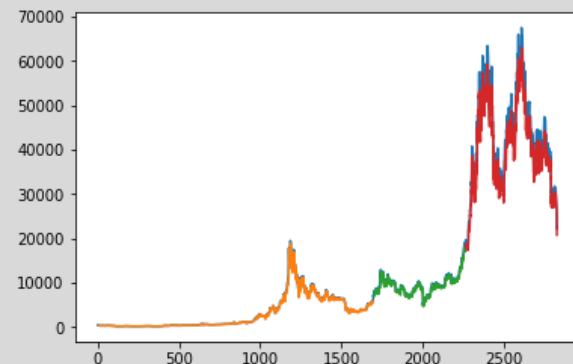
Figure-49: BTC_06.png
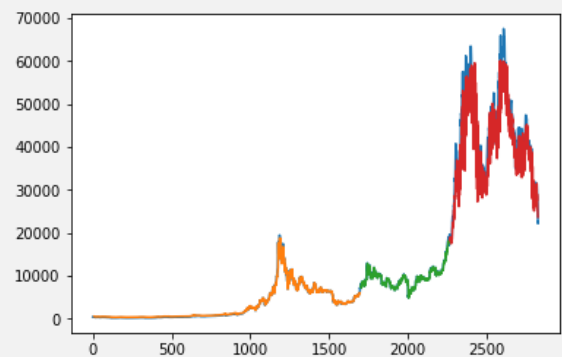
Figure-50: BTC_07.png
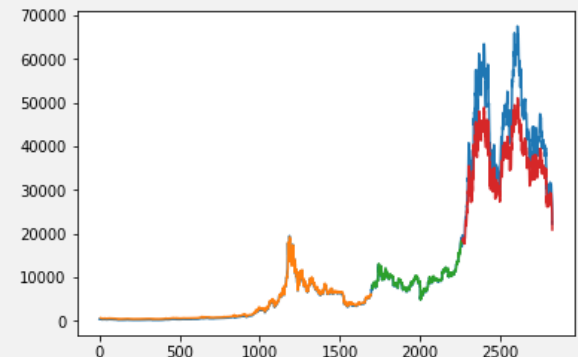
Figure-51: BTC_08.png
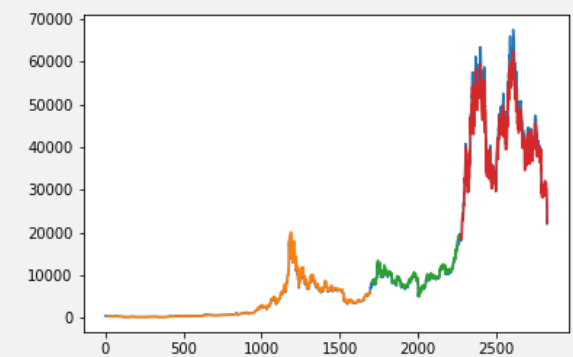
Figure-52: BTC_09.png
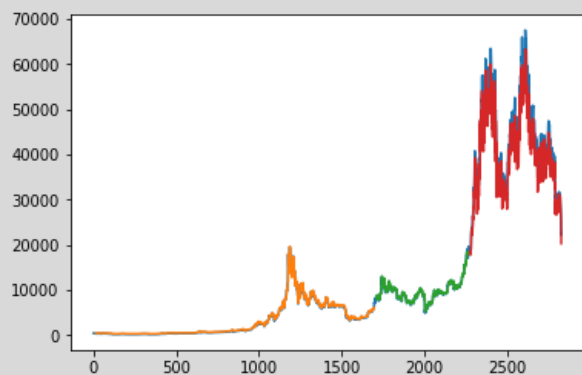
Figure-53: BTC_10.png
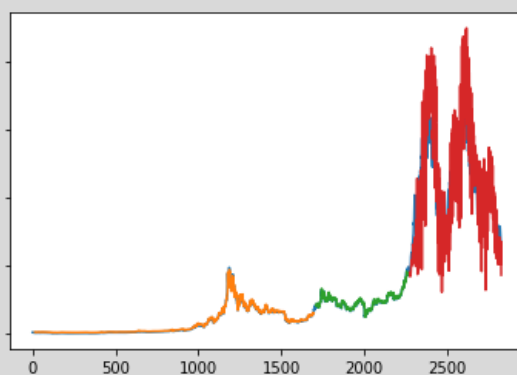
Figure-54: BTC_11.png
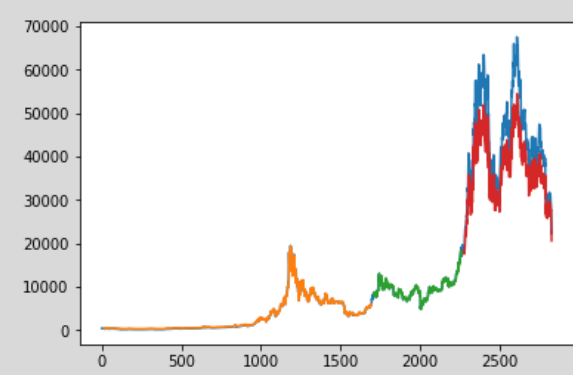
Figure-55: BTC_12.png
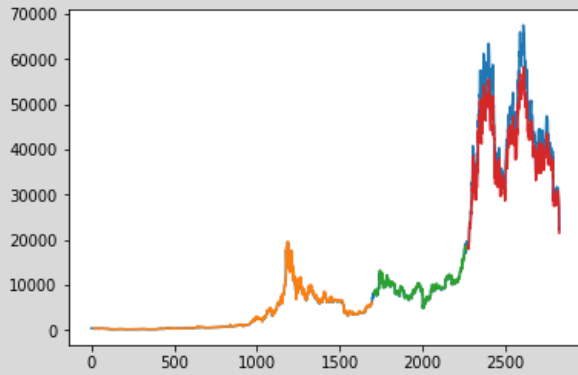
Figure-56: BTC_13.png

Figure-57: BTC_14.png

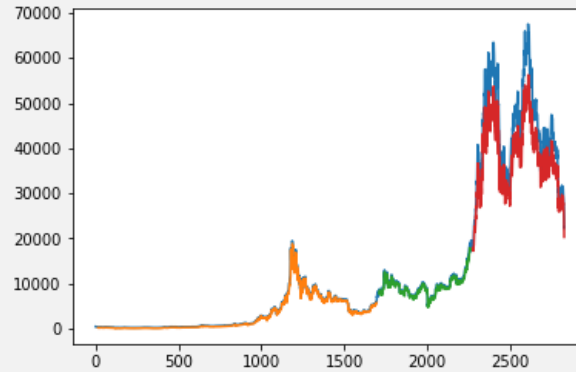Figure-58: BTC_15.png
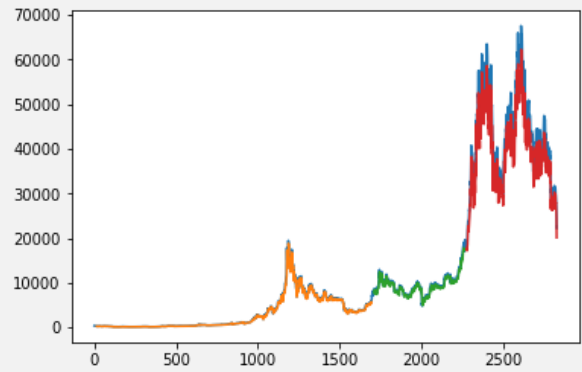

Figure-59: BTC_16.png

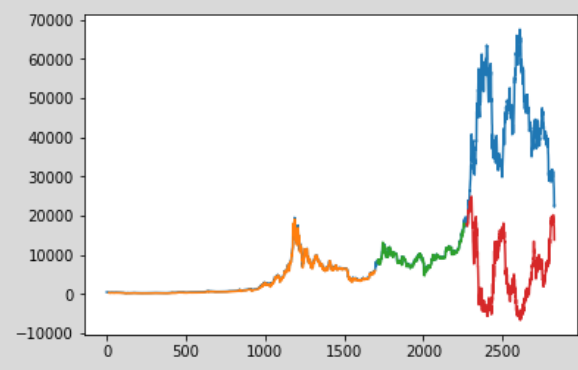Figure-60: BTC_17.png

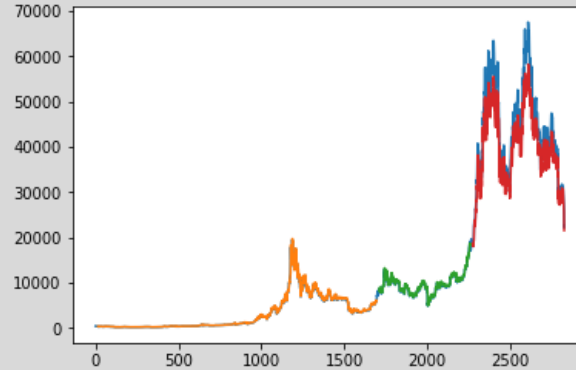Figure-61: BTC_18.png
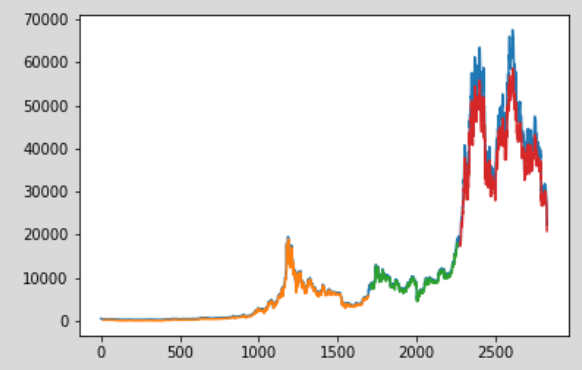

Figure-62: BTC_19.png

Figure-63: BTC_20.png

Figure-64: BTC_21.png

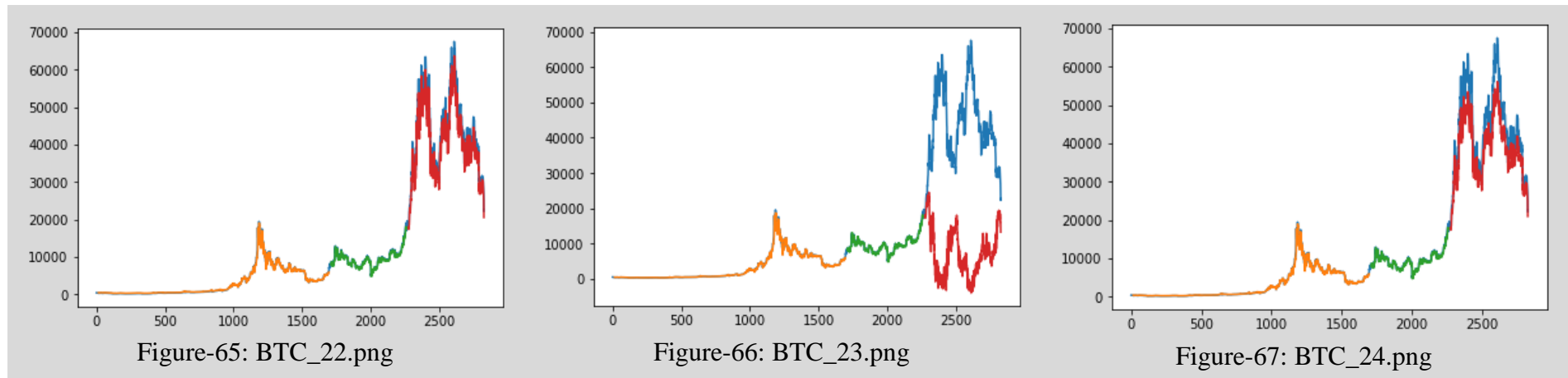Figure-65: BTC_22.png   Figure-66: BTC_23.png   Figure-67: BTC_24.png

Discussions: As crypto currencies are highly dependent on others factor (like world politics, socio-economic situation or even on some ones twitter), so it too difficult to predict the future price only with previous data. But if we look at the data, there may have some points.

1. For Ethereum data set, best accuracy gained when I use the parameter (time = 7 days, LSTM unit = 64, epochs = 100, batch size = 16).

2. Using different parameters (on Ethereum data set), the RMSE value for test data are too high. But the shape of prediction is pretty much similar with the actual value except (fig-22, fig-26, fig-30, fig-34, fig-38, and fig-42), where I used (epochs = 500, batch size = 16).

3. For Bitcoin the test RMSE value is huge 2392.9758, When applying the parameter (time = 14 days, LSTM unit = 128, epochs = 100, batch size = 64). But in maximum case the actual price and predicted price has almost same shape except (fig-51, fig-54, fig-58, fig-62, fig-66).

References:

- https://www.kaspersky.com/resource-center/definitions/what-is-cryptocurrency
- https://en.wikipedia.org/wiki/Long_short-term_memory
- https://en.wikipedia.org/wiki/Time_series
- https://machinelearningmastery.com/stacked-long-short-term-memory-networks/
- https://machinelearningmastery.com/difference-between-a-batch-and-an-epoch/
- https://finance.yahoo.com/
- Source code: https://github.com/AhmedDiderRahat/crypto_price_prediction/tree/main/code