REPORT

{INTRODUCTION}

One of the most trendy topic, in this period, are crypto coins. In order to improve our know-how in this topic, we choose this for the Project.
The goal of this project is to predict the price of the Bitcoin (BTC) in Dollars (USD). We choose Bitcoin because it is the most important and representative crypto coin in this moment.
The information about Bitcoin - used in the project - comes from Bitstamp Exchange (i.e. one of the most popular Exchange in all the Crypto Sector): an Exchange is a Web Site that allows his users to buy Crypto coins using real money or others Crypto.
There are two methods to predict price development:

- **Sentimental Analysis**: This approach examines the the underlying forces of an economy. Factors like government regulations and media influence play a major role in deciding the cryptocurrency prices.
- **Technical Analysis**: This methodology is used for forecasting the direction of prices through the study of past market data. The historical data that has been collected for many years is used as a training set for prediction.

Our aspiration is to do technical analysis and we have asked for Crypto expert advice for to reach the goal.

{DATA EXTRACTION}

For Data Extraction we have used the CryptoWatch API that is a REST Web Service. This Web Service provides a long list of function which provides several informations about Crypto, Exchanges, Market Trades etc.
In order to extract the right data for our purposes, we have used exclusively the **OHLC** function.
This function takes in input coin, exchange and optionally:

- time interval from which starts the query;
- some candles types (i.e. the time-interval of the candle).

The output of OHLC function is a series of candles, regarding the coin and the exchange chosen, divided in types starting from a date chosen in the function call.

Some information about candle:

- in the Crypto Coins ambient for candle is intended a variation of a Crypto Price in a time interval;

- the reason of the name "candle" is given from the drawn object in the chart that reminds of a candle. It represents the price variation.

Usually, a candle is synthesized in 6 parameters (this is the CryptoWatch Representation, too):
- **CloseTime**: it is the UNIX timestamp for the end of candle time interval.
- **OpenPrice:** it is the starting price of the candle.
- **HighPrice:** it is the highest price in the time interval.
- **LowPrice:** It is the lowest price in the time interval.
- **Volume:** it is the amount of cryptocoins (BTC in our case) that has been traded during the time interval (a trade is a transaction of coins between two wallets in a specific exchange).
- **ClosePrice:** It is the coin price at the end of candle time interval.

For our project we have choosen the 12 hour candles because they are the best trade-of between quality of data for prediction and numbers of samples into the Dataset.

For Data Extraction we have built several classes starting from the **Dataset_Creator** class that allows to download (in a JSON file) the result of a Query to the CryptoWatch REST API.
This class uses other classes - built by us - in order to perform his goal like the **Extractor** class for retrieving the result of a Query and the **JSON_Saver** class to save the result of the query in a json file.

In summary, in this phase of the project we used **“Dataset_Creator”** class for downloading the 12-hour candles data starting from the year 2011.

For this phase it is coded:
- Extractor;
- Time_Stamp_Converter;
- Periods_Maker;
- JSON_Saver;
- Dataset_Creator.

{DATASET BUILDING}

In order to make a dataset from the previous downloaded data, we have used,again, the **“Dataset_Creator”** class.
In particular, this class has a method that allows to build a dataset - from a JSON file - and saves it in a **pkl** file, accessible in future.

Specifically, the building of dataset consists in the creation of 2 Matrix:
- **Observations Matrix,** this matrix contains all the useful data in order to predict the price of the Bitcoin at the end of the candle.
- **Labels Matrix,** this matrix (i.e. Column Vector) contains all the “ClosePrice” values for every candle coming from the Dataset.

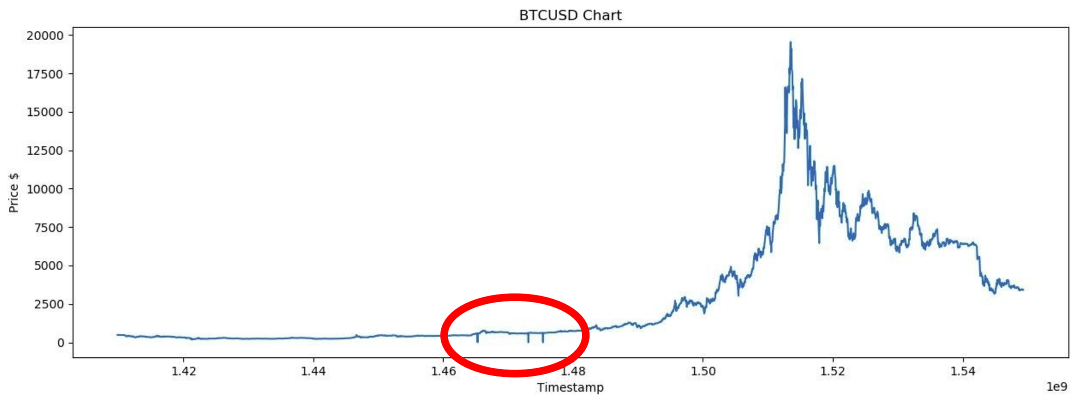
Moreover, we voluntarily remove a feature into the dataset because it is not documented from the authors of the CryptoWatch REST API.

For this section it is coded:

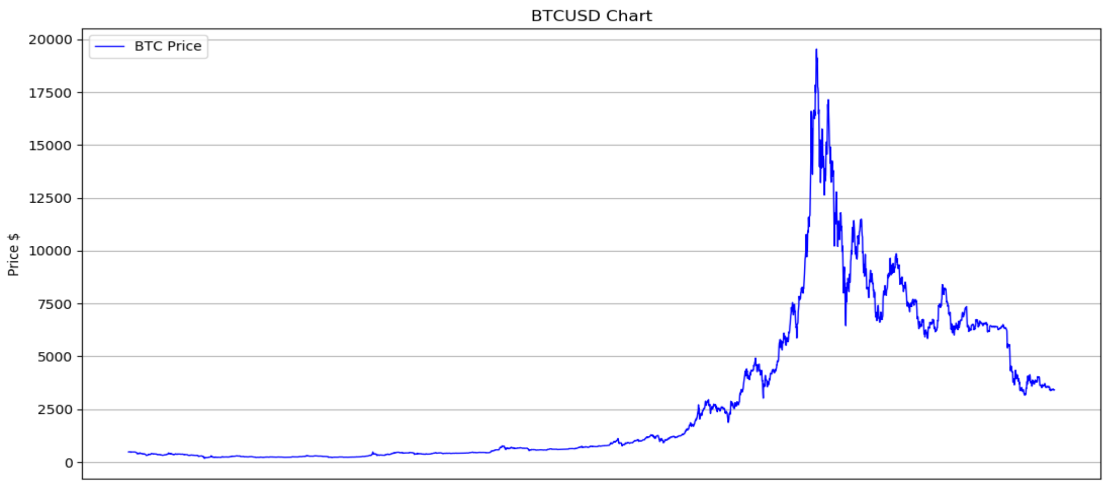
- Dataset_Creator;
- Dataset_Maker.

{DATA VISUALIZATION}

Firstly, we have plotted close time VS close price to check if our data are correct:



But we have realized that some candles are dirty because of API Cryptowatch: they have close price setted to 0.0.
So, we have created the new dataset removing this dirty candles and we have plotted again:



Descriptive statistic is a summary statistic that quantitatively describes or summarizes features of a collection of information. We have used several instruments to describe distribution shape, central tendencies and dispersion indeces computed in **Index** class.

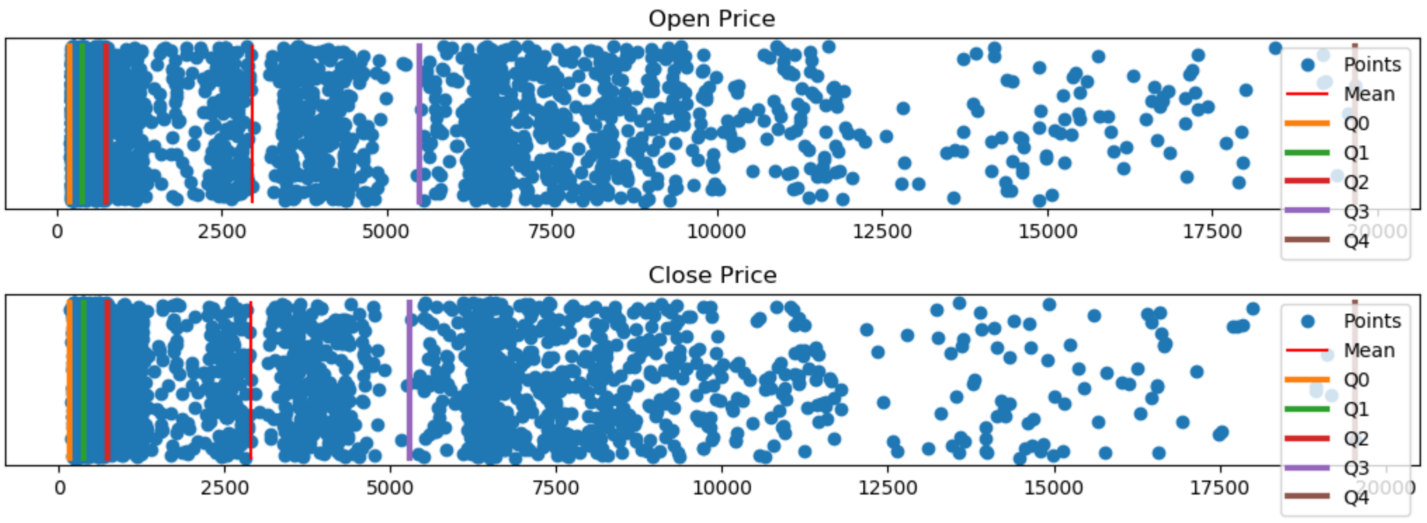
After extracting data, thye are converted in numpy array to perform computation; data are 12-hour candle from 01-01-2011, 00:00 to 04-02-2019, 01:00 previously saved in a .pkl file by **Dataset_Creator.** The data are Open Price’s vector and Close Price’s vector.

In order to show data, it is used dotplot: data are shown with dots whose positions are determined by own values and are “spreaded” along y axis for that random numbers from 0 and vector’s size are generated. On each dotplot significant points are located such as:

- mean;
- first quartile (Q0): corresponding to minimum;

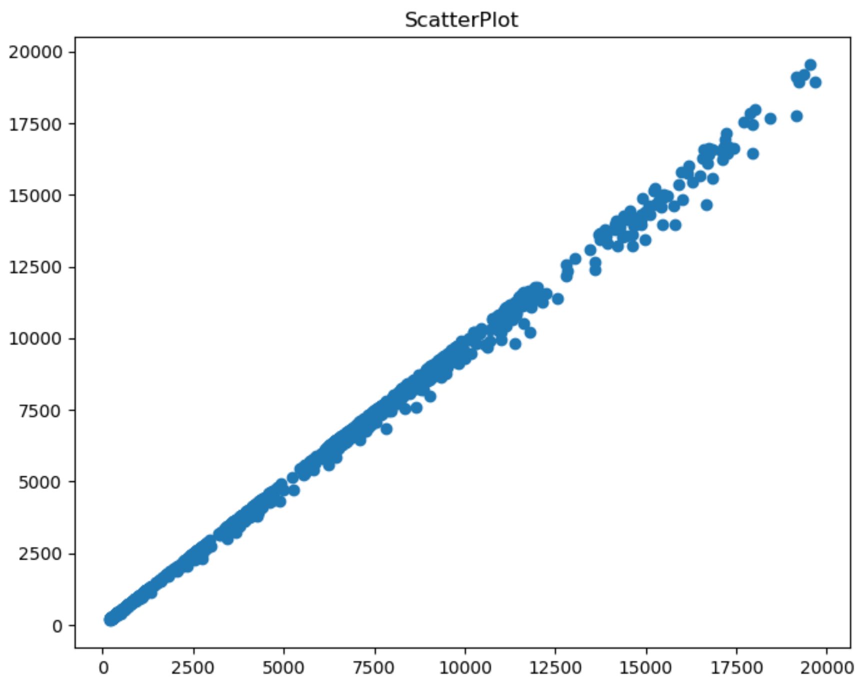
- second quartile (Q1): corresponding to ¼ data;
- third quartile (Q3): corresponding to median;
- fourth quartile (Q4): corresponding to ¾ data;
- fifth quartile (Q5): corresponding to maximum.

It can be observed that mean and median are in according to dotplot's density.



Easiest dispersion's measure is the range (i.e. differencing between maximum and minimum) but it is not very "expressive". Thus, variance and standard deviation are measured.

In order to measure correlation between data series, there are several instruments. Firstly, the scatter plot is shown: data are aligned along a straight line.



The covariance is computed through the covariance matrix which reports:

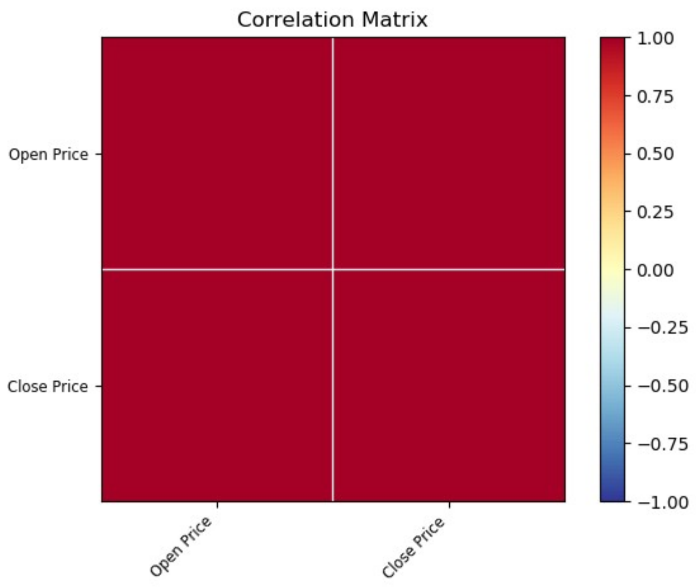
- Cov(X,X) and Cov(X,Y) on first row;

- Cov(Y,X) and Cov(YY) on second row.

The main diagonal contains the variance of two data series; the other diagonal contains the covariance: this value is positive and so the characters grow each other.
Moreover, these correlation coefficients are calculated:

- Pearson: its values is almost 1 and so implies an exact linear relationship;
- Spearman rank-order: its values is almost 1 and so implies an exact monotonic relationship;
- Kendall: its values is almost 1 and so implies strong agreement.

For each index it is calculated also the p-value that roughly indicates the probability of an uncorrelated system.
Finally, the matrix correlation is computed that reports correlation (Pearson index) for each pair of data; it is shown and it can be observed strong correlation between data series, again.



For this section it is coded:

- Index.py;
- Graph.py.

{LEARNING}

Firstly we have splitted the Dataset into Training-Set, Validation-Set and Test-Set, using our **Dataset_Creator** class.
Particularly, this split has been done as follows:

- 67% of Dataset for Training and Validation Set;
- 33% of Dataset for Test-Set.

We have splitted the Dataset in Training-Set and Test-Set because the Training-Set is useful for fitting the Algorithm, instead Test-Set is useful for evaluate the performance of algorithm with data never seen before.

We have introduced the Validation Test for Training reasons: in the Learning phase we have compared several models. Consequently, the Validation-Set has been needed to choose the best Model.

In order to improve the performance of the Algorithm, we have used a Variable Validation-Set using Cross-Validation techniques: specifically, we have chosen the K-Fold Validation (with k=5): doing this, we decrease the probability of choosing a unfortunately configuration.

In addition to this, we applied the PCA trasformation to the Dataset because it orders features based on the importance (this is very useful when we will plot the Hypothesis plot: we obtain a more significant graphics because we'll plot only the most important features), and deletes the non-independent features.

The last refining on the dataset is the Normalization: we have applied the Normalization to every feature of the dataset in order to increase the algorithm execution speed-up
For this task we used our **Cleaner** class.

In order to predict the price, we have drafted two models' list:

1. the list contains sixteen models (geometric loci are straight lines, parables and circumferences to approximate the data):

   [$\mathbf{x_1}$=CloseTime, $\mathbf{x_2}$=OpenPrice, $\mathbf{x_3}$=HighPrice, $\mathbf{x_4}$=LowPrice, $\mathbf{x_5}$=Volume]
   MODEL 1: $h\theta(x) = \theta_0 + \theta_1 x_1$
   MODEL 2: $h\theta(x) = \theta_0 + \theta_1 x_1^2$
   MODEL 3: $h\theta(x) = \theta_0 + \theta_1 x_1^3$
   MODEL 4: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$
   MODEL 5: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2$
   MODEL 6: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2$
   MODEL 7: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2$
   MODEL 8: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_5$
   MODEL 9: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2 + \theta_3 x_5$
   MODEL 10: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2 + \theta_3 x_5$
   MODEL 11: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2 + \theta_3 x_5^2$
   MODEL 12: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5$
   MODEL 13: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5$
   MODEL 14: $h\theta(x) = \theta_0 + \theta_1 x_1^2 + \theta_2 x_2^2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5$
   MODEL 15: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5^2$
   MODEL 16: $h\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2^2 + \theta_3 x_{3+} + \theta_4 x_{4+} + \theta_5 x_5$

2. the list is done with the power set of the features where for each model the geometric loci considered are straight line, parabola, circumference and those of grade 3.

In all model decision we have given emphasis on the CloseTime and OpenPrice features.

The training of Models uses our **Trainer** class which uses the **sklearn LinearRegression** class and the our **Dataset_Projector** class.

Our learning strategy is:

TEST NUMBER	MODEL LIST	PCA APPLICATION
TEST 1	**FULL LIST**	**NO**
TEST 2	**FULL LIST**	**YES**
TEST 3	**RESTRICTED LIST**	**NO**
TEST 4	**RESTRICTED LIST**	**YES**

We have done 4 Test (or Trainings), in order to try different Training parameters and discover the better model.
We report all the Final Error for every test as follows:

TEST NUMBER	FINAL ERROR
1	77.31282746660311
2	77.90236824669812
3	77.90236824669812
4	77.90236824669812

It needs to note that we have compared different Test (Training) for the Final Algorithm Error (we take the Test which has lowest Final Algorithm Error).
Instead, into a Training we have compared the Validation errors (we take the model that has lowest validation error) in order to choose the best model of a single Training.
The best Test has been Test Number 1 with the following Final error: **77.31282746660311.**

In Test number 1 it has been used Full Model List (list generated from the power set) and the PCA has not been applied.
The best model for Test 1 is the following:

MODEL 96: $h\theta(x) = \theta_0 + \theta_1 x_2 + \theta_2 x_3 + \theta_3 x_4$

This model has the following Training Errors:

- **Training Error:** 3.2867006968429857
- **Convalidation Error**: 1.4962860742694075

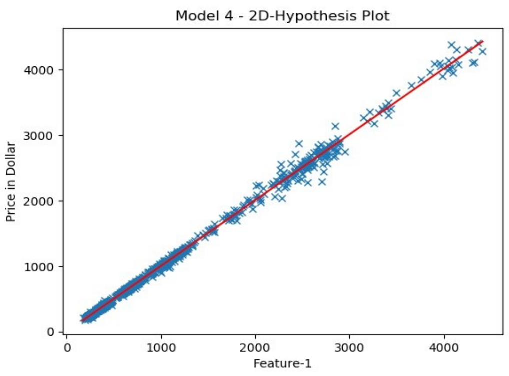
Below 20 Training Errors coming from Test 1 are listed :

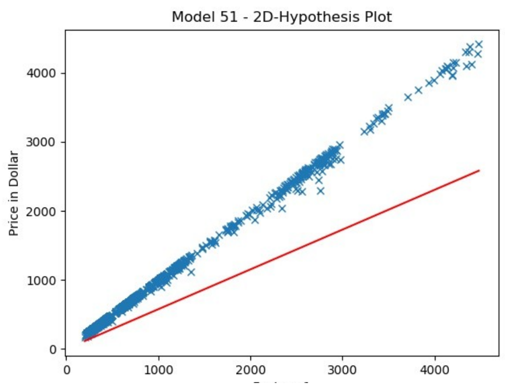
MODEL NUMBER	TRAIN ERROR	CONVALIDATION ERROR
96	3.2867006968429857	1.4962860742694075
97	3.3391998692474867	2.26552050770026
98	4.77929735684345	3.3024447818150424
99	5.424860217820947	3.805844615312863
100	3.3411887488364465	1.9301818086161135

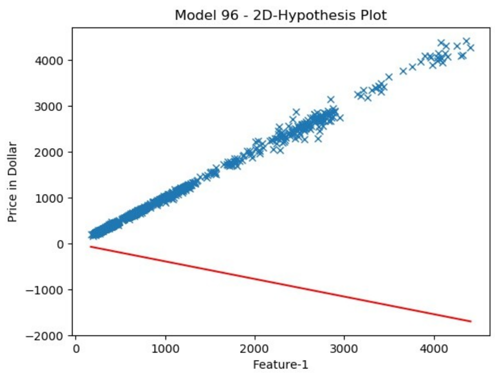
101	3.7318628373363487	3.127316359875666
102	3.693679387437472	2.9050483491462415
103	5.419213984996395	4.899390325128143
104	30.235211054631105	82.0036729710848
105	3.6816958299062224	2.9234087056957914
106	4.801254017800702	3.433953729970067
107	5.866006906988005	3.3805264922017644
108	5.49013196319411	5.186999020781832
109	29.641504940732883	76.67654600445456
110	4.837632618997467	3.435185290114451
111	3.2519610291748497	1.9625376507638335
112	5.944820932155511	3.2252004437741553
113	3.805549366537692	3.2513148289663665
114	29.648765496541753	77.20693553954804
115	4.8564925523358315	3.297689497787407
116	3.1225783122319233	1.5146803523267311

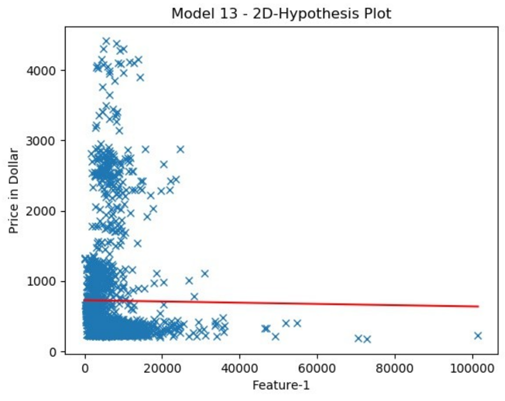
Below a subset of Test 1 **Hypothesis** plot and **Degree vs Error** Graphics are shown:

2D GRAPHICS

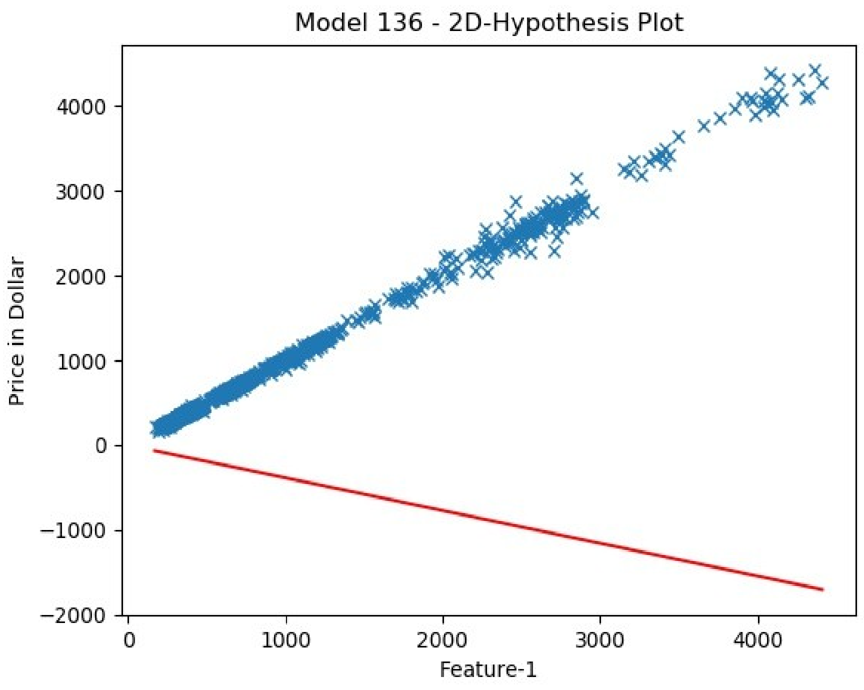




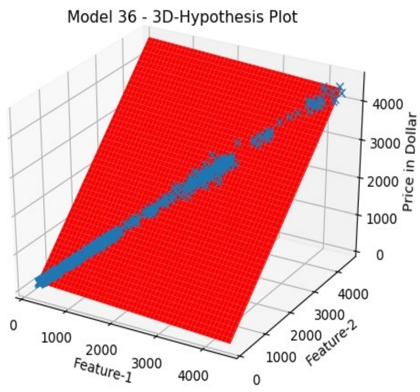


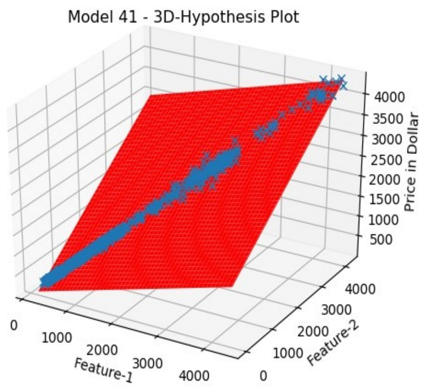


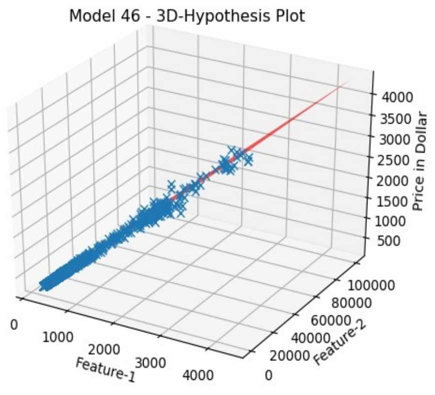
BEST MODEL 2D HYPOTHESIS GRAPHIC

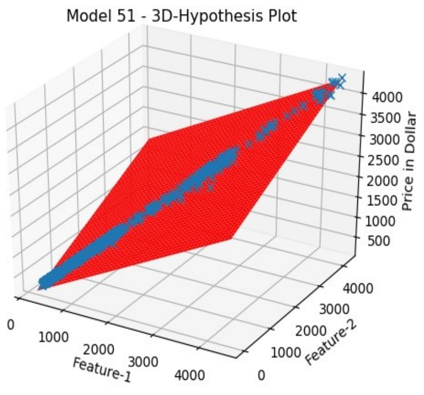


3D GRAPHICS

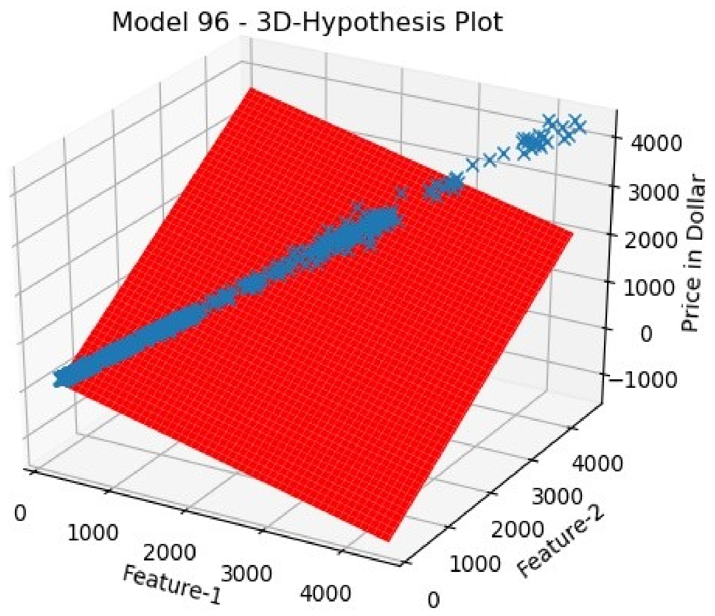




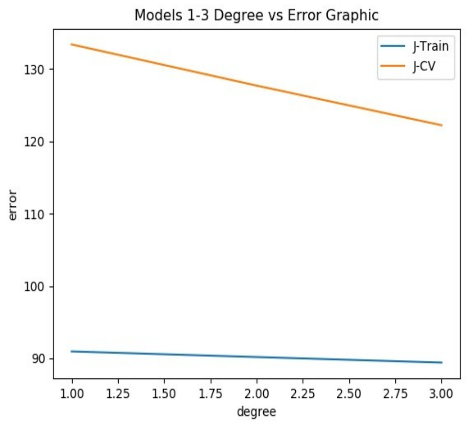


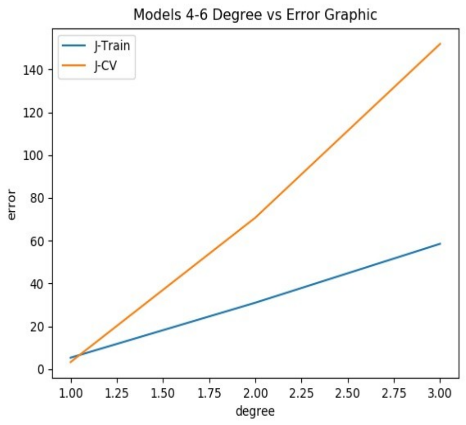


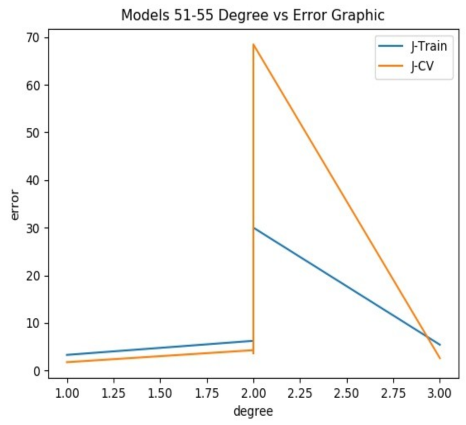
BEST MODEL 3D HYPOTHESIS GRAPHIC

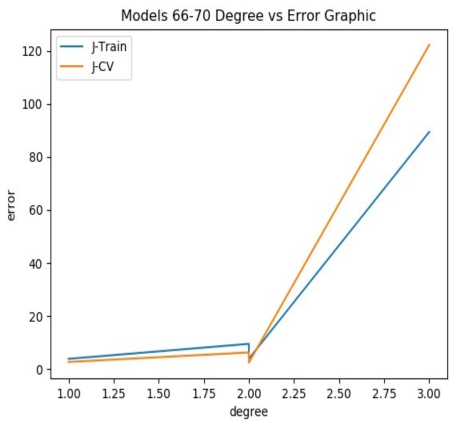


DEGREE vs ERROR GRAPHICS

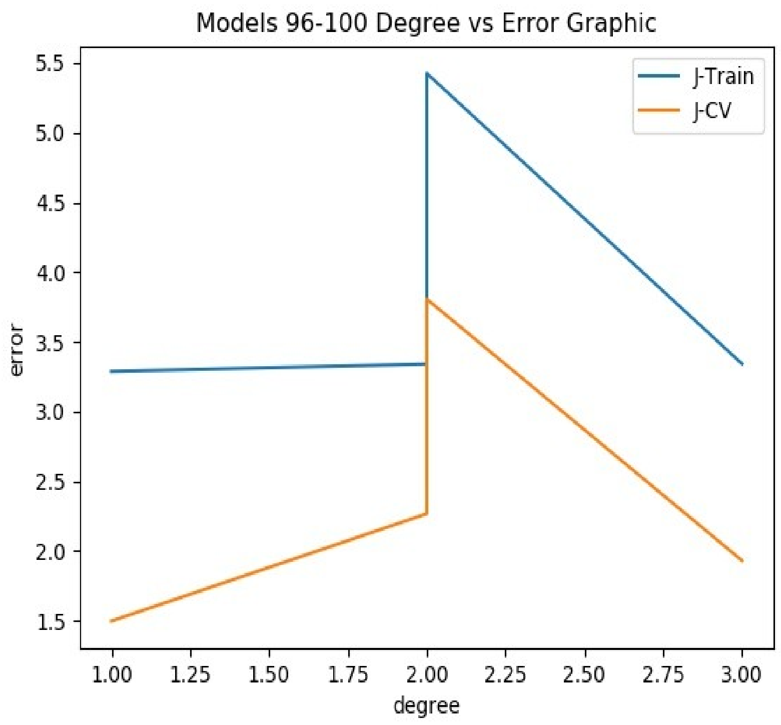








BEST MODEL DEGREE vs ERROR GRAPHIC:



At first look this graphics may be strange. If we take a look to, for example, the 2D Graphics, we can see that the best model (Model 96) seems to fit data worse than,for example, Model number 4. But we remember that, in this training, has not been applied the PCA.
Actually, the features have not been ordered for importance and consequently 2D/3D Hypothesis plot lose information value.
In addition to this, we remember that this Graphics are some approximations of the Learning trend: we can plot only up to 3 dimension (only 2 feature to put in the graphic) while we can have up to 5 features in a Model.

Taking a look to the **Degree vs Error** Graphic, we can draw the following conclusions:

- Increase Best Model (Model 96) Degree can only get worse the performance: the higher order models (coming from Model 96) worse the Convalidation Error.
- In the Graphic of Model 66-70 performance grow until to second degree; after this degree the training error and convalidation increase both.
- The Graphic Model 4-6 is a perfect example of over fitting (over-fitting is when you have excellent performance in the Training Set but your algorithm falls to generalizes).

It can be noted that we avoid to add in the report all the graphics and errors (more than 100) for layout reason.
Moreover, we could not plot the Graphics that have feature with big numeric interval (i.e. timestamp) because when we have attempted to plot these graphics, we have obtained a Memory Error coming from the Graphic Library (i.e. Matplotlib).
The cause of this error is that feature which have big numeric interval is hard to represent in computer and this, probably, overflows the Matplotlib resources.

For avoiding this error we have removed some training options from the learning program that cause this error: for example, <Graphic: yes, List: full, PCA: true>.
Moreover, Matplotlib has also a slightly slow execution time when it plots the 3D Graphics for the Models which has not the CloseTime feature, but don’t worry it can take at max 3 seconds. The reason of slow is amenable to the big numeric intervals of the features.

There is another experiment that we have tried in order to improve algorithm performance.
We have tried to apply the Normalization (Feature Scaling and Mean Normalization) to the Dataset before the Learning.
The result are very bad and strange: we have obtained a very very low Final Algorithm Error but the Bitcoin Predicted price was very very far from the real price.
After reflecting a lot, we have realized that the Normalization techniques are more suitable for the Gradient Descent Learning Algorithm (with these techniques the Algorithm can converge quickest). Moreover, we have discovered that the library which we have used to perform the prediction is based on a Learning Algorithm (Normal Equation) different by Gradient Descent.
So we have tried to avoid to perform normalization techniques on our Dataset and consequently the performance of our Algorithm became good.

In summary, the training consists in the following phases:

- Every Model has been trained using the **“fit”** method of LinearRegression class; this method has been applied to the Training-Set using the k-cross validation strategy.
  Of course, the Training-Set needs to be restricted in order to have a perfect match with the model that we want to train.
  This service was performed by the **Dataset_Projetor** class.
- For every Model, we have calculated the Training-Error and the Validation-Error (the error calculated on the Validation Set).
  The Error Function used is the MAE (Mean Absolute Error).
- Finally, we have chosen the Model with the lowest Validation-Errror.

With the goal of a future inference, we have used the **“save_the_best_estimator”** method of **Trainer** class for saving in a pkl file the best estimator (the corresponding LinearRegression object) and the best estimator features.

In this phase it is coded:
- Trainer, Dataset_Projector;
- Curve;
- Recorder_Degree;
- Model_Printer;
- Learn.py.

{PERFORMANCE EVALUATION}

For the performance evaluation we have used the MAE error function.
We have calculated the error using the **“mean_absolute_error”** on the Test-Set.
The result has became the final Error of our Algorithm.
The best fit model is MODEL 96 that has follow errors:
- **Training Error:** 3.2867006968429857

- **Convalidation Error**: 1.4962860742694075

The Test Number 1 has Final error as **77.31282746660311.**

In this phase it is coded:

- Learn.py

{INFERENCE}

It is the final phase of the project for which we have built quick command line program that allows:

- automatic download of last opened candle and automatic prediction of its Close Price;
- user’s input of an opened candle (only with some specific parameters chosen in the Training phase) in order to return in output a prediction of the ClosePrice of that candle.

In this phase it is coded:

- Cryptowolf.py

{CONCLUSION}

Our algorithm works discretely well but we know that it has great margins for improvement.

Infact, we have limited ourself to the course know-how but in future we could expand the project maybe replacing Linear Regression algorithm with a better Machine Learning Algorithm, for example Neural Networks based Algorithm.
A last idea for increase our project performance could be the building of new features: for example it could be a good idea add to every candle a feature based on Tweet sentimental analysis (i.e. using doc2vec algorithm).
Anyhow, this experience allows us to explore the theory of course as well as getting used to working in a team: this is been our very target.
We achieved it and we are very happy for this. :)

{AUTHORS}

Gabriele Marino
Rosario Scalia
Maria Ausilia Napoli Spatafora