Advance Data Science

Step by step guide for LSTM Model

Step 1 : Importing Required Libraries

```
[ ] import os
    import numpy as np
    import pandas as pd
    import tensorflow as tf
    from tensorflow import keras

    import seaborn as sns
    from pylab import rcParams
    import matplotlib.pyplot as plt
    from matplotlib import rc
    from sklearn.preprocessing import MinMaxScaler
    from tensorflow.keras.layers import Bidirectional, Dropout, Activation, Dense, LSTM
    from tensorflow.python.keras.layers import CuDNNLSTM
    from tensorflow.keras.models import Sequential

    %matplotlib inline

    sns.set(style='whitegrid', palette='muted', font_scale=1.5)

    rcParams['figure.figsize'] = 14, 8

    RANDOM_SEED = 42

    np.random.seed(RANDOM_SEED)
```

Fig1 : Importing required library

Os module in python is for creating and managing directory structure, fetching the content and identif y current directory.

NumPy is library in python for large multi dimensional array and matrix and for mathematical functio n as well.

Pandas is library in python for manipulating and analysis the data mainly for the numerical calculatio n.

Matplptlib is a plotting library in python for visualization.

Seaborn is a data visualization library based on matplotlib.

Scikit-learn is a ML librry specially for classification, regression and clustering.

Tensorflow is a ML library for training and inference of neural network.

Keras is python library for AI/ML act as inference for tensorflow.

Run the cell it will import all the above mention libraries .

Step 2: Load dataset

```
[ ]  from google.colab import files
     files.upload()
```

Fig 2: Upload the file

While running the above code section, it as for file to choose for upload . it will upload to the working area of your colab .

Step 3:Reading data as pandas dataframe

```
[ ]  df = pd.read_csv('coin_Bitcoin.csv', parse_dates=['Date'])
```

Fig 3: Reading dataframe as pandas dataframe

While running the above code it will read the dataframe.

Step4:Data cleaning

```
[ ]  df = df.sort_values('Date')
```

Fig 4: Date wise sorting the data

The above line sort the data in descending format.

Now sequentially execute following code.

```
[ ] coinbit.isnull().sum()# checking null values

SNo          0
Name         0
Symbol       0
Date         0
High         0
Low          0
Open         0
Close        0
Volume       0
Marketcap    0
dtype: int64
```

Fig5: Checking for null values in the data set

```
[ ] df.duplicated().sum()

0
```

Fig 6: Checking for Duplicate row

```
coinbit=coinbit.dropna()
coinbit=coinbit.drop(columns=['New_Price'])
```

Fig 7: Dropping the unused column

```
[ ] coinbit.isnull().sum()# checking null values

SNo          0
Name         0
Symbol       0
Date         0
High         0
Low          0
Open         0
Close        0
Volume       0
Marketcap    0
dtype: int64
```

Fig 8: Checking Null values

Step 5:EDA

Sequentially execute the following code ,the output and the work is respective to their code base.

```
df.head(10)
```

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
0	1	Bitcoin	BTC	2013-04-29 23:59:59	147.488007	134.000000	134.444000	144.539993	0.0	1.603769e+09
1	2	Bitcoin	BTC	2013-04-30 23:59:59	146.929993	134.050003	144.000000	139.000000	0.0	1.542813e+09
2	3	Bitcoin	BTC	2013-05-01 23:59:59	139.889999	107.720001	139.000000	116.989998	0.0	1.298955e+09
3	4	Bitcoin	BTC	2013-05-02 23:59:59	125.599998	92.281898	116.379997	105.209999	0.0	1.168517e+09
4	5	Bitcoin	BTC	2013-05-03 23:59:59	108.127998	79.099998	106.250000	97.750000	0.0	1.085995e+09
5	6	Bitcoin	BTC	2013-05-04 23:59:59	115.000000	92.500000	98.099998	112.500000	0.0	1.250317e+09
6	7	Bitcoin	BTC	2013-05-05 23:59:59	118.800003	107.142998	112.900002	115.910004	0.0	1.288693e+09
7	8	Bitcoin	BTC	2013-05-06 23:59:59	124.663002	106.639999	115.980003	112.300003	0.0	1.249023e+09
8	9	Bitcoin	BTC	2013-05-07 23:59:59	113.444000	97.699997	112.250000	111.500000	0.0	1.240594e+09
9	10	Bitcoin	BTC	2013-05-08 23:59:59	115.779999	109.599998	109.599998	113.566002	0.0	1.264049e+09

Fig 10: Reading 10 rows from head

```
df.tail(10)
```

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
2981	2982	Bitcoin	BTC	2021-06-27 23:59:59	34656.127356	32071.757148	32287.523211	34649.644588	3.551164e+10	6.494617e+11
2982	2983	Bitcoin	BTC	2021-06-28 23:59:59	35219.891791	33902.075892	34679.122222	34434.335314	3.389252e+10	6.454428e+11
2983	2984	Bitcoin	BTC	2021-06-29 23:59:59	36542.111018	34252.484892	34475.559697	35867.777735	3.790146e+10	6.723334e+11
2984	2985	Bitcoin	BTC	2021-06-30 23:59:59	36074.759757	34086.151878	35908.388054	35040.837249	3.405904e+10	6.568525e+11
2985	2986	Bitcoin	BTC	2021-07-01 23:59:59	35035.982712	32883.781226	35035.982712	33572.117653	3.783896e+10	6.293393e+11
2986	2987	Bitcoin	BTC	2021-07-02 23:59:59	33939.588699	32770.680780	33549.600177	33897.048590	3.872897e+10	6.354508e+11
2987	2988	Bitcoin	BTC	2021-07-03 23:59:59	34909.259899	33402.696536	33854.421362	34668.548402	2.438396e+10	6.499397e+11
2988	2989	Bitcoin	BTC	2021-07-04 23:59:59	35937.567147	34396.477458	34665.564866	35287.779766	2.492431e+10	6.615748e+11
2989	2990	Bitcoin	BTC	2021-07-05 23:59:59	35284.344430	33213.661034	35284.344430	33746.002456	2.672155e+10	6.326962e+11

Fig 11: Reading 10 rows from tail of data set

```
[ ] df.shape
```

```
(2991, 10)
```

Fig12: Counting rows and column of data set

```
[ ]  df.describe

<bound method NDFrame.describe of        SNo     Name Symbol                 Date          High           Low  \
0        1  Bitcoin    BTC 2013-04-29 23:59:59    147.488007    134.000000
1        2  Bitcoin    BTC 2013-04-30 23:59:59    146.929993    134.050003
2        3  Bitcoin    BTC 2013-05-01 23:59:59    139.889999    107.720001
3        4  Bitcoin    BTC 2013-05-02 23:59:59    125.599998     92.281898
4        5  Bitcoin    BTC 2013-05-03 23:59:59    108.127998     79.099998
...    ...      ...    ...                 ...           ...           ...
2986  2987  Bitcoin    BTC 2021-07-02 23:59:59  33939.588699  32770.680780
2987  2988  Bitcoin    BTC 2021-07-03 23:59:59  34909.259899  33402.696536
2988  2989  Bitcoin    BTC 2021-07-04 23:59:59  35937.567147  34396.477458
2989  2990  Bitcoin    BTC 2021-07-05 23:59:59  35284.344430  33213.661034
2990  2991  Bitcoin    BTC 2021-07-06 23:59:59  35038.536363  33599.916169

              Open         Close        Volume     Marketcap
0       134.444000    144.539993  0.000000e+00  1.603769e+09
1       144.000000    139.000000  0.000000e+00  1.542813e+09
2       139.000000    116.989998  0.000000e+00  1.298955e+09
3       116.379997    105.209999  0.000000e+00  1.168517e+09
4       106.250000     97.750000  0.000000e+00  1.085995e+09
...            ...           ...           ...           ...
2986  33549.600177  33897.048590  3.872897e+10  6.354508e+11
```

✓ 1s completed at 11:01 PM

Fig 13: Describing the data and its property

```
[ ]  df.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 2991 entries, 0 to 2990
Data columns (total 10 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   SNo        2991 non-null   int64
 1   Name       2991 non-null   object
 2   Symbol     2991 non-null   object
 3   Date       2991 non-null   datetime64[ns]
 4   High       2991 non-null   float64
 5   Low        2991 non-null   float64
 6   Open       2991 non-null   float64
 7   Close      2991 non-null   float64
 8   Volume     2991 non-null   float64
 9   Marketcap  2991 non-null   float64
dtypes: datetime64[ns](1), float64(6), int64(1), object(2)
memory usage: 257.0+ KB
```

Fig 14: looking into the info of data for data type and null values

```
[ ]  df.dtypes
```

```
SNo                   int64
Name                 object
Symbol               object
Date         datetime64[ns]
High                float64
Low                 float64
Open                float64
Close               float64
Volume              float64
Marketcap           float64
dtype: object
```

Fig 15: Looking into the datatype of dataset

```
[ ]  df.isna()
```

	SNo	Name	Symbol	Date	High	Low	Open	Close	Volume	Marketcap
0	False	False	False	False	False	False	False	False	False	False
1	False	False	False	False	False	False	False	False	False	False
2	False	False	False	False	False	False	False	False	False	False
3	False	False	False	False	False	False	False	False	False	False
4	False	False	False	False	False	False	False	False	False	False
...	...	...	...	...	...	...	...	...	...	...
2986	False	False	False	False	False	False	False	False	False	False
2987	False	False	False	False	False	False	False	False	False	False
2988	False	False	False	False	False	False	False	False	False	False
2989	False	False	False	False	False	False	False	False	False	False
2990	False	False	False	False	False	False	False	False	False	False

✓ 1s completed at 11:01 PM

Fig 16: Checking for NaN values

```
[ ] df.duplicated().sum()

0
```

Fig 17: Checking for duplicate values

```
df.value_counts()

SNo   Name     Symbol  Date                 High          Low           Open          Close         Volume        Marketcap
1     Bitcoin  BTC     2013-04-29 23:59:59  147.488007    134.000000    134.444000    144.539993    0.000000e+00  1.603769e+09    1
1998  Bitcoin  BTC     2018-10-17 23:59:59  6601.210000   6517.450000   6590.520000   6544.430000   4.088420e+09  1.133993e+11    1
1989  Bitcoin  BTC     2018-10-08 23:59:59  6675.060000   6576.040000   6600.190000   6652.230000   3.979460e+09  1.151629e+11    1
1990  Bitcoin  BTC     2018-10-09 23:59:59  6661.410000   6606.940000   6653.080000   6642.640000   3.580810e+09  1.150078e+11    1
1991  Bitcoin  BTC     2018-10-10 23:59:59  6640.290000   6538.960000   6640.290000   6585.530000   3.787650e+09  1.140308e+11    1
                                                                                                                               ..
1000  Bitcoin  BTC     2016-01-23 23:59:59  394.542999    381.980988    382.433990    387.490997    5.624740e+07  5.858060e+09    1
1001  Bitcoin  BTC     2016-01-24 23:59:59  405.484985    387.510010    388.101990    402.971008    5.482480e+07  6.093788e+09    1
1002  Bitcoin  BTC     2016-01-25 23:59:59  402.316986    388.553986    402.316986    391.726013    5.906240e+07  5.925345e+09    1
1003  Bitcoin  BTC     2016-01-26 23:59:59  397.765991    390.575012    392.002014    392.153015    5.814700e+07  5.933373e+09    1
2991  Bitcoin  BTC     2021-07-06 23:59:59  35038.536363  33599.916169  33723.509655  34235.193451  2.650126e+10  6.418992e+11    1
Length: 2991, dtype: int64
```

Fig18: Counting for the entire data

```
df.max()

SNo                          2991
Name                      Bitcoin
Symbol                        BTC
Date          2021-07-06 23:59:59
High              64863.098908
Low               62208.964366
Open              63523.754869
Close              63503.45793
Volume        350967941479.059998
Marketcap     1186364044140.27002
dtype: object
```

```
[ ] print("All Time High Price:",max(coinbit['Close']))
    print("Highest Number of Bitcoin units traded during the minute:",max(coinbit['Volume']))

All Time High Price: 63503.45793019
Highest Number of Bitcoin units traded during the minute: 350967941479.06
```

Fig 19: looking for maximum values for each row

Step 6: Visualization analysis

Sequentially execute the following cell ,the output and the work is respective to their code base. And visualize the output.

```
sns.pairplot(df)
```

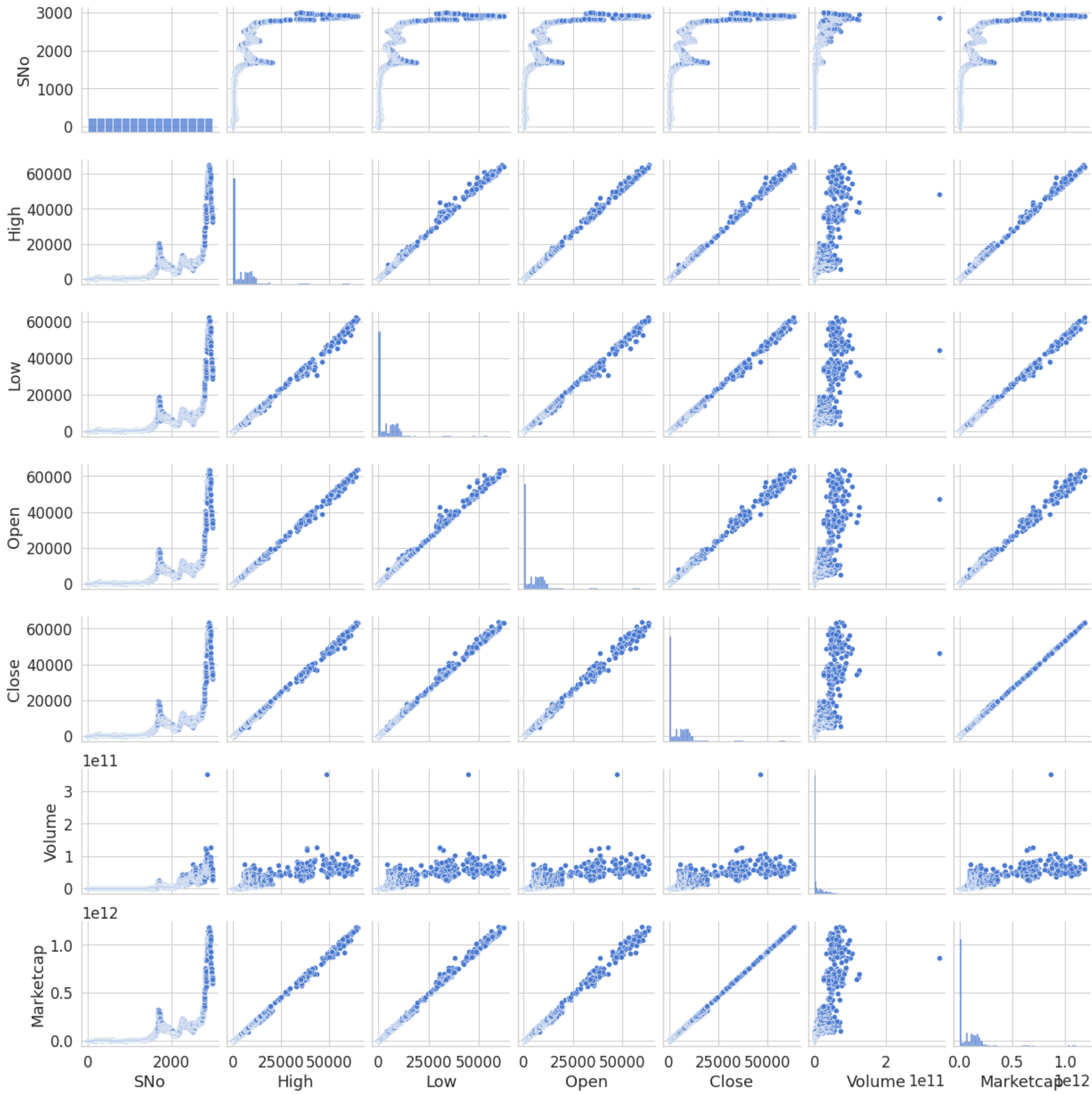

Fig 20 : Pair plot of each and every row

```
sns.boxplot(x=df["Close"])
```

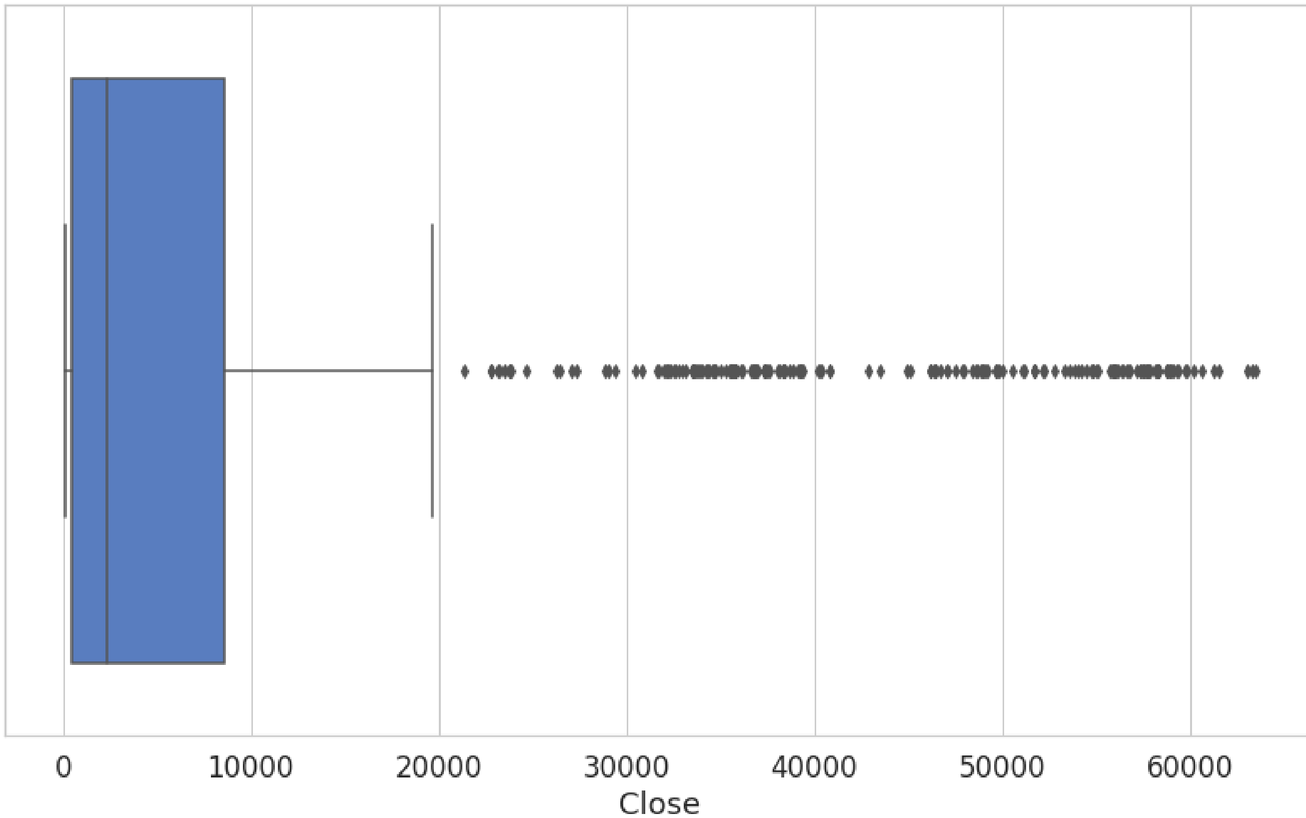

Fig 21: Boxplot for Close price

Draw a single horizontal boxplot, assigning the data directly to the coordinate variable:

```
[ ] boxplot = df.boxplot(column=['Low','High','Open','Close'])
```

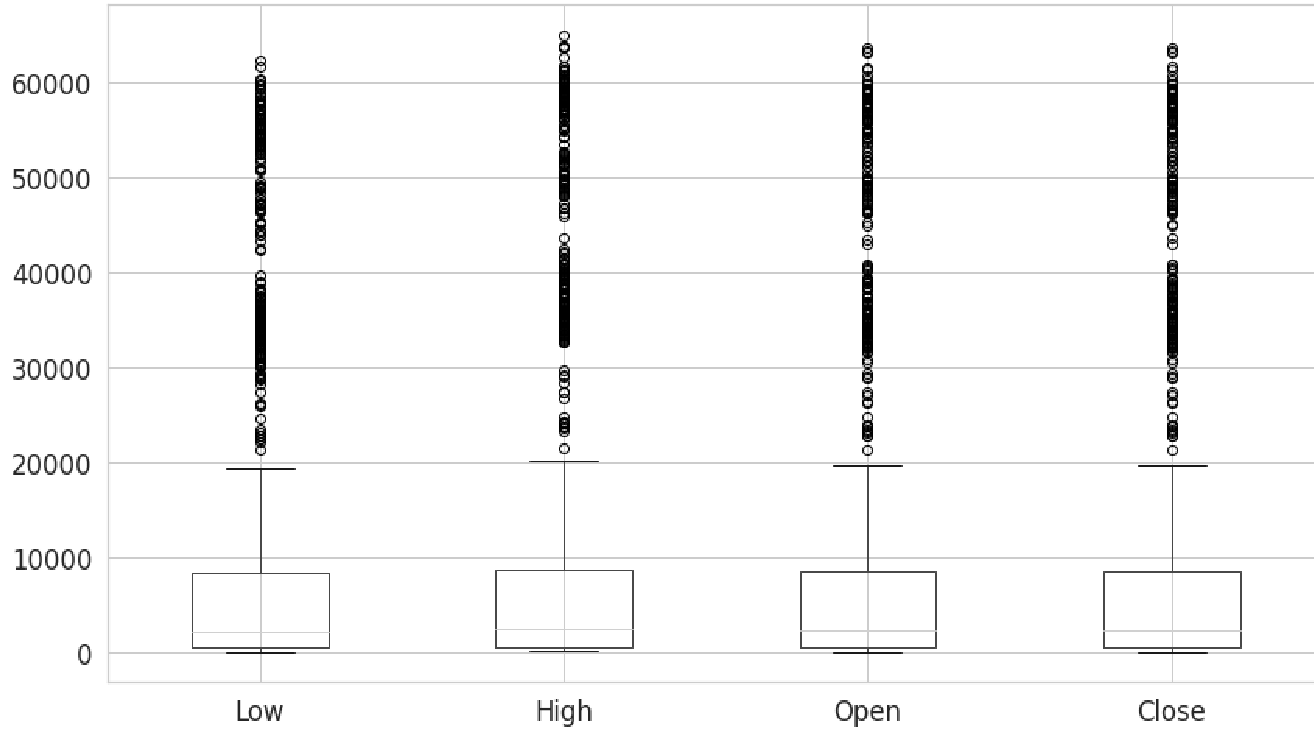

Fig 22: Boxplot for Low, High, Open and Close combined

Group by a categorical variable, referencing columns in a dataframe:

```
sns.boxplot(data=df[["Open", "Close" ,"High" ,"Low"]], orient="h")
```

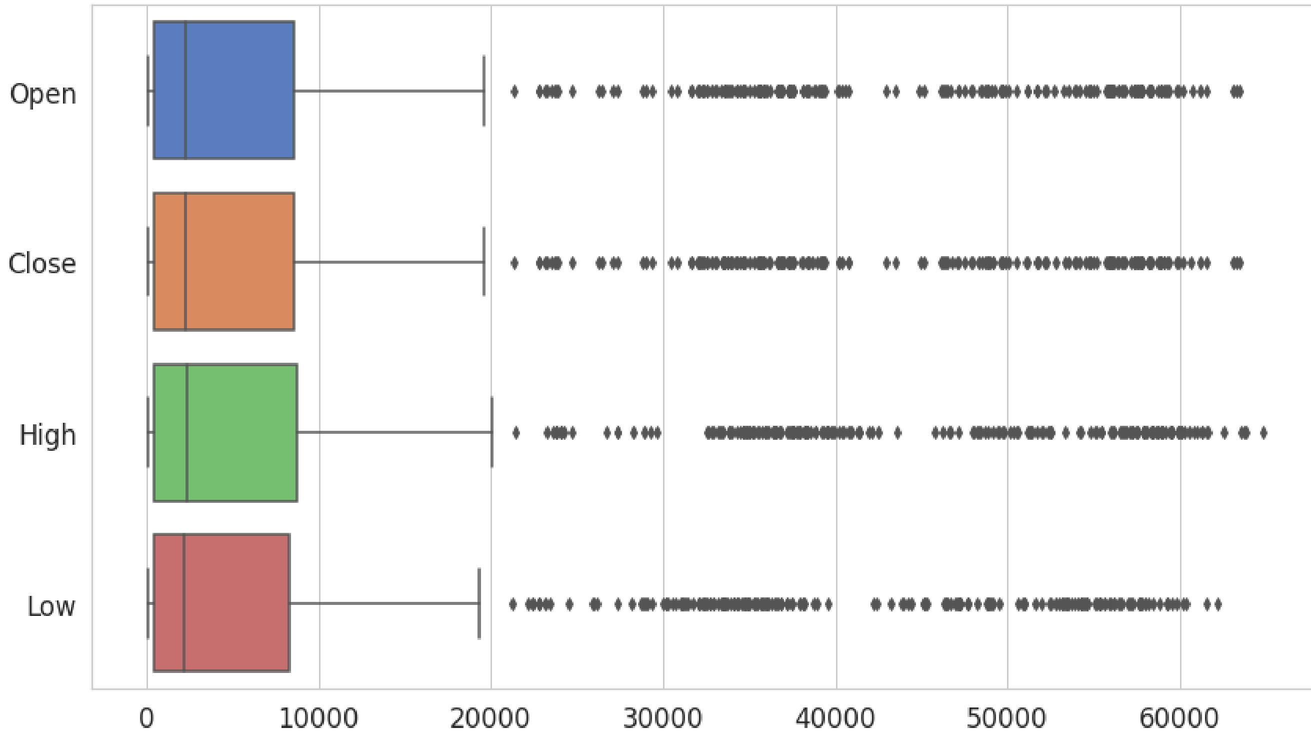

Fig 23: Boxplot for Low, High, Open and Close combined in horizontal orientation

Group by a categorical variable, referencing columns in a dataframe:

```
sns.boxplot(data=df[["Open", "Close" ,"High" ,"Low","Volume", "Marketcap"]], orient="h")
```

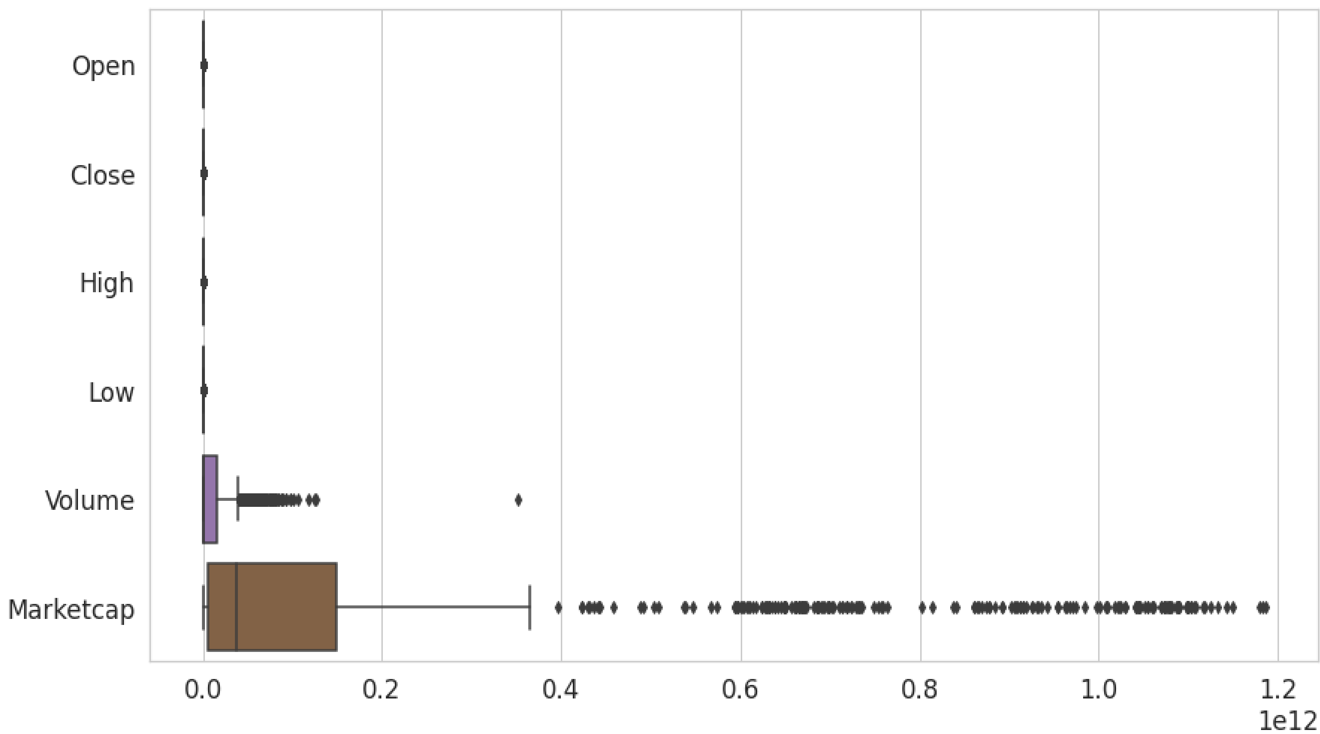

Fig 24: Boxplot for Low, High, Open , Close, Volume and Market Capitalization combined

Group by a categorical variable, referencing columns in a dataframe:

```
sns.boxplot(
    data=df, x="Close",
    notch=True, showcaps=False,
    flierprops={"marker": "x"},
    boxprops={"facecolor": (.4, .6, .8, .5)},
    medianprops={"color": "coral"},
)
```

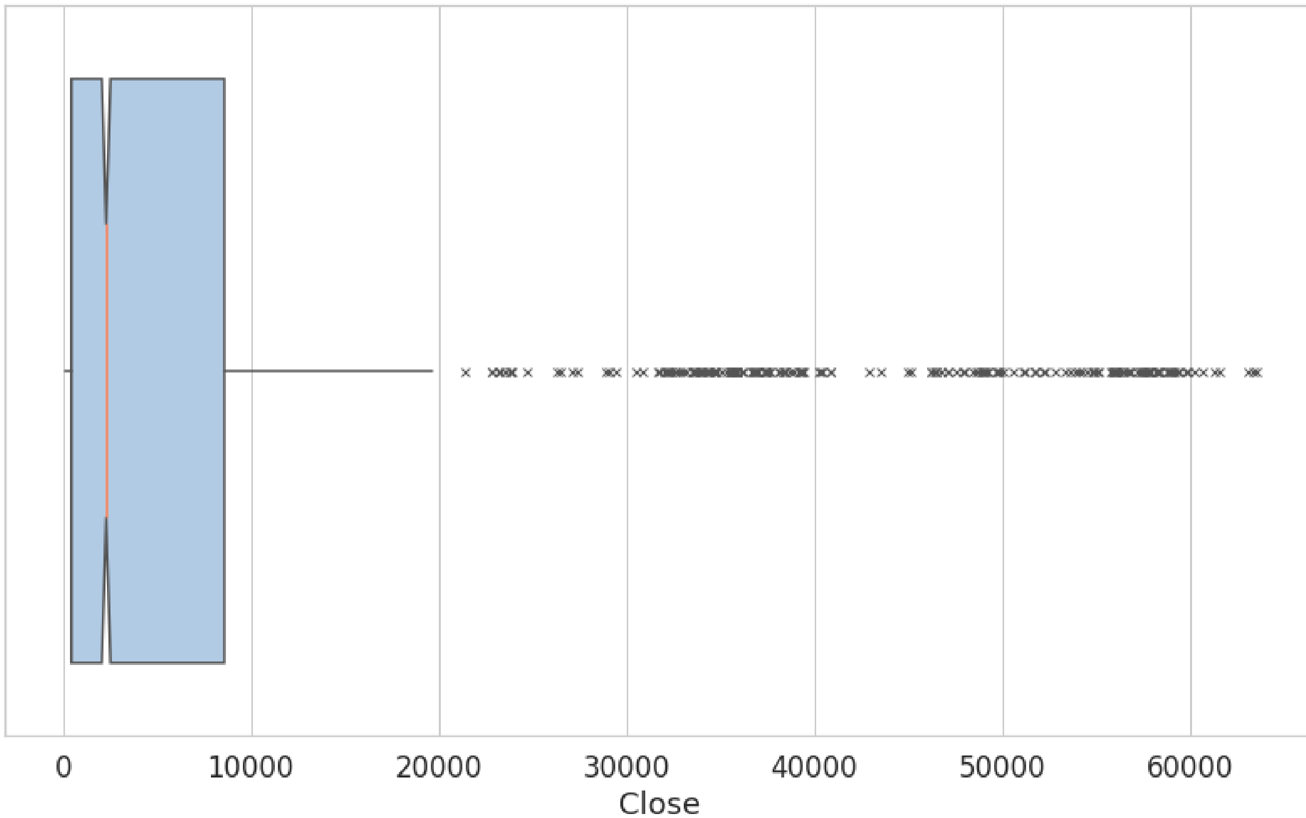

Fig 25: Boxplot Close value with additional information

Pass additional keyword arguments to matplotlib:

```
ax = df.plot(x='Date', y='Close');
ax.set_xlabel("Date")
ax.set_ylabel("Close Price (USD)")
```

```
Text(0, 0.5, 'Close Price (USD)')
```

Fig 26: Plotting Close value (As training and testing data)

Step 7: Normalization

Execute the following cell and analyze the output

```
[ ]  scaler = MinMaxScaler()

     close_price = df.Close.values.reshape(-1, 1)

     scaled_close = scaler.fit_transform(close_price)
```

Fig 27: Min Max scaler for reshaping the data

```
[ ]  scaled_close.shape

     (2991, 1)
```

```
[ ] np.isnan(scaled_close).any()

    False
```

Fig 28: scaling and additional filter for NaN values

```
[ ] scaled_close = scaled_close.reshape(-1, 1)
```

```
[ ] np.isnan(scaled_close).any()

    False
```

Fig 29: scaling and additional filter for NaN values

Step 8: Sequence building

Run the following cell and analyze the result

```
SEQ_LEN = 100

def to_sequences(data, seq_len):
    d = []

    for index in range(len(data) - seq_len):
        d.append(data[index: index + seq_len])

    return np.array(d)

def preprocess(data_raw, seq_len, train_split):

    data = to_sequences(data_raw, seq_len)

    num_train = int(train_split * data.shape[0])

    X_train = data[:num_train, :-1, :]
    y_train = data[:num_train, -1, :]

    X_test = data[num_train:, :-1, :]
    y_test = data[num_train:, -1, :]

    return X_train, y_train, X_test, y_test


X_train, y_train, X_test, y_test = preprocess(scaled_close, SEQ_LEN, train_split = 0.80)
```

```
[ ] X_train.shape # training data

    (2312, 99, 1)

    X_test.shape # testing data

    (579, 99, 1)
```

Fig 32: data pre-processing for training and testing

Now seperate our training data into our inputs and our outputs in time steps of time_steps. Where we will look at time_steps amount of data before we make our prediction of what the output for y will be.

Splitting our training data into training and validation.

Now we implement our actual model. We start with an LSTM input layer with 100 hidden units. We a dd a dropout of 0.2 before our Dense output layer with a linear activation and a shape of 1 (as we are out putting our expected price). We are using mean squared error to calculate our loss and adam as our opti mizer.

Step9:Model Building

```
[ ] from keras.layers import Input, LSTM, Dense, TimeDistributed, Activation, BatchNormalization, Dropout, Bidirectional
    from keras.models import Sequential
    from keras.utils import Sequence
    from keras.layers import CuDNNLSTM
    DROPOUT = 0.2
    WINDOW_SIZE = SEQ_LEN - 1

    model = keras.Sequential()

    model.add(Bidirectional(CuDNNLSTM(WINDOW_SIZE, return_sequences=True),
                            input_shape=(WINDOW_SIZE, X_train.shape[-1])))
    model.add(Dropout(rate=DROPOUT))

    model.add(Bidirectional(CuDNNLSTM((WINDOW_SIZE * 2), return_sequences=True)))
    model.add(Dropout(rate=DROPOUT))

    model.add(Bidirectional(CuDNNLSTM(WINDOW_SIZE, return_sequences=False)))

    model.add(Dense(units=1))

    model.add(Activation('linear'))
```

```
[ ] model.compile(
        loss='mean_squared_error',
        optimizer='adam'
    )
```

Fig 34: Model Building with necessary layers.

All models I have built so far do not allow for operating on sequence data. Fortunately, I have use a special class of Neural Network models known as **Recurrent Neural Networks (RNNs)** just for this purpose. *RNNs* allow using the output from the model as a new input for the same model. The process can be repeated indefinitely.

One serious limitation of *RNNs* is the inability of capturing longterm dependencies in a sequence One way to handle the situation is by using an **Long shortterm memory (LSTM)** variant of *RNN*.The default LSTM behavior is remembering information for prolonged periods of time. Let's see how to use LSTM in Keras.

Bidirectional RNNs allows you to train on the sequence data in forward and backward (reversed) direction. In practice, this approach works well with LSTMs.

CuDNNLSTM is a "Fast LSTM implementation backed by cuDNN". Personally, I think it is a good example of leaky abstraction, but it is crazy fast!

Our output layer has a single neuron (predicted Bitcoin price). We use Linear activation function which activation is proportional to the input.

Step 10: Training the Model

Run the below cell to train the model

```
[ ] import tensorflow as tf
    from tensorflow.keras.models import Sequential
    from tensorflow.keras.layers import Dense, Dropout, LSTM #, CuDNNLSTM
    BATCH_SIZE = 64

    history = model.fit(
        X_train,
        y_train,
        epochs=20,
        batch_size=BATCH_SIZE,
        shuffle=False,
        validation_split=0.1
    )
```

```
Epoch 1/20
33/33 [==============================] - 2s 46ms/step - loss: 3.5198e-04 - val_loss: 3.6522e-04
Epoch 2/20
33/33 [==============================] - 1s 41ms/step - loss: 3.9730e-04 - val_loss: 1.0688e-04
Epoch 3/20
33/33 [==============================] - 1s 41ms/step - loss: 1.8247e-04 - val_loss: 2.4138e-04
Epoch 4/20
33/33 [==============================] - 1s 42ms/step - loss: 1.6361e-04 - val_loss: 0.0015
Epoch 5/20
33/33 [==============================] - 1s 42ms/step - loss: 3.0279e-04 - val_loss: 1.8994e-04
Epoch 6/20
33/33 [==============================] - 1s 42ms/step - loss: 4.1041e-04 - val_loss: 2.0402e-04
Epoch 7/20
```

0s completed at 12:07 AM

Fig35: Training and model fitting

Fit our model now on our X and y training/validation data. We take advantage of keras' early stopping class so that once we are no longer recieving improvements the model will take its best weights and sto p.Now breaking our testing data up into time steps and splitting it again into our inputs and our expected out puts.Then predict on the inputs and then scale both the inputs and outputs back up, now were ready to see ho w we did!

Step11:Model evaluation

```
[ ] model.evaluate(X_test, y_test)
```

```
19/19 [==============================] - 0s 21ms/step - loss: 8.4954e-04
0.0008495371439494193
```

Fig 36: Evaluating model

Step 12: Plotting Train and test data

```
[ ] plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('model loss')
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train', 'test'], loc='upper left')
    plt.show()
```

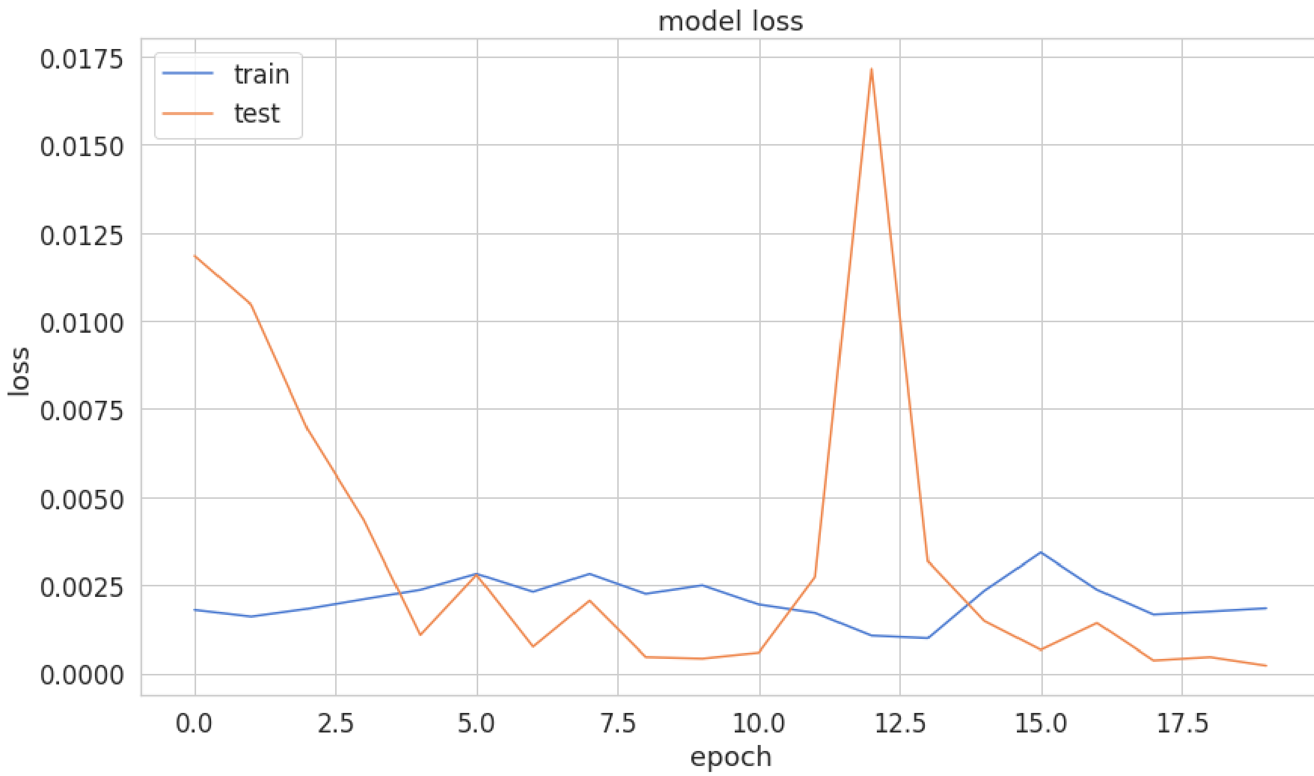

Fig 37: Plotting training and testing data

Step 13:Prediction and plotting the output

```
y_hat = model.predict(X_test)

y_test_inverse = scaler.inverse_transform(y_test)
y_hat_inverse = scaler.inverse_transform(y_hat)

plt.plot(y_test_inverse, label="Actual Price", color='green')
plt.plot(y_hat_inverse, label="Predicted Price", color='red')

plt.title('Bitcoin price prediction')
plt.xlabel('Time [days]')
plt.ylabel('Price')
plt.legend(loc='best')

plt.show();
```

Fig 38: Prediction for LSTM Model

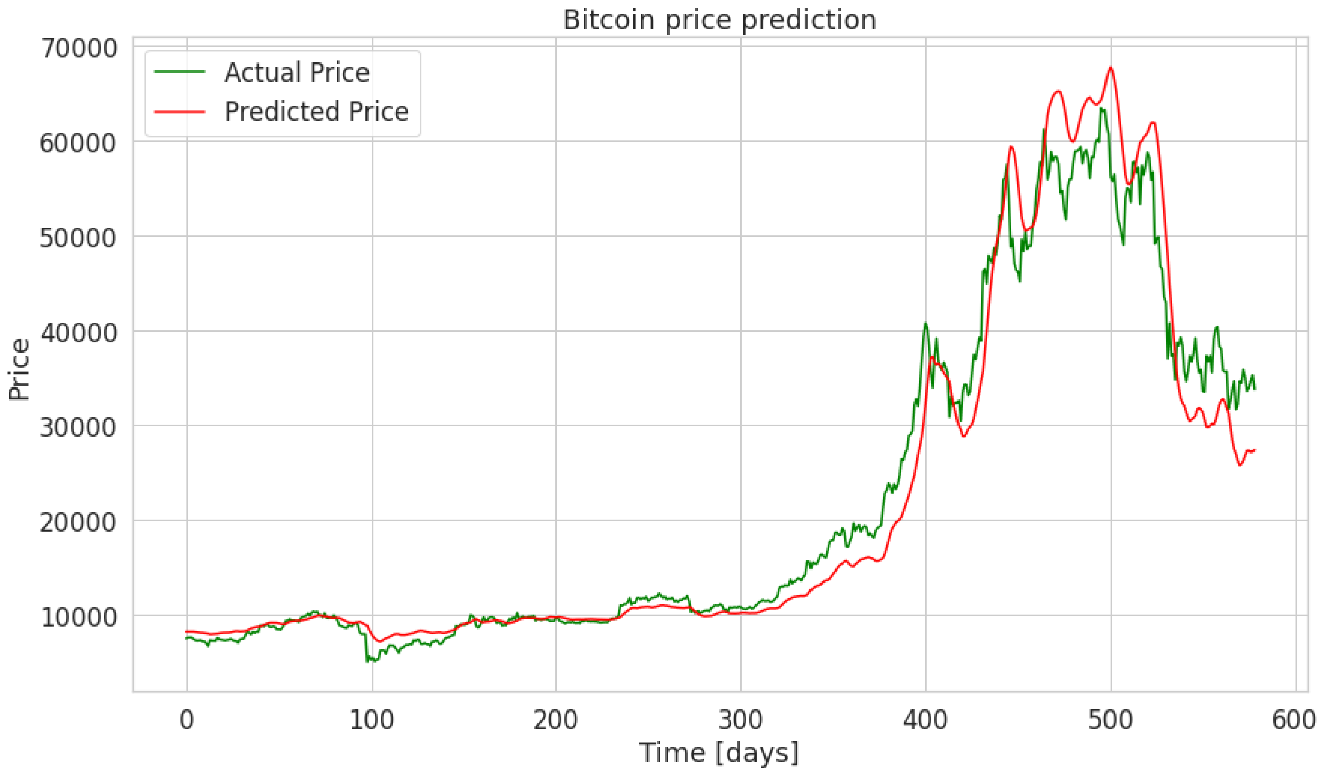

Fig 39: Predicted price visualization along with close price for LSTM model