Machine Learning Engineer Nanodegree

Capstone Project

Moustafa Banbouk
May 12th, 2018

I. Definition

Project Overview

The scope of this project is to predict the price of Cryptocurrencies, a relatively new type of currency that started in 2008 with Bitcoin. The hype of cryptocurrencies returned back in 2017 as we witnessed an increase of some cryptocurrencies like bitcoin by more than 2400% and then a great decrease as demonstrated in the following graph.

Cryptocurrencies belong to a highly vulnerable market that changes on daily basis. The frequency of price change for cryptocurrencies is very high as we are seeing a 30% increase or decrease in one day in some cases. The increase or

decrease of cryptocurrencies depend on various parameters including marketcap, liquidity, value proposition, number of coins in circulation but at the end all these factors are influenced by the number of buyers and sellers competing for a particular cryptocurrency coin. Similar to stock prices, if we can predict high increase or decrease, we will be able to invest in a smarter manner and optimize our trading strategy. We can run our prediction algorithm on multiple currencies and decide which coins we should, which should we buy and which to keep.

In order to solve the prediction problem under discussion and since we have the involvement of independent variables (historical price of an altcoin) and dependent variable (predicted price), we will be using machine learning regression algorithms to predict altcoin prices. Historically, regression started with astronomical observations to predict orbits of celestial bodies by Legendre and Gauss and since then, we had great progress to increase precision and efficiency.

There are numerous papers written on Cryptocurrency trading, the most famous of which is a reddit post by "joskye", a cryptocurrency investor and holder.

- Title: The Intelligent Investors Guide to Cryptocurrency
- Link: https://np.reddit.com/r/Particl/comments/7f28ja/the_intelligent_investors_guide_to_cryptocurrency/

As for the academic research in the field, I would like to refer to the following videos on which my algorithms were based:

The project will be based on the following videos from Siraj Raval and Sandeep Sharma.

Predicting Stock Prices - Learn Python for Data Science #4

How to Predict Stock Prices Easily - Intro to Deep Learning #7

Predicting Stock Price: A Machine Learning Project

As for the referenced academic paper, the project is based on the "Stock Prediction using Machine Learning a Review Paper" with the below details:

- Paper Title: "Stock Prediction using Machine Learning a Review Paper"
- Institute: Information Technology (I.T.), Vidyalankar Institute of Technology, Mumbai, Maharashtra, India
- Published in: The International Journal of Computer Applications in April 2017

Regarding the data source, we will be downloading fresh cryptocurrency historical data from the well known coinmarket cryptocurrency market capitalizations monitoring website http://www.coinmarketcap.com through the site's JSON APIs.

Problem Statement

As highlighted in the previous paragraph, the main problem with cryptocurrency is their high volatility and therefore, high fluctuations in their value. As an investor, we need to know when to buy, sell or hold a particular investment in a particular cryptocurrency coin. The main decision differentiator is the expected value of such cryptocurrency that can be predicted using machine learning algorithms.

Since the problem under discussion involves predicting altcoin price, we will be testing various regression algorithms and select the best model with the highest precision (using mean-squared-error metric). Python scikit library provides multiple algorithms that will help with regression prediction including but not limited to Linear Regression, Bayesian Ridge Regression, Random Forest Regression …

Metrics

There are multiple metrics that can be used to compare machine learning models, among these metrics, we have R^2 and MSE in which R focusses on linear relationships between variables in contrary to MSE that don't have dependency on the type of relationships between variables.

Since the problem under discussion is a supervised learning / regression problem, we will be using MSE (Mean-Squared-Error) which will measure the average of the squares of errors or deviations between the predicted altcoin price and the actual one. The MSE is a measure of the quality of an estimator – it is always non-negative and values closer to zero are better.

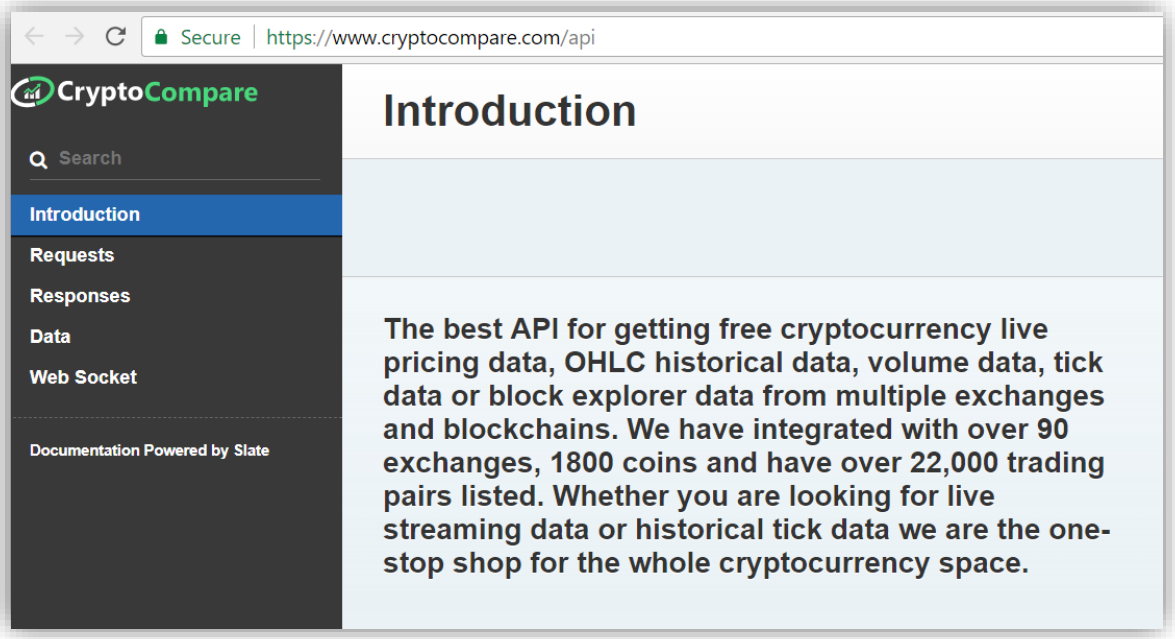
In order to compare between the performance of various machine learning algorithms, we will be comparing the average MSE and selecting the algorithm with the least MSE. In our problem of predicting the altcoin price, we will be measuring the weekly average MSE value for multiple weeks and calculate the average value of all MSEs as a performance metric.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

II. Analysis

Data Exploration

After performing a simple search on the Internet, I found an online portal that aggregates data from many exchanges and offer many query parameters (Link: https://www.cryptocompare.com/api)



The CryptoCompare API portal provides sample request/response query APIs that can provide Coins List, Recent Price, Daily average price, Hourly average price and other. In this project, we will be querying the average daily price of Ethereum cryptocurrency. In the following paragraphs, we will be analyzing a sample output from CryptoCompare site:

We will be using the **histoday** API request to provide the Ethereum cryptocurrency (**fsym=ETH**) price in USD (**tsym=USD**) of the last 700 days (**limit=700**) before a unix formatted timestamp corresponding to 17th of February 2018 (**toTs=1518825600.0**)

- Query for daily average price: https://min-api.cryptocompare.com/data/histoday?fsym=ETH&tsym=USD&limit=700&aggregate=1&toTs=1518825600.0
- Output:


```
{"Response":"Success","Type":100,"Aggregated":false,"Data":
[{"time":1458345600,"close":10.55,"high":11.19,"low":9.8,"open":
10.75,"volumefrom":48704.1,"volumeto":512234.39},{"time":1458432
000,"close":10.06,"high":10.85,"low":9.52,"open":10.55,"volumefr
om":47028.95,"volumeto":469675.86},
{"time":1458518400,"close":11.97,"high":11.99,"low":9.93,"open":
10.06,"volumefrom":73367.01,"volumeto":819534.02},
{"time":1458604800,"close":10.96,"high":12.2,"low":10.95,"open":
11.97,"volumefrom":63118.64,"volumeto":727727.88},
{"time":1458691200,"close":12.29,"high":12.5,"low":10.96,"open":
10.96,"volumefrom":54811.26,"volumeto":654062.63},

...
```


The above output is self-explanatory in which the first line in the output confirms a successful query and the next lines will give the closing price, high price, low price, opening price, volume exchange on a particular date. For the sake of this project, we will be only extracting the closing price per date and ignore the other parameters. On analyzing the output of this query, we didn't find any abnormalities like timestamps without values and therefore, no need for special treatment of the output data; however, we will be storing only relevant information (i.e. the closing price for a particular day presented as a Unix timestamp).

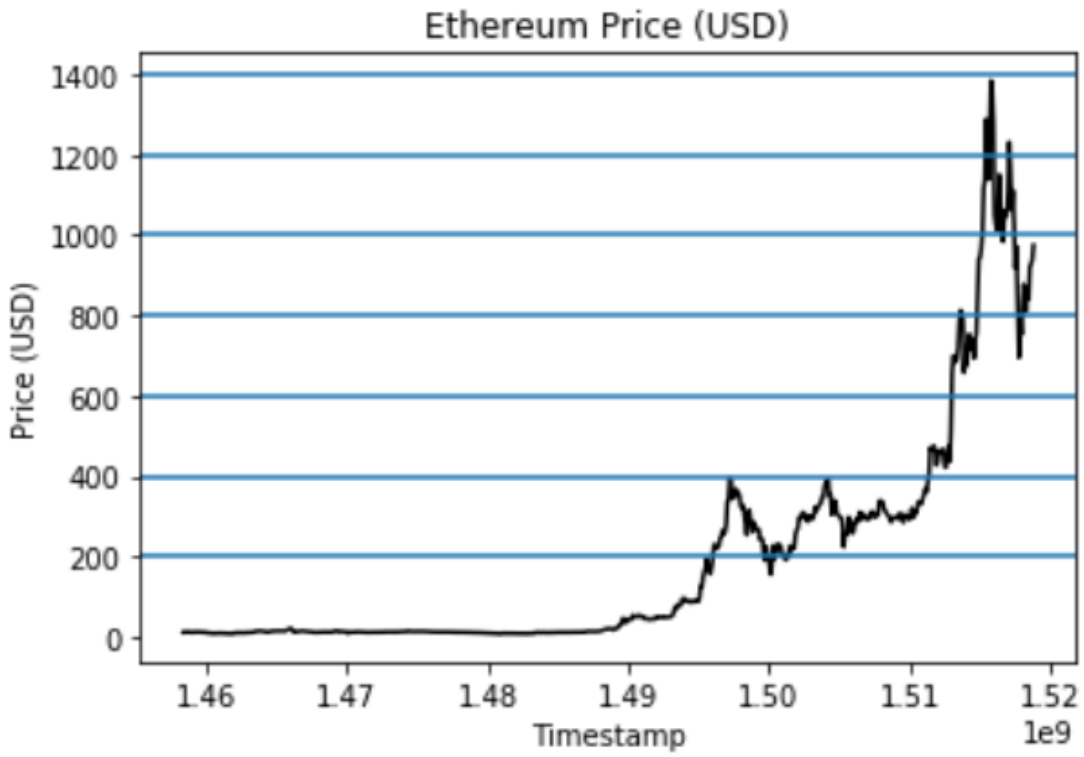
To facilitate API calling, we will be using the following Python function:

```
def histoDay(self, currency, baseCurrency = 'BTC', limit = 60,
aggregate = 0, toTs = ""):
        url = 'https://min-api.cryptocompare.com/data/histoday?fsym='
        url += currency + '&tsym='
        url += baseCurrency + '&limit='
        url += str(limit) + '&aggregate=' + str(aggregate)
        url += '&toTs=' + str(toTs)
        print(url)
        strOut = self.query(url)
        strOut =
str(strOut).replace("true","True").replace("false","False")[2:][:-1]
        return strOut
```

Exploratory Visualization

Since the exercise under discussion involves prediction of Ethereum price per each day, we will be exploring the provided data by using pyplot as per the

below. It is clearly shown that the provided data doesn't include any outliers or missing data and therefore no special pre-processing is required. In addition, the last peak shows the latest increase in Ethereum price in December 2017 and then the sudden loss in January 2018 indicating accurate data.

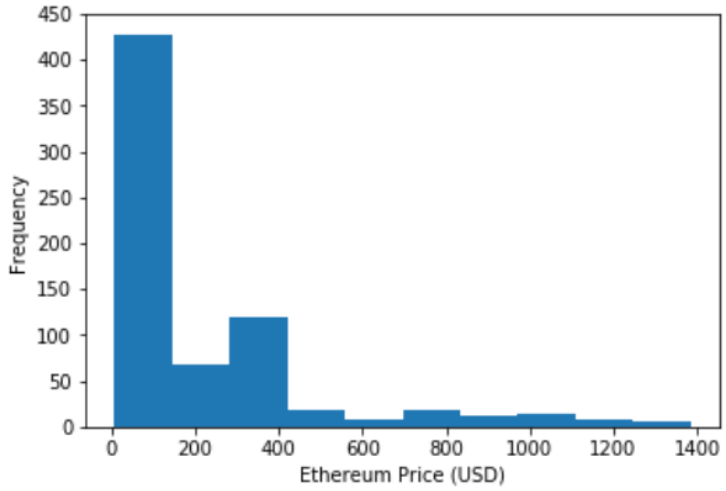


From another angle, we will try to visualize the frequency of Ethereum price using histogram in addition to calculating the standard deviation of the Ethereum closing price:

```
In [28]: # Important Stats
x1 = dfCC2['timestamp']
y1 = dfCC2['eth_close']

plt.hist(y1)
plt.xlabel("Ethereum Price (USD)")
plt.ylabel("Frequency")

plt.show()
stdY1 = np.std(y1)
print('Standard deviation for closing price of Ethereum: ', stdY1)
```




```
Standard deviation for closing price of Ethereum:  281.813041403
```

As can be seen in the above graph distribution is not gaussian and most of the Ethereum price concentration is less than 400 and the standard deviation is 281.813 USD.

Algorithms and Techniques

We will be using machine learning to predict the expected cryptocurrency value based on historical values of the same altcoin and other altcoins. To achieve this goal, we will be using machine learning algorithms and compare among their performance. This will allow us to derive the best algorithm that can be used in such an endeavor. As we are dealing with continuous numerical data, the machine learning problem at hand is a pure regression problem and therefore, we will be using supervised learning regression algorithms to predict expected cryptocurrency future values.

As input features, we will be using the following:

1. Cryptocurrency cost before 1 day, 2 days, 3 days … 8 days
2. Cryptocurrency cost before 2 weeks
3. Cryptocurrency cost before 3 weeks
4. Cryptocurrency cost before 4 weeks
5. Cryptocurrency cost before 8 weeks

The output feature is simply the predicted cryptocurrency cost for a particular date.

The following table demonstrate high level differences between the most known regression models:

Algorithm	**Problem Type**	**Results interpretable by you?**	**Easy to explain algorithm to others?**	**Average predictive accuracy**	**Training speed**	**Amount of parameter tuning needed**	**Performs well with small number of observations?**	**Features might need scaling?**
KNN	Either	Yes	Yes	Lower	Fast	Minimal	No	Yes
Linear regression	Regression	Yes	Yes	Lower	Fast	None (excluding regularization)	Yes	No (unless regularized)
Decision trees	Either	Somewhat	Somewhat	Lower	Fast	Some	No	No
Random Forests	Either	A little	No	Higher	Slow	Some	No	No
AdaBoost	Either	A little	No	Higher	Slow	Some	No	No

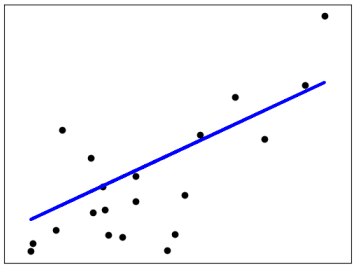
Regarding the machine learning algorithms to be used, we will be mainly using the following algorithms:

1. Linear Regression
2. Bayesian Ridge Regression
3. Random Forest Regression
4. Linear Ridge Regression
5. Linear Lasso Regression
6. Linear Lasso Lars Regression

As a strategy in selecting the best model, we will be first using the default parameters and then try to optimize the best model by playing with the corresponding hyperparameters.

As a rule of thumb in the world of predictive analytics, we start with linear models as it is the simplest among regression models with practically no hyperparameters to tweak. **Linear regression** fits a linear model with coefficients $w = (w_1, ..., w_p)$ to minimize the residual sum of squares between the observed responses in the dataset, and the responses predicted by the linear approximation. Mathematically it solves a problem of the form:

$$\min_w ||Xw - y||_2^2$$



The other regression models employ advanced techniques to optimize a solution better than that provided by the simple linear regression model.

Linear Ridge Regression: addresses some of the problems of Ordinary Least Squares by imposing a penalty on the size of coefficients. The ridge coefficients minimize a penalized residual sum of squares,

$$\min_w ||Xw - y||_2^2 + \alpha||w||_2^2$$

Here, $\alpha \geq 0$ is a complexity parameter that controls the amount of shrinkage: the larger the value of α, the greater the amount of shrinkage and thus the coefficients become more robust to collinearity.

A minor variation from Linear Ridge Regression, Sci Kit Learn provides **Linear Ridge CV** in which RidgeCV implements ridge regression with built-in cross-validation of the alpha parameter.

Another important linear model is the **Linear Lasso Regression** model is a linear model that estimates sparse coefficients. It is useful in some contexts due to its

tendency to prefer solutions with fewer parameter values, effectively reducing the number of variables upon which the given solution is dependent.

Mathematically, it consists of a linear model trained with ℓ_1 prior as regularizer. The objective function to minimize is:

$$\min_w \frac{1}{2n_{samples}}||Xw - y||_2^2 + \alpha||w||_1$$

The lasso estimate thus solves the minimization of the least-squares penalty with $\alpha||w||_1$ added, where α is a constant and $||w||_1$ is the ℓ_1-norm of the parameter vector.

As a variant of the Linear Lasso Regression, we have **Linear Lasso Lars Regression** which is implemented using the LARS algorithm, and unlike the implementation based on coordinate_descent, this yields the exact solution, which is piecewise linear as a function of the norm of its coefficients.

We also have another predictive model based on conditional probability which is the **Bayesian Ridge Regression** that estimates a probabilistic model of the regression problem. The prior for the parameter w is given by a spherical Gaussian:

$$p(w|\lambda) = \mathcal{N}(w|0, \lambda^{-1}\mathbf{I_p})$$

The priors over α and λ are chosen to be gamma distributions, the conjugate prior for the precision of the Gaussian.

Finally, we have a **Random Forest Estimator** that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if bootstrap=True (default).

Benchmark

As a main parameter to compare between the performance of all models and since this is a regression problem, we will be using the basic linear regression model as a benchmark to compare the accuracy of our other models with.

III. Methodology

Data Preprocessing

The main data source from which we are extracting our data (api.cryptocompare.com) provides the close, high, low, open prices of a particular altcoin on a particular day. The required features include the following:

1. Cryptocurrency cost before 1 day, 2 days, 3 days … 8 days
2. Cryptocurrency cost before 2 weeks
3. Cryptocurrency cost before 3 weeks
4. Cryptocurrency cost before 4 weeks
5. Cryptocurrency cost before 8 weeks

Consequently, we developed a function loadCryptos() that will read the altcoin attributes from api.cryptocompare.com and build a table that will append the target features besides to the original features downloaded from api.cryptocompare.com:

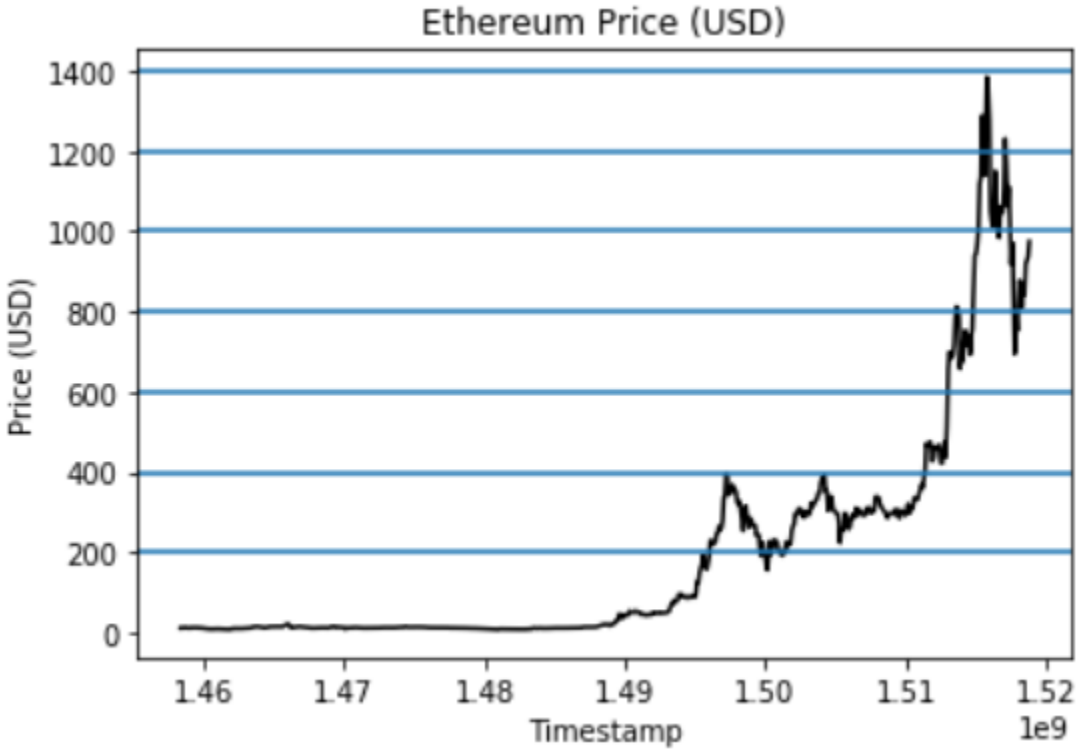
```
Function: loadCryptos
Overview: This function derives closing, high, low and opening price in
addition to the volume exchanged of the top 9 cryptocurrencies through
CryptoCompare's histoday() API and store the output in a Dataframe
Input:
       tsLastDayLastDAy: The timestamp of the last day to be read from
     CryptCompare histoday() API
       daysToBeAnalyzed: Total number of days to be read
       dfCC: The dataframe in which we are planning to store the
     Cryptocurrency prices
Output:
       DataFrame that includes the cryptocurrency prices
```

This function will return a dataframe with the following format:

_high	bch_low	...	iot_close_before_3days	iot_close_before_4days	iot_close_before_5days	iot_close_before_6days	iot_close_before_7days	iot_c
	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN
	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN
	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN
	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN
	0.0	...	NaN	NaN	NaN	NaN	NaN	NaN

With regards to abnormalities, we couldn't detect any missing or outlier values and this was clearly seen during exploratory visualization. In addition, since the models that we have selected don't require any scaling (although I've build

scaling options in my code), we didn't normalize our data and used the original USD cost values throughout the analysis.



After generation of the data in clear DataFrame, we split the data to 80% training and 20% testing and then we applied our machine learning models to learn from training data and measure accuracy from our testing data.

Implementation

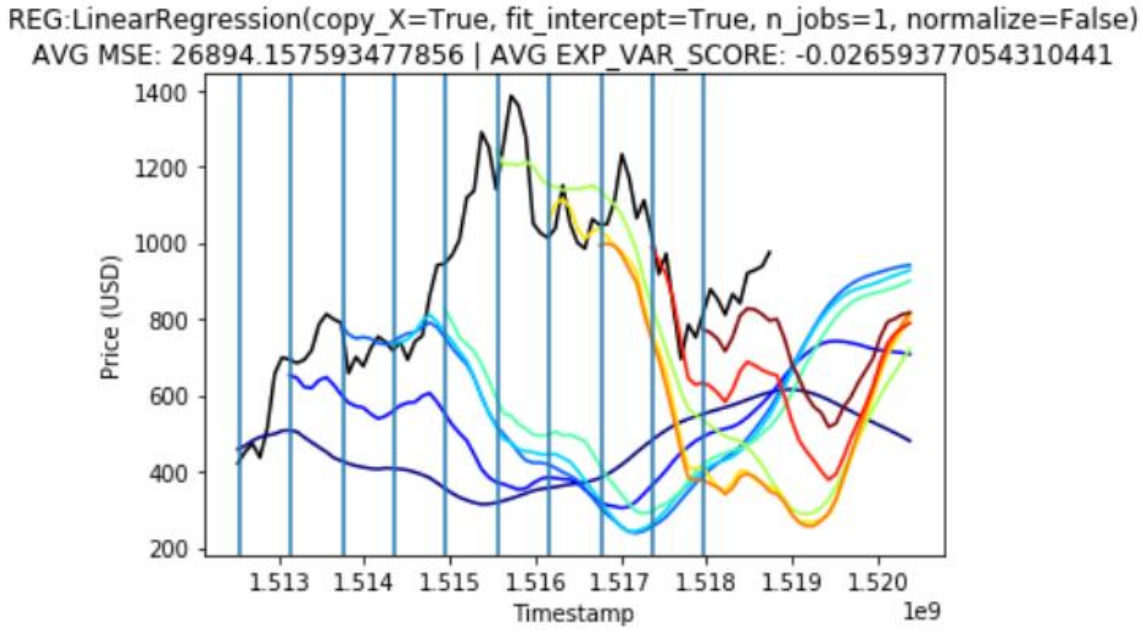
Comparing with other similar projects, we clearly see that this is a simple regression supervised learning problem that can be solved through Python's "Sci Kit Learn" library (In our case, we used SKLearn v0.18.1) one of the following algorithms and their parameters:

1. Linear Regression (Default Parameters)
2. Bayesian Ridge Regression (Default Parameters)
3. Random Forest Regression (max_depth=2, random_state=0)
4. Linear Ridge Regression (Alpha = 0.5)
5. Linear Ridge CV Regression (alphas=[0.1, 1.0, 10.0])
6. Linear Lasso Regression (alpha = 0.1)
7. Linear Lasso Lars Regression (alpha = 0.1)

In order to have a generic method for running the required prediction, we developed a function **predictCryptoPrice()** that will:

1. Read the test data
2. Divide the test data into multiple chunks (In our case 10 chunks)
3. Using a given regression model regr, we will predict the altcoin't price starting from the first timestamp (Unix DateTime format) in each chunk and for a number of days equal to daysToPredict parameter
4. Return the mean squared error (MSE) for each chunk and the average MSE of all chunks which will be used as a performance metric to compare the performance of the various models

We will later save the output of predictCryptoPrice() function and plot the multiple predictions in one graph using various colors as shown in the below graph obtained when using Linear Regression model default values (The x-axis in this graph represent the date in which the altcoin cost is calculated or predicted and the date format used is Unix timestamp format. The y-axis represents the altcoin's price in USD):



The black graph represents the actual cost of the training data and the colored graphs represents the predicted cost starting from various timestamps.

Function: `predictCryptoPrice`

Overview: predictCryptoPrice() function is used to predict upcoming cryptocurrency cost given a certain machine learning trained model. This function divides data to multiple timeslots and predict output values.

Input:

- regr: Machine Learning regression algorithm/model to be used for data prediction
- daysToPredict: Total days to predict
- crypto: The cryptocurrency under analysis
- inDfCC: The dataframe that stores the Cryptocurrency prices
- inLsCCFeatures: Input features to be used during training
- inPercentageOfTesting: Percentage of test data

Output:

- totalNumberOfDays: Total number of days to predict
- totalNumberOfChunks: Total number of timeslots to predict
- initialPredictionDate: the first prediction date to start from
- finalPredictionDate: The last prediction date to end our prediction
- MSE_list_avg: List of the average Mean Square Errors performance metrics for all timeslots
- exp_var_score_list_avg: List of the average Explained Variance Scores performance metrics for all timeslots
- regr: Machine Learning regression algorithm to be used while training data
- dfCC_CC_X: Input Features for all samples
- dfCC_CC_y: Output value for all samples
- dfCC_CC_X_train: Input Features for training samples
- dfCC_CC_y_train: Output value for training samples
- dfCC_CC_X_test: Input Features for testing samples
- dfCC_CC_y_test: Output value for testing samples
- dfCC_CC_X_test_before_list: Input Features for testing samples before prediction date (predicted values) for all timeslots
- dfCC_CC_y_test_before_list: Output values for testing samples before prediction date (predicted values) for all timeslots
- dfCC_CC_X_test_after_list: Input Features for testing samples after prediction date (predicted values) for all timeslots
- dfCC_CC_y_test_after_list: Output values for testing samples after prediction date (predicted values) for all timeslots
- dfCC_CC_X_test_list: Input Features for testing samples for all timeslots
- dfCC_CC_y_test_list: Output values for testing samples (predicted values) for all timeslots
- dfCC_CC_y_test_after_1week_pred_list: List of Output values (predicted values) for testing samples after one week of prediction date
- dfCC_CC_X_test_after_1week_pred_list: List of Input features for testing samples after one week of prediction dat
- dfCC_CC_y_test_after_1week_or_list: List of Output values (original values) for testing samples after one week of prediction date
- dfCC_CC_X_test_after_1week_or_list: List of Input features (Original values) for testing samples after one week of prediction date
- MSE_list: List of Mean Square Errors performance metrics for all timeslots
- exp_var_score_list: List of Explained Variance Scores performance metrics for all timeslots

It should be noted that there were no complications in implementing the regression algorithm. The only difficulty I faced was related to learning how to programmatically add new columns and rows.

Refinement

After running prediction modeling using the 6 pre-selected ML algorithms, we compared all the obtained results and provided the output through the below table showing the average MSE of various prediction series (PS. We are performing prediction every one week from the provided test)

ID	Model	Parameters	Avg. MSE	Avg. EXP_VAR_SCORE
1	Linear Regression	Default	26894.15759350	-0.02659377
2	Bayesian Ridge	Default	27337.24954620	-0.02151480
3	Random Forest	max_depth=2, random_state=0	317180.75365800	0.00000000
4	Linear Ridge	Alpha = 0.5	26894.19238870	-0.02659232
5	Linear Ridge CV	alphas=[0.1, 1.0, 10.0]	26894.85611190	-0.02656488
6	Linear Lasso	alpha = 0.1	26898.26207790	-0.02682159
7	Linear Lasso Lars	alpha = 0.1	31248.47845650	-0.42020684

As can be seen from the above graphs, we have similarity of performance between the following algorithms:

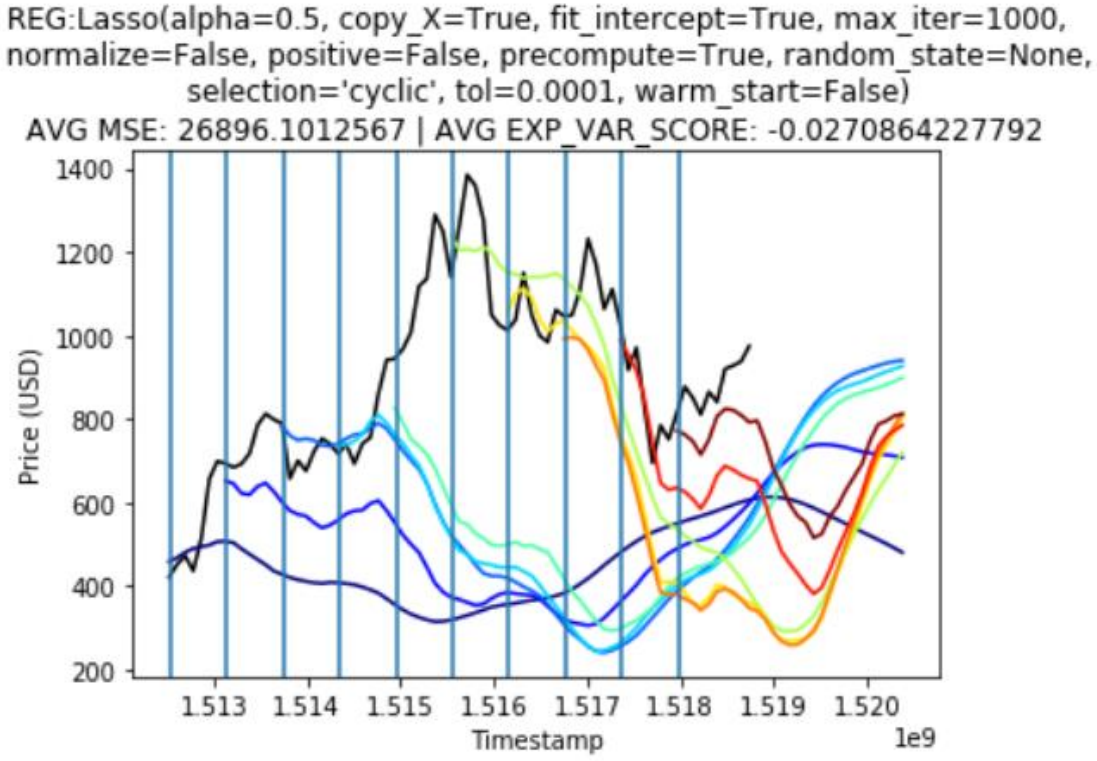
1. Linear Regression
2. Bayesian Ridge Regression
3. Linear Ridge Regression
4. Linear Lasso Regression

IV. Results

Model Evaluation and Validation

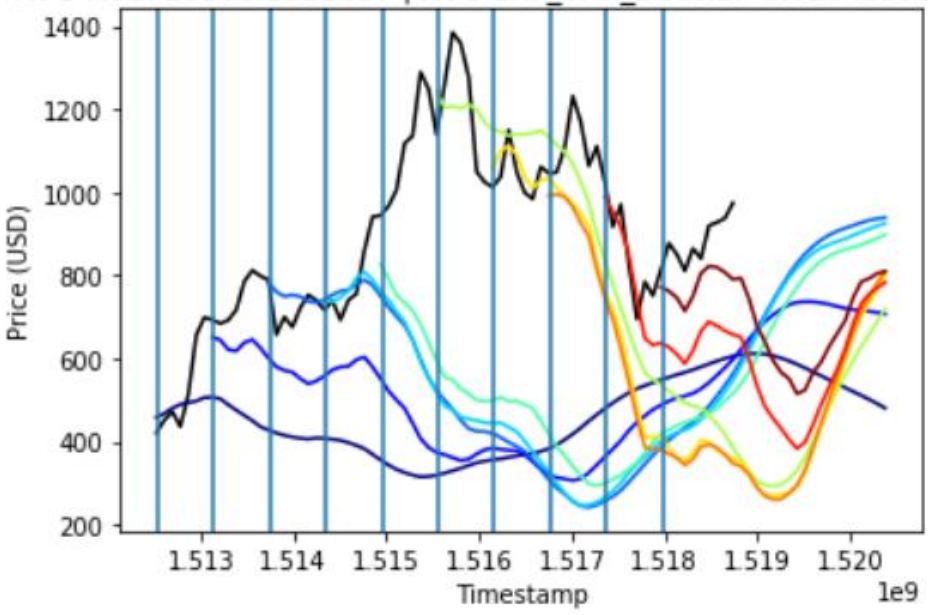
Since the performance of 4 models is similar, we selected one of these models randomly: “Linear Lasso” and tried to optimize it by running multiple iterations based on different hyper parameter values.

Trial #1: Alpha=0.5 / max_iter=1000 / precompute=True



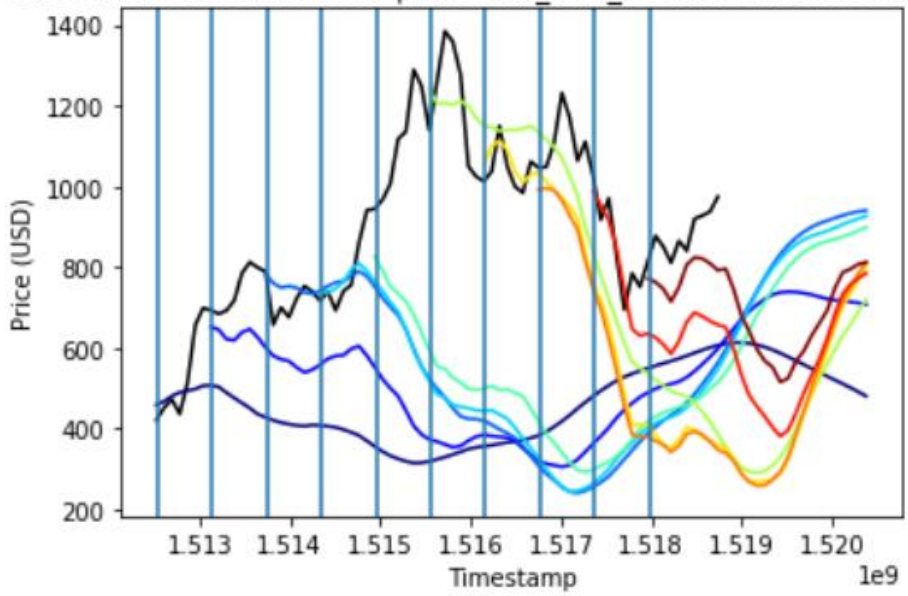
Trial #2: alpha=0.7 / max_iter=10000 / precompute=True

REG:Lasso(alpha=0.7, copy_X=True, fit_intercept=True, max_iter=10000,
normalize=False, positive=False, precompute=True, random_state=None,
selection='cyclic', tol=0.0001, warm_start=False)
AVG MSE: 26908.1322887 | AVG EXP_VAR_SCORE: -0.0274359509142

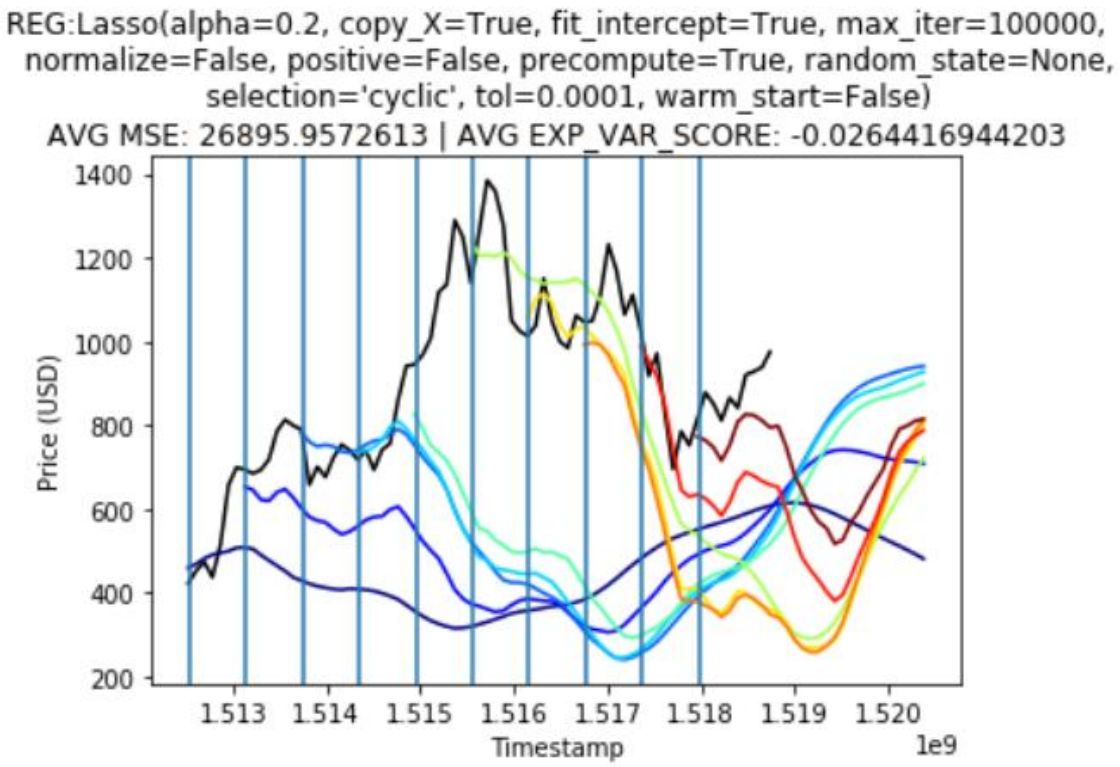


Trial #3: Lasso with alpha=0.5 / max_iter=100000 / precompute=True

REG:Lasso(alpha=0.5, copy_X=True, fit_intercept=True, max_iter=100000,
normalize=False, positive=False, precompute=True, random_state=None,
selection='cyclic', tol=0.0001, warm_start=False)
AVG MSE: 26896.1012567 | AVG EXP_VAR_SCORE: -0.0270864227792



Trial #4: Lasso with alpha=0.2 / max_iter=100000 / precompute=True



ID	Model	Trial	Parameters	Avg. MSE	Avg. EXP_VAR_SCORE
1	Linear Lasso	1	Alpha=0.5 / max_iter=1000 / precompute=True	26896.10125670	-0.02708642
2	Linear Lasso	2	alpha=0.7 / max_iter=10000 / precompute=True	26908.13228870	-0.02743595
3	Linear Lasso	3	Lasso with alpha=0.5 / max_iter=100000 / precompute=True	26896.10125670	-0.02708642
4	Linear Lasso	4	Lasso with alpha=0.2 / max_iter=100000 / precompute=True	26895.95726130	-0.02644169

In order to validate the model's predicted values, we randomly select 5 samples from the testing data and compare the predicted value with the actual one. The below code snipped and output shows an error value of less than 10%.

In [160]:
```
### Checking the robsutness and accuracy of the Lasso model from Trial #4
import random

print ('Row\t', 'Predicted\t\t', 'Actual\t', '\tError (%)')
for x in range(5):
    rowIndex = random.randint(1,60)
    X1 = dfCC_CC_X_test.iloc[rowIndex:rowIndex+1,1:]
    y1Actual = float(dfCC_CC_y_test.iloc[rowIndex:rowIndex+1,1:].eth_close_after_1day)
    y1Predict = regr.predict(X1)[0]
    print(rowIndex,'\t', y1Predict, '\t', y1Actual, '\t', abs((y1Actual-y1Predict)/y1Actual)*100)
```

```
Row      Predicted              Actual         Error (%)
43       1070.1371127129685     1037.36         3.159666144151363
50       1063.0087692439051     1048.58         1.3760294153908341
11       712.7761144421363      785.99          9.314862219349319
56       988.1472847918471      1026.19         3.707180464451311
45       1018.7533842553498     1049.09         2.89170764611712
```

Justification

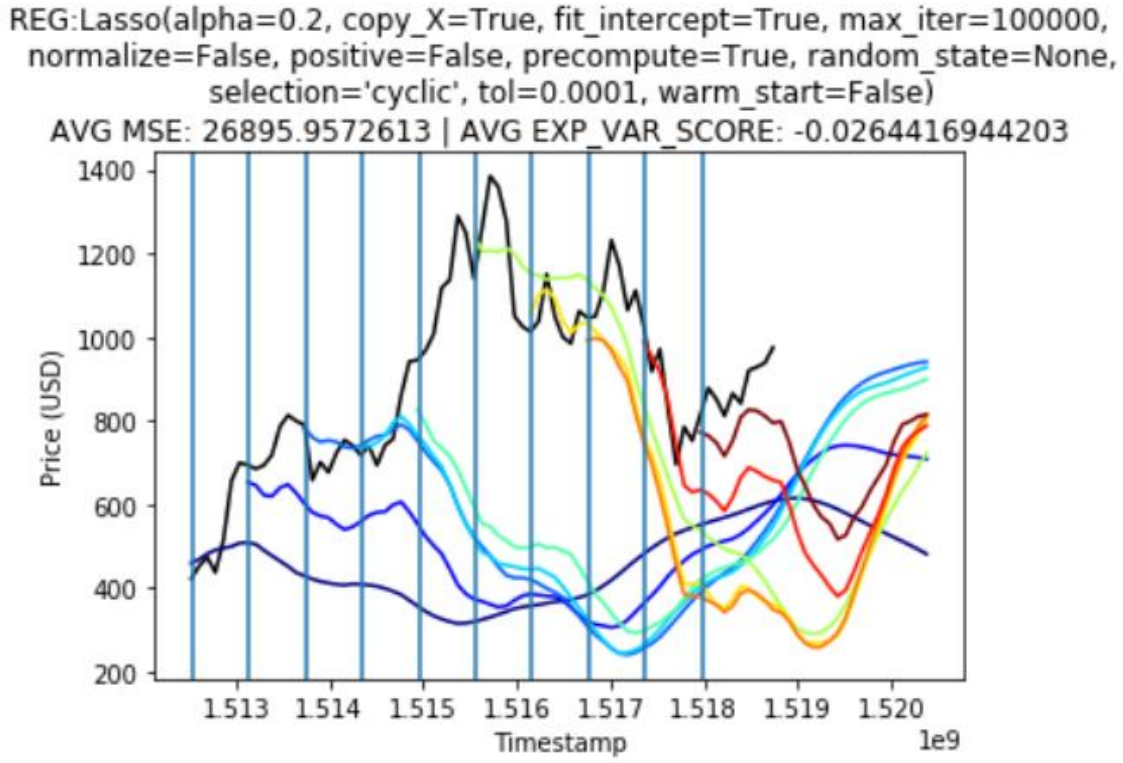
As can be seen in the previous sections and after optimizing the hyperparameters of the Lasso linear regression model, we can see that the regression model with the hyper parameters (Trial #4: Lasso with alpha=0.2 / max_iter=100000 / precompute=True) gave the best MSE value 26895.9572613 (Decrease of 0.1439954 from the original Lasso Linear MSE); however, this MSE value is still more than the original Simple linear regression (our benchmark model) MSE value of 26894.1575935 and therefore the simple linear regression model can be selected as the best optimized model.

V. Conclusion

Free-Form Visualization

As can be seen in the previous section, an important visualization thought this project implementation was showing the predicted cost of a particular altcoin. In the same graph, we managed to visualize the predicted price of an altcoin starting from a particular date. The x-axis in this graph represent the date in which the altcoin cost is calculated or predicted and the date format used is Unix timestamp format (The black graph represents the actual cost of the training data and the colored graphs represents the predicted cost starting from various

timestamps. A peculiar point to see here is the prediction of a steep decrease in the value of Ethereum altcoin after it increased to its highest peak).



Reflection

We can summarize the full project in the following steps:

- **Step 1 – Importing Data:** Importing major cryptocurrencies prices from the Internet
- **Step 2 – Data Pre-processing**: Adding columns representing the ML model features
- **Step 3 – Data Splitting:** Diving imported data between training and test data
- **Step 4 – Model Training:** Train ML Model using provided Data
- **Step 5 – Data Prediction:** Predict upcoming cryptocurrency cost using multiple ML models
- **Step 6 – ML Model Selection:** Selecting the ML Model with the best performance
- **Step 7 – ML Model Optimization:** Optimizing the ML model by tweaking the model's hyperparameters

It should be noted that the most time-consuming step in the whole project was the data pre-processing step as we had to add multiple features and calculate their values from the historically imported data. As for the most important step (Step 4 - Model Training), this step was relatively easy as the related ML models libraries are pre-defined in Python. One last remark in this analysis is that there was not a big difference in performance between the standard Linear regression algorithm and the other algorithms.

Improvement

After reading multiple articles about stock prediction, I can confidently say that there is a great possibility of improvement by employing the below techniques:

1. **Technique #1 - Using Deep Learning:** Instead of using classing machine learning algorithms, we can use deep learning and try to optimize its hyperparameters to increase performance. This type of supervised learning will require a huge dataset which is available but it also needs more CPU/GPU power that can be arranged.
2. **Technique #2 – Using Social Network Sentiment Analysis:** In addition to the historical cost of altcoins, we can add new features related to the sentiment analysis of social networks like twitter or Facebook. This will give us a direction on how the cryptocurrency market is moving.
3. **Technique #3 – High Frequency Prediction:** Instead of predicting the next day altcoin price, we can predict the expected cost for the next minute or even second. This proved to have a higher accuracy as external factors will not have huge impact during a very short period; however, high resources and accurate data are required.
4. **Technique #4 – Using Reinforcement Learning:** Instead of treating the problem as a classical supervised learning regression problem, we can build a bot that will use exploration and exploitation to take buy or sell decisions.