

Bitcoin Price Prediction using LSTM Model

Kasjan Śmigielski

February 4, 2025

Contents

1	Introduction	2
2	EDA	2
2.1	Data Collection	2
2.2	Sample values	2
2.3	Unique values	3
2.4	Descriptive statistics	3
2.5	Missing values	3
2.6	Histograms	4
2.7	Correlations	4
2.8	Outliers	5
2.9	Summary of EDA	6
3	Modeling	7
3.1	Data Processing	7
3.2	Data Splitting	7
3.3	LSTM Architecture	9
3.4	Model Training	9
3.5	Training results	10
4	Evaluation and Results	12
4.1	Error Metrics	12
4.2	Results Visualization	12
5	Conclusion and Improvements	13

List of Tables

1	Five sample values from Bitcoin dataset.	3
2	Unique values from Bitcoin dataset.	3
3	Data before normalization.	7
4	Data after normalization.	7

List of Figures

1	Histograms for columns: 'Open', 'High', 'Low', 'Volume'. . . .	4
2	Correlation matrix between Bitcoin prices.	5
3	share of outliers shown in sample boxplots.	6
4	LSTM model training results.	10
5	Real vs prediction closing price.	12

Abstract

The aim of this project was to create a predictive model that forecasts Bitcoin prices using an LSTM neural network. This document outlines the process of data collection, preparation, model building, and result analysis.

1 Introduction

This project focuses on predicting future Bitcoin prices using an advanced predictive model based on the Long Short-Term Memory (LSTM) neural network. Bitcoin is one of the most popular cryptocurrencies, known for its high price volatility.

2 EDA

2.1 Data Collection

Historical data on Bitcoin prices was collected from the *Kaggle* platform. The dataset includes daily price information from 2014-09-17 to 2022-03-25.

2.2 Sample values

The data was transformed into `DataFrame` using `pandas` library. The table below shows 5 examples of records from the loaded dataset:

Date	Open	High	Low	Close	Adj Close	Volume
2015-09-20	231.4	232.37	230.91	231.21	231.21	14444700
2020-11-12	15701.3	16305.0	15534.77	16276.34	16276.34	34175758344
2018-11-18	5559.74	5653.61	5559.74	5623.54	5623.54	4159680000
2015-05-27	237.07	238.64	236.7	237.28	237.28	18837000
2016-03-26	417.36	418.99	416.26	417.95	417.95	44650400

Table 1: Five sample values from Bitcoin dataset.

2.3 Unique values

There are **2747 records in the loaded dataset**, and almost all records are unique as shown in the table below:

Item	Number of unique values	Percentage
Date	2747	100.00
Volume	2747	100.00
Low	2746	99.96
Close	2744	99.89
High	2744	99.89
Adj Close	2744	99.89
Open	2743	99.85

Table 2: Unique values from Bitcoin dataset.

2.4 Descriptive statistics

Main information read from basic descriptive statistics (all data included from one day):

- Min. bitcoin price = 171.51 USD.
- Max. bitcoin price = 68789.62 USD.
- Average opening price of bitcoin = 11668.6 USD.
- Average closing price of bitcoin = 11682.89 USD.
- Difference between the average closing and opening price = 14.29 USD.

2.5 Missing values

In the loaded Bitcoin dataset - **there were no missing values**.

2.6 Histograms

Below are example of histograms showing that most Bitcoin prices were well below 20000 USD:

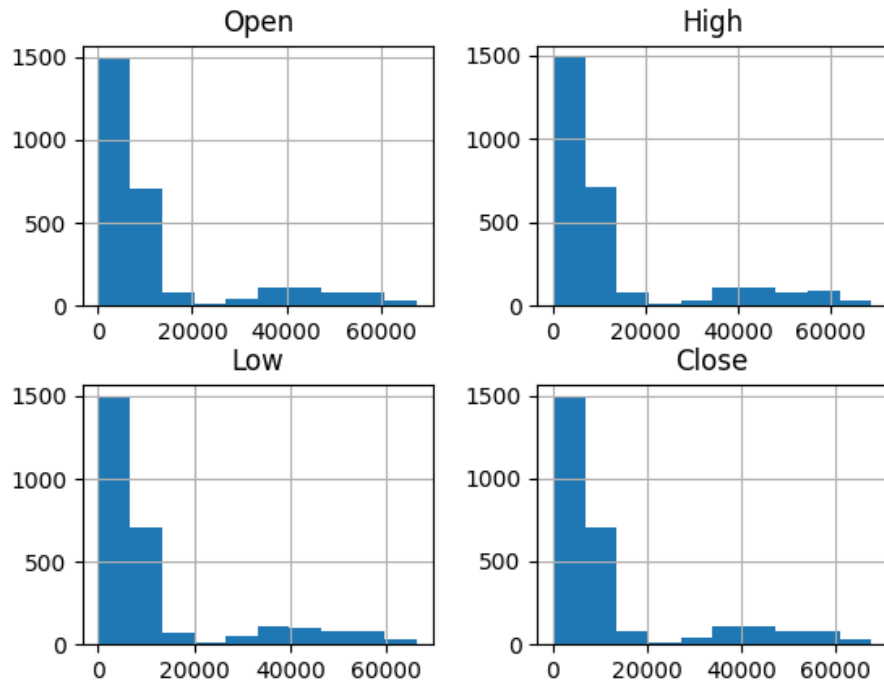


Figure 1: Histograms for columns: 'Open', 'High', 'Low', 'Volume'.

2.7 Correlations

The correlation matrix below confirms the fact that each bitcoin price is closely dependent on each other (maximum positive correlation):

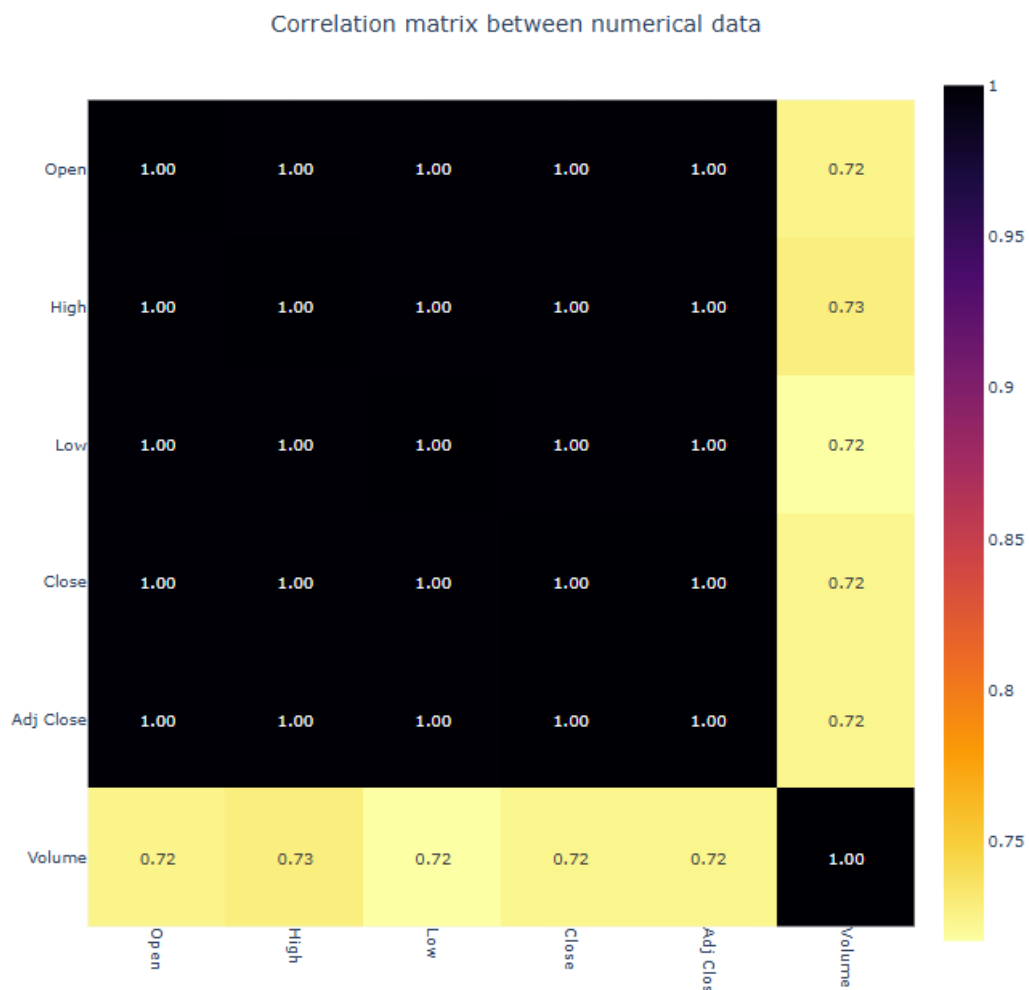


Figure 2: Correlation matrix between Bitcoin prices.

2.8 Outliers

In the sample boxplots below you can see a huge number of outliers in the data - this is caused by the specificity of the financial market:

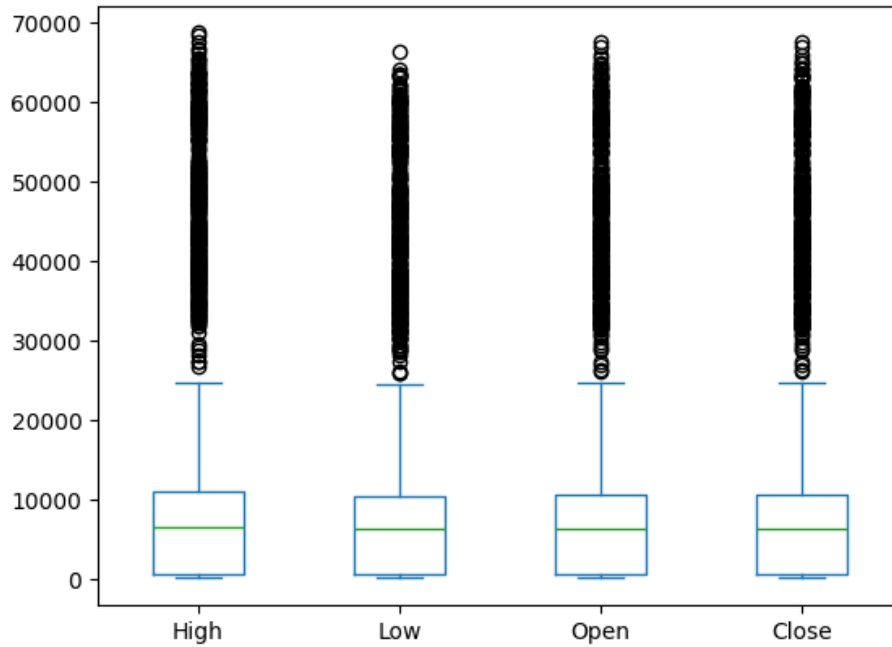


Figure 3: share of outliers shown in sample boxplots.

2.9 Summary of EDA

EDA preparation was aimed at getting to know the cryptocurrency domain better so that we could then better train predictive models.

Main conclusions:

- No missing values.
- Very large number of outliers.

3 Modeling

3.1 Data Processing

The data was processed to remove missing values and normalized using the `MinMaxScaler` from the `scikit-learn` library. In the tables below which are used before and after normalization:

Date	Open	High	Low	Close	Adj Close	Volume
2014-09-17	465.864014	468.174011	452.421997	457.334015	457.334015	21056800
2014-09-18	456.859985	456.859985	413.104004	424.440002	424.440002	34483200
2014-09-19	424.102997	427.834991	384.532013	394.795990	394.795990	37919700
2014-09-20	394.673004	423.295990	389.882996	408.903992	408.903992	36863600
2014-09-21	408.084991	412.425995	393.181000	398.821014	398.821014	26580100

Table 3: Data before normalization.

Date	Open	High	Low	Close	Adj Close	Volume
2014-09-17	0.004289	0.003739	0.004243	0.004144	0.004144	0.000043
2014-09-18	0.004155	0.003574	0.003649	0.003655	0.003655	0.000081
2014-09-19	0.003669	0.003151	0.003217	0.003216	0.003216	0.000091
2014-09-20	0.003232	0.003085	0.003298	0.003425	0.003425	0.000088
2014-09-21	0.003431	0.002927	0.003348	0.003275	0.003275	0.000059

Table 4: Data after normalization.

When using an LSTM that is sensitive to the scale of the data, normalization helps in stable model training and ensures that the data is in the appropriate range for the activation functions used in the neural networks.

3.2 Data Splitting

In the next step, the data was divided into training, validation and test sets to enable a comprehensive evaluation of the model and **target column *Volume* was set.**

At the very beginning, a sequence was prepared in the form of a time window **a size of 60 time units** (in this case days):

```

def create_sequences(data, window_size):
    x, y = [], []

    for i in range(len(data) - window_size):
        x.append(data[i:(i + window_size)])
        y.append(data[i + window_size, 3])

    return np.array(x), np.array(y)

window_size = 60

all_features = df[['Open', 'High', 'Low', 'Close', 'Volume']].values

x_sequences, y_sequences = create_sequences(all_features,
                                             window_size)

```

This time window size was used due to the identification of long-term trends.

Then the data was divided into three sets in the following proportions:

- Train set = 70%
- Validate set = 15%
- Test set = 15%

and it was implemented in the functionality:

```

train_size = int(len(x_sequences) * 0.7)
val_size = int(len(x_sequences) * 0.15)

x_train, x_val, x_test = x_sequences[:train_size],
    x_sequences[train_size:train_size + val_size],
    x_sequences[train_size + val_size:]
y_train, y_val, y_test = y_sequences[:train_size],
    y_sequences[train_size:train_size + val_size],
    y_sequences[train_size + val_size:]

```

This division into proportions was used due to the balance between model training and its evaluation.

3.3 LSTM Architecture

The LSTM model was built using the TensorFlow/Keras library. It includes *number* of LSTM layers and additional Dense layers.

The use of the LSTM model is presented in the implementation below:

```
input_shape = (x_train.shape[1], x_train.shape[2])

model = Sequential()
model.add(LSTM(units=50, return_sequences=True,
               input_shape=input_shape))
model.add(LSTM(units=50))
model.add(Dense(1))
```

LSTM was used because it is a type of recurrent network layer (RNN) that is particularly useful for processing data sequences and finding long-term dependencies. The Dense layers, at the end of the LSTM chain, allow the transformation of features obtained from the sequence into the final prediction (e.g. the next closing price).

3.4 Model Training

Before training the model, the optimizer and loss function were selected:

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

Justification for choice of optimizer and loss:

- Optimizer Adam (Adaptive Moment Estimation) - due to its versatility and good performance on a wide range of problems, including time series forecasting such as cryptocurrency prices.
- Loss Function MSE (Mean Squared Error) - due to sensitivity to outliers. MSE imposes larger penalties for larger errors, which is beneficial in situations where large deviations are undesirable, as is typical in price prediction tasks.

Then the model was trained, as summarized in the implementation below:

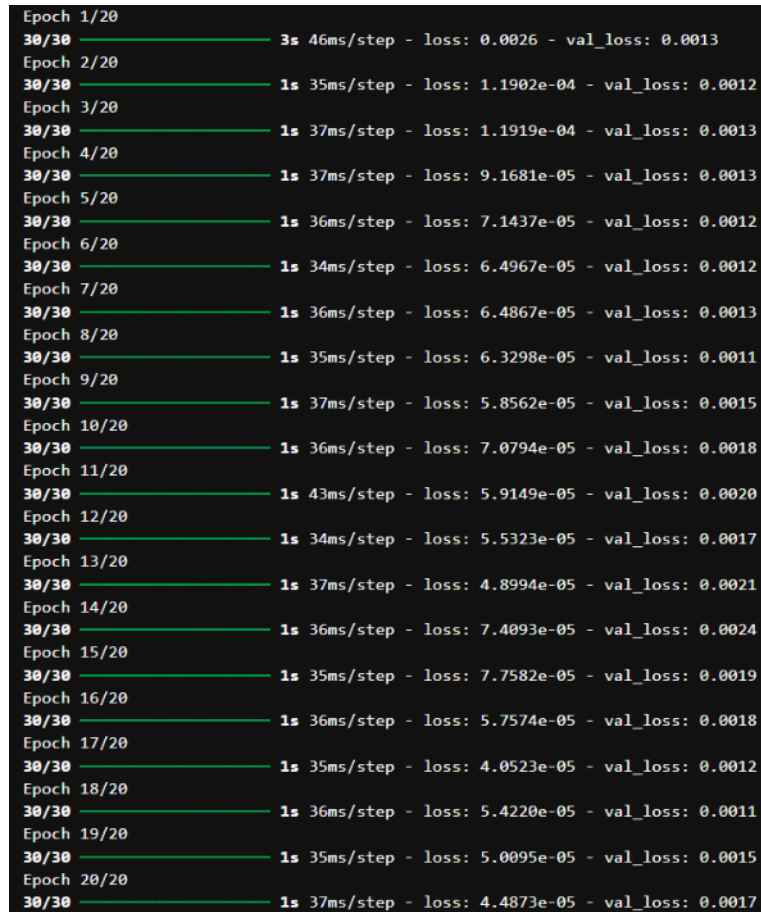
```
history = model.fit(x_train, y_train, epochs=20,  
                    batch_size=64, validation_data=(x_val, y_val))
```

Justification for choice of batch size and epochs:

- **batch size** - a batch size of 64 is a popular choice because it makes good use of hardware resources such as GPU memory.
- **epochs** - 20 epochs is often enough for the model to learn patterns without the risk of overfitting.

3.5 Training results

The training results are presented below:



Epoch	30/30	Time	loss	val_loss
Epoch 1/20	3s	46ms/step	0.0026	0.0013
Epoch 2/20	1s	35ms/step	1.1902e-04	0.0012
Epoch 3/20	1s	37ms/step	1.1919e-04	0.0013
Epoch 4/20	1s	37ms/step	9.1681e-05	0.0013
Epoch 5/20	1s	36ms/step	7.1437e-05	0.0012
Epoch 6/20	1s	34ms/step	6.4967e-05	0.0012
Epoch 7/20	1s	36ms/step	6.4867e-05	0.0013
Epoch 8/20	1s	35ms/step	6.3298e-05	0.0011
Epoch 9/20	1s	37ms/step	5.8562e-05	0.0015
Epoch 10/20	1s	36ms/step	7.0794e-05	0.0018
Epoch 11/20	1s	43ms/step	5.9149e-05	0.0020
Epoch 12/20	1s	34ms/step	5.5323e-05	0.0017
Epoch 13/20	1s	37ms/step	4.8994e-05	0.0021
Epoch 14/20	1s	36ms/step	7.4093e-05	0.0024
Epoch 15/20	1s	35ms/step	7.7582e-05	0.0019
Epoch 16/20	1s	36ms/step	5.7574e-05	0.0018
Epoch 17/20	1s	35ms/step	4.0523e-05	0.0012
Epoch 18/20	1s	36ms/step	5.4220e-05	0.0011
Epoch 19/20	1s	35ms/step	5.0095e-05	0.0015
Epoch 20/20	1s	37ms/step	4.4873e-05	0.0017

Figure 4: LSTM model training results.

Conclusions and Recommendations:

1. Initial Improvement:

During the initial epochs, both `loss` and `val_loss` decrease significantly, indicating that the model is effectively learning the patterns embedded in the data.

2. Stability in Middle Epochs:

- The `loss` values continue to decline consistently, which is promising.
- `val_loss` fluctuates slightly but remains within a stable range (0.0011 - 0.0018), suggesting that the model is either stabilizing or nearing the point of loss minimization.

3. Potential Overfitting Towards the End of Training:

- In the final epochs, `val_loss` exhibits slightly more variability, with an increase during the latter stages (from epoch 10 to 14). Ultimately, in epoch 20, `val_loss` rises to 0.0017, while `loss` decreases to 4.4873e-05.
- This may indicate some degree of overfitting, where the model learns the training data too well but does not generalize as effectively to the validation data.

To mitigate potential overfitting, consider techniques such as early stopping or implementing regularization methods like dropout to enhance the model's generalization capabilities.

4 Evaluation and Results

Below are the final results and a visualized Bitcoin closing price prediction.

4.1 Error Metrics

Two error metrics were selected to assess the model's performance: **Mean Squared Error (MSE)** and **Mean Absolute Error (MAE)**. The results are given below:

- **MSE:** 0.027970717990139192
- **MAE:** 0.15622346289979314

4.2 Results Visualization

Results were visualized with a plot showing actual and predicted Bitcoin prices. Figure 5 presents the comparison of both values.

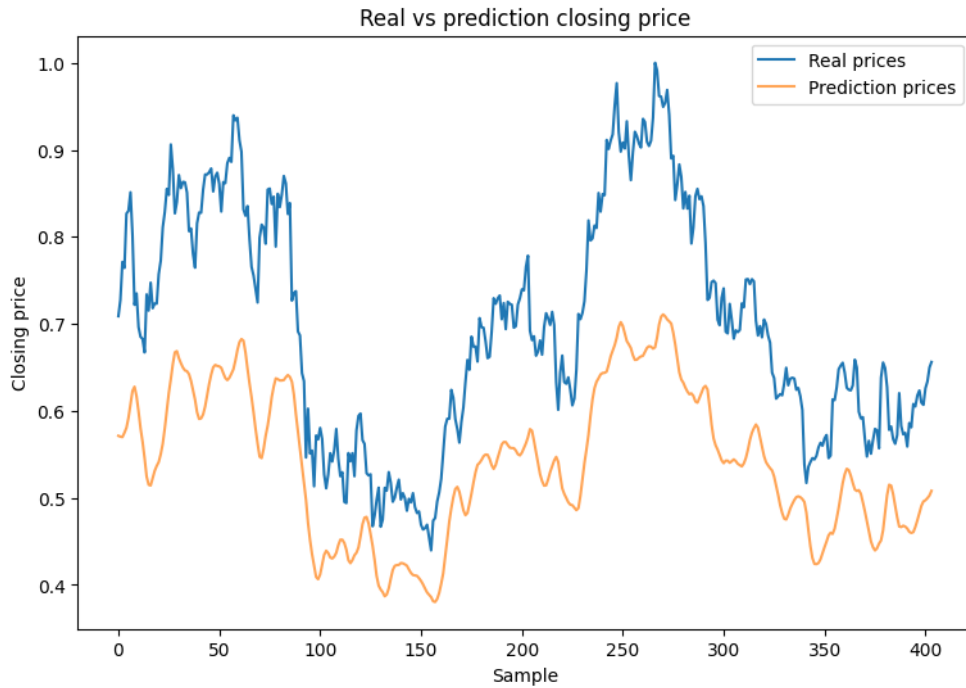


Figure 5: Real vs prediction closing price.

5 Conclusion and Improvements

Based on the analysis of the LSTM model's training results for predicting Bitcoin prices, several key conclusions and recommendations can be drawn:

1. Model Effectiveness:

The LSTM model has shown its capability in predicting Bitcoin prices with an acceptable level of accuracy, as indicated by the *overall model quality*. The Mean Squared Error (MSE) of 0.0163 and Mean Absolute Error (MAE) of 0.1159 suggest that the model captures the data patterns reasonably well.

2. Early Learning Success:

During the early epochs, the model exhibited significant improvement, with both the training loss and validation loss decreasing sharply. This indicates the model's ability to learn and adapt to the underlying patterns within the training data efficiently.

3. Mid-Epoch Stability:

As training progressed, the model demonstrated stability. The consistent decline in training loss and the minor fluctuations in validation loss (ranging between 0.0011 and 0.0018) suggest that the model was either stabilizing or reaching a minimal loss plateau.

4. Potential Overfitting:

Towards the end of the training process, an increase in the variability of the validation loss was noted. The rise in validation loss during later epochs suggests possible overfitting, where the model starts to memorize the training data rather than generalize from it.

5. Recommendations for Improvement:

To enhance the model's performance and address overfitting, several strategies are recommended:

- **Increase Dataset Size:** Expanding the dataset can provide more information for the model to learn intricate patterns.
- **Hyperparameter Tuning:** Fine-tuning the number of LSTM units, the sequence length, epochs, and batch sizes can optimize performance.

- **Model Architecture Adjustment:** Modifying the number and arrangement of layers can help capture the data complexity more effectively.
- **Cross-Validation:** Implementing cross-validation techniques will better assess model stability across different datasets splits.
- **Feature Engineering:** Introducing additional relevant features could enrich the model with more predictive power, provided these features contribute to Bitcoin price movements.

Implementing these strategies may lead to improvements in model accuracy and generalization in future iterations.