Creating an AI-based trading bot for cryptocurrencies markets

Side personal project

Sébastien Cararo

Gap Year – November 2020

« I'd like to live as a poor man with lots of money"

Pablo Picasso (1881 – 1973) - Spanish painter

Table of contents

I – Abstract

I created a Bitcoin trading bot in Python using the Poloniex API : the program uses Deep Learning to estimate *bearish* and *bullish* trend probabilities, which are then used to define a set of decision conditions to decide whether we should buy, sell or just wait. The system was able to generate profit on backtested simulations and is currently being tested for validation in real-life trading on the *Poloniex* exchange. The backtested simulations also yielded profit on more recent data, and I was able to observe a significant positive correlation between the validation and test sets, which confirms even more the robustness and the accuracy of the results. The constructed pipeline is very interesting in the sense that it only needs a dataset of basic historical characteristics (asset value, volume exchanged, min and max value within candles), from which it computes predictive variables and identify profitable strategies. We could even think about applying the process to other markets (other cryptocurrencies, stock market, energy, raw materials, etc…) to create similar trading bots on other platforms.

II – Opening remarks

This report contains information and techniques concerning the creation of a full pipeline from scratch to deploy a cryptocurrency trading bot, based on Poloniex data and trading fees. The process is easily generalizable and further development could allow to apply the techniques to other financial markets such as stock markets, commodity markets, and any other similar field. The following report is furnished along with multiple result metrics, all contained in a directory called *Results,* which display all the results presented in these lines. For any information or question concerning this work, please contact me at sebcararo@hotmail.fr.

As this field is largely exploited by extremely competent people around the world, and that I do not pretend that I made any sufficiently significant breakthrough in this high-tech subject, I also decided to publish parts of the code on my github page at https://github.com/Seb943/Tbpolo so that anybody can have a look at the source codes.

Along this report I tried to explain clearly the notions that were introduced, in order to make this report intelligible towards a non-scientific audience, but I also tried to display some technical points that were crucial in the development of the project.

III – Introduction

As a 22-years old engineering student, I was always interested in pursuing my studies and my career towards analytics-based fields, especially financial sectors. During my first years of higher education I studied *stem* fields such as mathematics and physics and I eventually moved towards Artificial Intelligence and quantitative market analysis courses during my Erasmus semestrer at the University of Trento (Italy – Master I level of education). I then started a gap year, during which I started to show some interest towards real-life applications of AI in financial markets fields : I fully conducted from scratch a first project about the sports betting market (https://seb943.github.io/projects.html), then I discovered the blockchain technology through web articles and trading bots examples, which motivated me to build my own program.

Bitcoin is a cryptocurrency firstly presented in november 2008 by Satoshi Nakamoto – an unknown person or group of people-, and which was launched later on during the year 2009. It is currently the largest held cryptocurrency asset across the world and, with as the date of the 04th of November, 2020 a market cap of 257 billion dollars. Some may think it can revolutionize the monetary system, as from its decentralized nature it cannot be controled by one single group of people, hence in the future it could change the way we see financial control power.

It can also be traded in order to bet on the price evolutions over time. Trading bots are a useful way to make profit without being emotionally involved in the decision process, as all transactions are realised inside the algorithm program. Removing the emotional side from trading can be of great help, as emotions - especially during stressful situations - might lead traders to make biased decisions under pressure. On the other hand, trading bots are lacking a few information, such as sentiment analysis, special events and news announces.

Also, a trading bot is a great tool for creating passive incomes, as from the moment it is entirely built and fully operating, it is supposed to be making trades autonomously and hence not requiring additionnal time investment from the user. Furthermore, it is a good way for non-specialists – like me – to start trading. Poloniex is a cryptocurrency platform that allows users to trade over dozens of cryptocurrencies and that provides a *python* API aswell.

I decided to conduct multiple studies throughout this work, in order to explore these possiblities and to choose the best one to make profit.

Along this report I distinguish two types of study that I performed : a *bullish* and a *bearish* one. *Bullish* is an usual term in finance which designates the period in which the price is going up, whereas *bearish* designates the period in which the price is going down. Understanding *bullish* trading is quite easy, as we just seek to sell our stock at an higher price than what we bought it. *Bearish* trading makes you bet on the fall of the price, and as I do not master enough this field I will focus my study on *bullish* trading only, however I put an informative section at the end of the report to exploit the *bearish* results aswell.

Finally, I chose to develop my study towards *bullish* trading for Bitcoin on the Poloniex cryptocurrency exchange. This report summarizes the multiples steps that I developed by myself, from the documentation part to creating predictive features, transforming the data (PCA), building and training the Deep Learning models, comparing results and models, to finally identifying profitable strategies and assessing their relevancy on the long-run.

IV – Documentation research

As I didn't have any finance-related education neither had done any project in this field before, I first decided to documentate myself towards the subject in order to discover the commonly used techniques and to get a first idea of what I would like my pipeline to look like. I was also largely inspired by some open-source github repository, which were great examples for me :

- Gekko repository : A bitcoin trading bot written in node [2]
- Catalyst repository : An Algorithmic Trading Library for Crypto-Assets in Python [3]
- Cyrpto trader (*cyrpo,* not *crypto..*) : Trading automation on poloniex cryptocoin exchange [4]
- PoloBot : A simple trading bot for Poloniex [5]
- Trading bot repository : Stock Trading Bot using Deep Q-Learning [6]

These repositories were very useful, particularly concerning the architecture of the codes : after studying them out, I could have a clear idea of the typical structure of trading bots, but also have a precise idea of

the key points that needed to be spent time on. I noticed that multiple strategies, even being profitable on simulations, weren't yielding profit in real-life. The factors were diverse, but the main one was the lack of robustness of the trading strategies or the fact that the strategies were only theoretical trading strategies (*e.g.* crossing moving averages) that were overused by traders, and eventually not yielding profit anymore. Also, I could figure out that it could be a good idea to build a decision process *based on* the market data (*i.e.* fitting an optimization process for trading strategies based on historical and recent data), which I tried to conduct in this study.

I also leaned towards more academic works, in order to get a larger view of the usual topics in finance. The papers that most influenced me were :

- Marouane Anane, Frédéric Abergel. Optimal high frequency strategy in an omniscient order book. 2014. ffhal-01006401f [7]
- An agent strategy for automated stock market trading combining price and order book information. Silaghi, Robu [8]
- Predicting short-term Bitcoin price fluctuations from buy and sell orders. Guo & Antulov-Fantulin, 2018 [9]
- Stock Trading Bot Using Deep Reinforcement Learning. Akhil Raj Azhikodan, Anvitha G. K. Bhat and Mamatha V. Jadhav [10]

These papers were very interesting in terms of modelization and formalization, and even though I didn't have enough background in this field to fully understand every part, it was very interesting to get to know the current areas of study concerning finance. I also learned a lot about financial markets organization (especially order books) and deep learning applications in finance [10].

V – Pipeline presentation

The multiple steps which are developed along this document can be summarized with the help of the following diagram, representing the main functions involved in each part – with the real-life deployment being currently realised:

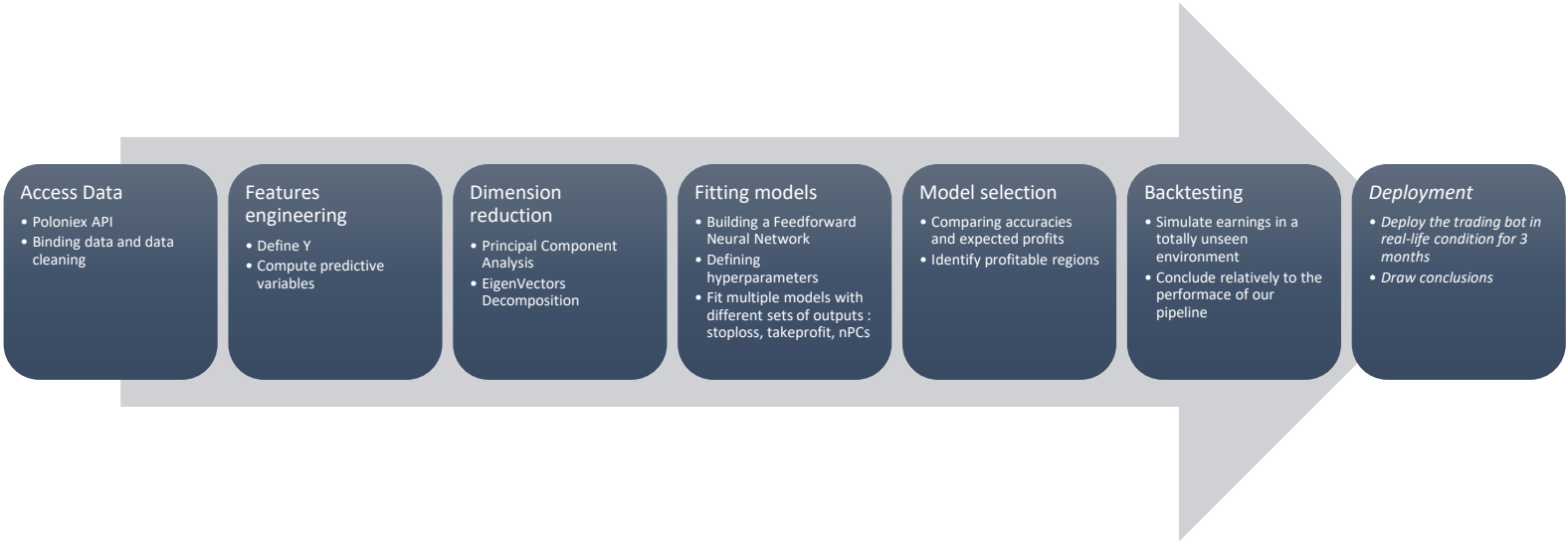


Figure 1 : Descriptive pipeline of the project

VI – Data source and software

The poloniex API in *python* ([11]) is a great tool to take buy and sell orders, but it also features multiples functions such as historical data requesting. The data is available throught different timesteps (in seconds : 300, 900, 1 800, 7 200, 14 400 and 86 400) from the year 2015. In order to obtain as precise as possible predictive models, especially for short-term variations such as 1%, I decided to focus on the shortest available period ($T = 300$ seconds). I then asked for the 5-min Bitcoin data from February 19th of 2015 until September 28th of 2020, containing the following information for each 5-minute timestep :

Field	Description
date	The UTC date for this candle in miliseconds since the Unix epoch.
high	The highest price for this asset within this candle.
low	The lowest price for this asset within this candle.
open	The price for this asset at the start of the candle.
close	The price for this asset at the end of the candle.
volume	The total amount of this asset transacted within this candle.
quoteVolume	The total amount of base currency transacted for this asset within this candle.
weightedAverage	The average price paid for this asset within this candle.

Figure 2 : Output for historical data request on Poloniex ([12] – Poloniex API documentation)

close	date	high	low	open	quoteVolume	volume	weightedAverage
7405.0	16/11/2017 08:35	7415.56705894	7360.00000007	7365.0	21.55844607	159209.20511175	7385.00375188
7399.00000001	16/11/2017 08:40	7415.56705884	7385.43573889	7415.0	27.49793162	203593.35868841	7403.95174087
7395.0	16/11/2017 08:45	7405.0	7395.0	7402.24401974	16.21509078	120005.59893429	7400.85890128
7405.0000001	16/11/2017 08:50	7410.00000004	7386.94453482	7395.0	7.80580921	57731.03404846	7395.90636861
7420.55037798	16/11/2017 08:55	7424.335108	7402.0	7408.9999999	34.31409057	254376.25602325	7413.1720176
7425.0	16/11/2017 09:00	7429.11735298	7410.00000009	7423.99999999	66.83573313	496078.05660959997	7422.34779776
7442.0	16/11/2017 09:05	7442.21324774	7421.86142151	7421.86142151	23.04951484	171420.83566639	7437.06914684
7446.99999997	16/11/2017 09:10	7449.0	7421.099420899999	7442.0	38.56054511	286883.06715657	7439.80839322
7438.0	16/11/2017 09:15	7450.0	7436.16022967	7446.99999997	57.40625173	427561.91204007	7448.00259823
7421.10681754	16/11/2017 09:20	7440.0	7411.19999903	7439.99999999	6.21939699999999	194754.94911932	7427.89581008
7430.99999988	16/11/2017 09:25	7439.99999985	7421.10681754	7421.10681754	23.47427649	174448.73187999	7431.48492582
7431.00000044	16/11/2017 09:30	7439.9999999	7430.0	7430.99999988	8.52377513	63370.67840717	7434.57886214
7437.47847159	16/11/2017 09:35	7443.2	7430.0	7430.2569002	2.75189469999999	169268.13912404	7439.73815613
7443.2	16/11/2017 09:40	7450.00000001	7435.79745242	7437.47847159	28.76639623	214173.24830107	7445.25823077
7450.0	16/11/2017 09:45	7450.00000011	7440.00000006	7443.2	10.27007034	76505.66749248	7449.38106163
7450.49650337	16/11/2017 09:50	7450.49650337	7441.0	7449.99999999	13.2621354	98789.23704975	7448.96911923
7457.28	16/11/2017 09:55	7457.28	7450.0	7450.49650337	15.25788605	113705.90816863	7452.27142187

Figure 3 : Overview of the data received after calling the API

The dataset was then splitted into three distinct datasets, which serve different purposes in the construction of the strategies (detailed in the upcoming parts) :

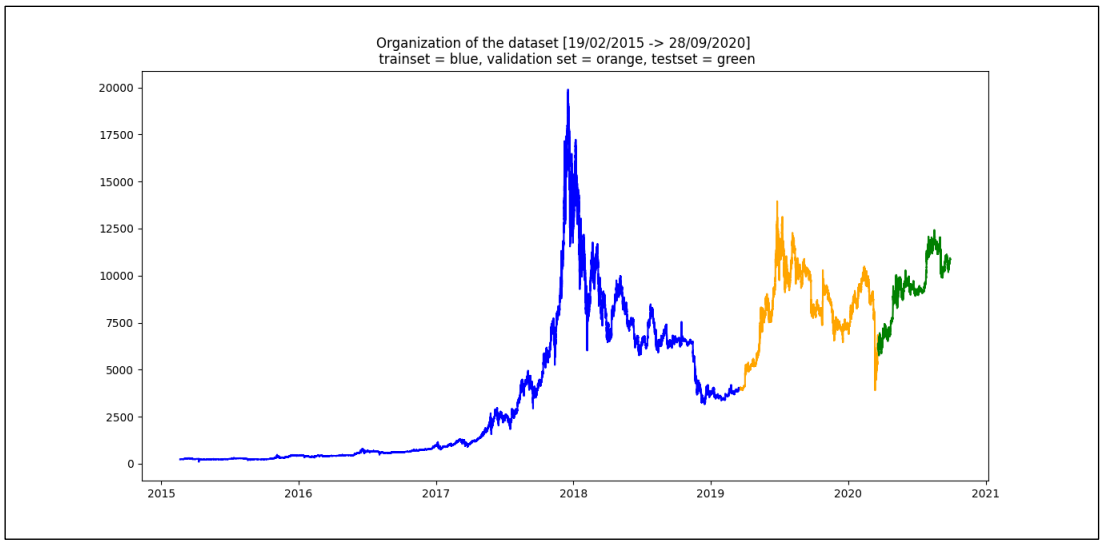


Figure 4 : Organization of the datasets into train/validation/test sets

The whole project was coded with Python 3.8.5 in the Pyzo IDE. I used a few usual *pip* packages, among which the following ones:

- *keras* 2.3.1 [13] and *tensorflow* 2.2.0 [14] for the deep learning architecture

- *matplotlib* 3.3.2 [15] for visualization tools

- *numpy* 1.18.5 [16] for calculations

- *pandas* 1.1.2 [17] for facilitated dataframe manipulation

- *poloniex* 0.0.17 [18] for data requesting

- *scikit-learn* 0.22.2.post1 [19] for PCA and linear regression used in the results part

- *pickle* 3.9.0 [20] to save and store deep learning models (*python* integrated tool)

- *scipy* 1.4.1 [21] for p-value computations in the results part

VII – Defining Y, computing predictive variables (X)

(a) Defining Y

As the trading process is generally binary (we either bet on an uptrend – *bullish* market – or on a downtrend – *bearish* market), I wanted to define a binary output such that the trading bot could know what is the predicted trend for the Bitcoin price evolution. In finance asset trading, it is common to define a couple *(stoploss, takeprofit)* when buying an asset in order to estimate the possible earnings or loss. It is generally a percentage of the initial price. For example, if we are betting on the Bitcoin going upwards (*bullish* strategy) with a stoploss of 1% and a takeprofit of 2% and that the price is 6100$ (21/03/2020 11:55 AM on Poloniex), the stoploss will be 0.99*6100 = 6039$ and the takeprofit will be 1.02*6100 = 6222$. The scheme will be like this one :

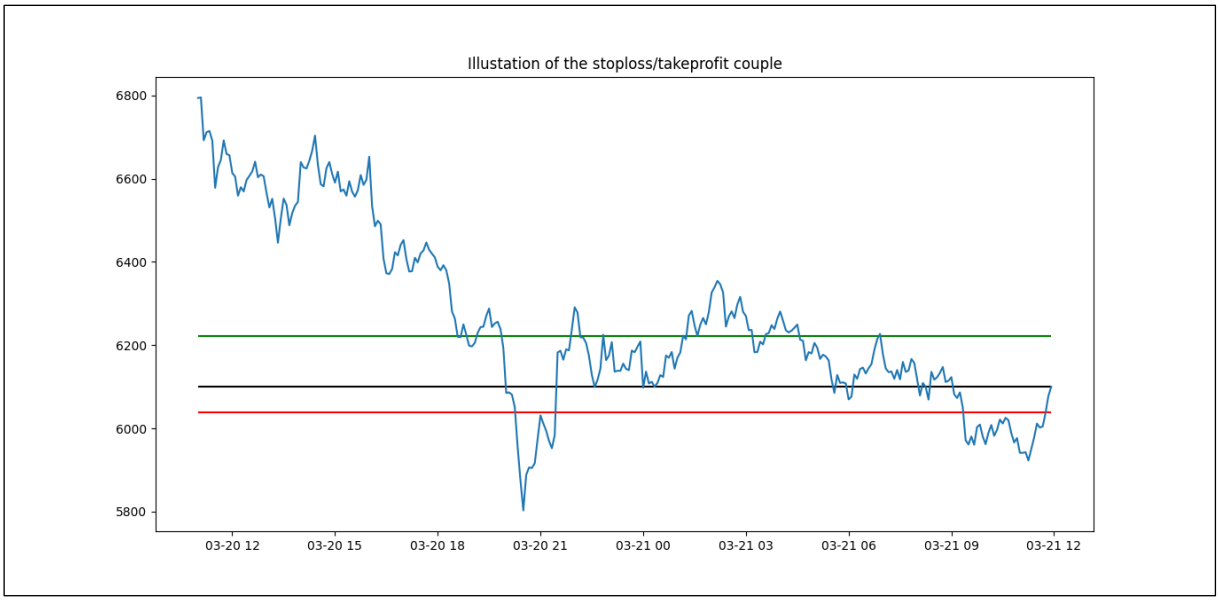


Figure 5 : Illustation of the stoploss (red) and takeprofit (green) notions when the price is located at the level of the black line

This being said, I first defined my *(stoploss, takeprofit)* couple as (0.01, 0.01), which means we would wait for the price to move 1% away from its initial price. The output *Y* would then have the value 1 if the price moves up to +1% before -1%, and 0 in the other case. With the same principle I could define Y for any set of *(stoploss, takeprofit)* couple by defining :

$$Y_{stoploss,takeprofit} = \begin{cases} 1 \; if \; takeprofit \; is \; reached \; before \; stoploss \\ 0 \; if \; stoploss \; if \; reached \; before \; takeprofit \\ -1 \; otherwise \end{cases}$$

I could then remove timesteps for which $Y_{stoploss,takeprofit}$ was equal to -1 – this corresponds to a situation where neither the takeprofit nor the stoploss were reached : the price is still contained in between the range (very recent observations).

I defined $Y_{stoploss,takeprofit}$ for *(stoploss, takeprofit)* $\in [0.01,0.02,0.03,0.04,0.05]^2$, obtaining 25 different outputs – hence 25 different possible trading strategies (this number of different strategies will evolve along the upcoming parts).

(b) Computing predictive variables (X)

In order to accurately predict the price trends, I had to create some finance-related variables that would describe as precisely as possible the signal evolution over time. To do this, the variables needed to reflect both short-term and long-term tendencies along with the volume and volatility evolutions. Therefore I documented myself towards such finance metrics, and I found out a few key variables that are commonly used for financial time-series analysis :

- Bodysize : This variable is computed from the difference between the closing price and the opening price. As our time step is T = 300 s, this variable contains the 5-minute price evolution in the period.
- Shadow size : The variable is computed from the difference between the maximum price and the minimum price in the period.
- Percent change : This is simply the relative evolution of the closing price in the last period of time, expressed as a percentage.
- *Minima* and *maxima* : I created the variables containing the minimum and maximum prices of the asset for different window sizes (3, 8, 21, 55, 144, 377, 987, 2584, 6765, 10946) in order to capture both short-term and long-term *extrema*. These window values are some of the Fibonacci

numbers, which are known for also modelizing buying and selling trends dynamics in financial markets [22].

- Volume and quote volume : I computed the rolling mean of the volume (total amount exchanged for the currency over a period of time) for the same different values of windows, expressed either in BTC units (*Volume*) or in USDT equivalent volume (*quoteVolume*).
- Moving averages [23] : I computed simple moving averages for several window sizes. A simple moving average for a window size *n* is computed by calculating the mean value of the signal over the last *n* observations. This type of average is widely spread in finance, as we can then capture different information by changing the window size. I chose to use simple moving averages, even though exponential moving averages are also very often used.
- Bollinger bands[24] : This tool is also widely used in market finance fields, it is mostly used to capture the volatility of the market. The Bollinger Bands are actually two signals, derived from the asset price: an upper band and a lower band. One can interprete their values in mutliple ways, especially according to the position of the asset price relatively to the bands. If the price moves towards the upper band then the asset is likely to be overbought (and *vice versa*). One can also be interested in the spread between the two bands, as a large spread may indicate a high volatility in the market. I computed the two bands, plus also the difference between the bands and eventually the relative difference between the asset price and the bottom band. I computed these variables for the very same different window sizes precedently described. The bollinger bands were only computed with the largest window size (10 946).

$$\text{BOLU} = \text{MA}(\text{TP}, n) + m * \sigma[\text{TP}, n]$$
$$\text{BOLD} = \text{MA}(\text{TP}, n) - m * \sigma[\text{TP}, n]$$
where:
$\text{BOLU} = \text{Upper Bollinger Band}$
$\text{BOLD} = \text{Lower Bollinger Band}$
$\text{MA} = \text{Moving average}$
$\text{TP (typical price)} = (\text{High} + \text{Low} + \text{Close}) \div 3$
$n = \text{Number of days in smoothing period (typically 20)}$
$m = \text{Number of standard deviations (typically 2)}$
$\sigma[\text{TP}, n] = \text{Standard Deviation over last } n \text{ periods of TP}$

- MACD [25] : The Moving Average Convergence Divergence (MACD) is the result of the substraction of a fast moving average by a slower one. It is recognized as being a good indicator of oversold/overbought assets. Usually we take the 26-periods and the 12-periods exponential moving average to compute it, however in my pipeline I computed MACDs columns from *window* and *int(window/3)*, for window in [3, 8, 21, 55, 144, 377, 987, 2584, 6765, 10946].
- RSI [26] : The Relative Strentgh Index (RSI), first introduced in 1978 (New Concepts in Technical Trading Systems, J. Welles Wilder Jr., [27]) , is one of the most used momentum indicator. It is computed from the following formula :

$$RSI = 100 - \frac{100}{1 + RS}$$

With *RS* being computed in the following way :

$$RS = \frac{\text{SMMA}(U, n)}{\text{SMMA}(D, n)}$$

where *SMMA(U, n)* is the smooth moving average of window size *n* computed only for up movements and *SMMA(D, n)* is the smooth moving average of window size *n* computed only for down movements. The value of RSI is then comprised between 0 and 100, with 0 meaning the asset is oversold and 100 in the case of an overbought asset. I also computed a set of RSIs variables, with the same different window sizes used during the MACD calculation (Fibonacci numbers). As there were no great interest into coding this function, I directly took the code from a stackoverflow post [28].

- Modulo 10, 50, 100, 500, 1000 : these variables help identify pschological support and resistance levels.

- Support and resistance trendlines : thanks to the *trendy* [29] package in Python, I was able to compute support and resistance levels for the signal in the form of linear lines (one above the signal and one below the signal). I used the slope from each line and used it as input for my model, plus I also used the expected value according to each line. I also used the difference between these two expected values from support/resistance line and the actual value, and the difference as a percentage of the price.

Figure 6 : Applying the segtrends function from the trendy package onto the bitcoin price at date = 27/12/2017 with a window range of 55 timesteps.

```python
def compute_variables1(df):
    df['bodysize'] = df['close'] - df['open']
    df['shadowsize'] = df['high'] - df['low']
    df['percentChange'] = df['close'].pct_change()
    for window in [3, 8, 21, 55, 144, 377, 987, 2584, 6765, 10946]:
        print(window)
        df = sma(df, window, targetcol = 'close', colname = 'slow_sma_{}'.format(window))
        df = sma(df, int(window//3), targetcol = 'close', colname = 'fast_sma_{}'.format(window))
        df = bbands(df, window)
        #df = ema(df, window, colname='ema_{}'.format(window))
        df = macd(df, fastcol='fast_sma_{}'.format(window), slowcol='slow_sma_{}'.format(window), colname='macd_{}'.format(window))
        df = rsi(df, window, colname = 'rsi_{}'.format(window))
        df["Min_{}".format(window)] = df["low"].rolling(window).min()
        df["Max_{}".format(window)] = df["high"].rolling(window).max()
        df["volume_{}".format(window)] = df["volume"].rolling(window).mean()
        df["quoteVolume_{}".format(window)] = df["quoteVolume"].rolling(window).mean()
        df["Slope_uptrend_{}".format(window)] = 0
        df["Slope_downtrend_{}".format(window)] = 0
        df["Upslope_diff_{}".format(window)] = 0
        df["Downslope_diff_{}".format(window)] = 0
        df["Upslope_pct_{}".format(window)] = 0
        df["Downslope_pct_{}".format(window)] = 0
        if (window > 3) & (window < 3000):
            for line in range(6*window + 1, len(df["close"])):
                if line % 10000 == 0:
                    print(line, "/", len(df["date"]),'-', window)
                a = segtrends(df["close"][line - 6*window:line], segments=2, charts=False)
                df["Slope_uptrend_{}".format(window)].iloc[line] = (a[1][1] - a[1][0]) / (a[0][1] - a[0][0])
                df["Slope_downtrend_{}".format(window)].iloc[line] = (a[3][1] - a[3][0]) / (a[2][1] - a[2][0])
                df["Upslope_diff_{}".format(window)] = a[4][-1] - df["close"].iloc[line]
                df["Downslope_diff_{}".format(window)] = a[5][-1] - df["close"].iloc[line]
                df["Upslope_pct_{}".format(window)] = (a[4][-1] - df["close"].iloc[line]) / df["close"].iloc[line]
                df["Downslope_pct_{}".format(window)] = (a[5][-1] - df["close"].iloc[line]) / df["close"].iloc[line]
    # (a) Add modulo 10, 100, 1000, 500, 50
    df["Modulo_10"] = df["close"].copy() % 10
    df["Modulo_100"] = df["close"].copy() % 100
    df["Modulo_1000"] = df["close"].copy() % 1000
    df["Modulo_500"] = df["close"].copy() % 500
    df["Modulo_50"] = df["close"].copy() % 50
    # (b) Add weekday and day of the month
    df["WeekDay"] = pd.to_datetime(df["date"]).dt.weekday
    df["Day"] = pd.to_datetime(df["date"]).dt.day
    #df.dropna(inplace=True)
    return(df)
```

Figure 7 : Python code screenshot for computing variables (44 lines)

VIII – Exploratory Data Analysis

After having computed all predictive variables, we can then have a first look at their shape in order to familiarize with the dataset. First we can display a few basic figures concerning the period of interest:

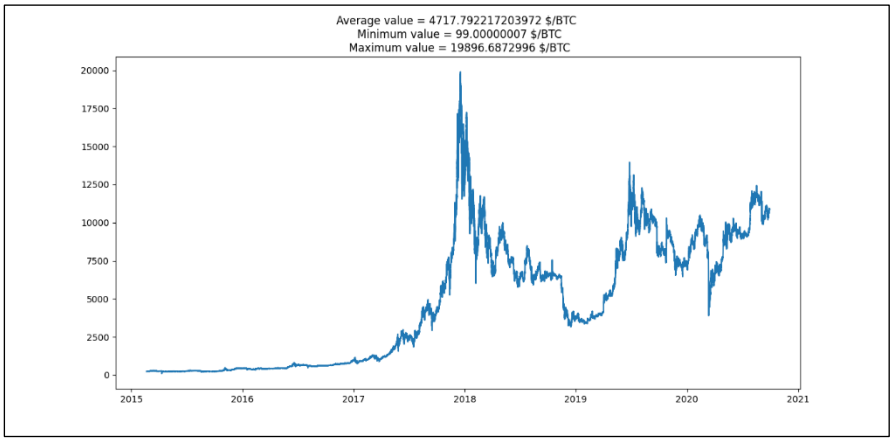
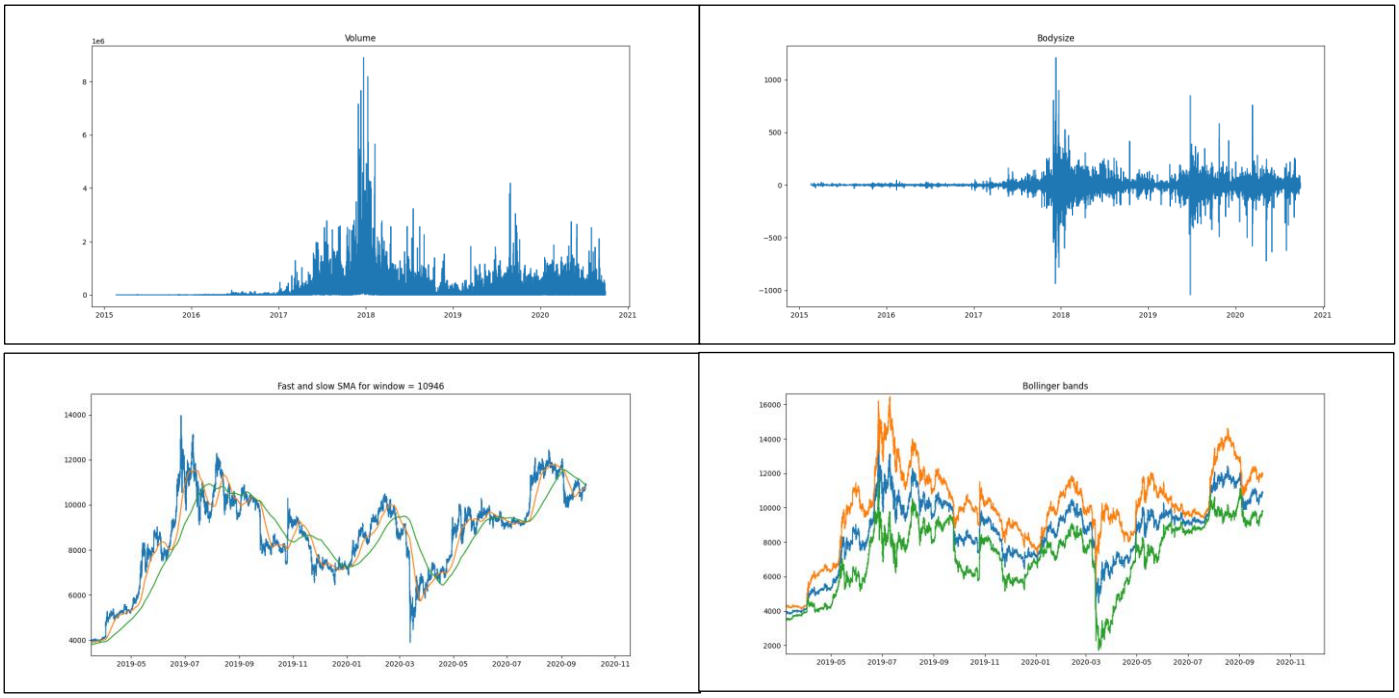


Figure 8 : Bitcoin price shape over our period of interest

Out of the 589 651 observations (time period T = 5 minutes), there were 319 557 observations (54.19%) where the price was higher than 24 hours before, and 266 922 observations (45.26%) where the price was lower than 24 hours before. The general trend is *bullish*, which we can easily infer from the plots. The Bitcoin price went from 225$ on the 19th of February in 2015 to 10 870$ on the 28th of September in 2020, which represents a 4 731 % increase over the 5 years time lap.

After having displayed the general trend, we can then have a look at the predictive variables that we implemented :

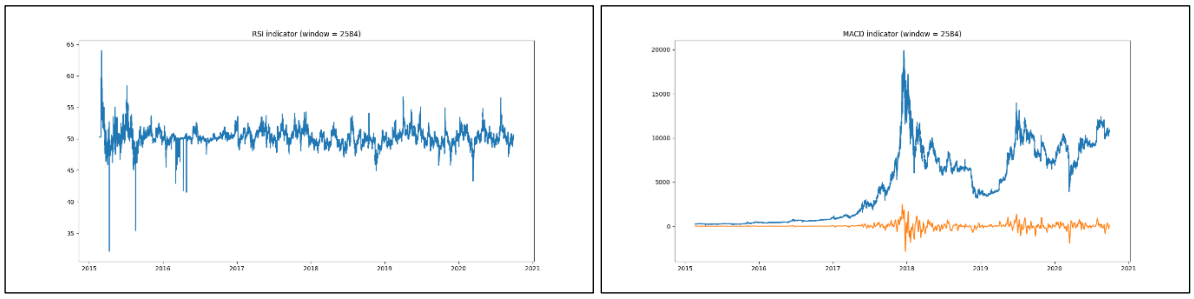


Figures 9 to 12 : Basic variables diplayed : volume, bodysize, moving averages (zoomed on the period May 2019-September 2020), Bollinger bands (also zoomed on the period May 2019-September 2020)

- ✓ Commentary : From the volume and bodysize plots we can distinguish at least two atypical spikes, concerning the months of December 2017 and May 2019. These periods illustrate the apogea of the last two *bull runs* [30], which are periods where the price was increasing very fast. In the meanwhile, the exchanged volumes were also very high, especially during the 2017 run, where the Bitcoin reached a new audience thanks to media reporting the incredible increasing trend. These bull runs are directly linked to a phenomenon called *halving* [31] : in synthesis, the Bitcoin network requires computing power to assess the many transactions made with the currency, which is supplied by individual people but also professionnal entities who provide their computing power in exchange of a reward being given for each block of transaction being validated. As the number of Bitcoin available was fixed at the creation, it was also decided that every time a new milestone of Bitcoins was being discovered, the reward would be splitted in half : this is the *halving* phenomenon – which is the main innovation of the Bitcoin technology. This *halving* results - with a few months delay - in an inflation, which was typically observed during this December of 2017 month. The bodysize plot shows that the fluctuations were also very high at these times.

  The moving averages plot illustrates the difference between the slow-MA and the fast-MA : the fast curve is following more accurately the Bitcoin price, whereas the slow-MA displays longer trends. The Bollinger bands plot displays the two bands : a lower curve and an upper one. As presented earlier, one usual way to interpret such plot is to have a look at whether or not the bands are close to the Bitcoin price : if the curves are far away, then it is common to say we are in a highly volatile period.

We can also have a look at the RSI and MACD plots, for visualization purposes :



Figures 13 and 14 : RSI and MACD

From the previous plots, we can also observe that many of the variables are highly correlated : this confirms a logic intuition, as all of these variables are derived from the same Bitcoin signal. To illustrate this impression, we can have a look at the Pearson correlation matrix [32] :

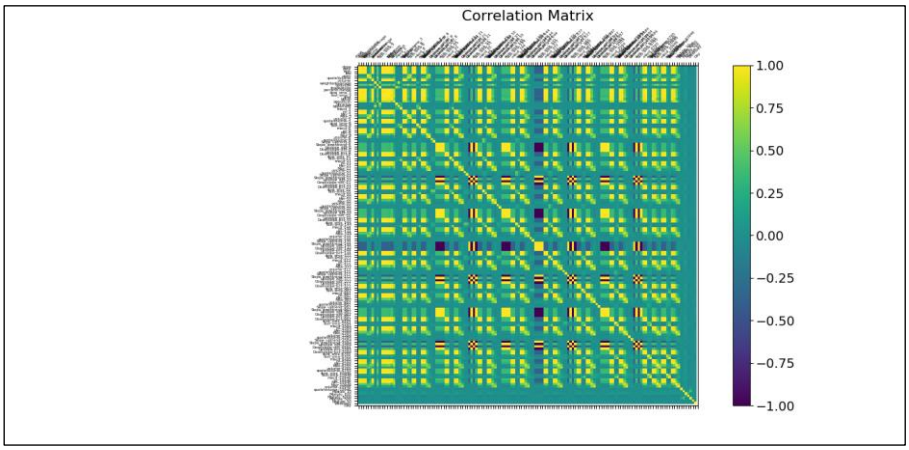
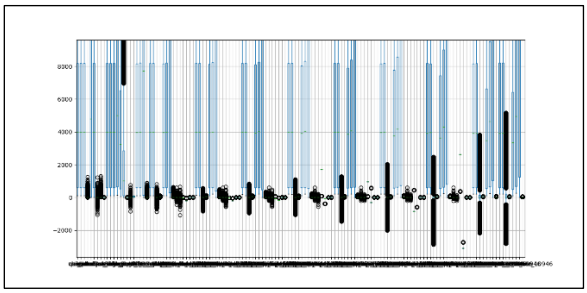
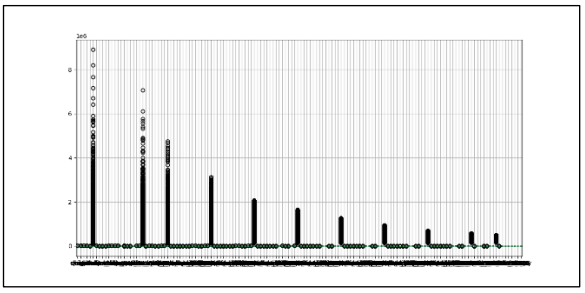


Figure 15 : Correlation matrix (Pearson method)

- ✓ Commentary : We can observe that many of the coefficients are either highly positively or highly negatively correlated, which *might* indicates that the same information is contained in these variables. To deal with such repetitiveness, we will apply a process called Principal Components Analysis (PCA) in section IX.

Finally, we can also have a look at the order of magnitude of each variable : as predictive models – such as Neural Networks – usually deal better with variables within the same order of magnitude, we better have a look at it in order to decide whether or not we might apply normalization to our set of variables.





Figures 16 and 17 : Boxplot of the variables (with and without zoom applied)

- ✓ Commentary : We can notice that the spectrum of values taken by the variables is very wide : some indicator variables are comprised between 0 and 1, whereas some variables are about 10^6 (*cf* left diagram *y*-axis scale), which justifies the use of normalization before fitting our models.

As a conclusion of this exploratory analysis, we can definitely confirm the use of normalization to deal with the differences of scale, but also the use of PCA before fitting the models in order to reduce variance in the fitting process, as the newly created Principal Components will be orthogonal.

IX – Dimension reduction using Principal Component Analysis (EigenVectors Decomposition)

Dimension reduction using Principal Component Analysis (*PCA*) is a widely used technique in AI to enhance performances of Machine Learning and Deep Learning models, by applying a transformation to the input data so that the information can be concatenated in less components that the raw variables. Actually, with this step we decompose the data along a set of optimized orthogonal directions. In this way, we can capture the necessary information in way less vectors that it would orignally be, and it also allows us to let the noise apart. The final goal is to reduce variance in the prediction process, by limiting the number of predictors and optimizing the information contained in each one.

To do this, the program seeks directions which maximize the variance along them and then projects the original variables onto them. This creates a set of components which are ordered by range of total variance – which is supposedly correlated with the importance of any direction : if there is a lot of variance along a direction then the value of the vector might be proheminent in the prediction process.

We pay attention to applying to PCA process to the train set only : by doing this, we can really assess the performances on validation sets and test sets, because then these two sets will be totally new for the predictors (otherwise we are in the case of a non-causal study).

```
df = pd.read_csv("./Preprocessed/Preprocessed_{}-2020-03-21_{}.csv".format(pair, period))
df = df.drop(df.columns[0], axis = 1).dropna()

trainset = df[df['date'] < '2019-03-21 12:00:00']

scale_fct = StandardScaler()
scale_fct.fit(trainset.drop('date', 1))
pk.dump(scale_fct, open('./Models/Scaler/scaler_{}_{}.pkl'.format(pair,period),'wb'))


pca = PCA(n_components=162)
pca.fit(scale_fct.transform(trainset.drop('date', 1)))
pk.dump(pca, open('./Models/PCA/pca_{}_{}.pkl'.format(pair,period),"wb"))


pca_scaler = StandardScaler()
pca_scaler.fit(pca.transform(scale_fct.transform(trainset.drop('date', 1))))
pk.dump(pca_scaler, open('./Models/PCA_scaler/pca_scaler_{}_{}.pkl'.format(pair,period),'wb'))
```

Figure 18 : PCA implementation in python : first we scale the variables, then we fit the PCA model, and finally we scale the PCs (it accelerates with the fitting process). All three scalers are saved using the pickle tool in python.

X – Machine learning

As presented earlier, the original dataset was splitted into three distinct datasets : train set, validation set and testset. The purpose there is first to identify possible profitable strategies thanks to the validation set, and then check on the testset that these strategies are effectively profitable when confronted to totally new data. That is why I will also study the correlation between every strategy's performance on the validation set and on the test set. With this process we can ensure that the strategies are robust and not the result on luck on the validation set.

(a) Model description

I chose to build a feedforward neural network, composed of 4 hidden layers and I used the sigmoid function [33] as activation function. The models were fitted using the Adam optimizer [34], the loss function was the binary cross entropy [35] and the metric was set to accuracy. The batch size was 500 samples (the trainset contains ~335 000 samples) and the number of epochs as 75 for each fitting process. I also fitted multiple models, with different number of Principal Components (10, 25, 40, 65, 80, 95, 110). To summarize with the previous parts, I recall that there were 25 differents possible outputs (depending on the values of *stoploss* and *takeprofit*), plus as we just explained I fitted 7 different models for each type of output : in the end there were 25*7 = 175 models fitted.

```
print(nPCs, stoploss, takeprofit)
X = trainset.drop('result', 1).drop('end',1).drop('date',1).drop('close',1).iloc[:, :nPCs]
y = trainset["result"]

# Build model and train it
classifier = Sequential()
#First Hidden Layer
classifier.add(Dense(32, activation='relu', kernel_initializer='random_normal', input_dim=nPCs))
#Second, third and fourth  hidden Layers
classifier.add(Dense(32, activation='relu', kernel_initializer='random_normal'))
classifier.add(Dense(16, activation='relu', kernel_initializer='random_normal'))
classifier.add(Dense(16, activation='relu', kernel_initializer='random_normal'))

#Output Layer
classifier.add(Dense(1, activation='sigmoid', kernel_initializer='random_normal'))
#Compiling the neural network
classifier.compile(optimizer ='adam',loss='binary_crossentropy', metrics =['accuracy'])
#Fitting the data to the training dataset
classifier.fit(X,y, batch_size=500, epochs=75, verbose =1)
```

```
>>> print(clf.summary())
Model: "sequential_10"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense_46 (Dense)             (None, 32)                352
_________________________________________________________________
dense_47 (Dense)             (None, 32)                1056
_________________________________________________________________
dense_48 (Dense)             (None, 16)                528
_________________________________________________________________
dense_49 (Dense)             (None, 16)                272
_________________________________________________________________
dense_50 (Dense)             (None, 1)                 17
=================================================================
Total params: 2,225
Trainable params: 2,225
Non-trainable params: 0
_________________________________________________________________
None
```

Figures 18 and 19 : Python script for fitting the model – Network architecture as contained in the python file

The neural network architecture has then the following shape :

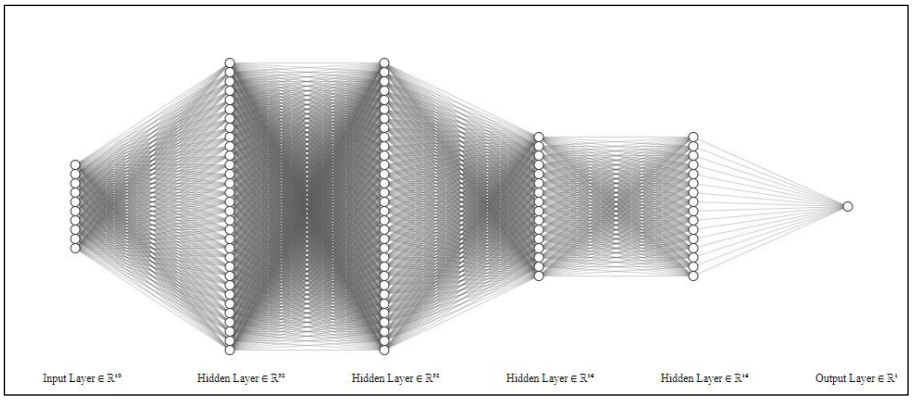


Figure 20 : Neural Network architecture (software used for modelling [36])

NB : The input layer dimension depends on the number of PCs we take for regression (10 in the illustration).

(b) Results

Now that we have created the multiple models with different sets of [stoploss, takeprofit, nPCs],we need to select the best models and to create trading strategies from these models. To do this, we will have a look at the estimated probability of the price going up (noted *proba1*) and study the regions in which we are able to identify profitable trading strategies. These steps will be described in the very next sections, where we first compute the predictions over the validation set, along with multiple necessary metrics (part I), then we create backtesting tools by summarizing the performance as a function of the section of *proba1* which we consider (part II), and finally we compare the performances of the various strategies between the validation set and the testset, to make sure the profitable strategies in the validation set are also yielding profit on the testset (robustness assessment – part III). For part I and II we consider only the validation set, then in part III we also consider the test set : there we assess the strategies on totally unseen data.

(I) Create backtesting tools

The first step is to create – for each set of [stoploss, takeprofit, nPCs] – a table that contains, for each timestep, the relevant information :

- the predicted class (0 or 1) : *prediction*
- the probabilities of 0 and 1 : *proba0* and *proba1*
- the closing price : *close*

- the ground truth result (0 or 1) : *result*
- the duration it took to the reach the result (this could be useful to seek for the best ROI.h^{-1}, which is not covered in this document but could be interesting to study aswell) : *duration*
- the earnings for the strategies after taking into consideration the 0.07% fees from Poloniex for each transaction (2 transactions per trade) : *EarningsPred0* and *EarningsPred1*

The recapitulative table has the following appearence and is the starting point for the rest of the strategy selection process :

prediction	proba0	proba1	result	date	end	close	duration	EarningsPred1	EarningsPred0	EarningsPred
[1]	0.26287460327148	0.7371253967285156	1	2019-11-23 08:10:00	2019-11-23 14:40:00	3685.68145793	0 days 06:30:00	0.008182818099999878	0.0	0.008182818099999878
[1]	0.382242739200592	0.617757260799408	1	2019-11-23 08:20:00	2019-11-23 14:45:00	3679.85	0 days 06:25:00	0.008182818099999878	0.0	0.008182818099999878
[1]	0.13590061664581	0.864099383354187	1	2019-11-23 08:25:00	2019-11-23 14:40:00	3687.0999998999996	0 days 06:15:00	0.008182818099999878	0.0	0.008182818099999878
[1]	0.064213275909423	0.9357867240905762	1	2019-11-23 08:30:00	2019-11-23 14:20:00	3687.0999998999996	0 days 05:50:00	0.008182818099999878	0.0	0.008182818099999878
[1]	0.057177782058715	0.9428222179412842	1	2019-11-23 08:35:00	2019-11-23 14:20:00	3689.69999981	0 days 05:45:00	0.008182818099999878	0.0	0.008182818099999878
[1]	0.202545583248138	0.7974544167518616	1	2019-11-23 08:40:00	2019-11-23 11:50:00	3690.35620203	0 days 03:10:00	0.008182818099999878	0.0	0.008182818099999878
[1]	0.082543194293975	0.9174568057060242	1	2019-11-23 08:45:00	2019-11-23 11:50:00	3690.35620203	0 days 03:05:00	0.008182818099999878	0.0	0.008182818099999878
[1]	0.095913529396057	0.9040864706039429	1	2019-11-23 08:50:00	2019-11-23 11:50:00	3691.88304624	0 days 03:00:00	0.008182818099999878	0.0	0.008182818099999878

Figure 21 : Table infered from the model's predictions for specific set of [stoploss, takeprofit, nPCs]

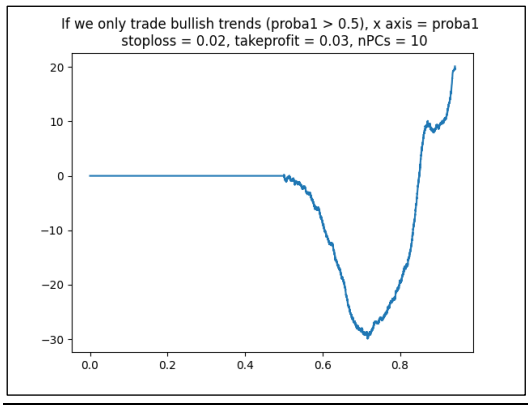
(II) Finding profitable strategies

Once we now have created the precedent table, the next step is – for each set of [stoploss, takeprofit, nPCs] – to create a recapitulative table that contains the performance of each segment – organized by steps of 0.05 for *proba1* - which gives us the information about where do the model performs well. The recapitulative table contains relevant information, that is :

- The segment which we consider : *e.g.* we consider observations where 0.7 < *proba1* < 0.8. Obviously the lower bound is above or equal 0.5, because we wouldn't invest money on a *bullish* trend if our model predicts a *bearish* trend : *Min* and *Max*
- The ROI% which we obtain when we consider only the selected segment : *ROI%*
- The number of trades of the selected segment : *nTrades*

The recapitulative table has the following appearence and corresponds to the following profile :

stoploss	takeprofit	nPCs	strategy	Min	Max	ROI%	nTrades
0.02	0.03	10	Bullish	0.5	0.55	-0.06956514160804399	3184
0.02	0.03	10	Bullish	0.5	0.6000000000000001	-0.1312262775609794	6560
0.02	0.03	10	Bullish	0.5	0.6500000000000001	-0.18757757536568737	10036
0.02	0.03	10	Bullish	0.5	0.7000000000000002	-0.20509595584342127	13718
0.02	0.03	10	Bullish	0.5	0.7500000000000002	-0.1525694315274319	17081
0.02	0.03	10	Bullish	0.5	0.8000000000000003	-0.09341938337789817	20945
0.02	0.03	10	Bullish	0.5	0.8500000000000003	0.0011720780243361197	25637
0.02	0.03	10	Bullish	0.5	0.9000000000000004	0.0301450370846401	29571
0.02	0.03	10	Bullish	0.5	0.9500000000000004	0.06351311625563641	31693
0.02	0.03	10	Bullish	0.5	1.0000000000000004	0.06351311625563641	31693
0.02	0.03	10	Bullish	0.55	0.6000000000000001	-0.18938061905213657	3376
0.02	0.03	10	Bullish	0.55	0.6500000000000001	-0.24241581078371663	6852



Figures 22 and 23 : Overview of a recapitulative table for a specific set of [stoploss, takeprofit, nPCs] – Plotting of the associated results

✓ Commentary: First we may observe that for *proba1* < 0.5, the bullish earnings are 0 : obviously we do not bet on a *bullish* trend if our estimator indicates a *bearish* trend, hence we simply do

not bet. Then, we can clearly identify two distinct zones: [0.5, 0.7] and [0.7, 1]. In the first zone, *i.e.* when 0.5 < *proba1* < 0.7, we notice that betting on the Bitcoin increase yielded some loss, whereas in the second zone we can clearly distinguish some profit being made. From this observation, we may want to select the times where *proba1* belongs to the second zone, in order to take a *bullish* trade then. In other words, when our estimator estimates that the probability of Bitcoin reaching + 3% before reaching – 2% is superior to 70%, then it is a good idea to buy Bitcoin.

Now that we have created all recapitulative tables for every set of [stoploss, takeprofit, nPCs], we can concatenate all of these strategies in one single file and rank them by order of ROI %. As there were 175 different models fitted (see *Section X (a)*) and that there are 55 different segments by model, there were in total 55*175 = 9625 different strategies to rank.

We obtain the following table :

stoploss	takeprofit	nPCs	strategy	Min	Max	ROI%	nTrades
0.05	0.05	10	Bullish	0.9500000000000004	1.0000000000000004	3.1693461612335287	10166
0.05	0.05	10	Bullish	0.9000000000000004	1.0000000000000004	2.793478783750797	12616
0.05	0.05	10	Bullish	0.8500000000000003	1.0000000000000004	2.6033532488468487	14482
0.05	0.05	10	Bullish	0.8000000000000003	1.0000000000000004	2.4642703240281523	16129
0.05	0.04	10	Bullish	0.9500000000000004	1.0000000000000004	2.3590686465184567	8367
0.05	0.05	10	Bullish	0.7500000000000002	1.0000000000000004	2.3522920057732764	17607
0.05	0.05	10	Bullish	0.7000000000000002	1.0000000000000004	2.2646436949442377	19008
0.05	0.05	10	Bullish	0.6500000000000001	1.0000000000000004	2.180118858782429	20393
0.04	0.05	10	Bullish	0.9500000000000004	1.0000000000000004	2.174667400535019	9439
0.05	0.05	10	Bullish	0.6000000000000001	1.0000000000000004	2.0968114796611363	21867
0.05	0.05	10	Bullish	0.55	1.0000000000000004	2.005898829788829	23393
0.04	0.05	10	Bullish	0.9000000000000004	1.0000000000000004	1.9284095920123805	12120
0.05	0.05	10	Bullish	0.5	1.0000000000000004	1.9146995649603504	25093

Figure 24 : Overview of the concatenation of the recapitulative tables

✓ Commentary : We can notice that the best strategies are obtained with the highest values of *stoploss* and *takeprofit*, which correspond to middle to long-term trading strategies. This is also due to the fact that the selected strategies yield impressive values of ROI%, and that these values are simply not achievable with lower values of takeprofit (*e.g.* we cannot obtain of ROI > 2% with a takeprofit of 0.02 because we always stop the trading at 2% profit – in the best case). Also, we notice that the best strategies are obtained with the highest values of *proba1*, which is quite intuitive and correlates the accuracy of the models : when the model is very confident about the price going up, then betting on an increase yields profit.

We can also have a quick look at the repartition of all the strategies' ROI as a function of the number of trades, where we observe that the average ROI is negative (mainly due to the fees), but we will see in the following section that we can still draw profitable strategies by selecting only the best strategies identified in the validation set :

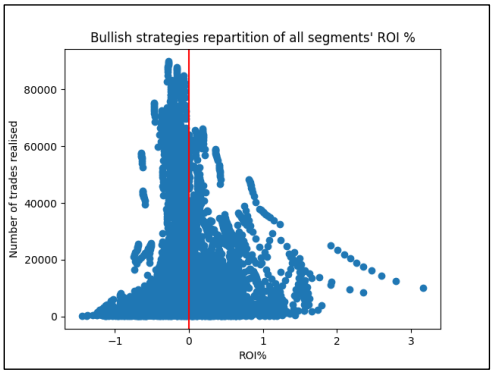


Figure 25 : Bullish strategies repartition of all segments ROI %, the average ROI is -0.076196 %

(III) Assessing these results, study correlation between validation set and testset

Within the last two steps, we created tools to identify the models and the zones that yielded profit on the validation set. That is great, but these strategies were selected *because* they made profit on the validation set, hence it could also be pure luck. To assess the accuracy and the robustness of such strategies, we will confront them to totally new observations, *i.e.* we will test these strategies onto the testset. We will first study the correlation between the two performances (validation set and testset) to ensure there is a positive correlation, which would help us to validate the robustness. Thanks to the *scipy* package in *python*, we can even compute the p-value relative to the correlation calculation, and draw conclusions regarding to the significance :

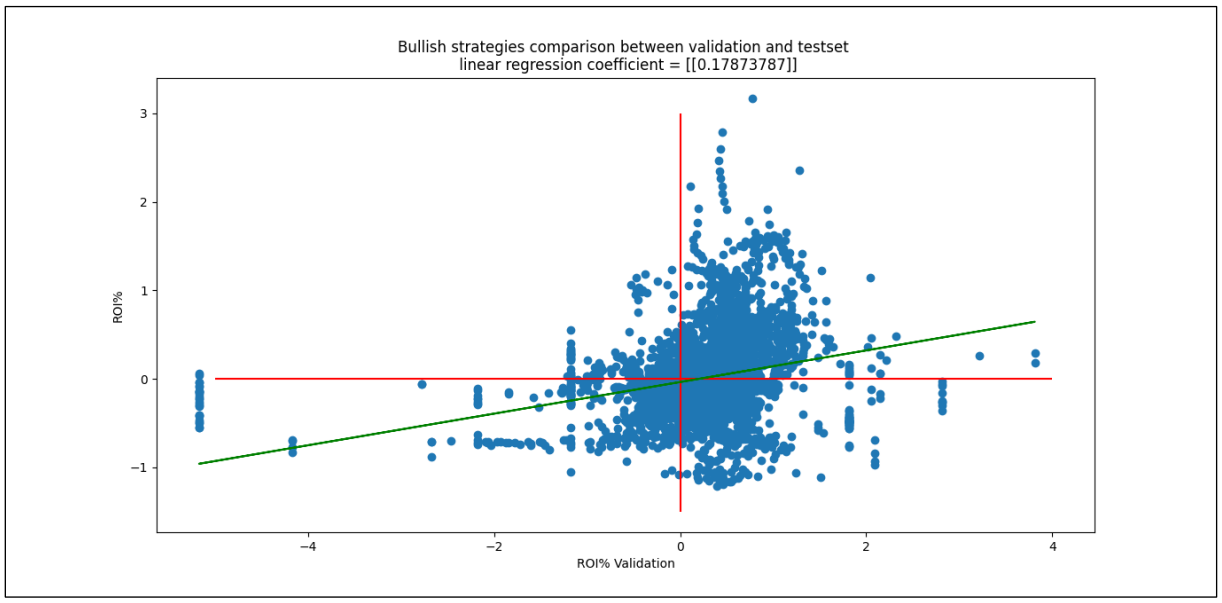


Figure 26 : Bullish strategies comparison between the two performances

```
# compute correlation significance
from scipy.stats.stats import pearsonr
c = pearsonr(df['ROI%Validation'], df['ROI%'])
print("Bullish strategy : The Pearson's correlation coefficient is {} with a p-value of {}".format(c[0], c[1]))
```

```
Bullish strategy : The Pearson's correlation coefficient is
0.27080928134117294 with a p-value of 1.3103696163389714e-99
```

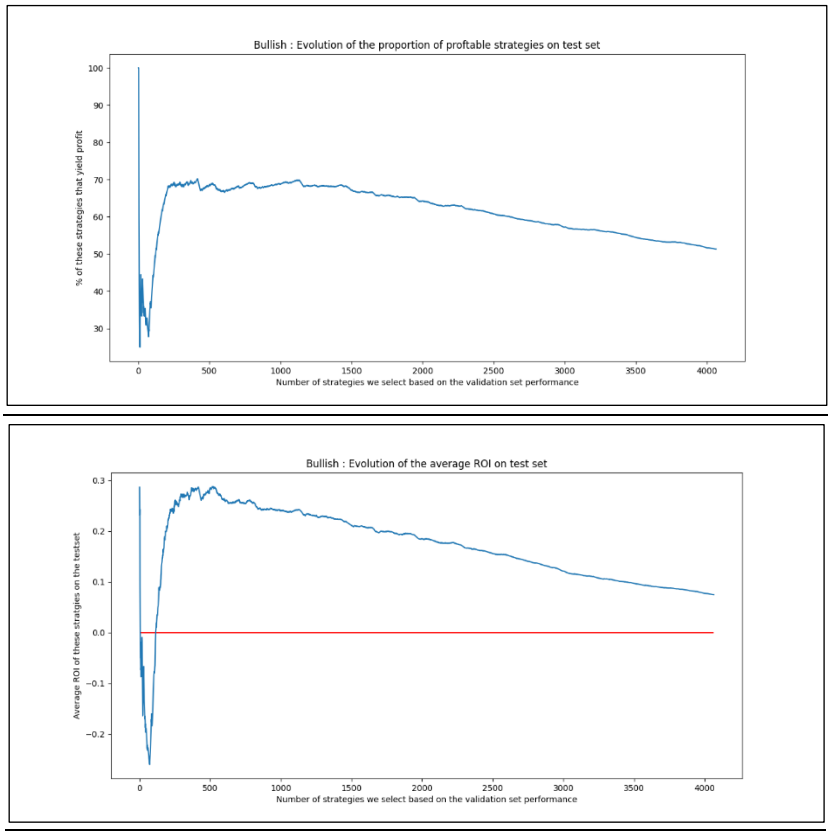
```
Bullish strategies, Spearman correlation between validation/test
                  ROI%  ROI%Validation
ROI%           1.00000         0.37119
ROI%Validation 0.37119         1.00000
Bullish strategies, Pearson correlation between validation/test
                   ROI%  ROI%Validation
ROI%           1.000000        0.270809
ROI%Validation 0.270809        1.000000
```

```
Bullish : Out of the 4064 strategies that yielded profit during the validation set,
2087 still yielded profit during the testset : 51.35334645669291 % of them
```

Figures 27 to 30 : Python script to compute the p-value from the Pearson correlation computation – Display of the values – Spearman and Pearson correlation matrices obtain with python integrated tool – Statistics related to the comparison

- ✓ Commentary : We can observe that there is a significant correlation between the two performances (p-value << 0.05) and that this correlation is a positive one (both Spearman and Pearson coefficients are positive), which tend to confirm the robustness of the trading strategies.

Finally, we could also think about having a look at the profile of the best methods, *i.e.* do the strategies with the highest validation set ROI's also perform best on the testset :



<u>Figures 31 and 32 :</u> Evolution of the proportion of profitable strategies on the test set (<u>NB :</u> the strategies come from the validation set performances and were ranked by order of ROI%) – Evolution of the average ROI% on test set

- ✓ <u>Commentary :</u> We can observe that the decreasing is almost always monotonous, which is a pretty good sign : this would indicate that the very best strategies on the validation set are also the very best strategies on the test set. Nontheless, we can also observe that if we select only the top 200 strategies, then the results on the test set are not profitable : this could be due to variance, to the difference of market conditions between the two sets, or that for an unknown phenomenon only the very best methods do not align with their validation set performances. However, we can clearly validate the robustness, as if we select a sufficiently large amount of strategies (*e.g.* the top 1000 strategies), we obtain an average performance that is largely satisfying, with a ROI > 0.2% on totally new data and approximately 65% of these strategies yielding profit.

As a conclusion, we can be satisfied with the process : we obtain a large number of strategies that are profitable on the long-run, and that can provide us a huge number of opportunities to make profit by betting on the Bitcoin increasing periods.

XI – Creating a trading bot on Poloniex

Once that we did all of the aforementioned steps, and are confident about the performance of our strategies, the next logical step is to deploy these strategies in real-life through an algorithm : this is the final trading bot. Thanks to the Poloniex API for *python*, It is quite feasible to deploy a loop to look for possible trades and to buy/sell in consequence.

Let's say we pick one specific strategy from our set of strategies and wish to deploy it. As explained in the previous parts, a trading strategy is defined by a model $Model_{takeprofit,stoploss,nPCs}$ and a segment $[Min, Max]$: to know if we should buy, we predict *proba1* through the model $Model_{takeprofit,stoploss,nPCs}$ and if *Min* < *proba1* < *Max* then we buy one unit (*amount* to be defined by the user). We finally wait for the price to either reach *takeprofit* or *stoploss*.

We have two choices in the deployment :

- Either we wish to have only one trade running (the strategy would then be easier to track for a human operator) : this is what I called *1-Trade-At-A-Time*
- Or we can choose to apply every trade, that is everytime the algorithm detects a possible trade, we deploy the process by buying and waiting for the takeprofit/stoploss to be reached : this is what I called *Multiple Simultaneous Trades*

The *Multiple Simultaneous Trades* choice is way harder to track for a human because of the many simultaneous active trades running, but on the other hand it allows to take a larger amount of trades, which would reduce the variance in the trading results.

We can explicit down there the algorithmical principle that we might wish to implement for real life trading, for the two choices :

- 1-Trade-At-A-Time pseudo-code :

```
# inputs : model, min, max, stoploss, takeprofit
While TRUE:
      If we have an active trade:
            if price < stoploss_active_trade:
                  sell our amount A at the highest available price
            else if price > takeprofit_active_trade:
                  sell our amount A at the highest available price
      If we don't have an active trade :
            request recent data from poloniex API
            compute estimated probability proba1 with model
            If min < proba1 < max :
                  buy Bitcoin for an amount A at the lowest available price
                  stoploss_active_trade = (1-stoploss)*price_Bitcoin
                  takeprofit_active_trade = (1+takeprofit)*price_Bitcoin
```

- Multiple Simultaneous Trades pseudo-code:

```
# inputs : model, min, max, stoploss, takeprofit
# L contains the list of active trades
While TRUE:
      for trade in L :
            if price < trade_stoploss:
                  sell our amount A at the highest available price
            else if price > trade_takeprofit:
                  sell our amount A at the highest available price
      request recent data from poloniex API
      compute estimated probability proba1
      if min < proba1 < max :
            buy Bitcoin for an amount A at the lowest available price
            add trade = (trade_stoploss, trade_takeprofit) to the list L
```

As we can see, the implementation is not very complicated : we simply need to compute the value of *proba1* and check if it fits in the segment of interest [Min, Max]. It would also be very easy to display the trade characteristics in the console to inform the operator, or to send a mail with the trade's information.

As a conclusion, we can say that we have now described the principle for deploying one specific strategy into real-life trading. To deploy multiple strategies at the same time, we could either run parallel scripts, or we could also create one single script – in order to collect the data just once for all strategies and apply the decision process sequentially for each strategy (*i.e.* for each strategy we compute *proba1* and check if it fits in the segment [Min, Max] of this specific strategy).

For illustration purpose, we can pick the best strategy (model = $Model_{0.05,0.05,10}$, Min = 0.95, Max = 1), and compare three investments over the validation set period:

- The first strategy with the choice of 1TAAT (unit per trade = 1)
- The first strategy with the coice of multiple simultaneous trades (unit per trade = 1)
- The Bitcoin

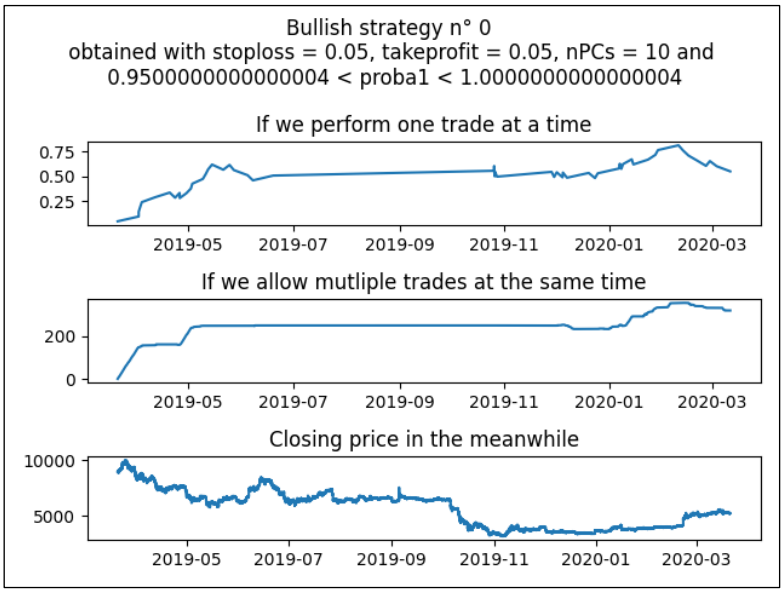


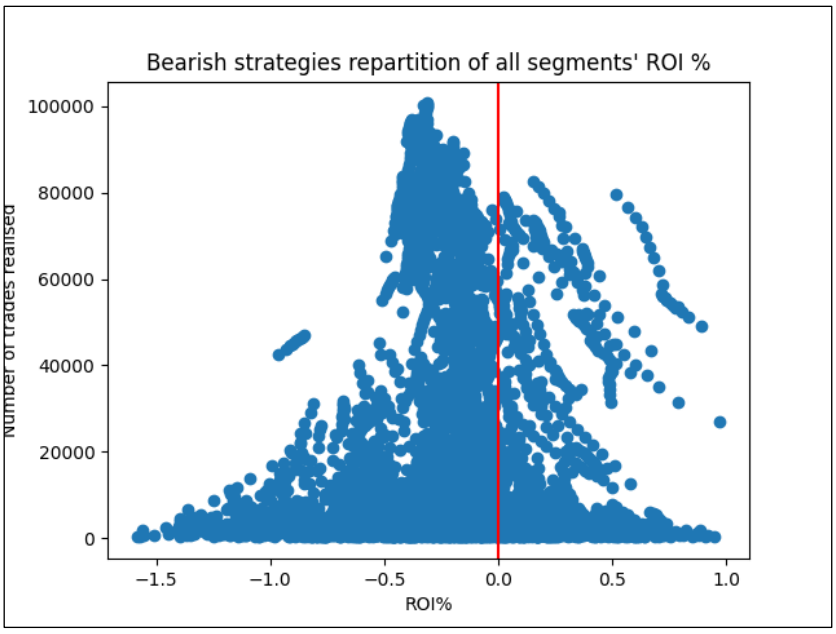
Figure 33 : Illustrative plots of the best strategy selected over the validation set

NB : The profit are computed at the date when the trade is taken, *e.g.* if we buy on January 1st and reached the takeprofit on March 1st, then the displayed curve will display an increasing spike at the date of January 1st. The plot's title contains "strategy n°0" because it was the output of a *python* script (*python* indexing starts at 0..)

XII – Bearish study

As announced earlier, the main point of my work is about finding *bullish* strategies, as I am not enough at ease with *bearish* trading, especially for deployment on the poloniex API (there are additionnal fees, and I am not enough at ease with the implementation of these fees into the backtesting). However, I also applied my pipeline to Bitcoin bearish trading – without taking additionnal fees into consideration, so this part is mainly informative - , by redefining the earnings according to the new case: now we win money if we accuratey identify decreasing trends. Hence, after applying all the steps I obtain the following results for bearish trading.

As during the bullish study, we can concatenate all bearish strategies, display their repartition and eventually rank them by order of ROI :

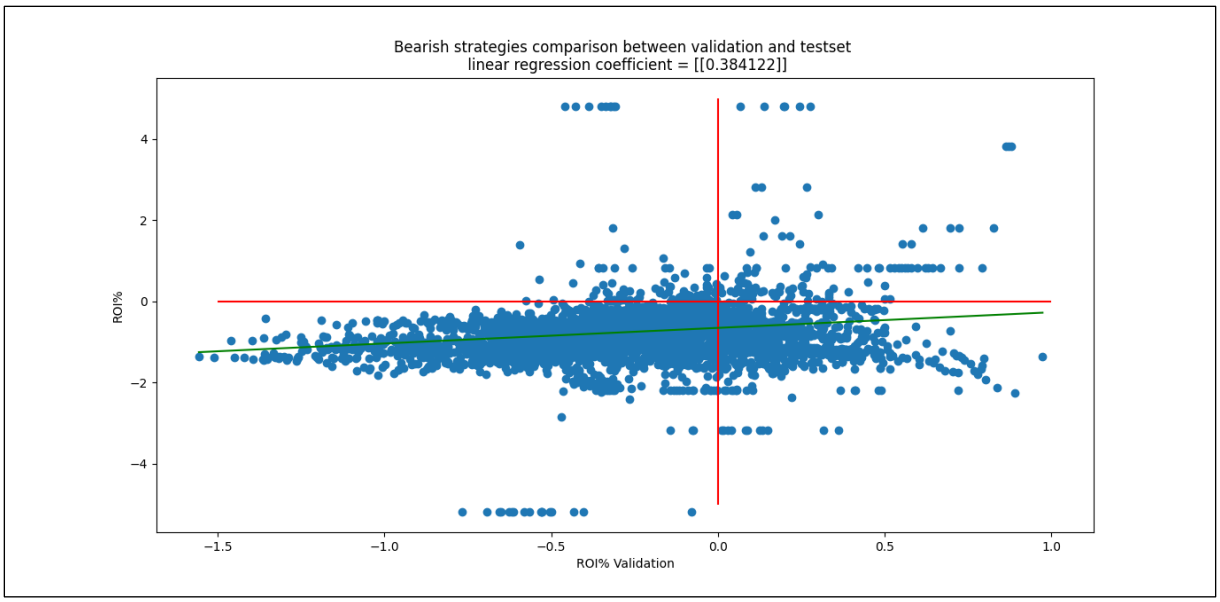


stoploss	takeprofit	nPCs	strategy	Min	Max	ROI%	nTrades
0.05	0.02	10	Bearish	0.9500000000000004	1.0000000000000004	0.9734746761094898	27053
0.05	0.05	95	Bearish	0.6500000000000001	0.7000000000000002	0.9479915197740126	354
0.05	0.02	95	Bearish	0.8000000000000003	0.8500000000000003	0.9184948834448826	508
0.03	0.04	95	Bearish	0.6000000000000001	0.6500000000000001	0.9090273381818158	363
0.05	0.05	65	Bearish	0.9500000000000004	1.0000000000000004	0.8900549405314102	48983
0.04	0.05	110	Bearish	0.9000000000000004	0.9500000000000004	0.8834881775557893	1837
0.05	0.02	95	Bearish	0.8000000000000003	0.9000000000000004	0.8810403995836877	1177
0.04	0.02	65	Bearish	0.6500000000000001	0.7000000000000002	0.8797823338388558	633
0.04	0.02	65	Bearish	0.6000000000000001	0.7000000000000002	0.871893618209712	1173
0.04	0.02	65	Bearish	0.6000000000000001	0.6500000000000001	0.8626462904444376	540
0.05	0.02	95	Bearish	0.8500000000000003	0.9000000000000004	0.8525996255904341	669

<u>Figures 34 and 35 :</u> Bearish strategies' repartition of ROIs (the average ROI is – 0.200906 %) – Overview of the best bearish strategies

- ✓ <u>Commentary :</u> As a general observation, we can note down that the performances are quite poor: only a minority of strategies yielded profit. We can also observe that the performances are less widespread than the bullish ones, and that even the best strategies are not exceeding 1% ROI. We also observe that the best methods were obtained with high values of takeprofit but with various values of stoploss (*<u>NB :</u>* for bearish study stoploss and takeprofit are inverted, *i.e.* stoploss just designates the lower bound while takeprofit designates the higher bound, so the meaning of the columns in figure 35 is inverted).

We can then also make predictions on the testset, and eventually compare the correlation between the two performances :



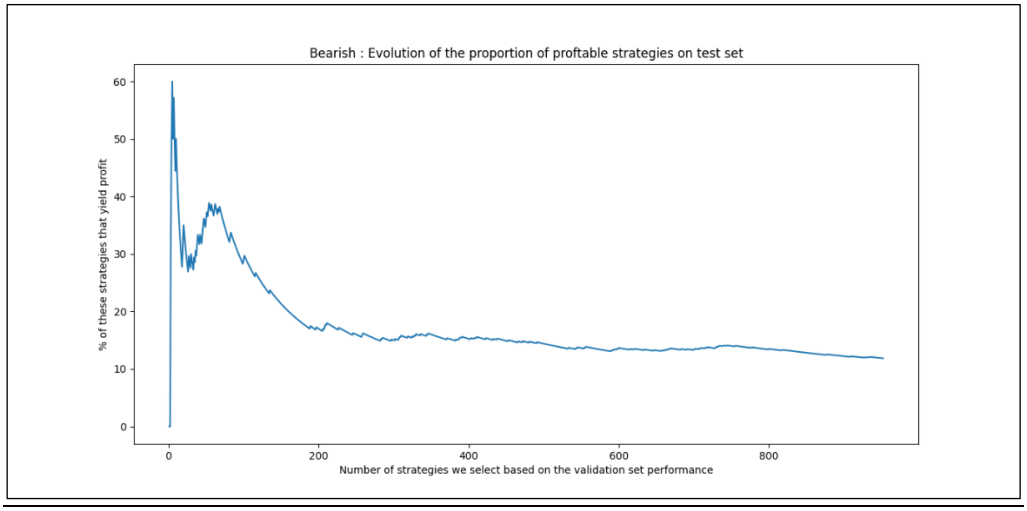

```
Bearish strategy : The Pearson's correlation coefficient is
0.18830370610962938 with a p-value of 1.485535553393473e-47
```

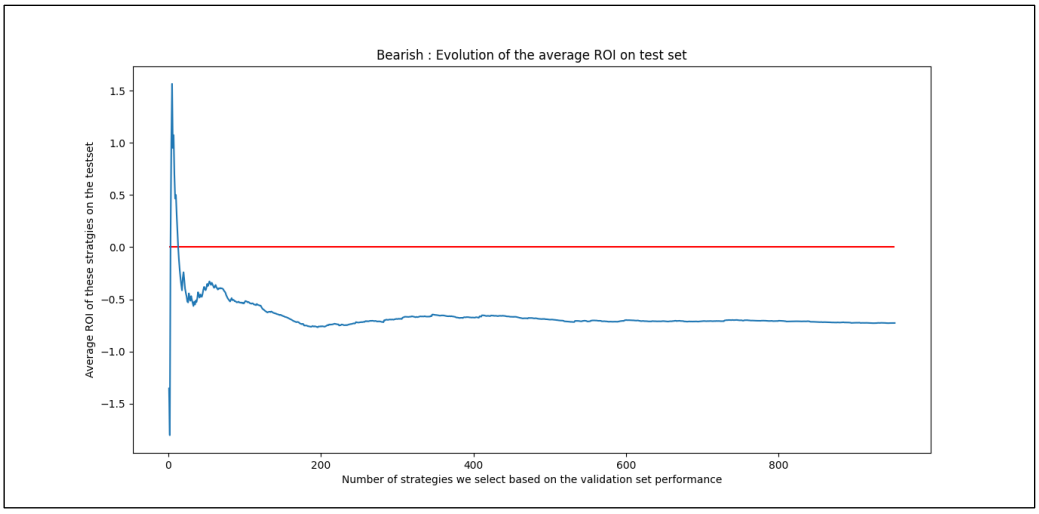
```
Bearish : Out of the 952 strategies that yielded profit duri
ng the validation set, 113 still yielded profit during the t
estset : 11.869747899159664 % of them
```

<u>Figures 36 to 38 :</u> Plotting of the ROI% on testset as a function of the performance on validation set for bearish strategies – Pearsons's correlation coefficient and p-value associated with the Pearson's coefficient computation - Statistics related to the comparison

- ✓ <u>Commentary :</u> We can observe that like during the validation set period, only few strategies yielded profit : we obtain only 113 strategies that yield profit in both sets. Concerning the correlation study, we still observe a significative positive correlation between the validation set and the test set, however as there are very few profitable strategies on the validation set it is difficult to draw conclusions regarding to the robustness and the accuracy of the strategies.

We could also think about having a look at the profile of the best methods, *i.e.* do the strategies with the highest validation set ROI's also perform better on the test set :





<u>Figures 39 and 40 :</u> Evolution of the proportion of profitable strategies on the test set (<u>NB :</u> the strategies come from the validation set performances and were ranked by order of ROI%) – Evolution of the average ROI% on test set

- ✓ <u>Commentary :</u> From the two displayed plots, we can observe that it not possible to make profit with the *bearish* strategies. There is only a minority of them yielding profit on the test set. This result confirms the previous plots of this part, particularly that there were a very limited range of profitable strategies (952 instead of 4064 for bullish study).

As a conclusion, we can recall that this *bearish* study was mainly informative but helped us to get an insight on the difference between the *bullish* and the *bearish* trading that we could obtain through our pipeline. It confirmed our choice of not taking this type of trading into consideration for building our trading bot, without even mentioning we didn't even take into consideration the additionnal fees.

XIII – Future improvements and limitations

After finishing this project, we can underline some ideas and future works that flourished along the construction of this trading bot, among which :

- Test with various metrics, take risk into consideration for the ranking of the strategies

- Merge mutiple strategies to form a single one (only one decision process) instead of just concatenating them, which then use different decision processes and different models for buying/selling

- Consider *bearish* trading by taking additionnal fees in order to perform long/short equity trading

- Deploy the trading bot and test in real-life conditions

We could also think about applying the pipeline to other currencies, as the principle should work aswell : the creation of models and strategies is directly derived from the market data, hence when applying the process to another market (stock markets, raw materials, electricity, etc…), the model and strategies would be numerically different but they would have many chances to work aswell !

As the main limitation of this project, we could point down the fact that this paper was backtested : all backtesting simulations are fundamentally wrong, because once we perform a trade in real-life, the market is changed. However, we could be confident that our pipeline would also work in real-life.

XIV – Conclusion

Concerning the aforementioned results, we can be pretty satisfied with the outputs, as the constructed pipeline successfully identified a set of profitable *bullish* strategies concerning the Bitcoin market. We proved the robustness of the *bullish* strategies by confronting them to totally unseen data through the testset, and we observed that on average they were still yielding some profit. We also observed a significative positive correlation between the two performances, and that the best strategies from the validation set were usually yielding great profit in the testset aswell.

The results are extremely encouraging in the sense that the generated strategies are directly derived from the market, and that in this sense they adapt to the market data. Hence, - as mentioned in the last part - it would be very interesting to apply the very same pipeline to other cryptocurrencies markets or even other traditional financial markets such as stock markets, raw materials, electricity, etc...

This project was very fulfilling, as I discovered by myself a new field that I previously never got into. I learned a lot about financial markets and the studies that are made about this field through the documentation phase, plus I also learned a lot about the many algorithmic tools that were deployed in the financial market, along with various analysis and prediction technologies.

The project made me confirm my wish of pursuing my studies with a Quantitative Finance MSc (Lille University – France) in order to graduate in this field and start my profesionnal career in this sector. This Master, which I am currently attending beside my cursus in my engineering school (Ecole Centrale de Lille), will surely help me to learn some technical points that will help me improve my work towards trading bot, and more generally towards the financial field.

XV – Sources

[1] Creative commons licenses

https://creativecommons.org/licenses/

[2] Gekko repository

https://github.com/askmike/gekko

[3] Catalyst repository

https://github.com/enigmampc/catalyst

[4] Cyrpto trader repository

https://github.com/timucin/cyrpto_trader

[5] PoloBot repository

https://github.com/steviecurrie/PoloBot

[6] Trading bot repository

https://github.com/pskrunner14/trading-bot

[7] Marouane Anane, Frédéric Abergel. Optimal high frequency strategy in an omniscient order book. 2014. ffhal-01006401f

https://hal.archives-ouvertes.fr/hal-01006401/file/ANANE_201402_Optimal_High_Frequency_Strategy.pdf

[8] An agent strategy for automated stock market trading combining price and order book information. Silaghi, Robu

https://www.researchgate.net/publication/224642405_An_agent_strategy_for_automated_stock_market_trading_combining_price_and_order_book_information

[9] Predicting short-term Bitcoin price fluctuations from buy and sell orders. Guo & Antulov-Fantulin, 2018

https://arxiv.org/pdf/1802.04065v1.pdf

[10] Stock Trading Bot Using Deep Reinforcement Learning Akhil Raj Azhikodan, Anvitha G. K. Bhat and Mamatha V. Jadhav

https://www.researchgate.net/publication/325385951_Stock_Trading_Bot_Using_Deep_Reinforcement_Learning

[11] – Poloniex API in Python

https://pypi.org/project/poloniex/

[12] – Poloniex API documentation

https://docs.poloniex.com/#returnchartdata

[13] *keras* pip package

https://pypi.org/project/Keras/

[14] *tensorflow* pip package

https://www.tensorflow.org/install/pip

[15] *Matplotlib* pip package

https://pypi.org/project/matplotlib/

[16] *Numpy* pip package

https://pypi.org/project/numpy/

[17] *Pandas* pip package

https://pypi.org/project/pandas/

[18] *Poloniex* pip package

https://pypi.org/project/poloniex/

[19] *scikit-learn* pip package

https://scikit-learn.org/stable/install.html

[20] *pickle* in python

https://docs.python.org/3/library/pickle.html

[21] *scipy* pip package

https://pypi.org/project/scipy/

[22] Fibonacci numbers in finance

https://en.cryptonomist.ch/2019/06/23/fibonacci-financial-trading/

[23] Moving averages

https://www.wallstreetmojo.com/moving-average-formula/

[24] Bollinger bands description

https://www.bollingerbands.com/bollinger-bands

[25] MACD description

https://www.investopedia.com/terms/m/macd.asp

[26] RSI description

https://www.investopedia.com/terms/r/rsi.asp

[27] New concepts in technical trading systems, J. Welles Wilder Jr, 1978

https://www.academia.edu/37588524/New_Concepts_in_Technical_Trading_Systems_pdf

[28] Stackoverflow question for RSI calculation

https://stackoverflow.com/questions/20526414/relative-strength-index-in-python-pandas/32346692#32346692

[29] *Trendy* package for python

https://github.com/dysonance/Trendy

[30] Bull run principle and description

https://cryptotips.eu/en/knowledge-base/what-is-a-bull-market-or-a-bull-run/

[31] Halving principle and description

https://www.foxbusiness.com/markets/bitcoin-what-is-halving

[32] Pearson correlation definition and illustrations

https://www.statisticshowto.com/probability-and-statistics/correlation-coefficient-formula/

[33] Sigmoid function

https://www.youtube.com/watch?v=WsFasV46KgQ

[34] Adam optimizer

https://keras.io/api/optimizers/

[35] Cross entropy function

https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html#:~:text=Cross%2Dentropy%20loss%2C%20or%20log,diverges%20from%20the%20actual%20label.

[36] Software used for modelling the neural network

http://alexlenail.me/NN-SVG/index.html