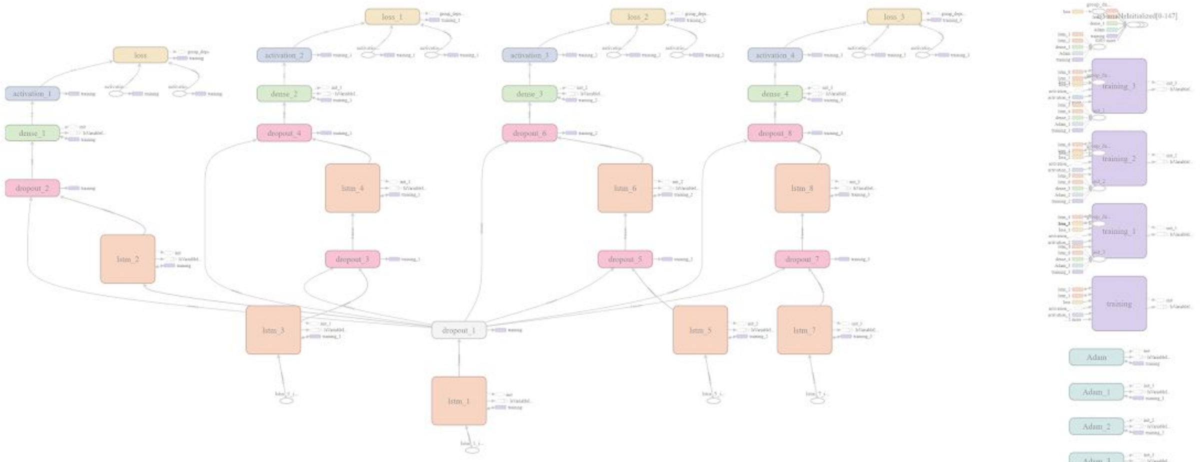


# DEEP LEARNING
# BITCOIN PRICE PREDICTOR

by Nadezhda Kozlova

# CONTENT

**DEFINITION** **2**
- Project Overview 2
- Problem Statement 2
- Metrics 3

**ANALYSIS** **3**
- Data Exploration 3
- Exploratory Visualization 5
- Algorithms and Techniques 7
- Benchmark 9

**METHODOLOGY** **11**
- Data Preprocessing 11
- Implementation 12
- Refinement 12

**RESULTS** **14**
- Model Evaluation and Validation 14
- Justification 15

**CONCLUSION** **15**
- Free-Form Visualization 15
- Reflection 17
- Improvement 18

**REFERENCES** **19**

## DEFINITION

### Project Overview

Cryptocurrencies are making waves in almost every industry and have been growing exponentially over the last few years. Bitcoin itself is one of the most popular and widely known cryptocurrency. It is mainly designed to remove the need of any third-party entities or financial institutions and thereby eliminate the possibility of fraud during the transactions.

For nowadays, cryptocurrencies have gathered significant investor attention. Even stock market using econometric tools to predict prices. In this project, with the main features such as the opening price, highest price, lowest price, closing price, and volume of currency I will try to accurately predict the next day closing price of Bitcoin in a given time period. Fortunately, there is complete historical data of Bitcoin price changes and time-series algorithms should be capable to use it for predictions. I will experiment with different machine learning algorithms with various architectures, particularly to Long Short-Term Memory models.

### Problem Statement

The challengable task of the project is to predict the next day Bitcoin closing price. Before starting to decide this problem I investigated similar issues in stock market, and technics used to solve it. My hypothetical solution was to use Long Short-Term Memory model. The key points my research is trying to answer does deep learning algorithms are capable to make working predictions on free-access data, and if simple recurrent neural network model overperforms benchmark.

First of all, I'm going to shift data on one point. My models will get the 0-th observation from x_train and try to predict the 0-th element from y_train (which is the first target observation of x_train). This way I'm creating a sequence - exactly what RNNs need for training. LSTM models will be trained on full data (+generated features), on plain data (which can be easily got from CoinMarketCap) and on simple data (just today's bitcoin price to predict tomorrow's)

## Metrics

I have used Root Mean Square Error (RMSE) as an evaluation metric for this project. Mean squared error is the arithmetic mean of the squares of an error function computed between estimated and actual (target) values, where the estimates here predictions. It's the square root of the average of squared differences between prediction and actual observation. RMSE is a pretty standard metric for measuring the error in regression issues with normalized data.

$$RMS = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(d_i - p_i)^2}$$

'n' is the number of examples
'p' represents vector of n predictions
'd'  represents vector of ground truth

I will fit the LSTMs model with normalized data, but evaluate RMSE on inverted data - because it's important exactly how much is the difference in USD

# ANALYSIS

## Data Exploration

Data, used for this project is being loaded from  CoinMarketCap (the most famous resource of cryptocurrencies information) It consists of (USD price, from 28.04.2013 to current date (fixed to 09.12.2018):

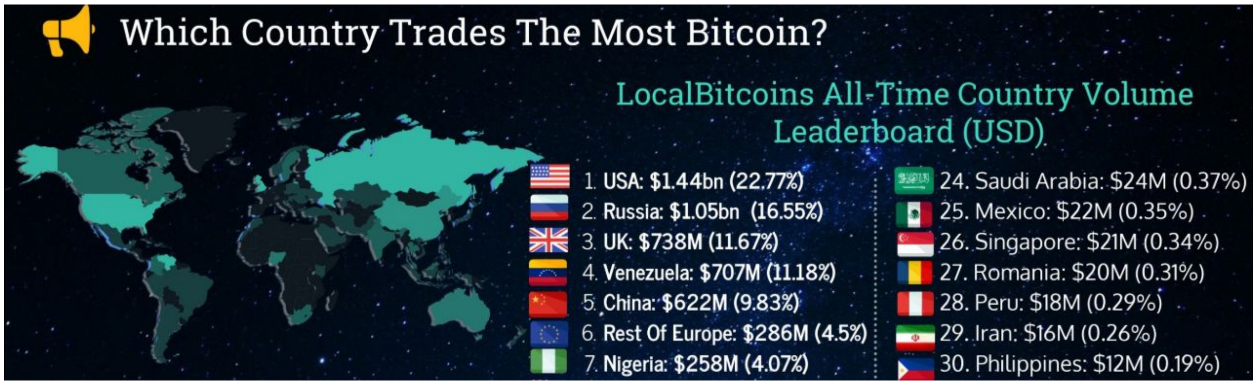
- Open: Earliest data in range (UTC time)
- High: hightes value during the day

- Low: lowest value during the day
- Close: Latest data in range (UTC time)
- Volume: circulating bitcoin volume on the market
- Market Cap: total value of cryptocurrencies on the market

There are a lot of bitcoin’s features not as cryptocurrency but as a blockchain system features that could be useful. From QUANDL.com I downloaded:

- Number of confirmed transactions per day
- Difficulty: A relative measure of how difficult it is to find a new block. The difficulty is adjusted periodically as a function of how much hashing power has been deployed by the network of miners
- Average block size (in MB)
- Miners Revenue: Historical data showing (number of bitcoins mined per day + transaction fees) * market price
- Bitcoin Cost Per Transaction

For me, feature engineering is a very interesting subject to start with. It is a pretty interesting way to improve the quality of the dataset. So I made weekdays and weekend feature and also holiday feature, in countries, where Bitcoin is most tradable (I took the first 5 countries, which covers more than 70% of world trade):


Which Country Trades The Most Bitcoin?
LocalBitcoins All-Time Country Volume Leaderboard (USD)

| | |
|---|---|
| 1. USA: $1.44bn (22.77%) | 24. Saudi Arabia: $24M (0.37%) |
| 2. Russia: $1.05bn (16.55%) | 25. Mexico: $22M (0.35%) |
| 3. UK: $738M (11.67%) | 26. Singapore: $21M (0.34%) |
| 4. Venezuela: $707M (11.18%) | 27. Romania: $20M (0.31%) |
| 5. China: $622M (9.83%) | 28. Peru: $18M (0.29%) |
| 6. Rest Of Europe: $286M (4.5%) | 29. Iran: $16M (0.26%) |
| 7. Nigeria: $258M (4.07%) | 30. Philippines: $12M (0.19%) |

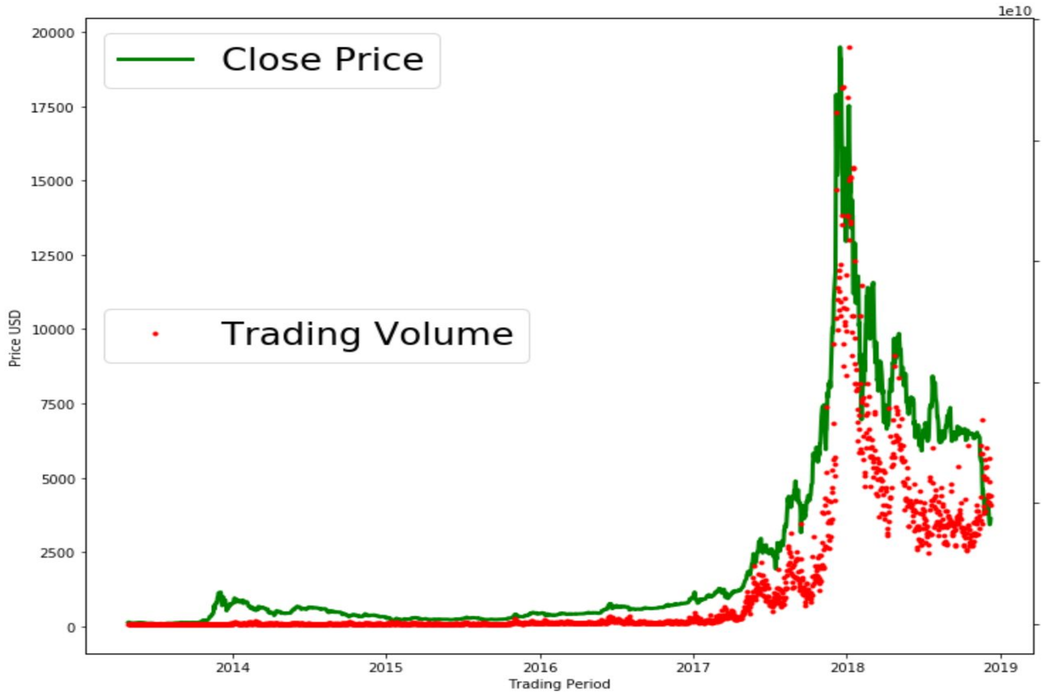
Also, I checked, which features is usually important in the stock market and generate a few, such as:

- Moving Average for 7 days: (rolling average or running average) is a calculation to analyze data points by creating a series of averages of different subsets of the full data set
- Stochastic Oscillator: The indicator picks one observation point in the current base and refers to all points in the defined range from where the highest and lowest points are considered for comparison

- Momentum (for 2, 7, 14 days): Pricing anomaly is term as “time series momentum,” which is remarkably consistent across very different asset classes and markets.

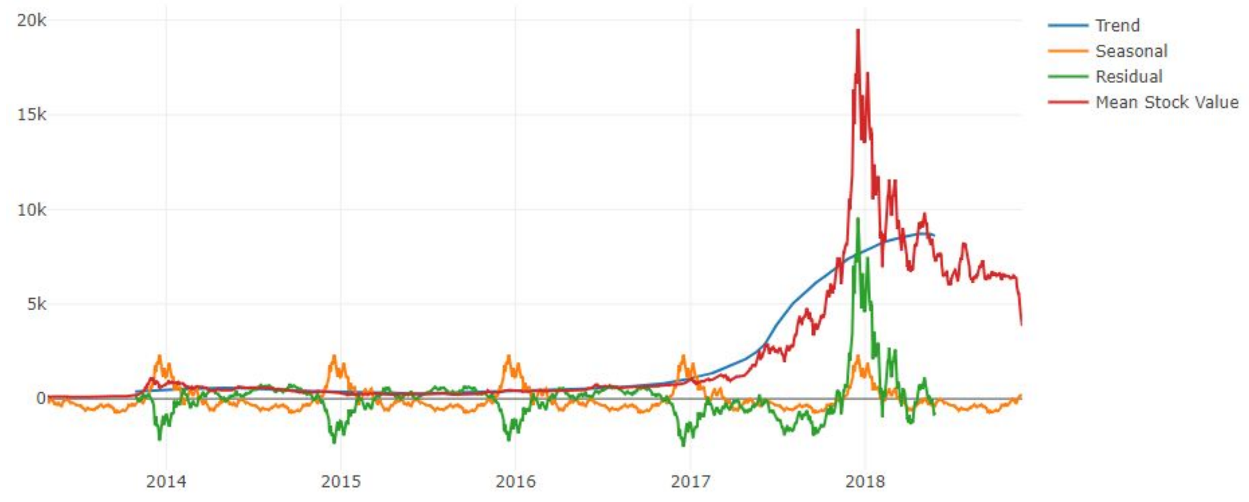
## Exploratory Visualization

Let’s take a closer look to data. Here I visualize Close  price and Trading volume, so we can see that currency volume is growing not only in a rise moments but also on it’s falls

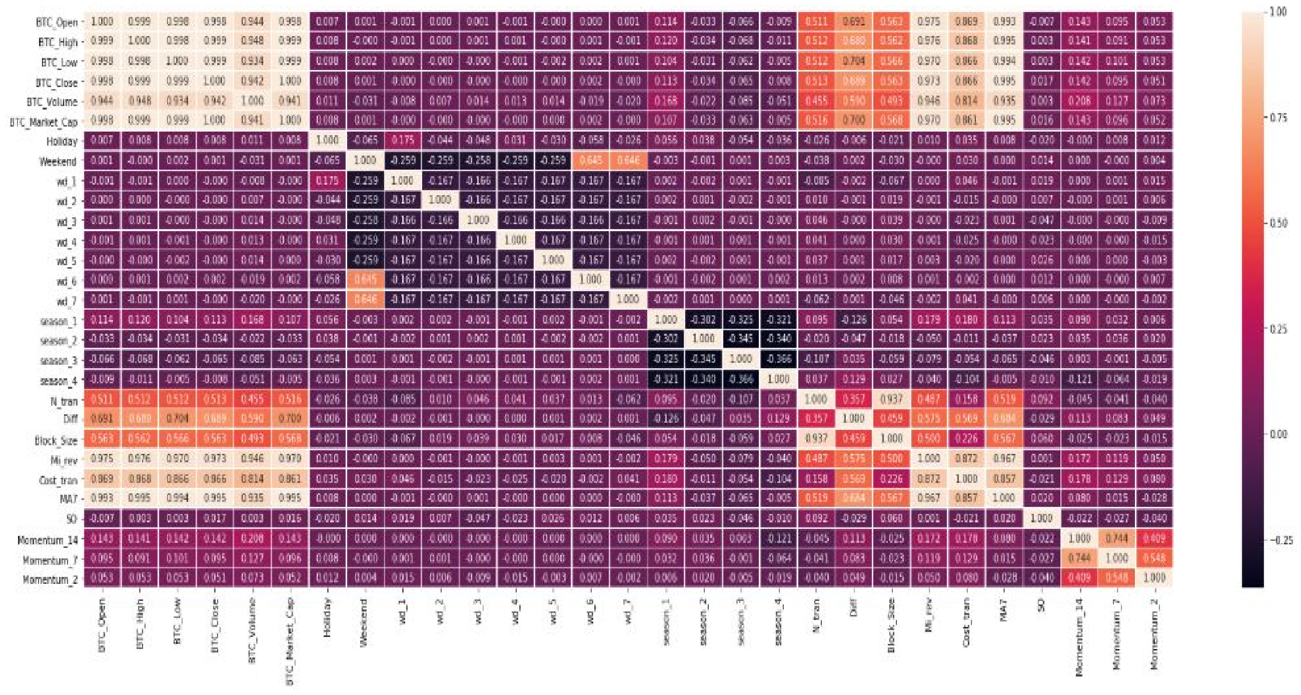


Most time series data can be described by three components. And those are trend, seasonality, and bias:

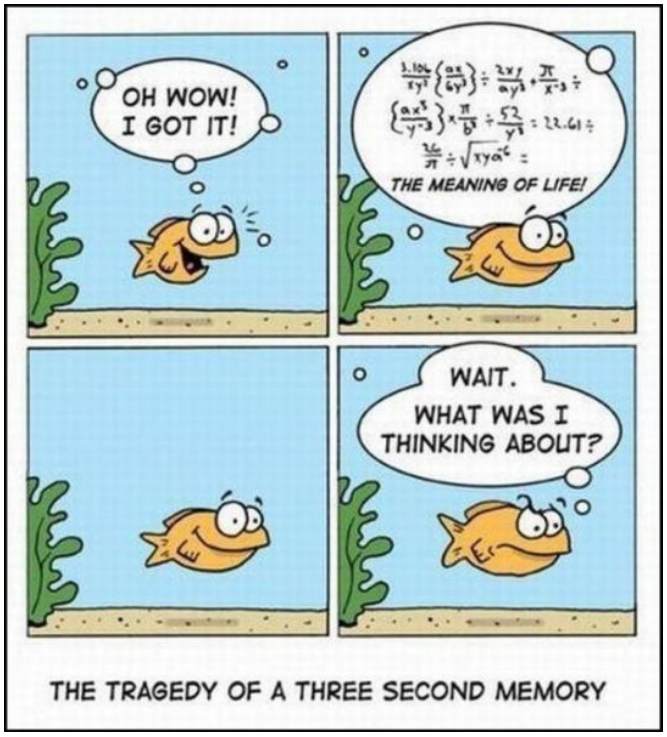
- Trend - a general systematic linear or (most often) nonlinear component that changes over time and does not repeat
- Seasonality -  a general systematic linear or (most often) nonlinear component that changes over time and does repeat. In other words, captures patterns that repeat every season
- Noise - a non-systematic component that is nor Trend/Seasonality within the data. It is what left.



I also made a heatmap in order to get insights into what features are correlated with the target - Bitcoin Close price. Unfortunately, there was no big impact:



## Algorithms and Techniques



For the time-series issue, is very important to memory previous states. Recurrent neural networks address this issue. They are networks with loops in them, allowing information to persist. But why exactly LSTM? Until we want to train RNNs via back-propagation, it looks good. As the gradient of our training samples gets propagated backward through our network so it can learn from its outcome, gradient gets weaker and weaker, by the time it gets to those neurons that are responsible for older data points in our time-series it has no juice to adjust them properly. This problem is called Vanishing Gradient. An LSTM cell is a type of RNN which stores important information about the past and forgets the non-important pieces, in this way when gradient back-propagates, it won't be consumed by unnecessary information.

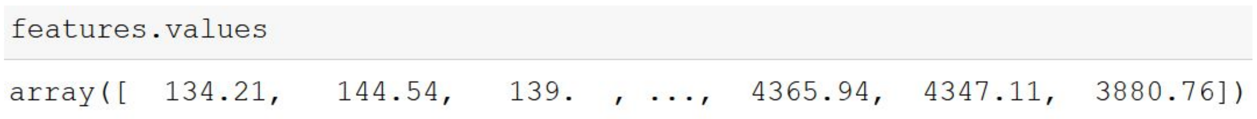
I chose Bitcoin prediction as a subject of this project because time-series and its technics were unknown by me. What was completely new and interesting, that LSTM itself has plenty of implementations and can be confused to newb researcher.

For my research, I've tried several Univariate(one variable measured over time) and Multivariate (multiple variables measured over time) time-series forecasting.

For example, let's take a closer look at Univariate data preparation.

The LSTM model will learn a function that maps a sequence of past observations as input to an output observation. As such, the sequence of observations must be transformed into multiple examples from which the LSTM can learn.

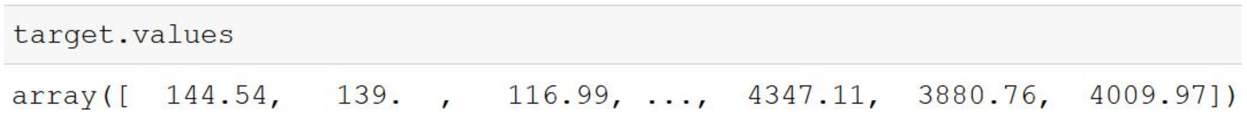
Bitcoin Close price for day t:



```
features.values

array([  134.21,   144.54,   139.  , ...,  4365.94,  4347.11,  3880.76])
```

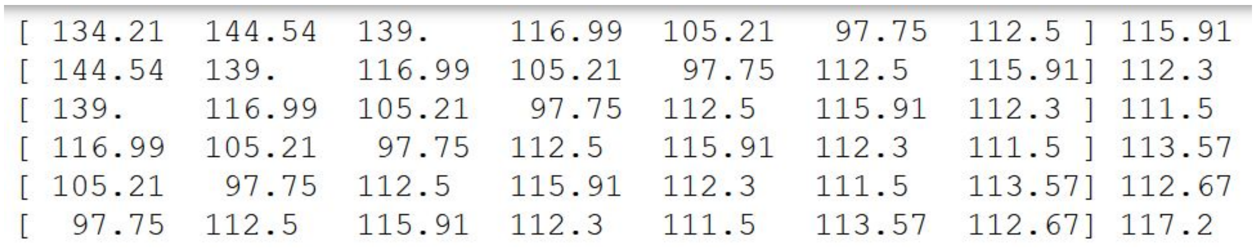
Bitcoin Close price for day t+1 (target feature)



```
target.values

array([  144.54,   139.  ,   116.99, ...,  4347.11,  3880.76,  4009.97])
```

A sample of prepared data (7-day sliding window):



```
[ 134.21  144.54  139.    116.99  105.21   97.75  112.5 ] 115.91
[ 144.54  139.    116.99  105.21   97.75  112.5   115.91] 112.3
[ 139.    116.99  105.21   97.75  112.5   115.91  112.3 ] 111.5
[ 116.99  105.21   97.75  112.5   115.91  112.3   111.5 ] 113.57
[ 105.21   97.75  112.5   115.91  112.3   111.5   113.57] 112.67
[  97.75  112.5   115.91  112.3   111.5   113.57  112.67] 117.2
```

Types of LSTM models I tried for this project:

UNIVARIATE and MULTIVARIATE:

- Vanilla LSTM. It is an LSTM model that has a single hidden layer of LSTM units, and an output layer used to make a prediction.
- Stacked LSTM. Multiple hidden LSTM layers can be stacked one on top of another
- Bidirectional LSTM. LSTM model allowed to learn the input sequence both forward and backward and concatenate both interpretations

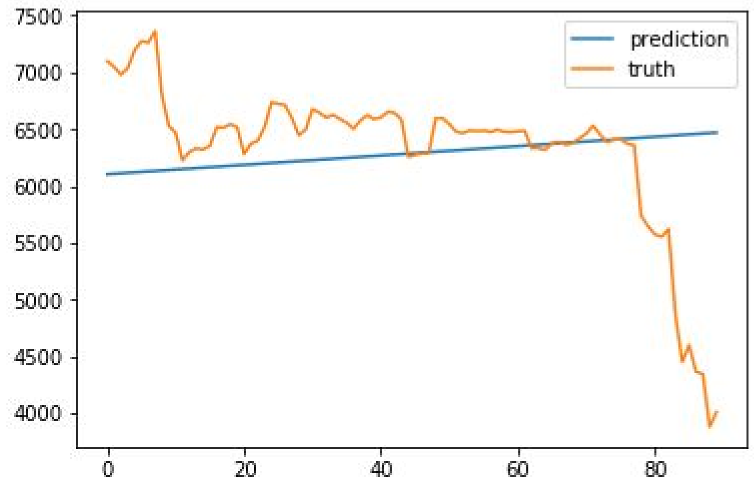
To build an optimal architecture of Neural Network there are some parameters can be tuned:

- Preprocessed input data (including train-test split, the sliding window size)
- Type of LSTM model
- Number of hidden layers
- Number of nodes per layer
- Optimizing function
- Number of epoch

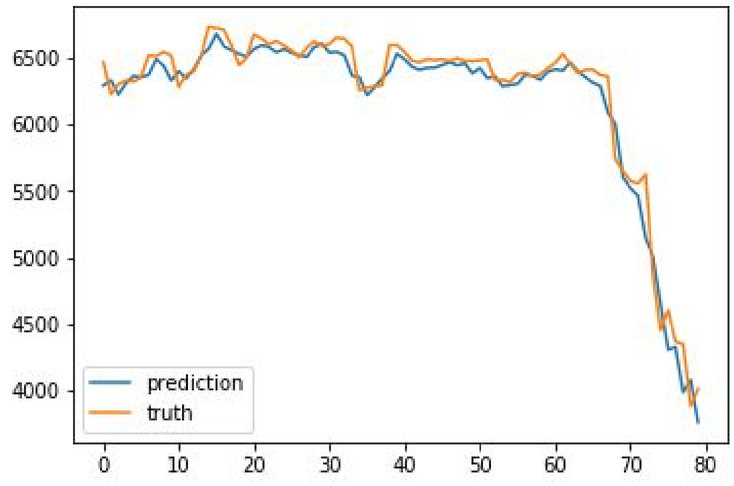
## Benchmark

I created a simple Linear Regression model as a benchmark for more advanced Deep Learning models. I will compare them by evaluating the RMSE of rescaled data.

I am trying to predict next day Bitcoin price, so if I go straight and built model just of the set of BTC close price, Linear regression doing a pretty bad job on test period with RMSE:732.290781026



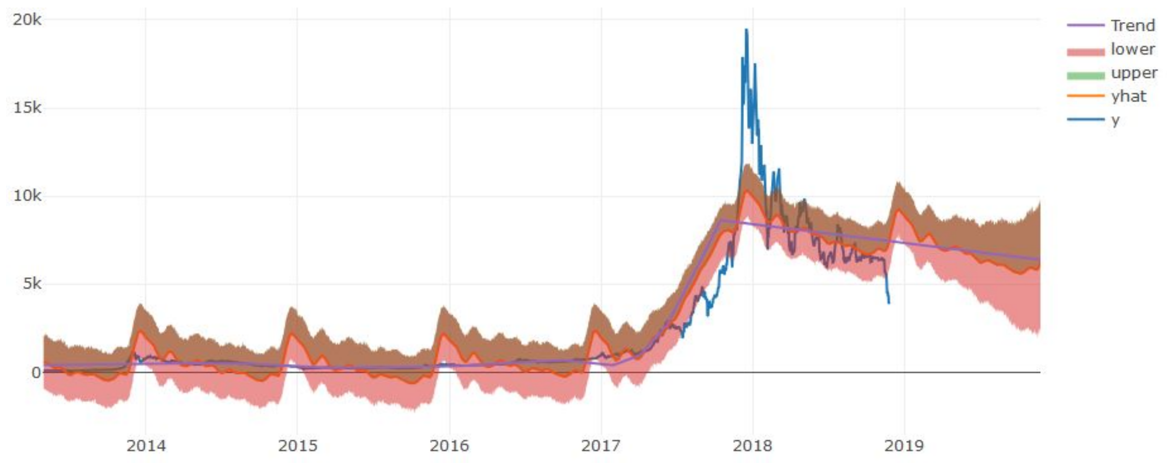
But if I use all created features and fit Linear Regression model with sliding windows (10 previous days features to predict one next day price) - it works much better RMSE:123.50143535010383



I’ve also tried several more nonlinear models with almost default configuration from An Empirical Comparison of Machine Learning Models for Time Series Forecasting 2010 list (non-deep-learning).
The results are:

| **MODELS** | **RMSE** |
|---|---|
| **nonlinear** | |
| KNeighborsRegressor | 673.014 |
| DecisionTreeRegressor | 289.560 |
| SVR | 5782.697 |
| **ensemble** | |
| BaggingRegressor | 149.876 |
| RandomForestRegressor | 152.230 |
| ExtraTreesRegressor | 191.452 |
| GradientBoostingRegressor | 160.861 |

Model, can be also very interesting to explore as a baseline - is Facebook Prophet.
Prophet is a procedure for forecasting time series data based on an additive model where non-linear trends are fit with yearly, weekly, and daily seasonality, plus holiday effects. It works best with time series that have strong seasonal effects and several seasons of historical data. Prophet is robust to missing data and shifts in the trend, and typically handles outliers well.
RMSE: 852.248806

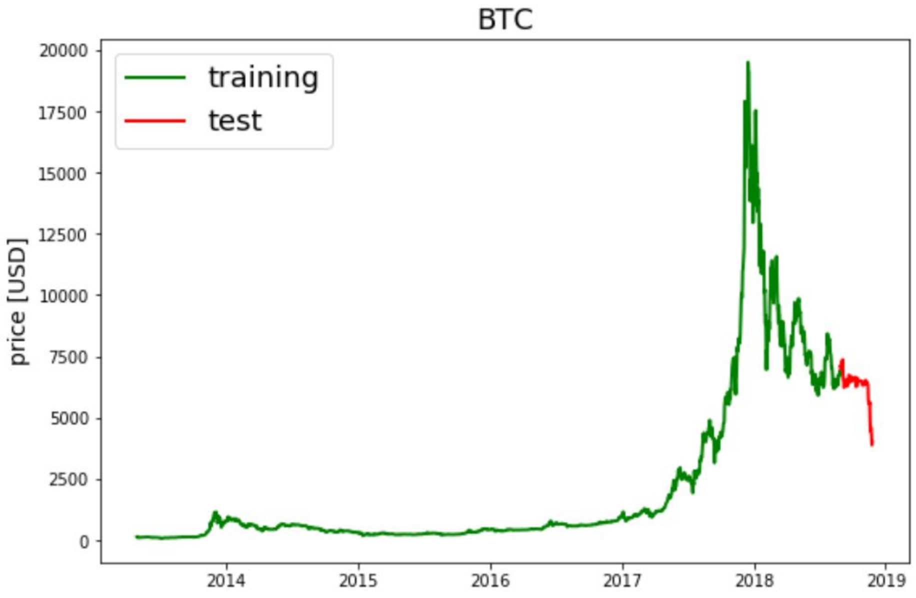


# METHODOLOGY

## Data Preprocessing

Preprocessing the data have several steps. After getting data and sort it by date, I check is any feature is absent and feature generation (DATA.ipynb) following preparation made:

- Load prepared dataset
- Creating training and test split (I chose 90 days for test period - it seems interesting to me, some part because this period almost hasn't rising trend but usually days prior to new year positively influence course, but it's arbitrary parameter. (train_test_split function)



- Normalize data. LSTMs are sensitive to the scale of the input data. It can be a good practice to rescale the data to the range of 0-to-1, also called normalizing. Data can be easily normalized using the MinMaxScaler preprocessing class from the scikit-learn library.
- Define sliding window split (window_data function)
  Example for data with 2 features, 3-day sliding window

| Prediction for 3rd day | Prediction for 4th day |
|---|---|
| input: | input: |
| [[feature1_day0 feature2_day0]<br>[feature1_day1 feature2_day1]<br>[feature1_day2 feature2_day2]] | [[feature1_day1 feature2_day1]<br>[feature1_day2 feature2_day2]<br>[feature1_day3 feature2_day3]] |

| output: | output: |
|---|---|
| [target_day3] | [target_day4] |

## Implementation

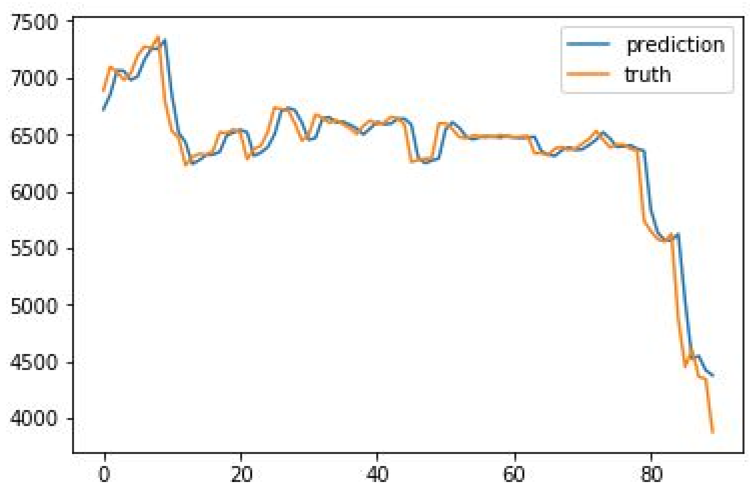
Once data is preprocessed, the implementation process can be defined as:

- Create model architecture
- Tuning training parameters
- Compile model
- Train model
- Plot validation error during training
- Make predictions
- Rescale predictions back to USD price
- Evaluate RMSE
- Plot actual and predicted prices
- Save model weights

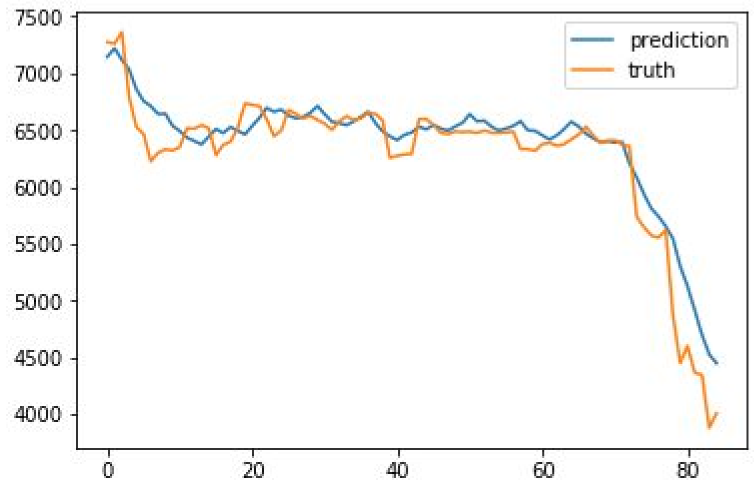
Across studies, I tried different architectures, mostly Long Short-Term Memory or LSTM stacked. All models on different data are saved to check their performance.

## Refinement

For the start, I build simple models on plain data - such as only Bitcoin Close price and Close price for the next day. Then, I build models on full data (with generated features). After several iterations on training, I noticed the result on simple data was actually better than on full data, I got confused.
Best LSTM model (Bidirectional) on simple data:



Best LSTM model (stacked with 5 “sliding windows” size):



With almost same RMSE around 175, prediction on simple data looks more logical.
So I consider, that I should be more sophisticated with data preparation especially with Stationarity. It’s not a simple moment, perfectly, if a cycle or trend is obvious, then it should be removed so that the model can focus on the underlying signal. I’ve tried to build larger and hierarchical models (stacked LSTM) suggested larger models can learn more, but it performs even worse. For this moment it’s not perfectly clear, but I guess I should be more careful with generating such features as “seasons” to avoid giving an extra trend. Finally, I decided to remove generated features, that has low correlated with the target. By removing it one by one and different combination, I build Final model. Then I tune hyperparameters such as the number of neurons and dropout value, also increase number of epochs and save best weights on validation.

Details of my project can be found in Jupiter notebooks:

- DATA. Everything about getting, exploring, feature engineering data
- BENCHMARK. Benchmark models and exploration

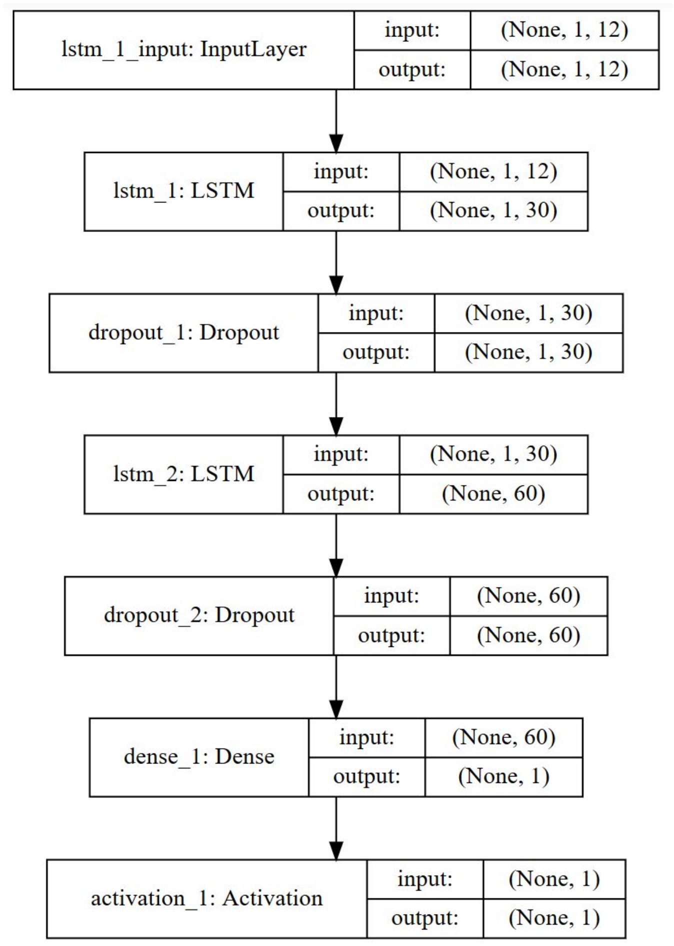
- LSTM_simple. LSTM models on simple data (such as only BTC Close price)
- LSTM_full. LSTM models on full data (with generated features)
- LSTM_less. LSTM models on finite data (including best model)

## RESULTS

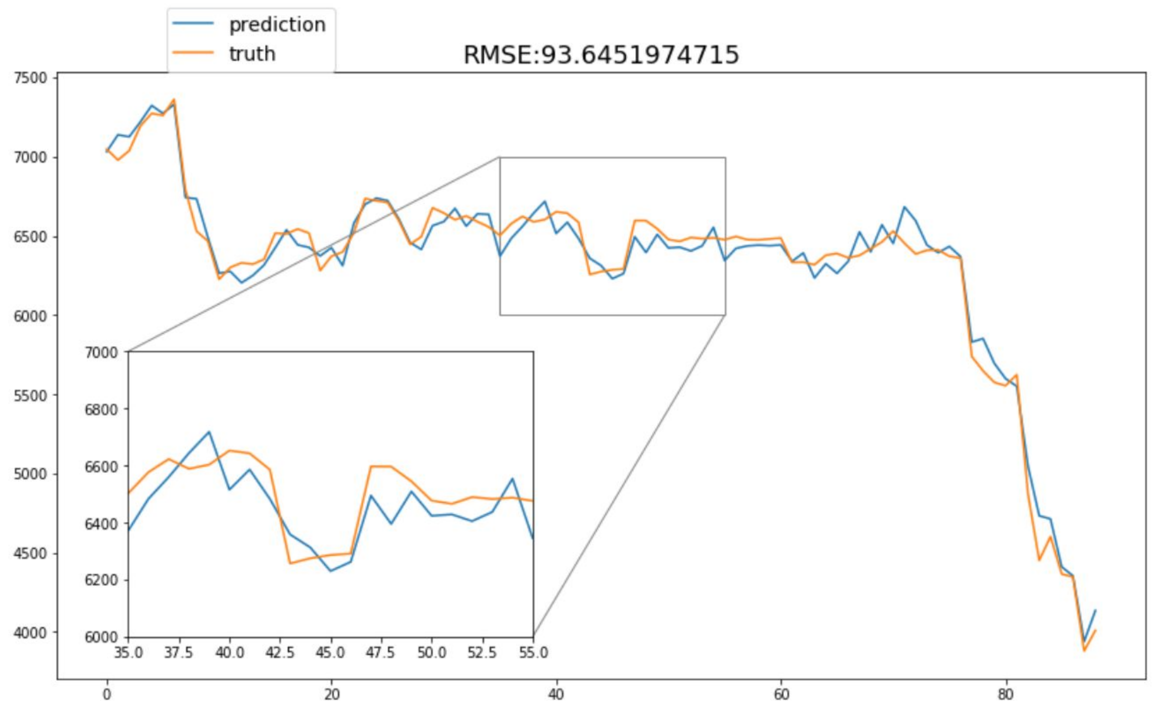
### Model Evaluation and Validation

Across studies, the best result was performed by Stacked LSTM with sliding window size equals one on data with less than generated features, with 30 units and dropout value of 0.3 in first LSTM layer, and 60 units and dropout equals 0,5 on second LSTM layer, with final Dense layer and Linear activation function. Final model looks reasonably good, it exceeds my expectations but still, due to the variability of Close price it can be understood only as a helper for the given problem (to predict next day price). Still, I feel not ready to be completely sure if my best model identified any underlying trends. And I'll discuss it in later visualizations.

Architecture:



The graph on the testing period:



## Justification

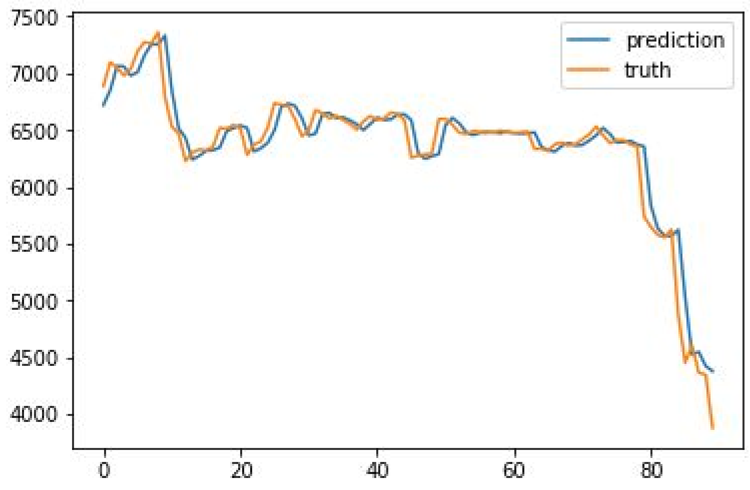
Compare to the benchmark simple linear model,  final LSTM model results are much stronger - 93.6 RMSE vs 732.3. That is a significant improvement. For sure, developed LSTM model generalizes better to the diverse dataset.

# CONCLUSION

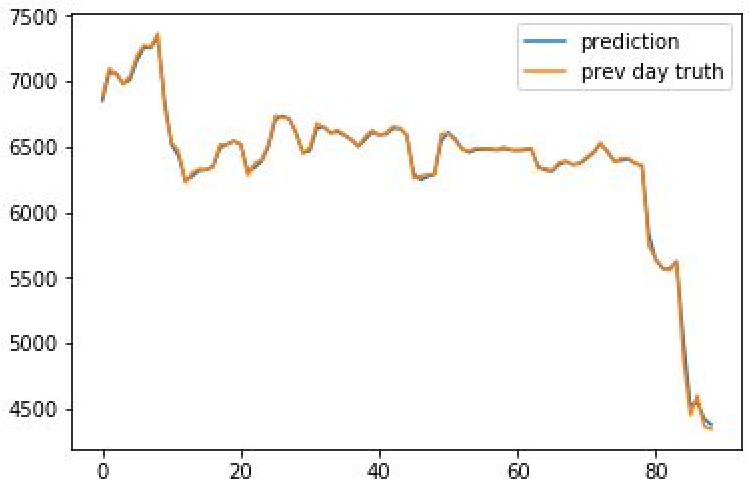
## Free-Form Visualization

I didn't expect I will found a perfect solution - I'm just a student researcher with simple LSTM model modifications, and cryptocurrencies are on air - so it can bring gold to experienced researches, but there still no signs of sophisticated solutions. But I'm surprised models works pretty fine and what is more interesting - It would be fun to check future prices and compare it with my best implementation.

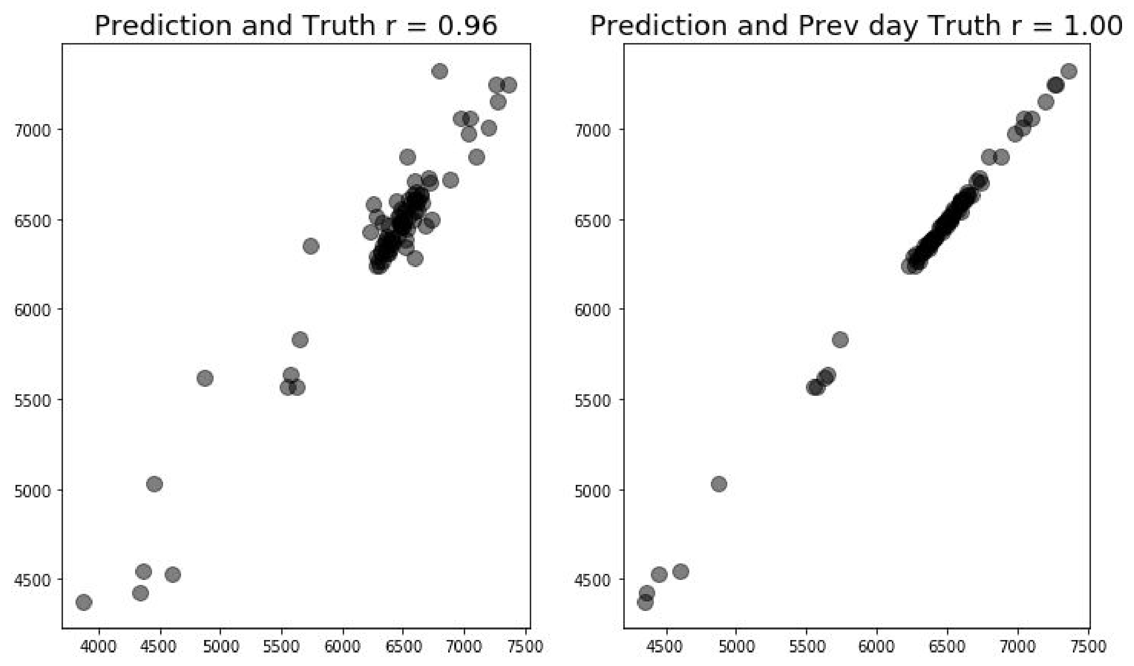
One thing is bothering me during this research. If we take a closer look on simple LSTM model on simple data, predictions look like:



Even visually, it obvious that if we shift actual date just for one day, this too lines will stick together:



And error falls from 175(which is still good for prediction) to 29 USD.
So I can make a conclusion, that this LSTM model predicting yesterday's price as today's. If we build the graph with Pearson correlation - it will look like this:



For the data such as Bitcoin price, LSTM is very oriented on yesterday's price. Even when my best model doesn't have such a strong correlation, it feels like I can't generally trust it my money, but it is the main reason to build such models - to make a profit on forecasting. This comparison gives me more thoughts about why my promising model can be not as perfect as it performs - for example, I just been lucky with the testing period.

## Reflection

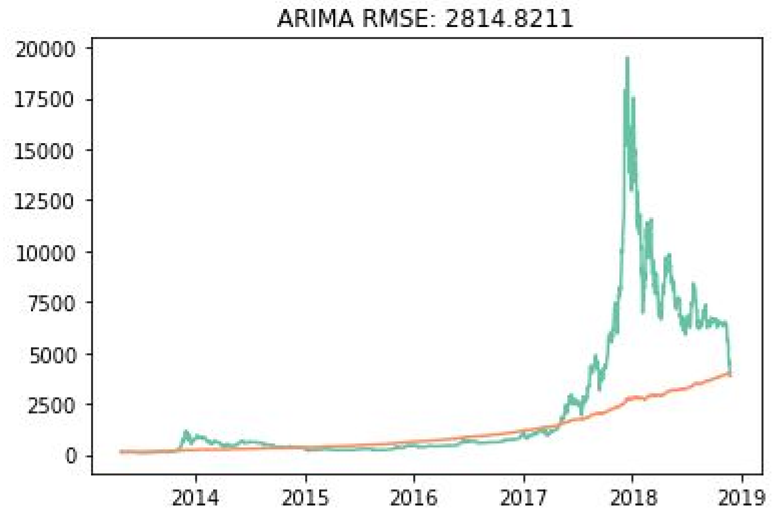
I made quality research during this project. Now, I can easily implement different LSTM architectures by Keras and visualize structures by Tensorboard. It was also a good practice for the future because now I know where to get a lot of interesting financial data for analysis. And how to prepare data for LSTM training - it should be normalized and can be split by sliding windows. My first hypothesis was the larger model with the bigger sliding window can make prediction more accurate, but in practice, it doesn't work. For such an unstable time-series as cryptocurrency price, it feels like more simple is better, and latest data affect most. Also, it is very important to check if data you're trying to use for training is stationarity.

The primary aim for me was to do a survey, how to work with time series - such as Bitcoin price. I made a conclusion, that actually I can't do much with just the price of the next day, it is also good to check what would be forecast for the longer period. I also realize, how practical it would be to check LSTM models in a completely different field - for example, natural language processing.

Resuming all the work I've done, I understood that LSTM is a huge field to explore.

## Improvement

I still believe the model can be improved by adding more reasonable data. For example, huge transactions can affect market (there were cases in Bitcoin history when society catch panic after someone had shifted 194,993 BTC in one transaction). Even more important features - is news about cryptocurrencies - from twitter, reddit, bitcointalk and other famous resources. In this case, we can use Sentiment analysis.
Next idea - is to try different models - in can be Convolutional neural network or even ConvLSTM. I even thought about deepmind's Wavenet (to predict price, not sound)
There are a lot of models can be used with time series data. For example, even simple implementation of Arima (AutoRegressive Integrated Moving Average) can give us some insights.



I still think, to make a real-world working prediction is possible, using a strong combination of different models and features, but in an enormous period of time and with great skills.
I realized, model 's hyperparameters are extremely sensitive to the results - so it would help to run some hyperparameter optimization technique (for example, Grid search / Random search), but it can use extremely long time.
Also, probably, it wasn't a perfect idea to use absolute price as a prediction - so I would also use the difference between Today and Tomorrow prices.

## REFERENCES

1) Predict the Future with MLPs, CNNs and LSTMs in Python, Jason Brownlee
2) The premier source for financial, economic, and alternative datasets QUANDL.com
3) Understanding LSTM
http://colah.github.io/posts/2015-08-Understanding-LSTMs/
4) Time Series Analysis: KERAS LSTM Deep Learning
https://www.business-science.io/timeseries-analysis/2018/04/18/keras-lstm-sunspots-time-series-prediction.html
5) Prediction of cryptocurrency prices
https://coinalysis.wordpress.com/2018/01/22/predict-cryptocurrency-prices-based-on-news-and-historical-price-data-2/
6) Anticipating cryptocurrency prices using machine learning
https://arxiv.org/pdf/1805.08550.pdf
7) Don't be fooled — Deceptive Cryptocurrency Price Predictions Using Deep Learning
https://hackernoon.com/dont-be-fooled-deceptive-cryptocurrency-price-predictions-using-deep-learning-bf27e4837151
8) A comprehensive beginner's guide to create a Time Series Forecast
https://www.analyticsvidhya.com/blog/2016/02/time-series-forecasting-codes-python/
9) Making Data Stationary
https://www.kdnuggets.com/2018/03/time-series-dummies-3-step-process.html
10) Predicting Bitcoin Prices by Using Rolling Window LSTM model
https://fintech.kdd2018.a.intuit.com/papers/DSF2018_paper_lee.pdf
11) 101 Formulaic Alphas
https://arxiv.org/ftp/arxiv/papers/1601/1601.00991.pdf
12) Facebook Prophet
https://facebook.github.io/prophet/docs/quick_start.html#python-api
13) Wavenet
https://deepmind.com/blog/wavenet-generative-model-raw-audio/
14) Plotly
https://plot.ly/python/time-series/
15) Stock market features
https://www.investopedia.com
16) Econometric tools in stock market
https://www.researchgate.net/publication/246006074_Econometric_Modelling_of_Stock_Market_Intraday_Activity
17) BTC mystery
https://www.coindesk.com/194993-btc-transaction-147m-mystery-and-speculation
18) additional information from bitcointalk, reddit and kaggle