2022

Algorithms against the market

Data science project to beat the market with artificial intelligence and machine learning

BY Peter Pichler

Inhaltsverzeichnis

Foreword

This project might feel intimidating to someone that isn't into coding, finance or statistics at all. But don't be deterred. Only a few short snippets of the full code are included to make this project as conceivable as possible. The goal of this paper is to simply show what technologies could be used to analyze and understand modern markets, and how the accompanying information advantage can be exploited to 'beat the market'. The goal that I have set myself in this project is almost of unachievable nature, and although the outcome of this project was more than surprising, I am educated enough to be cautious and consider this a fading state of success. As the irrationalities and dynamics of modern markets are as complex and unpredictable as human nature itself, I don't claim, and never will claim to have found the key to tame the idiosyncratic monster that we call 'market'.

Why analyze the market?

Market analysis enables investors to identify the intrinsic worth of a security even before investing in it. Analysts try to find out activity of an instrument/sector/market in future, in order to determine if the investment implies an attractive return.

By analyzing and predicting the market to great efficiency, market participants could therefore gain extremely attractive returns on seemingly unimpressive investment choices. And since cryptocurrencies are at its peak, opportunities to earn money from a suddenly skyrocketing asset, seem as realistic as never before. At least in a theoretical perspective.

So, is it really that simple?

Is there a potential analysis tool that could make you rich in seconds?

Probably not, since the amount of information that has to be evaluated in order to predict the market is horrendous. The market can be considered efficient, at least according to modern capital market theories, but the market is certainly far from persistently predictable. Although research has shown, that inside trading and certain information advantages can result in a higher return, to process more general information has so far not concluded any evidence that to do so would increase your chances to 'beat the market'. But I'd say we give it our best try anyway.

What should we try to analyze?

The modern world we live in contains a large variety of technologies that we can use to predict and understand the world around us. But the only way to make something measurable and therefore usable for analysis is to entail a system of values to describe it. In a second step we than can use the values and patterns that naturally occur in the system to analyze its behavior to understand interdependencies, correlation, and trends in the system.

Luckily, modern financial markets fulfill those preconditions extensively. A variety of assets which strong interdependencies, correlation and trends that not only are not only consistently present in our lives, but also at the tip of our fingers, as most information about the stock market is a google search away.

As easy it is to access the information, as nearly impossible it seems to understand the markets mechanics, when we're not using technology to do so.

Algorithms have a significant advantage over all humans that participate in the stock market - they decide much faster than humans and the decision is rational. They can evaluate more information in seconds than a human is able to process over his/her whole lifetime.

But before we do choose an algorithm, I'd say we choose an asset to analyze. And what would be more prominent and present than the crypto pioneer Bitcoin?

No other asset has created multimillionaires from old hard drives and dusty USB-sticks. Since the idea of decentralized currency fueled the release of Bitcoin Satoshi Nakamoto in 2009, it skyrocketed from a value of a couple cents to more than 60.000 USD at its highest point. Bitcoin was disputed for a long time, as an asset that should be part of a conservative investor's portfolio, but since the value hasn't dropped to near 0 so after the hype in 2021, and still seems to be relatively consistent in its market performance and has therefore conquered many hearts of conservatives and fundamentalists.

What are Bitcoins expectations at the beginning of 2022?

So, Bitcoin seems to be an asset that is worth analyzing and predicting, as the best that can happen, is that I become millionaires by doing so. On the other hand, the worst thing that could happen is that we lose a bigger part of our money, should Bitcoin crash completely. Unfortunately, that is a very realistic scenario, at least according to Charlie Munger, who is the vice chairman of Berkshire Hathaway. To keep our optimism, other Billionaires/Investors such as Tim Draper completely contradict such doomsday scenarios, as he seems to believe that Bitcoin will rise to 250.000 USD in the course of 2022.

Since Bitcoin doesn't really provide realistic value in our modern economy yet, the development of its price is completely dependent on the expectations of the growth or decline of its market capitalization. The asset is basically only worth what people are willing to pay for it. So, we could win or lose very big, as no one really can make an educated guess where Bitcoin is going to be in a couple days, weeks or months.

What is the budget for my project?

Since I personally am a student, and don't have a lot of money to burn by trying something that more than half the financial world has seemingly tried before me. I wanted to evaluate how safe and efficient my final idea would work, before letting the thing that I came up with play with my real money. So, I decided to declare a virtual portfolio of 10000 USD as my guinea pig for this experiment and started to research the best ways for a university student with decent programming experience to analyze the market.

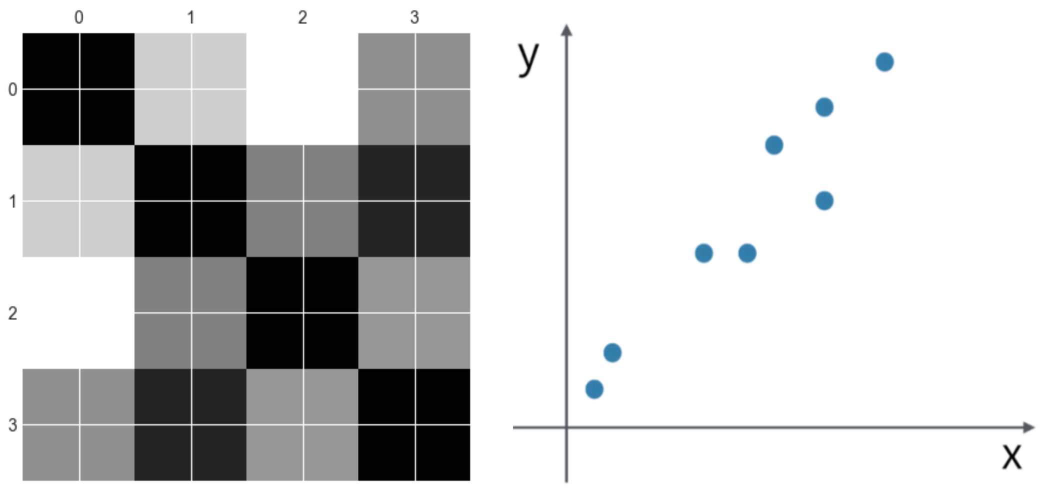
How should we analyze the market?

The first steps are the hardest or the steepest in such projects, as it was very important to figure out a good reliable model that could be used to understand if the price of bitcoin was going to increase or fall in the next cycle. Since bitcoin is not an asset that behaves according to fixed

rules, I decided to use more than just one model to predict the price that bitcoin will have (but more on that later)
To increase my security on the investments, I decided to choose a relatively conservative approach regarding bitcoins analysis, and that was to capture the value of the asset in a broader financial equilibrium. I wanted to be able to establish if Bitcoin was over- or undervalued, in a broad context of assets that correlate to Bitcoin. This might sound confusing at first, but I'm sure that you'll understand the concept in the explanation that follows.

Linear Regression Model

Why would a linear regression model work?



Different assets play different roles in the portfolio of investors. Conservative investment strategies imply, that the risk of a portfolio should be diversified, to achieve the best possible relation of risk and return. In a crisis for example, the price of gold normally rises, as it is widely considered to be one of the most resilient assets, although not very rentable. Vice versa, the price of Gold tendentially decreases when the stock market is on the rise. This behavior can also be established, considering the results of the correlation tests of BTC and Gold, as well as BTC and the S&P 500.

As the S&P and BTC are largely considered to be assets that are more attractive during an economic boom, the price of S&P and BTC correlate strongly positive, as BTC and Gold correlate slightly negative. This implies that if the investors on public markets tend to bullish behavior, the price of certain assets rises, because the demand for these assets increases.

The fundamental logic of the linear regression model is, that certain assets have a correlation, because of their shared relevance as assets in modern financial markets.

The aim of the linear regression model therefore is to consistently monitor the pricing data of the correlating assets in the stock market and analyze whether the financial equilibrium implies that BTC is currently under- or overvalued.

Which correlating assets to choose?

To cover a big part of the financial equilibrium I decided to choose some of the biggest and most relevant financial indexes and test their correlation to bitcoin.

While testing it became very clear that the correlation consistently changes, depending on what time period I choose to test. That’s why I decided to use less than a year, in order to be more inclusive of trends that occurred in recent weeks. In this case it was relatively important to find a rational compromise between long term interdependencies of the assets, and the recent market trend. After doing multiple correlation tests in the range of 10-52 weeks, I noticed that test data of around 25 weeks of bitcoin would be the best approach, as it is statistically robust, but still relatively inclusive of recent developments.

The code that would finally run in my server was written so that it would optimize this range on a daily basis and adjust to extreme changes in market trends faster if the market would do huge jumps. But that is just a nerdy sidenote.

This is the perfect moment to set a reminder:

To see specifics about the code and look more closely to my greenhorn approach to create a robust model, you can always look at the (python) codes that were used to create the part of the project that is concerned with machine learning and data automation.

Correlation Tests

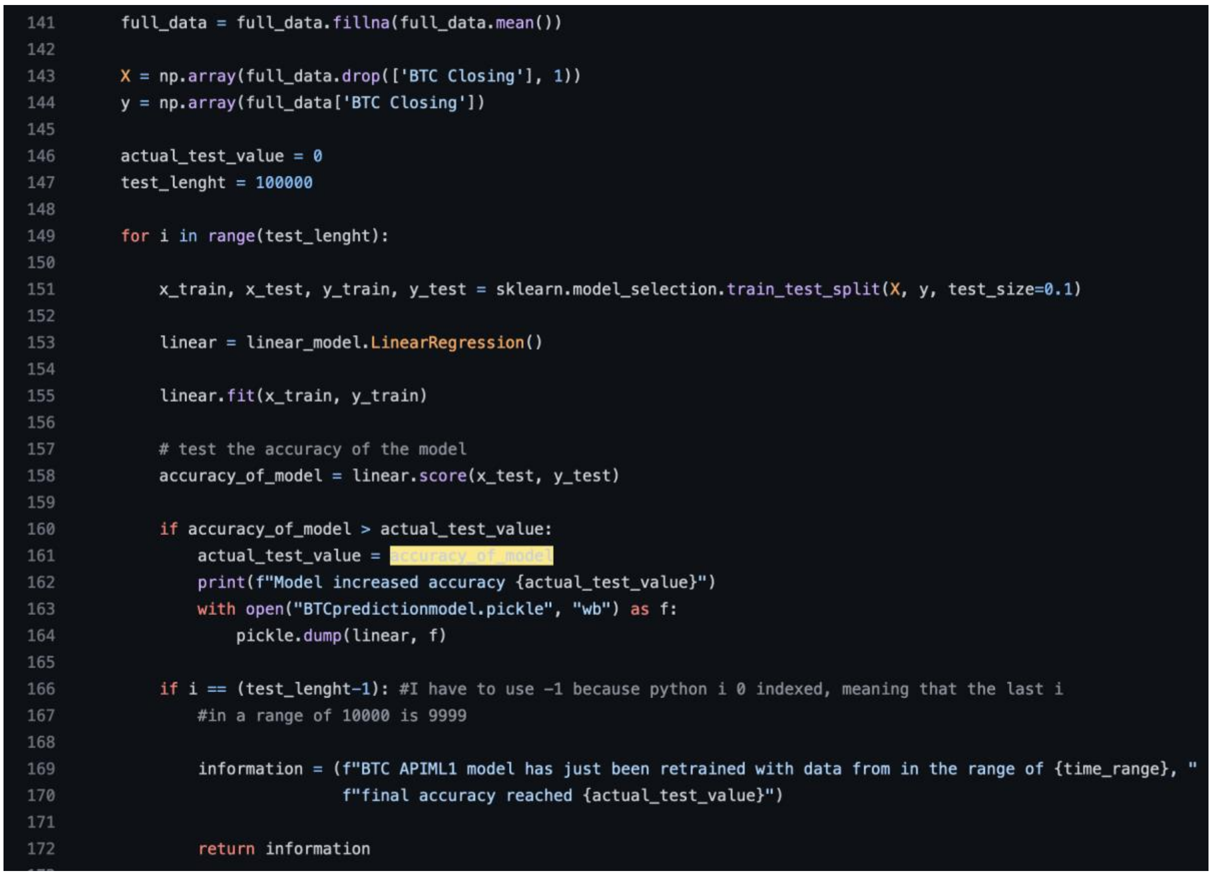
```
Results

the correlation between BTC and the S&P500 (SPY) amounts to:
 0.8321742852518388
the correlation between BTC and Gold (GC=F) amounts to:
-0.19452515112363158
the correlation between BTC and western bonds (MXBIX) amounts to:
 0.6664411973760584
```

As can be seen in the results of the correlation calculations within the chosen time range, the values vary from a strong correlation of ca 0.83 of the s&p500 and 0.67 of western bonds, to a slight negative correlation of -0.19 of gold.

This foundation was sufficient to build a model with, if I would also implement the consistent retraining of the model, as, distribution tests of the correlation showed that the correlation of the factors is consistently changing.

Linear regression model snippet (full code on GitHub)

The linear regression model would be created and saved as a bitstream object in the virtual environment that the BOT operates in. To achieve the highest possible accuracy, the model was optimized a total of 100000 times, every time that the model was trained, resulting in an average accuracy of the predictions of the model around 99 %. This means that the model was able to predict every price that bitcoin historically had over the test range, with an error rate of only 1%. So, it was safe to say that the data was chosen well, and the model worked impressively well.



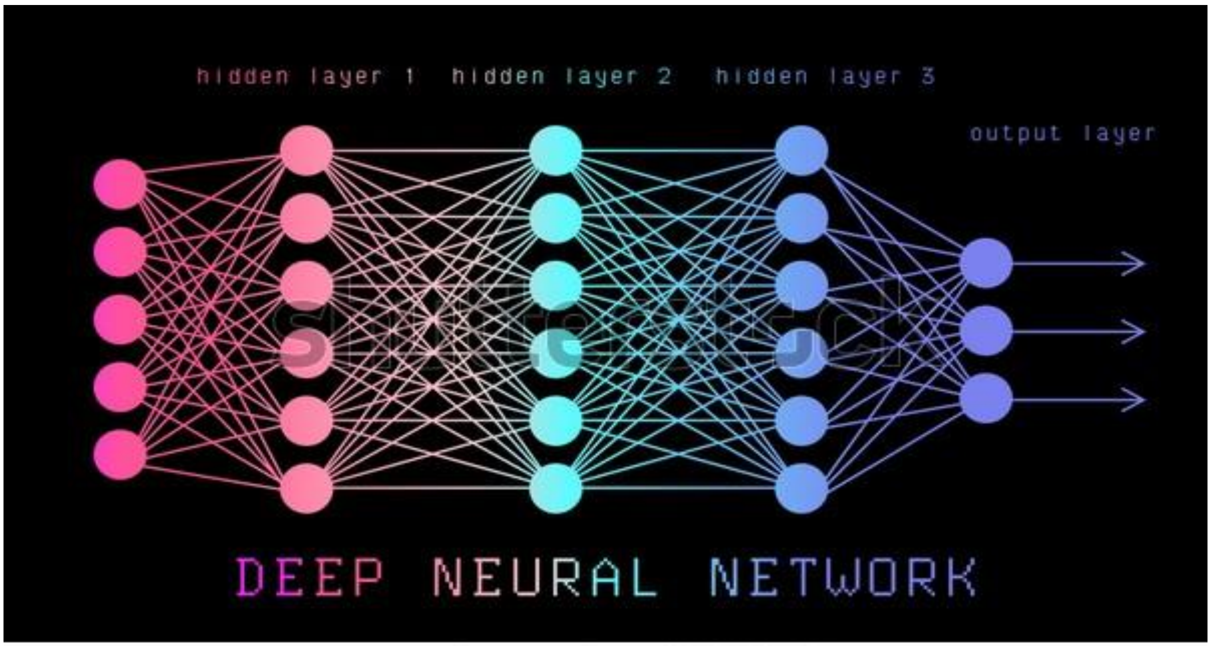
```python
    full_data = full_data.fillna(full_data.mean())

    X = np.array(full_data.drop(['BTC Closing'], 1))
    y = np.array(full_data['BTC Closing'])

    actual_test_value = 0
    test_lenght = 100000

    for i in range(test_lenght):

        x_train, x_test, y_train, y_test = sklearn.model_selection.train_test_split(X, y, test_size=0.1)

        linear = linear_model.LinearRegression()

        linear.fit(x_train, y_train)

        # test the accuracy of the model
        accuracy_of_model = linear.score(x_test, y_test)

        if accuracy_of_model > actual_test_value:
            actual_test_value = accuracy_of_model
            print(f"Model increased accuracy {actual_test_value}")
            with open("BTCpredictionmodel.pickle", "wb") as f:
                pickle.dump(linear, f)

        if i == (test_lenght-1): #I have to use -1 because python i 0 indexed, meaning that the last i
            #in a range of 10000 is 9999

            information = (f"BTC APIML1 model has just been retrained with data from in the range of {time_range}, "
                           f"final accuracy reached {actual_test_value}")

            return information
```

But to only understand if BTC was over- or undervalued in a broader financial context seems to be almost boring and certainly not a very innovative approach to solve the problem at hand. That’s why I chose to employ a second model, that could be considered the ‘partner in crime’ of the bot that analyzes linear regression. This next model could be the brain of the operation to say the least, as its fundamental logic is a considerable part of what we call ‘artificial intelligence’ or AI.

Neural network

Why would a neural network model work?

Neural nets are a means of doing machine learning, in which a computer learns to perform some tasks by analyzing training examples. Usually, the examples have been hand-labeled in advance. An object recognition system, for instance, might be fed thousands of labeled images of cars, houses, coffee cups, and so on, and it would find visual patterns in the images that consistently correlate with particular labels. Modeled loosely on the human brain, a neural net consists of thousands or even millions of simple processing nodes that are densely interconnected. Most of today's neural nets are organized into layers of nodes, and they're "feed-forward," meaning that data moves through them in only one direction.
An individual node might be connected to several nodes in the layer beneath it, from which it receives data, and several nodes in the layer above it, to which it sends data.

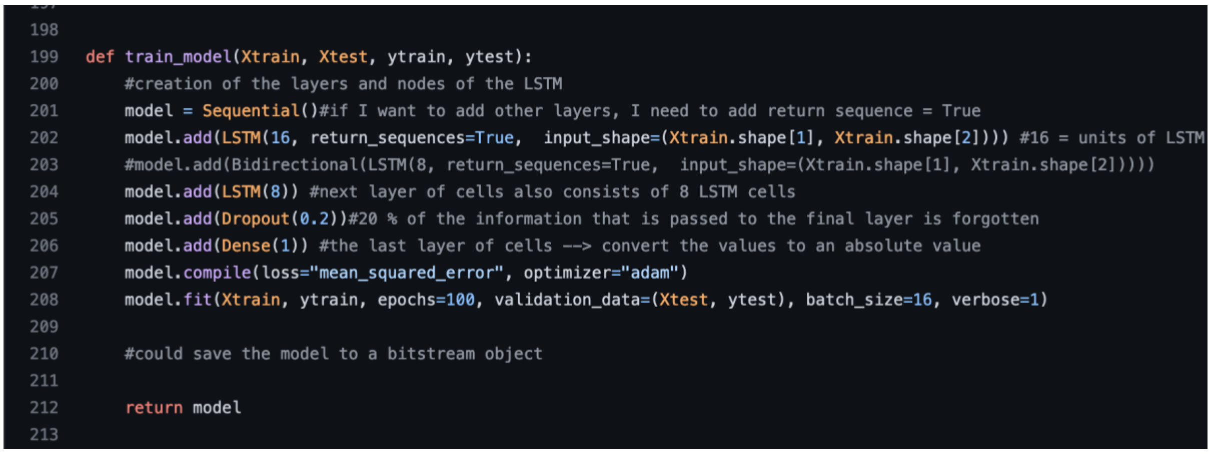


To each of its incoming connections, a node will assign a number known as a "weight." When the network is active, the node receives a different data item — a different number — over each of its connections and multiplies it by the associated weight. It then adds the resulting products together, yielding a single number. If that number is below a threshold value, the node passes no data to the next layer. If the number exceeds the threshold value, the node "fires," which in today's neural nets generally means sending the number — the sum of the weighted inputs — along all its outgoing connections. When a neural net is being trained, all of its weights and thresholds are initially set to random values. Training data is fed to the bottom layer — the input layer — and it passes through the succeeding layers, getting multiplied and added together in complex ways, until it finally arrives, radically transformed, at the output layer. During training, the weights and thresholds are continually adjusted until training data with the same labels consistently yield similar outputs.

In a nutshell

This all might sound very abstract to someone that doesn’t have any connection to such technologies. Basically, a neural network reflects the behavior of the human brain, allowing computer programs to recognize patterns and solve common problems in the fields of AI, machine learning, and deep learning. Therefore, the neural network, that we are aiming to create is an algorithm that can detect patterns in the historical prices, and to therefore predict what the most likely pattern of the next price of Bitcoin would be.

Neural network snippet (full code on GitHub)



```

def train_model(Xtrain, Xtest, ytrain, ytest):
    #creation of the layers and nodes of the LSTM
    model = Sequential()#if I want to add other layers, I need to add return sequence = True
    model.add(LSTM(16, return_sequences=True,  input_shape=(Xtrain.shape[1], Xtrain.shape[2]))) #16 = units of LSTM
    #model.add(Bidirectional(LSTM(8, return_sequences=True,  input_shape=(Xtrain.shape[1], Xtrain.shape[2]))))
    model.add(LSTM(8)) #next layer of cells also consists of 8 LSTM cells
    model.add(Dropout(0.2))#20 % of the information that is passed to the final layer is forgotten
    model.add(Dense(1)) #the last layer of cells --> convert the values to an absolute value
    model.compile(loss="mean_squared_error", optimizer="adam")
    model.fit(Xtrain, ytrain, epochs=100, validation_data=(Xtest, ytest), batch_size=16, verbose=1)

    #could save the model to a bitstream object

    return model

```

In the context of my project, the neural network will be trained to determine if the next price of bitcoin is going to increase or decrease, based on a year of historical data, and the patterns that emerge, considering historical price data of 30 days as input to every datapoint where the price of Bitcoin consequently increased or decreased. The regarding information would be compressed and analyzed across 3 layers with 16, 8 and a final, dense layer respectively.

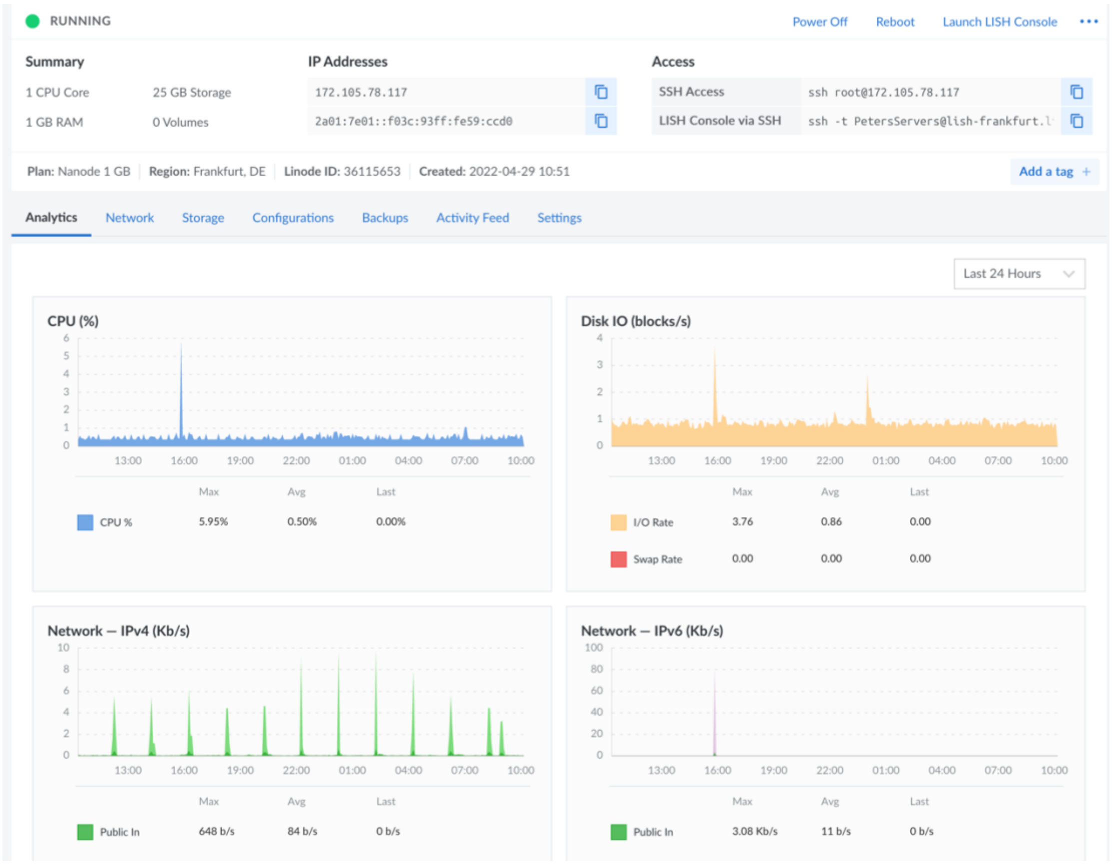
The code of the described bot can be found on my GitHub @peterssservers

Further requirements

At this point I created the backend that I needed for my trading bot to (in the best case) make informed decisions on the market. But that was clearly not enough, as it is necessary for such an operation to work, I had to deploy the different bots on servers, and connect them via databases that could store and deliver the data that was necessary to bring the project to live.

Luckily, I spend a lot of time in cyber security anyway, so I had some Linux servers that I used for pen testing and network scans etc., that easily had the capacity to host the bots that I created for an indefinite amount of time. To fulfill the need for the underlying database structure of the bots I simply bought an additional shared CPU of an Ubuntu 17.4 server and installed a MySQL database service on it. I then created all the relational databases and tables that I needed and whitelisted my bots IP to store and retrieve data.

Graph X: My Ubuntu Server that hosts the necessary database


RUNNING

Power Off · Reboot · Launch LISH Console

Summary

1 CPU Core · 25 GB Storage

1 GB RAM · 0 Volumes

IP Addresses

```
172.105.78.117
2a01:7e01::f03c:93ff:fe59:ccd0
```

Access

SSH Access	`ssh root@172.105.78.117`
LISH Console via SSH	`ssh -t PetersServers@lish-frankfurt.l`

Plan: Nanode 1 GB | **Region:** Frankfurt, DE | **Linode ID:** 36115653 | **Created:** 2022-04-29 10:51

Add a tag +

Analytics · Network · Storage · Configurations · Backups · Activity Feed · Settings

Last 24 Hours

CPU (%)

	Max	Avg	Last
🟦 CPU %	5.95%	0.50%	0.00%

Disk IO (blocks/s)

	Max	Avg	Last
🟨 I/O Rate	3.76	0.86	0.00
🟥 Swap Rate	0.00	0.00	0.00

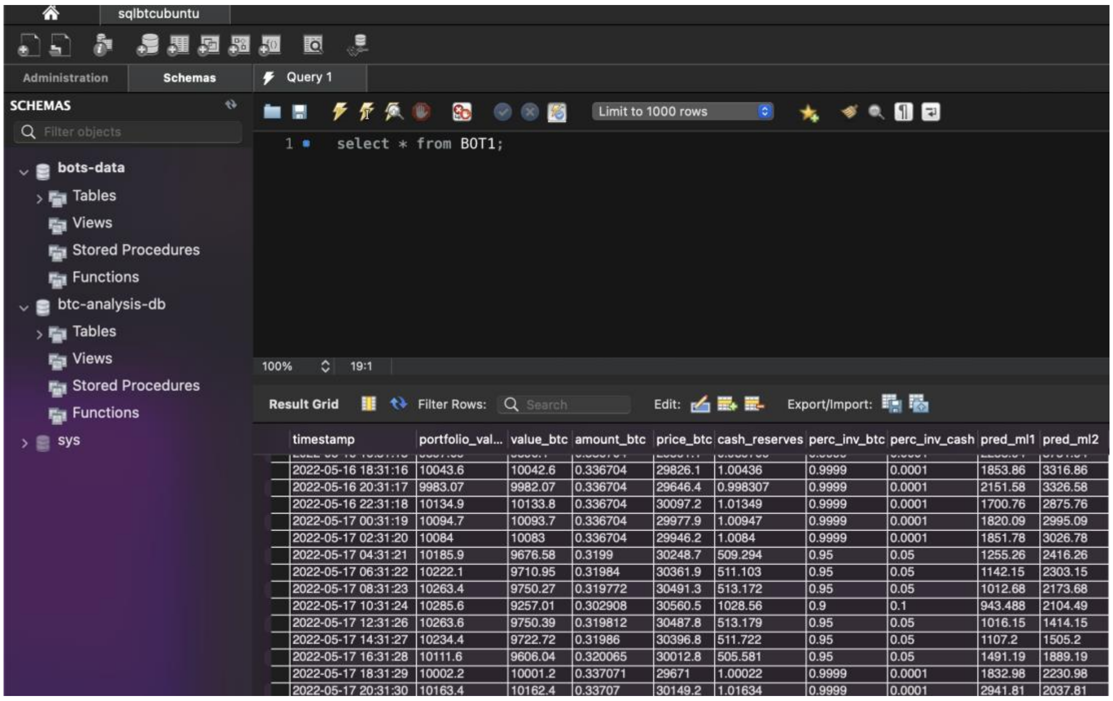
Network — IPv4 (Kb/s)

	Max	Avg	Last
🟩 Public In	648 b/s	84 b/s	0 b/s

Network — IPv6 (Kb/s)

	Max	Avg	Last
🟩 Public In	3.08 Kb/s	11 b/s	0 b/s

Graph X: the MySQL Databases of the bots and corresponding data


sqlbtcubuntu

Administration · Schemas · Query 1

SCHEMAS

- bots-data
  - Tables
  - Views
  - Stored Procedures
  - Functions
- btc-analysis-db
  - Tables
  - Views
  - Stored Procedures
  - Functions
- sys

Limit to 1000 rows

```
select * from BOT1;
```

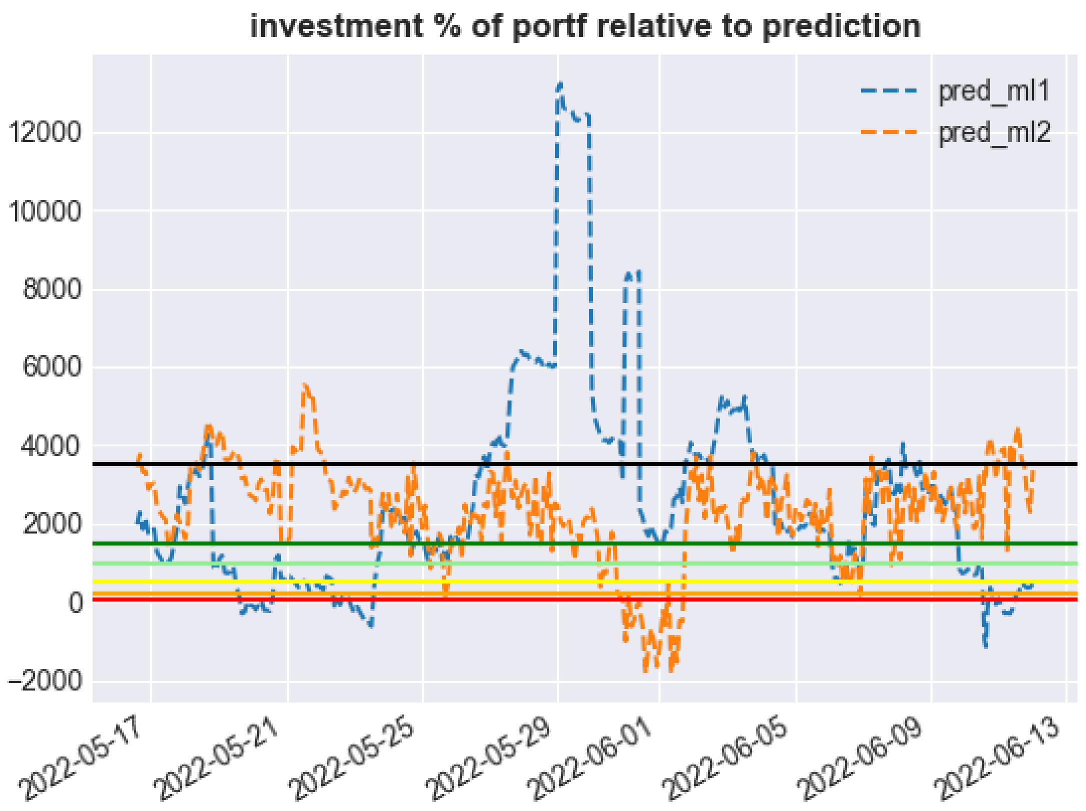
100% · 19:1

Result Grid · Filter Rows: · Edit: · Export/Import:

timestamp	portfolio_val...	value_btc	amount_btc	price_btc	cash_reserves	perc_inv_btc	perc_inv_cash	pred_ml1	pred_ml2
2022-05-16 18:31:16	10043.6	10042.6	0.336704	29826.1	1.00436	0.9999	0.0001	1853.86	3316.86
2022-05-16 20:31:17	9983.07	9982.07	0.336704	29646.4	0.998307	0.9999	0.0001	2151.58	3326.58
2022-05-16 22:31:18	10134.9	10133.8	0.336704	30097.2	1.01349	0.9999	0.0001	1700.76	2875.76
2022-05-17 00:31:19	10094.7	10093.7	0.336704	29977.9	1.00947	0.9999	0.0001	1820.09	2995.09
2022-05-17 02:31:20	10084	10083	0.336704	29946.2	1.0084	0.9999	0.0001	1851.78	3026.78
2022-05-17 04:31:21	10185.9	9676.58	0.3199	30248.7	509.294	0.95	0.05	1255.26	2416.26
2022-05-17 06:31:22	10222.1	9710.95	0.31984	30361.9	511.103	0.95	0.05	1142.15	2303.15
2022-05-17 08:31:23	10263.4	9750.27	0.319772	30491.3	513.172	0.95	0.05	1012.68	2173.68
2022-05-17 10:31:24	10285.6	9257.01	0.302908	30560.5	1028.56	0.9	0.1	943.488	2104.49
2022-05-17 12:31:26	10263.6	9750.39	0.319812	30487.8	513.179	0.95	0.05	1016.15	1414.15
2022-05-17 14:31:27	10234.4	9722.72	0.31986	30396.8	511.722	0.95	0.05	1107.2	1505.2
2022-05-17 16:31:28	10111.6	9606.04	0.320065	30012.8	505.581	0.95	0.05	1491.19	1889.19
2022-05-17 18:31:29	10002.2	10001.2	0.337071	29671	1.00022	0.9999	0.0001	1832.98	2230.98
2022-05-17 20:31:30	10163.4	10162.4	0.33707	30149.2	1.01634	0.9999	0.0001	2941.81	2037.81

The Trading Bot

After creating models that generate the data the I needed for my trading bot, I had to define a model that invests, or retreats from the market according to the predictions of the data. It was very apparent that I had to create a trading bot that is consequent with my conservative approach to market analysis. So, I decided to create a model that consistently considers the data of both models in time intervals of 2 hours. The logic of the model in this context was, that the predictions of the models both must reach a certain threshold of rise, for the bot to consider the prediction as true and invest the implied amount of the threshold. According to this design, the bot will exit the market if the neural network predicts a fall, but the Linear Regression model doesn't, but on the other hand the bot will only invest the whole of the portfolio if both predictions surpass the regarding threshold.

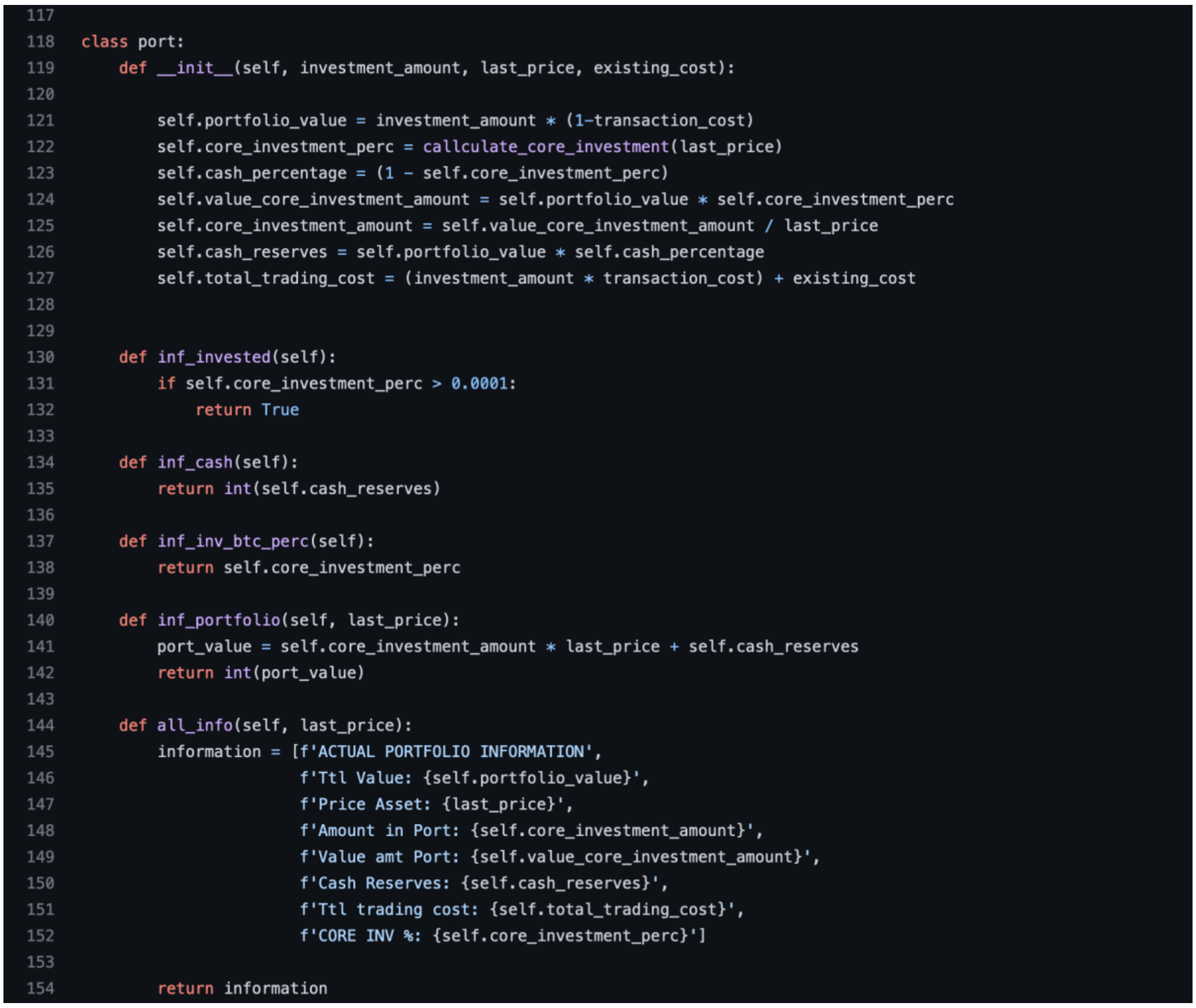


In order to stop the model from investing when the predictions and the real price seem increasingly irrational, I added a Threshold after 3500 $ difference of price predictions and actual price. In the case that that happened I wanted the model to retreat from the market and maybe invest at a point where the market has leveled off. In order to prevent investment within a complete market crash, the bot activates a web-socket connection that would consistently evaluate extreme price dips and exit the market at any time if the price would fall under a threshold.

Threshold invested	**Color**	**Numeric limit in USD**
0 %	Black	3500
99.99 %	Green	1500
95.00 %	Light green	1000
90.00 %	Yellow	500
80.00 %	Orange	200
65.00 %	Red	50
50.00 %	Black	0

trading bot snippet (full code on GitHub)

The trading bot consists of a lot of different mechanics that edit and influence a portfolio that was initialized when creating the bot. Part of the bot consisted of interacting with the databases, as well as with historical price data and a WebSocket to be able to monitor live pricing data.



```python

class port:
    def __init__(self, investment_amount, last_price, existing_cost):

        self.portfolio_value = investment_amount * (1-transaction_cost)
        self.core_investment_perc = callculate_core_investment(last_price)
        self.cash_percentage = (1 - self.core_investment_perc)
        self.value_core_investment_amount = self.portfolio_value * self.core_investment_perc
        self.core_investment_amount = self.value_core_investment_amount / last_price
        self.cash_reserves = self.portfolio_value * self.cash_percentage
        self.total_trading_cost = (investment_amount * transaction_cost) + existing_cost


    def inf_invested(self):
        if self.core_investment_perc > 0.0001:
            return True

    def inf_cash(self):
        return int(self.cash_reserves)

    def inf_inv_btc_perc(self):
        return self.core_investment_perc

    def inf_portfolio(self, last_price):
        port_value = self.core_investment_amount * last_price + self.cash_reserves
        return int(port_value)

    def all_info(self, last_price):
        information = [f'ACTUAL PORTFOLIO INFORMATION',
                       f'Ttl Value: {self.portfolio_value}',
                       f'Price Asset: {last_price}',
                       f'Amount in Port: {self.core_investment_amount}',
                       f'Value amt Port: {self.value_core_investment_amount}',
                       f'Cash Reserves: {self.cash_reserves}',
                       f'Ttl trading cost: {self.total_trading_cost}',
                       f'CORE INV %: {self.core_investment_perc}']

        return information
```

The core part of the bot, from a programming perspective was a portfolio object, which was being edited and influenced according to the information stored from the two models that powered the backend of the structure.

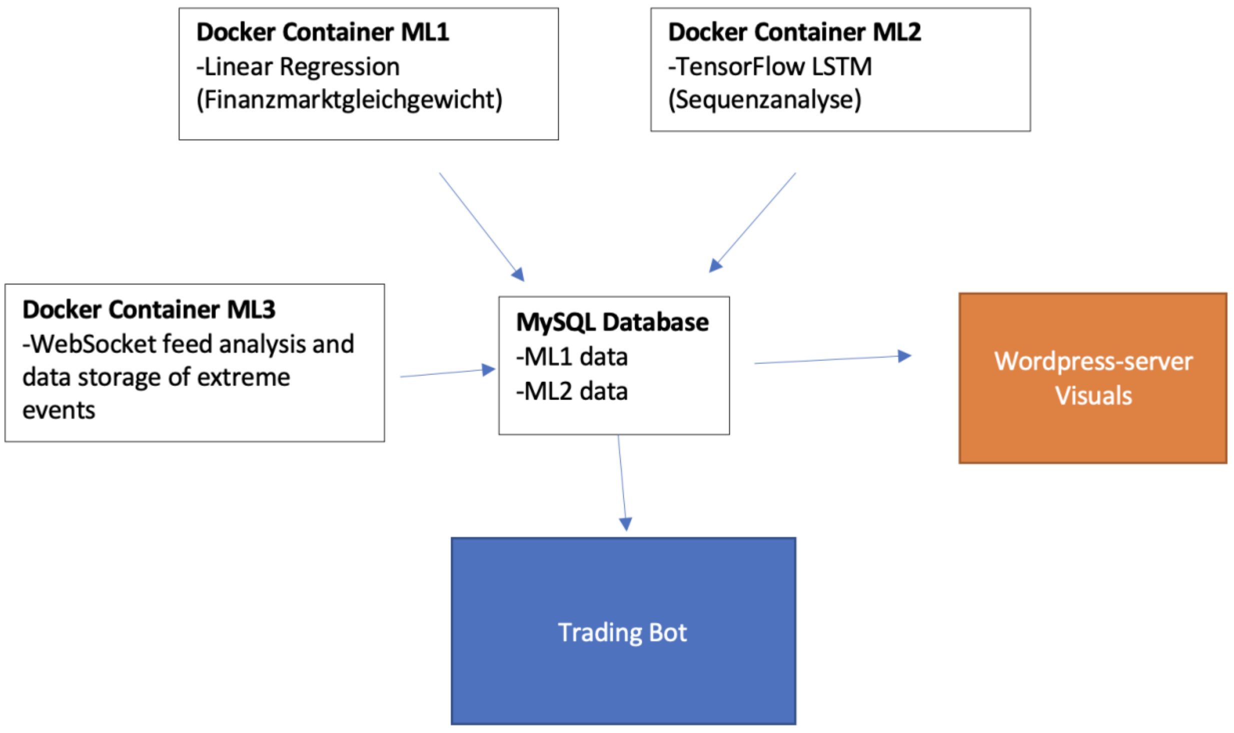
So there really are a lot of moving parts in this concept, and just reading through it might be confusing. That’s why the next chapter aims to clarify the structure as a whole, before we start to dive into the machinations and results that this project has produced.

Overview of the network

At this point in time, the foundation was created to let my model go live, and flourish. Al the databases were created, all the bots were alive and well, and the only thing left to do was to let them conquer the world of data that I prepared for them. Disclaimer: I am fully aware that this sounds extremely nerdy.

To provide a good and visual understanding of how the whole network of bots, databases etc. would work, I decided to create a graph that represents this structure.

Graph X: Overview of the whole Network



The network of bots worked according to the structure that is displayed. Every bot was running in a Docker Container, generating data, and storing it in a central database. Consequentially the data was accessed by the trading bot to make decisions whether to buy or to sell the asset. I also launched a WordPress server that displayed the events that were happening in real time to be able to access information regarding the project from wherever I was.

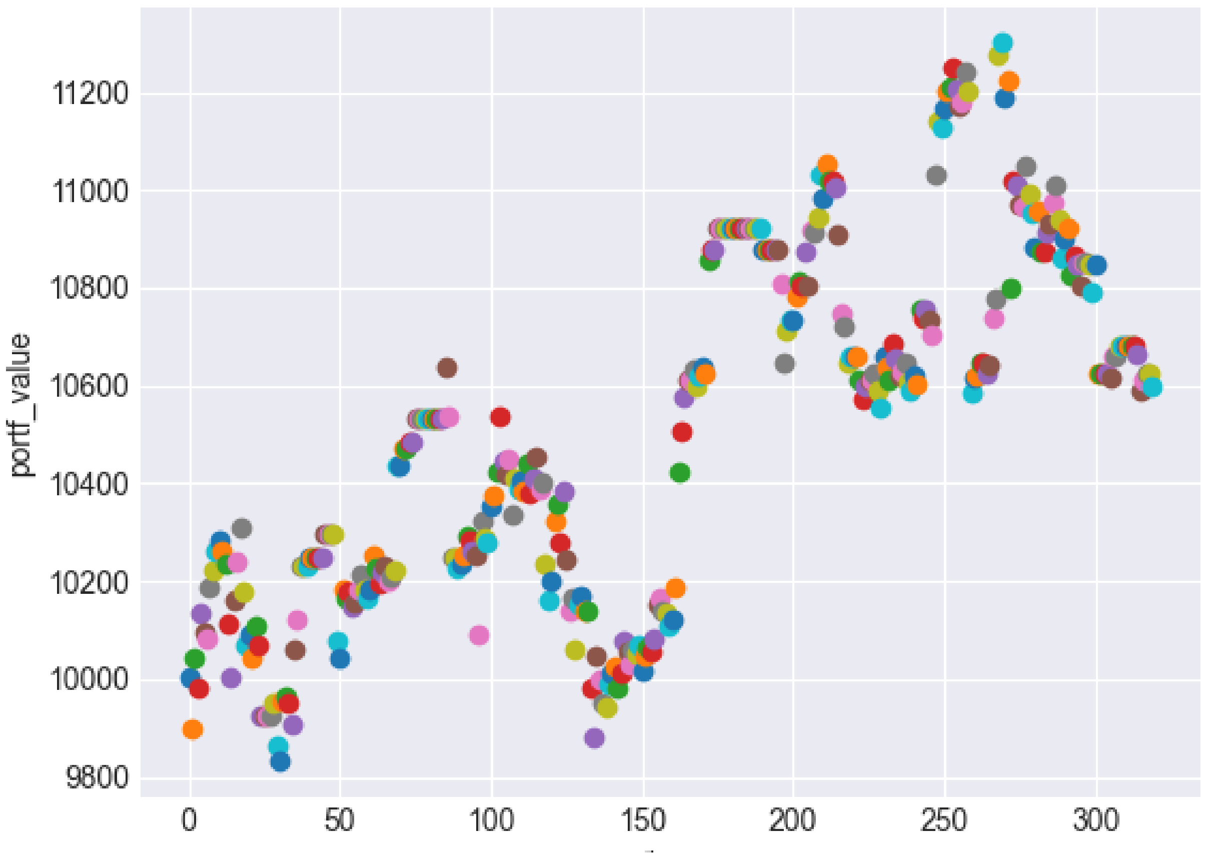
Results

The project ran a total of 27 days, where certain parameters of regarding runtime and activation were consistently adjusted by me to increase the performance of the model. The final version of the bot fundamentally analyzed the asset, as well as the market 12 times a day, but was consistently active due to a WebSocket connection that permanently analyzed the data-stream to prevent full investment during a complete market crash.

Activations and Volume

Total Bot activations	320
Total number of trades	195
Total BTC trading volume	10.776959025400004
BTC volume bought	5.408011512700003
BTC volume sold	5.3689475127

Graph X: Activations over time and portfolio value

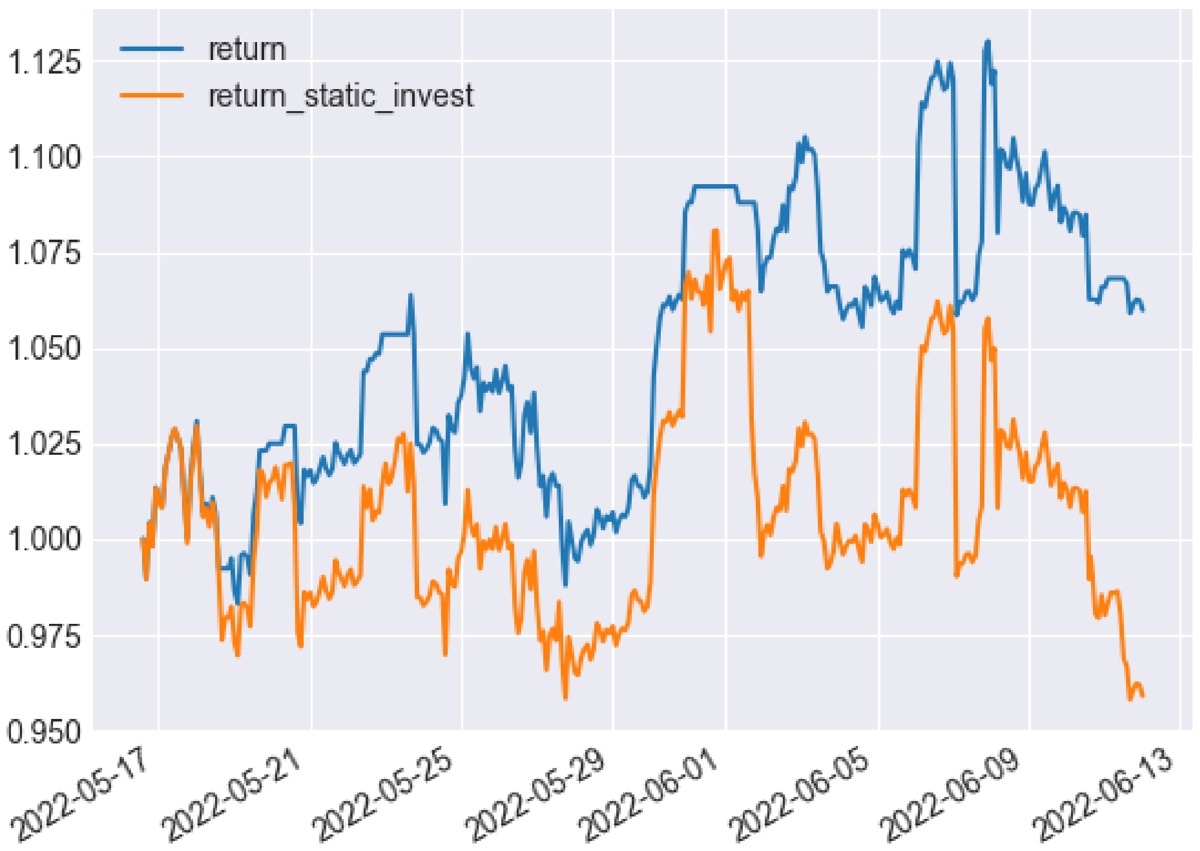


As can be seen in table x, the Bot was activated a number of 320 times. It completed a number of 195 trades within a total trading volume of 10.78 BTC, whereas the volume of BTC bought slightly surpasses the volume of BTC sold.

Return bot vs market

Buy and hold	0.95905759
Bot return	1.05984
Beat market	10.07824059 %
Portfolio value	10598.4
Trading cost	306.99892097
Profit Bot	289.40107903

Graph X: Bot vs Buy and hold



As can be seen in table return bot vs market, the bot achieved a total return of 1.05984, whereas the price of the asset fell 0.95905759. Resulting in the bot beating the market for 10.07824059 %. The total return of the bot was 5,98 %, with trading cost eating up a little more than that, the final revenue was 2,89 %.

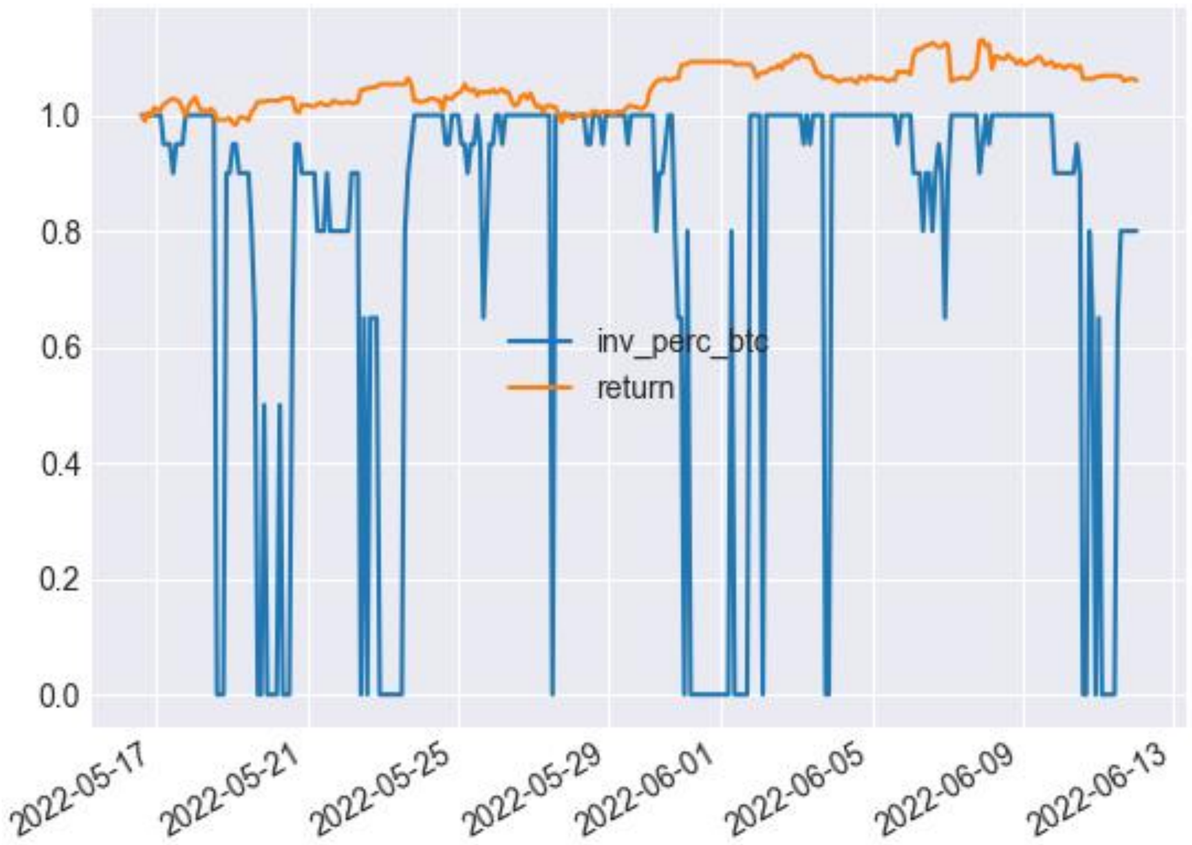
My target of beating the market was surprisingly successful. The model seemed to work better than I ever dreamed of when I first had the idea. On a closer look to the behavior of the model in the market it becomes relatively clear why it is so successful, so let's look.

Behind the scenes of the trading bot

Table X: Distribution of investment percent

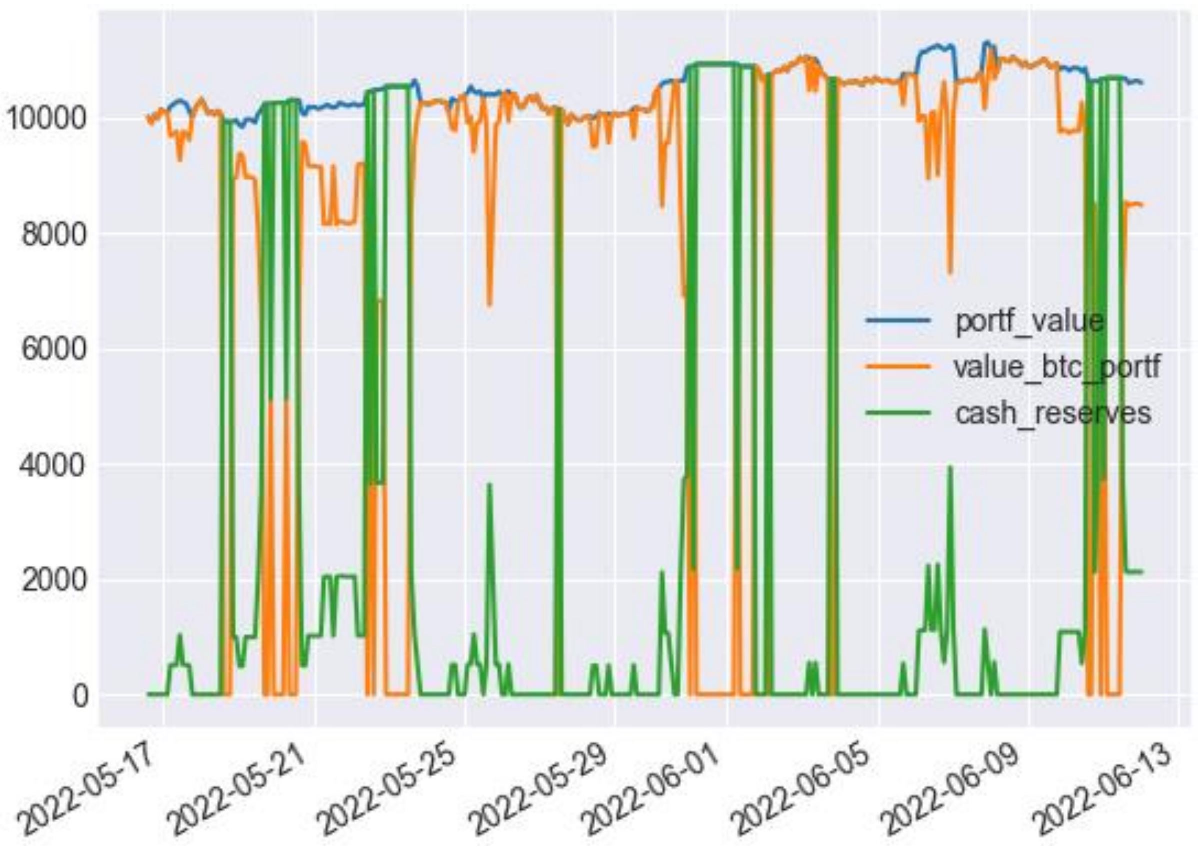
Investment %	**Occurrence**
0.9999	169
0.95	33
0.90	37
0.80	25
0.65	13
0.50	2
0.0001	154
Total cycles	**320**

Graph X: Total portfolio development



As can be seen in Graph, the bot chose to increasingly withdraw from the market, when bitcoin was overvalued, or very near its value according to the predictions of the models.
The table X shows that the decision of complete investment in the market occurred a total of 169/320 cycles, whereas complete withdrawal only occurred in 154/320 cycles.

Graph X: Cash amount vs BTC in portfolio



Graph X displays composition of the portfolio according to the investment decisions of the trading bot. The graph clearly shows that the cash reserves in the portfolio increase, and the amount of BTC in the portfolio decrease, as the asset tends to be overvalued or near it's predicted value.

The combination of the models managed to provide data that was valuable to the trading bot, and therefore the bot could make informed decisions whether the asset was going to rise or fall in the current financial equilibrium and the historical development of the price data.

Conclusion

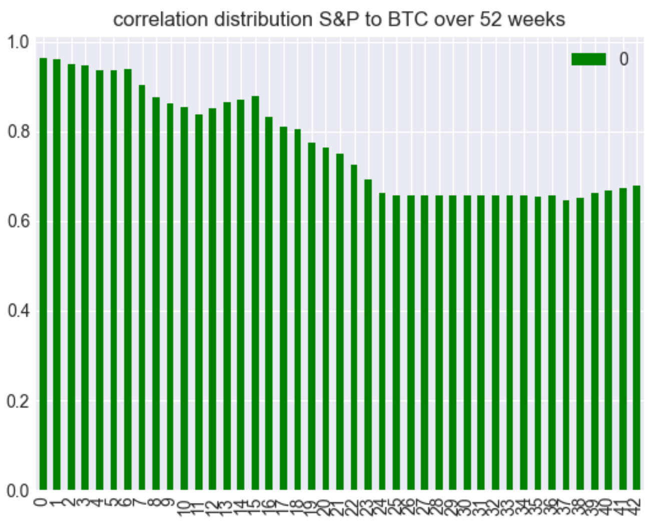
At the end of my little programming experiment, it was safe to say that my guinea pig was alive and well. The portfolio of 10000 € that I declared at the beginning of my journey increased by almost 6% and provided me with a total revenue of around 280 €, considering trading cost, as well as the cost of renting the necessary computational power. Although the experiment was entirely hypothetical, I personally consider it to be a huge victory that improved my skills and speed in programming, was well as an endless list of technologies that I had to use in order to make this project happen.

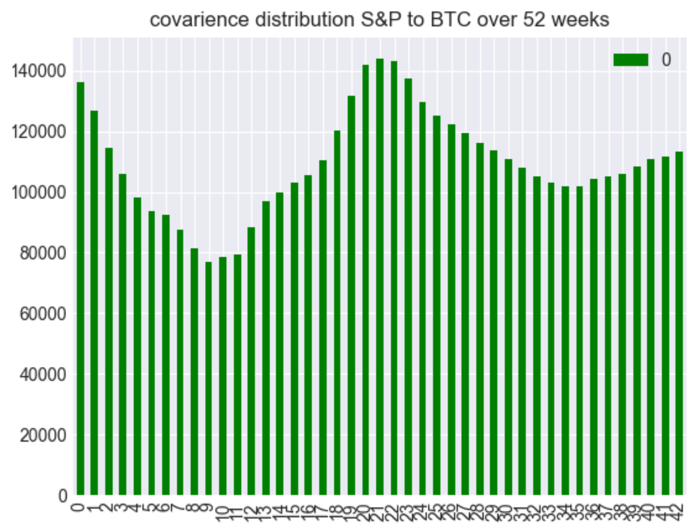
I'm pretty sure that the project still holds a future, as I consider really taking it to the next level and investing some money into the idea if, one day, I have the funds to do so. Till then I have to save up some money and try to increase the efficiency of the model. I want to thank my mam and dad for always believing in my… (just kidding).

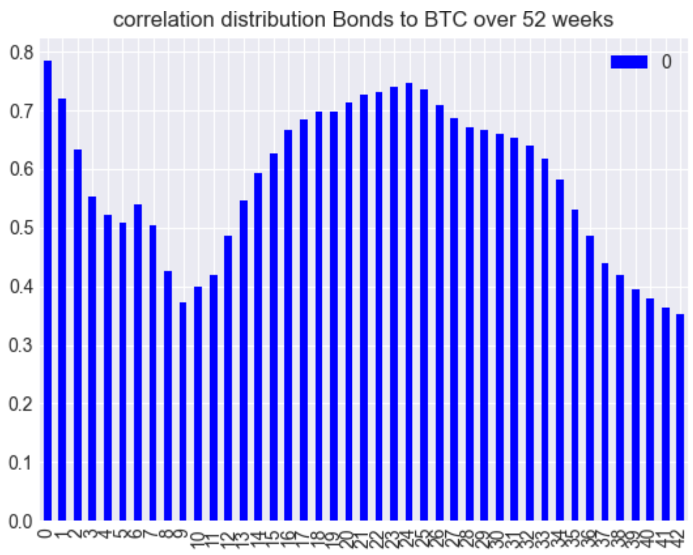
If you want to learn more about the project/idea or if you just want to have a discussion hit me up (peterserver@yahoo.com). Peace.

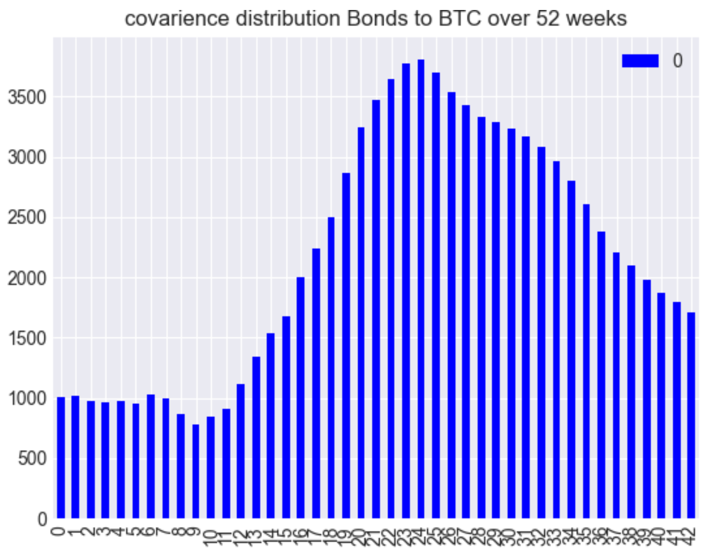
Appendix

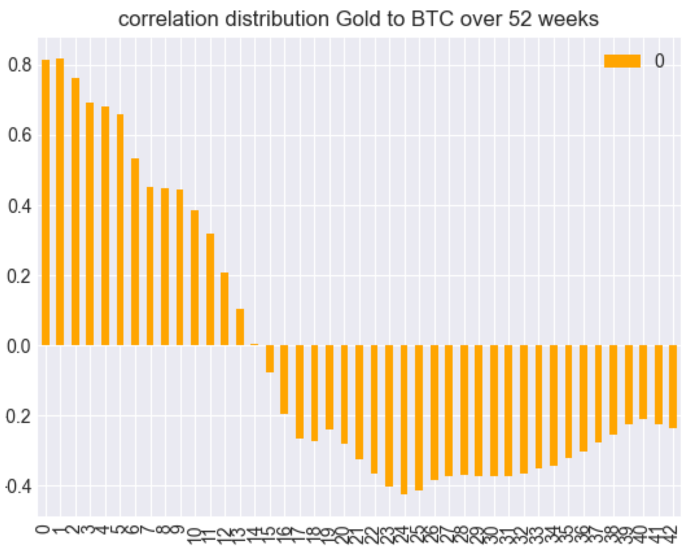
Graph1: correlation distribution over time

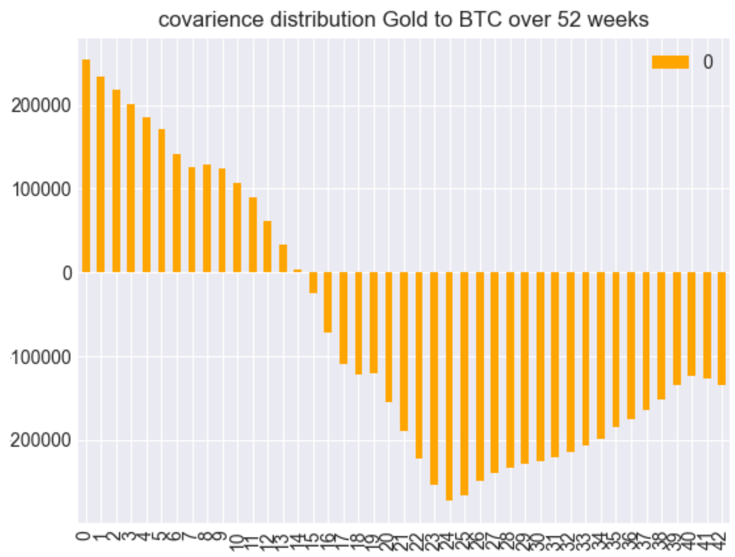


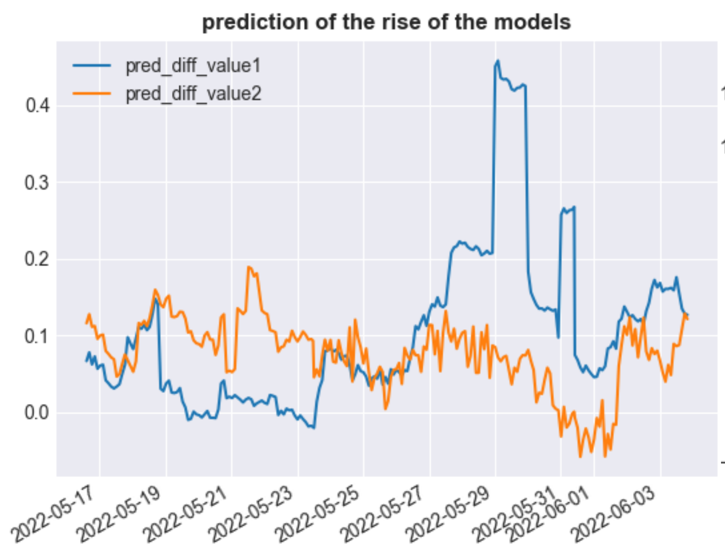


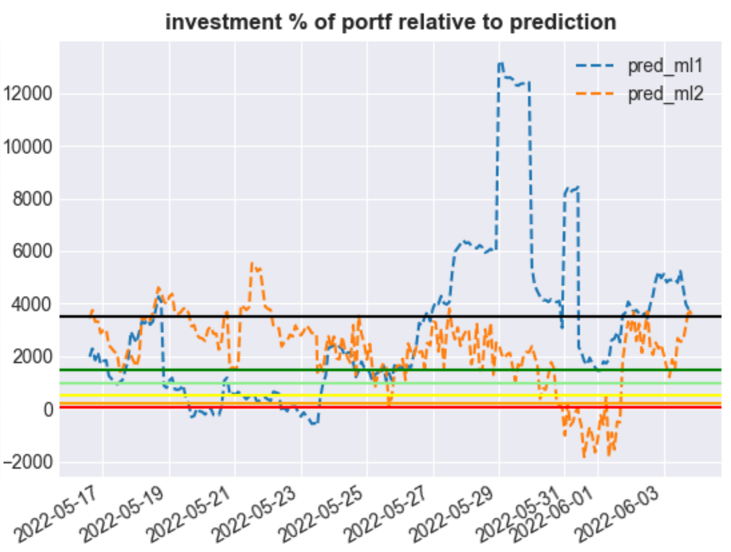


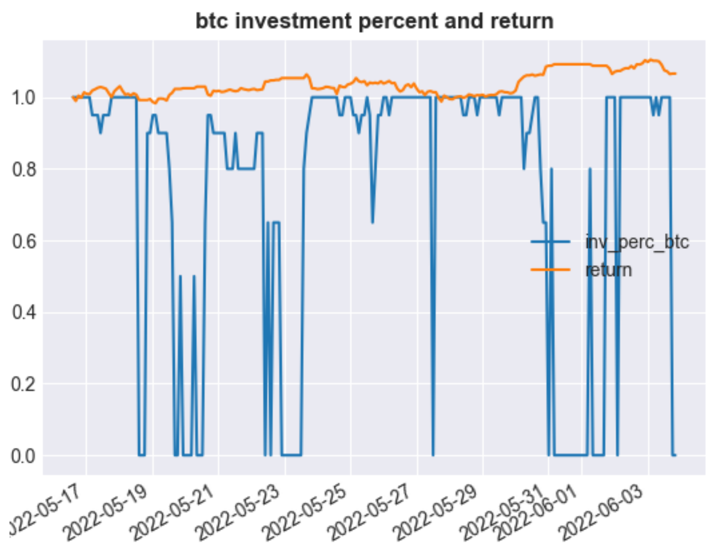


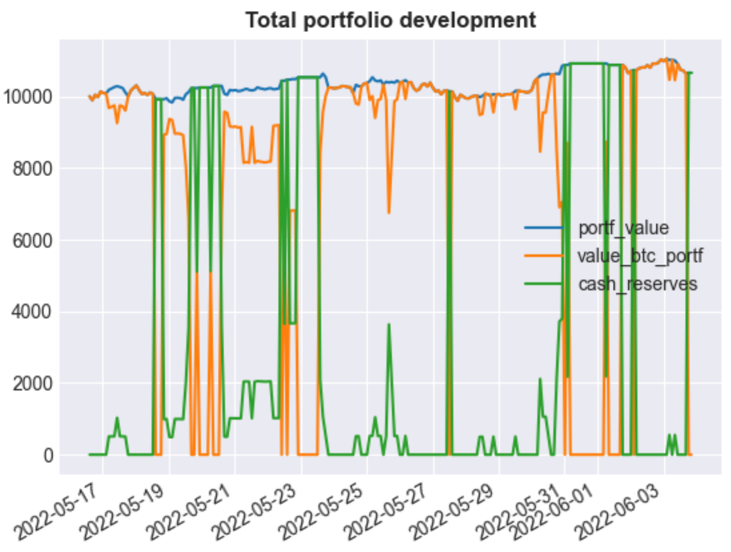


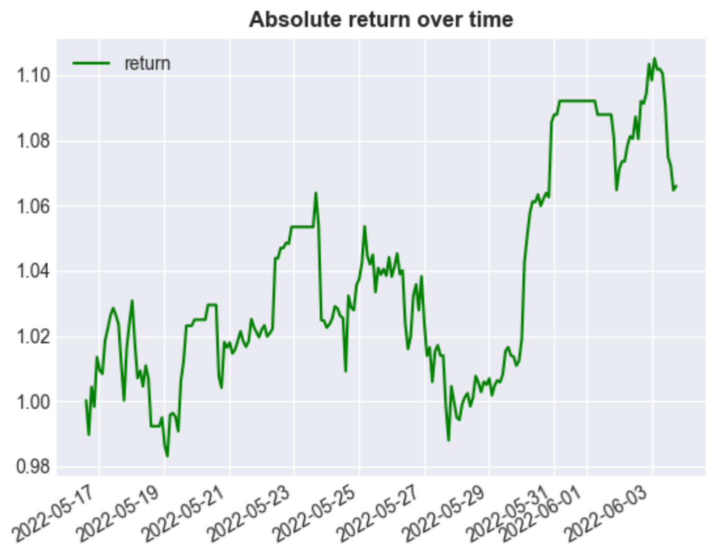


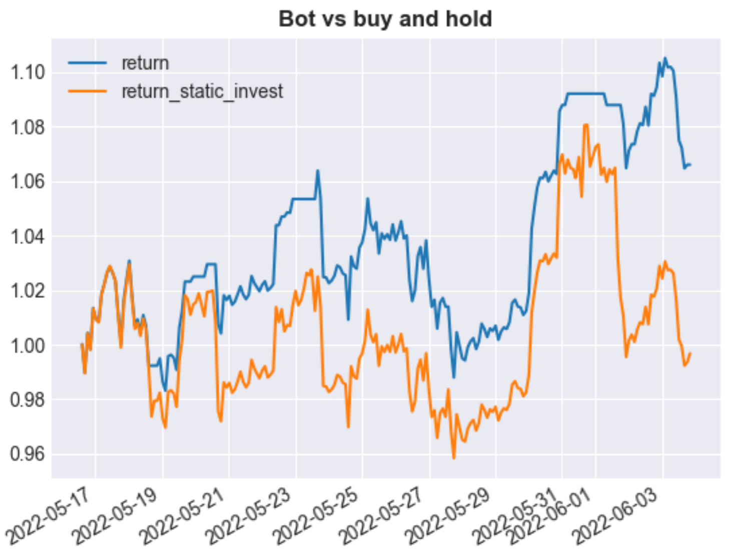

prediction of the rise of the models
pred_diff_value1
pred_diff_value2


investment % of portf relative to prediction
pred_ml1
pred_ml2


btc investment percent and return
inv_perc_btc
return


Total portfolio development
portf_value
value_btc_portf
cash_reserves


Absolute return over time
return


Bot vs buy and hold
return
return_static_invest