DataScientest

# Py The Dip

The Online Portfolio Allocation Project
May 2022 Continuous Data Science Course

**Final Report**

| | |
|---|---|
| **Team Members** | Negah Bremer, Tobias Kessel |
| **Mentor** | Jérémy Robert |

## 1. Introduction

“Open an account at *InvestAlot.*”

“With our app, you can easily find information on stocks! It is completely free of charge!”

YouTube Ad after YouTube Ad of companies trying to get us to invest in the stock market! That is all we see these days. Once you talk to people about stocks, everyone has their own opinion on what is right or wrong. It is either evil and risky or the best chance at making money.

One question that always comes to mind is knowing which stock to choose. How certain are investor decisions? Is it just a combination of gut feeling and luck? Or can we break down decision-making into numbers and algorithms?

Regarding stock market analysis and prediction, different models and methods, such as time series analysis, might seem promising at first. But how accurate are they? Is it worth the risk? How do investors make decisions regarding portfolios? Can you decide which stocks to invest in using an algorithm?

In the following chapters, we will present our method and strategies.

## 2. Project Description

This project aims to implement and benchmark appropriate algorithms for optimum portfolio selection (PS). To facilitate this, the model must recognize different market behaviors and stock relations (correlated, anti-correlated) and adapt its strategy to maximize the ROI.

As the first step, we gathered appropriate datasets representing different market situations. We had to bring the datasets to a level of conformity and uniform structure.

## 3. Data Collection

Our chosen data source was Yahoo Finance (Yfinances). This module supplies sufficient information for our model, like a continuous and consistent time series of opening- and closing prices as well as min- and max-values and trading volume of any given trading day. The stock selection of our first dataset contained the top 25 assets determined by their respective market cap[1].

The initial imported dataset consisted of the following features:

[1] https://companiesmarketcap.com/

| | |
|---|---|
| ● Ticker: | Unique abbreviated ticker name (i.e., AAPL for the stock APPLE) |
| ● Date: | The date for each row |
| ● Close: | Daily closing price |

Note that the combination of 'Ticker' and 'Date' creates a unique identifier for each dataset row. Furthermore, the trading currency is the dollar.

We exported the relevant raw data as CSV. We then chose the stocks that were active between 2010 and 2022, which left us with the following 15 stocks:

| | |
|---|---|
| ● AAPL: | Apple |
| ● MSFT: | Microsoft |
| ● AMZN: | Amazon |
| ● BRKB: | Berkshire-Hathaway B |
| ● UNH: | United Health |
| ● JNJ: | Johnson & Johnson |
| ● V: | VISA |
| ● TSM: | TSMC |
| ● NVDA: | NVIDIA |
| ● XOM: | Exxon Mobile |
| ● WMT: | Walmart |
| ● PG: | Procter & Gamble |
| ● JPM: | JPMorgan Chase |
| ● MA: | Mastercard |
| ● HD: | Home Depot |

## 4. Initial Data Analysis

The data trading information obtained from Yahoo finance is displayed below. To reduce data quality issues within the original raw data, we limited our company stock selection to include companies whose stock has traded since 2010. Each stock selected, therefore, contains 3,145 data points. The data is limited to daily closing prices for various companies from different industries to ensure the broader applicability of the model developed through this project. Here you can see a snippet of the raw data below:

| index | Date | AAPL | MSFT | AMZN | BRK-B | UNH |
|---|---|---|---|---|---|---|
| **0** | 2010-01-04 | 6.535085678 | 23.80022812 | 6.695000172 | 66.22000122 | 26.14505768 |
| **1** | 2010-01-05 | 6.546383381 | 23.80791664 | 6.734499931 | 66.54000092 | 26.10359573 |
| **2** | 2010-01-06 | 6.442254543 | 23.66180801 | 6.612500191 | 66.19999695 | 26.3606472 |
| **3** | 2010-01-07 | 6.430346012 | 23.41573524 | 6.5 | 66.45999908 | 27.37229347 |
| **4** | 2010-01-08 | 6.473095894 | 23.5772171 | 6.676000118 | 66.44000244 | 27.11523247 |

Table 1: Raw data structure

You can also view a snippet of the descriptive statistics of the data used. As you can see, there are significant differences in prices (means, minimums, and maximums) and significant differences in the volatility of the stocks included (as can be seen in the different standard deviations).

| Statistics | AAPL | MSFT | AMZN | BRK-B | UNH |
|---|---|---|---|---|---|
| count | 3145 | 3145 | 3145 | 3145 | 3145 |
| mean | 45.557981 | 89.466519 | 56.741044 | 158.980032 | 164.128877 |
| std | 44.318566 | 85.600014 | 53.927295 | 67.208053 | 130.038835 |
| min | 5.864508 | 17.857403 | 5.4305 | 64.940002 | 23.211792 |
| 25% | 16.448139 | 25.566662 | 13.075 | 99.209999 | 49.580936 |
| 50% | 26.655201 | 48.373642 | 33.239498 | 144.860001 | 116.371002 |
| 75% | 50.637501 | 122.509216 | 90.391998 | 204.570007 | 242.243042 |
| max | 181.511703 | 341.606354 | 186.570496 | 359.570007 | 544.069763 |

Table 2: Statistical description of the raw dataset

We also analyzed the various graphs to understand the data at hand better.

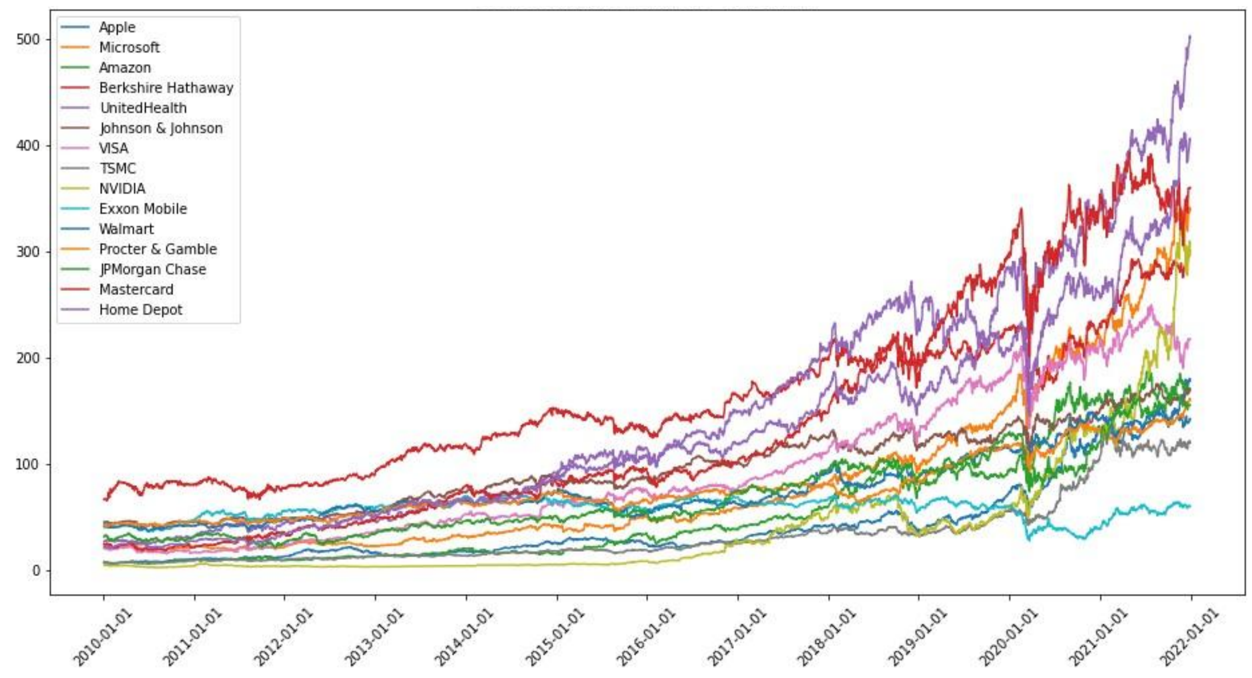


Fig. 1: Stock prices from 1st of January 2010 to 31st of December 2021

Here, we see the graph of the dataset. We can see that a couple of the stock prices diverge significantly from each. One can hardly see the details of the closing price information per stock. That is why we decided to plot the closing prices separately to see how they vary from one company to the other.

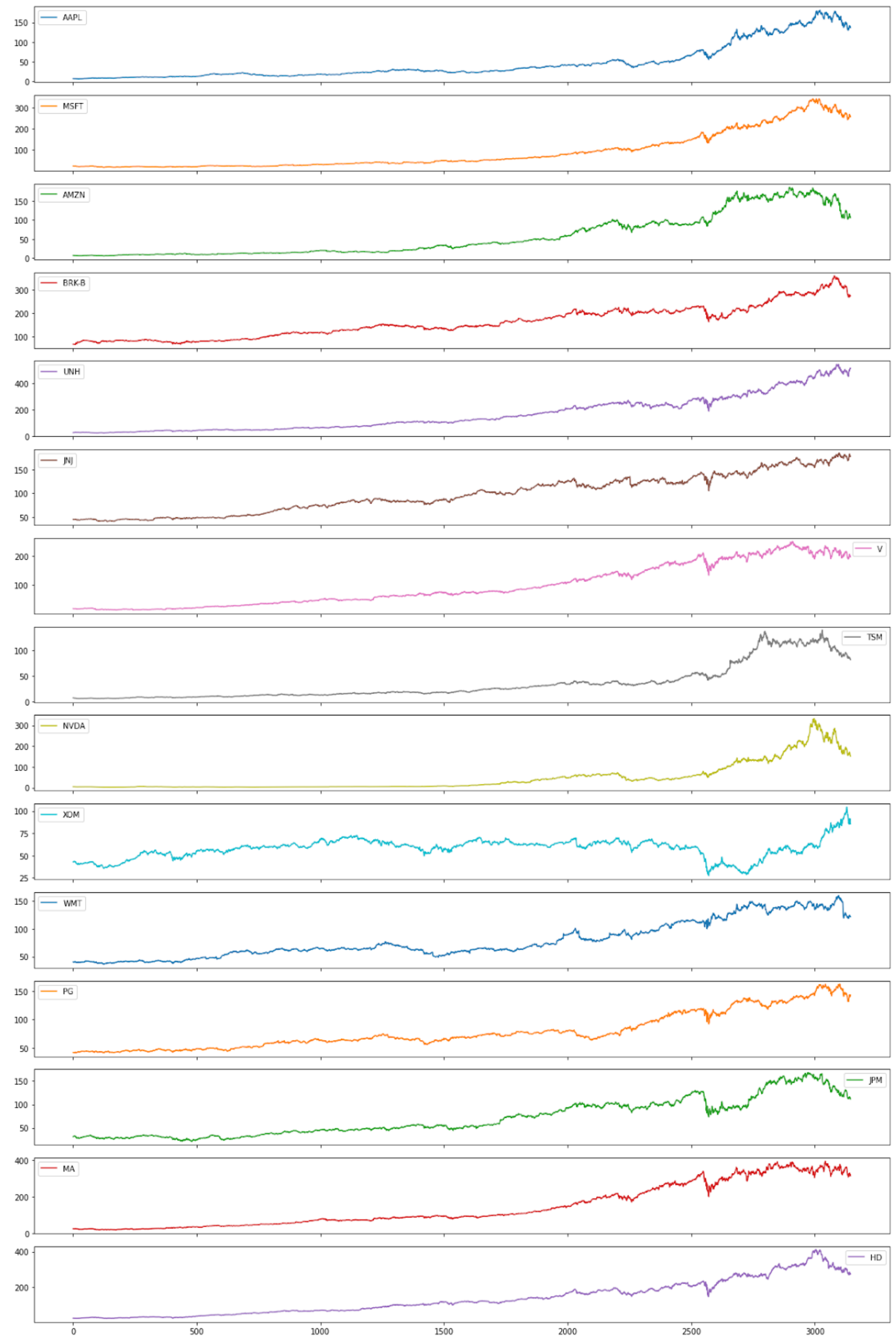


Fig. 2: Stock prices from 1st of January 2010 to 31st of December 2021 for all stock prices. Figure 2 helps us see the plots of the closing prices next to each other and examine the evolution, differences, and similarities between the stocks. We can see if the prices go up and drop simultaneously.

Figure 1 and Figure 2 are not normalized. The values on the y-axis in Figure 2 for each plot are different, and Figure 1 is not that interpretable. Therefore, we scaled the data and plotted the data again, which helped us see the volatility of the stocks and find whether there were any outliers.

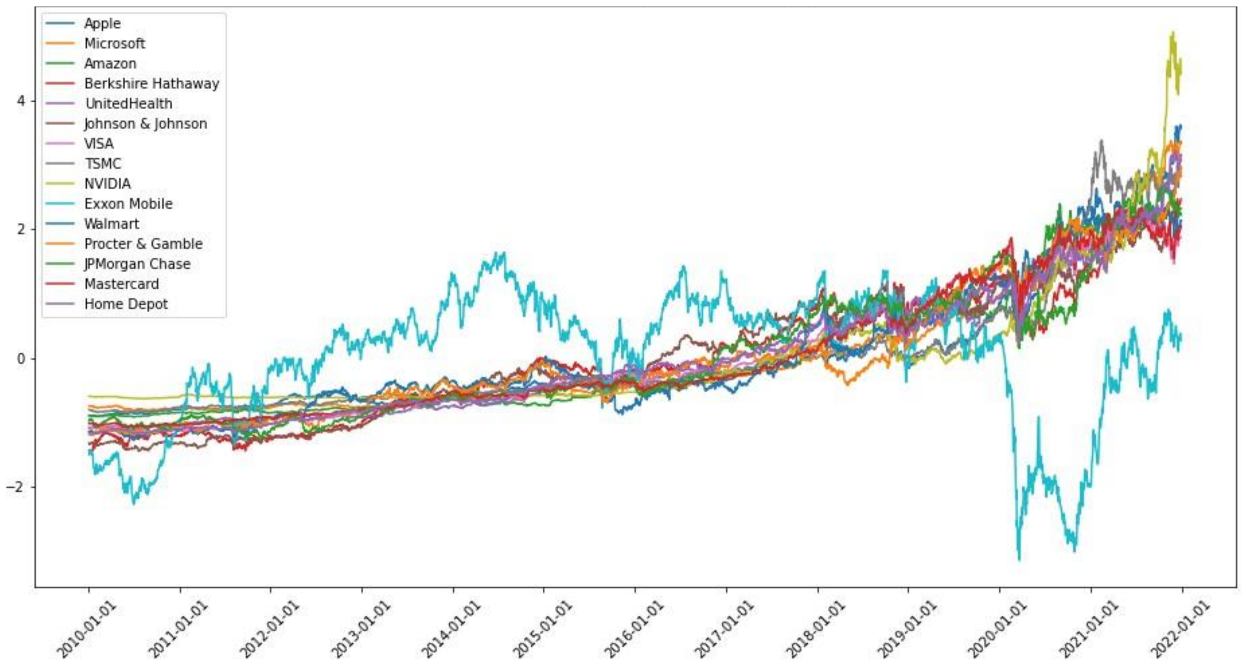


Fig. 3: Normalized Stock prices from 1st of January 2010 to 31st of December 2021

Here, we can see that XOM (Exxon Mobile) behaves entirely differently than the other stocks. We can see that XOM moves in the opposite direction in some periods. The volatility of XOM is also much higher than the other stock data presented in the graph.

We then examined the correlation between the plots and realized that our theories on XOM were valid.

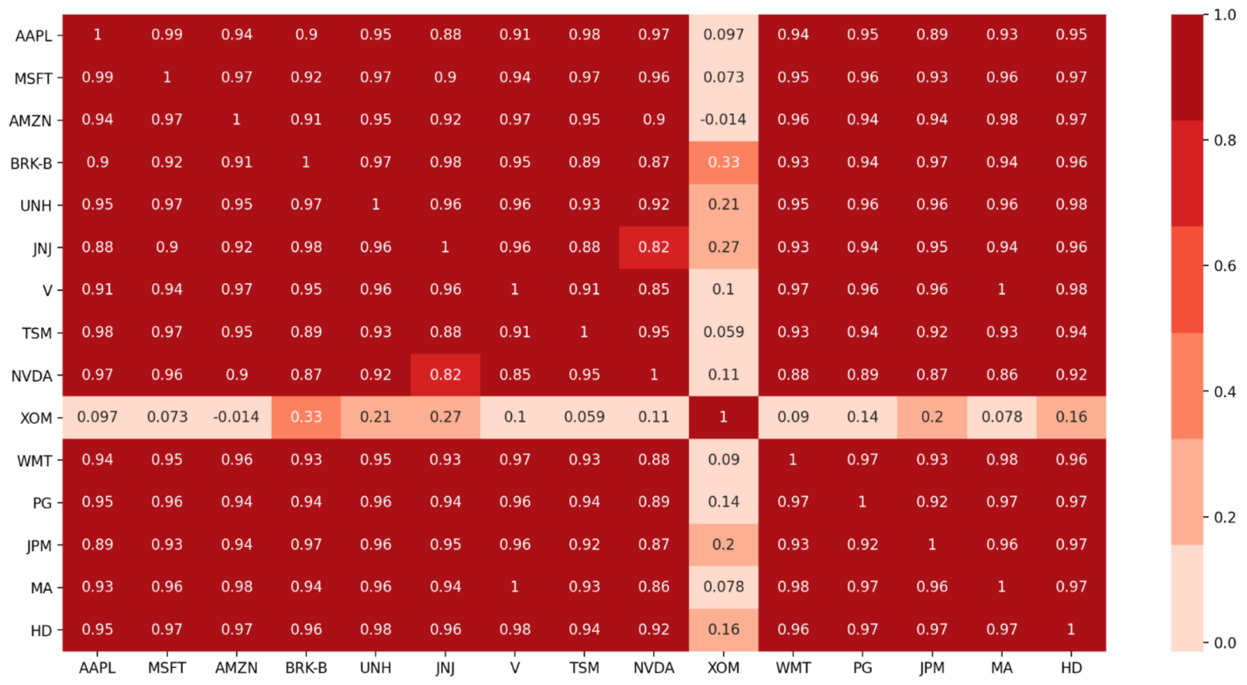


Fig. 4: Correlation heatmap of all stocks (Exxon Mobile [XOM] stands out very clearly).

Figure 4 shows that the closing prices of nearly all stock companies correlate strongly. The only outlier here is XOM.

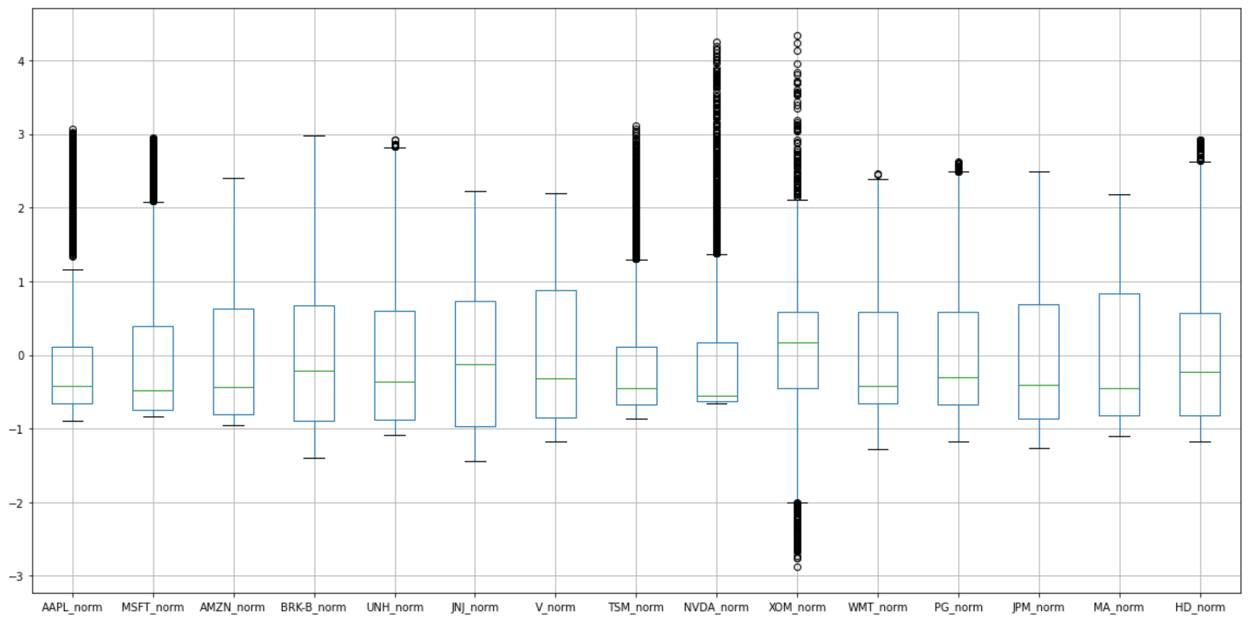


Fig. 5: Box plot of all stocks (Normalized)

To better examine the distribution of the closing prices, we created a box plot of the normalized closing prices for each stock. Here we can see again that the outliers of XOM are different from the others in that they move in both directions.

## 5. Data Preparation / Feature Generation

We used raw data from YFinance and saved it as a CSV-File. The imported dataset consisted of the following features, which has more features compared to the initial dataset mentioned in section 4.

| | |
|---|---|
| ● Ticker: | Unique abbreviated ticker name (i.e., AAPL for the stock APPLE) |
| ● Date: | The date for each row |
| ● Open: | Daily opening price |
| ● Close: | Daily closing price |
| ● Volume: | Trading volume |
| ● High: | Maximum daily trading price |
| ● Low: | Minimum daily trading price |
| ● Dividends: | Number of dividends that got paid at the given day |
| ● Stock Splits: | Determines if a stock split occurred at the given day. (A stock split of 4 means the stock got split with a ratio of 1:4) |

Special attention was put on the fact that future data must not be projected into a present data point. For future pipeline integration, a class called '**stock_feature_generator**' was created that contains several functions for feature generation. This class receives a list of stocks and a threshold value (see Threshold Markers) as input.
The .**fit**-method takes the raw data from YFinance as an argument, and the .**transform**-method returns the transformed data.

The following features are created:

### 5.1. Datetime Features

To get more information from the date feature, we converted the column to datetime and generated several new numerical features like *Year*, *Month*, *Day,* and *Weekday*. The feature *'Weekday'* was subsequently transformed into its corresponding weekday name (column: '*Weekday_Str*') and then discretized into distinct columns.

## 5.2. Rolling Averages

We calculated the rolling averages with a window of 10 and 20 days of the closing prices (.rolling) and set the parameter 'center' to "False" so as not to draw data from future stock development into the present data points.

## 5.3. High and Low Features

A feature called '*High_Low*' was generated that contains the difference between the highest and lowest daily prices and a feature called '*High_Low_2*', which is the mean value of the highest and lowest daily prices, in other words (*'High'* + *'Low'*)/2.

## 5.4. Threshold Markers

This value flags the points where the rolling average (10 or 20 days) exceeds or undershoots the '*Close'*-value of that day by a given threshold.

## 5.5. Intersect Markers

Intersect markers mark where the rolling average (10 or 20 days) intersects with that day's *'Close'* value. The following two figures show the markers for an example of the Berkshire-Hathaway stock with a 10- and 20-day rolling average. The corresponding markers depict values that over- or under-shoot more than 10% of the set threshold and the intersect markers.

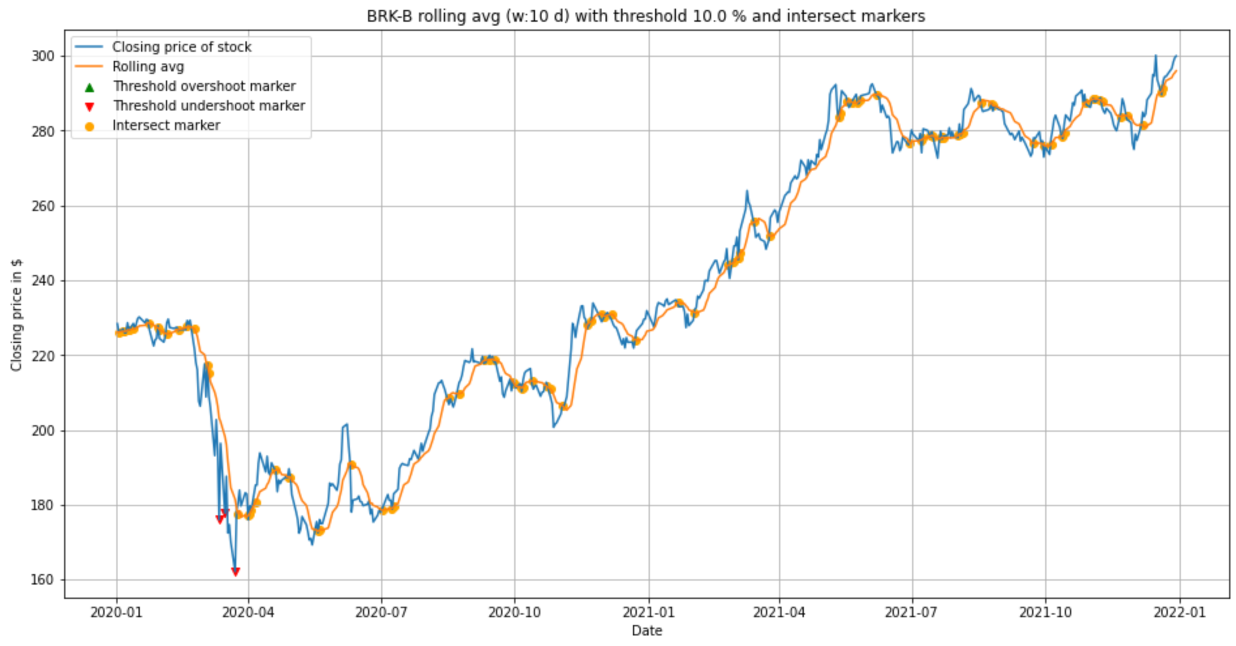


Fig. 6: Example of Berkshire-Hathaway B closing price with a 10-day rolling average and markers

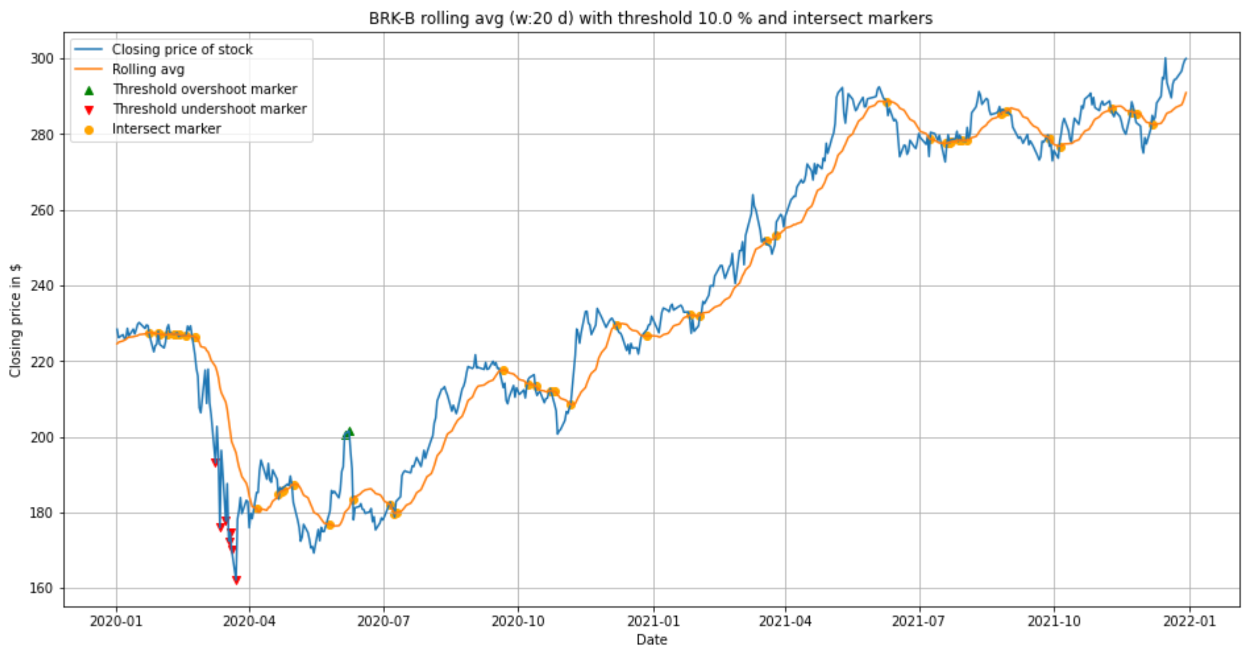


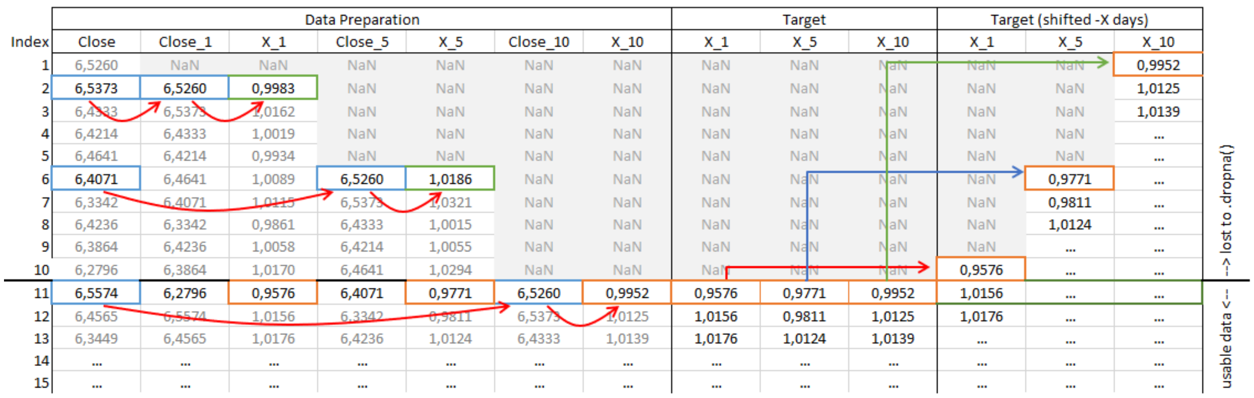
Fig. 7: Example of Berkshire-Hathaway B closing price with a 20-day rolling average and markers

## 5.6. Miscellaneous Time-Shifted Features

The features "High", "Low", and closing price were all shifted into the future (forwards) by 1,5 and 10 days.

## 5.7. Target Variable 'X'

As we want to predict the rise or fall of the closing price of a stock by classifying the data, we need to create features that reflect the stock development.
The feature *'X'* shows the ratio of the closing price to the closing price of a day formerly specified by a desired window. We chose three window values to create the opportunity for exploring the influence of a time delay: 1, 5, and 10 days.
If the ratio X is greater than one, the stock price rises and a value smaller than 1 indicates a price fall. Depending on the result, we created the target for the modeling. The target can either have the value '0' or '1', with '0' meaning a decrease in the closing price and '1' an increase. The label was then shifted back in time by the same number of days specified by the window.


--> lost to .dropna()
usable data <--

| | Data Preparation | | | | | | | Target | | | Target (shifted -X days) | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Index | Close | Close_1 | X_1 | Close_5 | X_5 | Close_10 | X_10 | X_1 | X_5 | X_10 | X_1 | X_5 | X_10 |
| 1 | 6,5260 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 0,9952 |
| 2 | 6,5373 | 6,5260 | 0,9983 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 1,0125 |
| 3 | 6,4333 | 6,5373 | 1,0162 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | 1,0139 |
| 4 | 6,4214 | 6,4333 | 1,0019 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... |
| 5 | 6,4641 | 6,4214 | 0,9934 | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | NaN | ... |
| 6 | 6,4071 | 6,4641 | 1,0089 | 6,5260 | 1,0186 | NaN | NaN | NaN | NaN | NaN | NaN | 0,9771 | ... |
| 7 | 6,3342 | 6,4071 | 1,0115 | 6,5373 | 1,0321 | NaN | NaN | NaN | NaN | NaN | NaN | 0,9811 | ... |
| 8 | 6,4236 | 6,3342 | 0,9861 | 6,4333 | 1,0015 | NaN | NaN | NaN | NaN | NaN | NaN | 1,0124 | ... |
| 9 | 6,3864 | 6,4236 | 1,0058 | 6,4214 | 1,0055 | NaN | NaN | NaN | NaN | NaN | NaN | ... | ... |
| 10 | 6,2796 | 6,3864 | 1,0170 | 6,4641 | 1,0294 | NaN | NaN | NaN | NaN | NaN | 0,9576 | ... | ... |
| 11 | 6,5574 | 6,2796 | 0,9576 | 6,4071 | 0,9771 | 6,5260 | 0,9952 | 0,9576 | 0,9771 | 0,9952 | 1,0156 | ... | ... |
| 12 | 6,4565 | 6,5574 | 1,0156 | 6,3342 | 0,9811 | 6,5373 | 1,0125 | 1,0156 | 0,9811 | 1,0125 | 1,0176 | ... | ... |
| 13 | 6,3449 | 6,4565 | 1,0176 | 6,4236 | 1,0124 | 6,4333 | 1,0139 | 1,0176 | 1,0124 | 1,0139 | ... | ... | ... |
| 14 | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 15 | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |

Table 3: Working principle of data preparation for variable 'X'

## 5.8. Dropping of NAs

The shifting of features and the calculation of X generated several NaN rows at the start of the dataset. We dropped the NaNs to generate a clean and uniform dataset. In total, we dropped 20 rows from the beginning of the record.

## 5.9. All Data Preparation and Feature Generation Steps

Section 5.9 helps understand the steps implemented in data preparation better. Here is a detailed chronological list of functions and classes defined for the first part of the project.

- **Defining the first functions:**
  - `get_yfin_data(date_out,select_tickers,delta=60,date_in=None)`
  - `save_csv_df(df,name)`
  - `load_csv_df(name)`
- **Defining the tickers:**
  - `tickers = {'AAPL':'Apple','MSFT':'Microsoft','AMZN':'Amazon','BRK-B':'Berkshire Hathaway','UNH':'UnitedHealth','JNJ':'Johnson & Johnson', 'V':'VISA','TSM':'TSMC','NVDA':'NVIDIA','XOM':'Exxon Mobile', 'WMT':'Walmart','PG':'Procter & Gamble','JPM':'JPMorgan Chase', 'MA':'Mastercard','HD':'Home Depot'}`
- **Defining the Class Data Feature Generator:** `stock_feature_generator()`
- A desired threshold and the tickers are first set.
  - Functions for creating new features:
    - `add_ra_10`: Create the rolling average of 10 days for closing price

- `add_ra_20`: Create the rolling average of 20 days for closing price
- `add_threshold_under_10`: Values that are 20% smaller than the threshold
- `add_threshold_under_20`: Values that are 10% smaller than the threshold
- `add_threshold_over_10`: Values that are 10% greater than the threshold
- `add_threshold_over_20`: Values that are 20% greater than the threshold
- `add_intersect`: Where data points and threshold meet
- `add_intersect_marker_10d`: Add a marker if C_rav_intersec has a change in sign
- `add_intersect_marker_20d`: Add a marker if C_rav_intersec has a change in sign
- `high_low`: Calculate the difference between *'High'* and *'Low'* (difference between the highest and lowest daily prices)
- `high_add_low_2`: Calculate the average of *'High'* and *'Low'*
- `calc_x_shift`: Calculates the price ratio X of the closing price to a value shifted backward (into past) by the given number of days (default shift=1). Since the first row(s)are the starting point, and there is no value for the ratio, we receive NaNs. If the ratio is greater than one, the price has risen, and if the ratio is smaller than one, the price has fallen. We shifted the target column back by the same amount.
- `translate_weekday`: Transforms the numeric values set for weekdays (0 to 4 for workdays) to actual weekdays (String)
- `dt_features`: Transforms 'Date' to datetime and then creates *'Day', 'Week', 'Month', 'Year'* and *'Weekday', 'Weekday_str'* (created through calling `translate_weekday`)
- `drop_NAs`: Drops all NaNs created by the shift by `calc_x_shift`
- `plot_intersect_thresh`: Shows all intersect and threshold markers on the Closing prices graph

- `time_shift(self,col_list, shift=1)`: Shifts the given columns into the future(forwards), creates a copy of the column that is shifted by the desired number of days
- `fit(self, data)`: Creates a copy of the input data
- `transform(self)`: Calls all the functions mentioned above

## 6. Model

In this section, we will explain our process in finding the right course of action regarding modeling.

### 6.1. Setting Main Goals for the Model

For the first step, we decided to work with the raw data extracted from YFinance. In this section, we only added the values *'X_1'*, *'X_5'*, and *'X_10'* to the dataset. At this stage, we set forecasting as the core goal of our model and decided to later build upon it or change the course depending on the outcome.

We wanted to use classic machine learning models, not time series models such as ARIMA or SARIMAX. The problem with the most common machine learning models, such as decision trees, is that they are only capable of interpolation and are not good at extrapolation. Extrapolation is a big part of forecasting that helps estimate a value outside the training scope. To extrapolate with the classic models and the time series data at hand, we needed to transform the dataset to simulate forecasting. In other words, we had to change the time series data into data suitable for supervised learning. For this simulation, we experienced a method defined by Jason Brownlee mentioned on the platform "Machine Learning Mastery." [2] The algorithm uses the python shift() function. The goal is to use the data from past data points and integrate them as present features.

After transforming the dataset, we took two courses of action: regression and classification. In other words, we could either work on predicting the closing prices or the labels for *'X_1'*, *'X_5'*, and *'X_10'* (whether the closing prices rise or fall in 1 day, 5 days, and 10 days). After discussing the results and upon further examination, we decided to set the main goal as choosing stocks to invest in and portfolio management depending on whether the market value falls or rises.

### 6.2. Overview of Modelling

As mentioned in section 6.1, the model goal we had set needed to be updated. The purpose was not the prediction of closing prices anymore.
We decided to expand the dataset by feature engineering for the following steps. (See section 5)
Since the values *'X_1'*, *'X_5'*, and *'X_10'* indicate whether the price falls or rises, we can consider the task a simple classification problem. This means that we do not need to look at the data as time series or to simulate a forecasting algorithm. At this stage, the goal is to pick a random data point and predict its label (0 or 1), which indicates whether the closing price for that data point has risen or fallen.

[2] https://machinelearningmastery.com/convert-time-series-supervised-learning-problem-python/

First, we needed to iterate over given tickers. In each iteration, desired classification models were trained. A grid search was performed for each model to find the hyperparameters that delivered the best results. This helped us analyze and decide which model and hyperparameters best predict the labels. The process is explained below in detail.

## 6.3. Choosing a suitable Model and Hyperparameter Analysis

The entire dataset was used for the training. However, in the first steps of the hyperparameter analysis, we focused on 4 selected stocks (UNH, NVDA, XOM, and BRK-B) between 01.02.2010 and 31.12.2021.
We chose XOM since it was a low performer compared to other stocks during that time period. We chose NVDA because there was a significant increase in the stock value during 2021. We chose UNH and BRK-B because there is nothing particularly abnormal about them. This way, we could ensure we had a good mix of different scenarios in our dataset.

The numeric values were scaled using a standard scaler since the linear models tend to be affected by non-scaled data. We saved the grid search result for each iteration for later use via joblib, which means we could ensure the conditions remained unchanged and that we did not need to retrain the model each time.

We performed a grid search with a 5-fold cross-validation for each stock that ran through several classifiers and their respective hyperparameters. The goal was to select the hyperparameter combinations that deliver the highest accuracy. Since we are now dealing with a supervised learning and classification model, not a time series, we could randomly split the dataset into training and testing data. (This means we set random_state to a specific number in the train_test_split method).

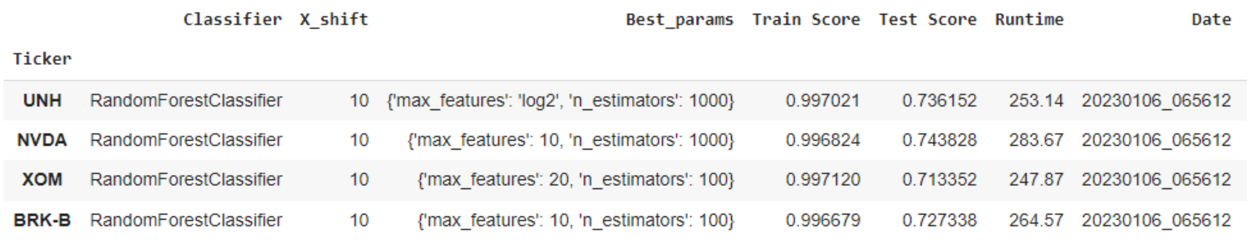
We used the following models and hyperparameters for this step[3]:

| Classifier | Hyperparameters | Range |
|---|---|---|
| RandomForestClassifier | n_estimators<br>max_features | [10,100,1000]<br>['sqrt','log2',1,5,10,20] |
| LogisticRegression | solver<br>penalty<br>c | ['newton-cg','lbfgs','libliniear','sag','saga]<br>['none','11','12','elasticnet']<br>[0.01,0.1,1,10,100] |
| KNeighborsClassifier | n_neighbors<br>metric<br>weights | [1,2,3,4,5,7,9,11,13,15,17,19,21]<br>['euclidian','manhattan','minkowski']<br>['uniform','distance'] |

Table 4: Classifiers and hyperparameters that were used for the first model analysis

We stored the training and testing accuracy in a CSV file for future evaluation. You can see the model scores under the columns '*Train Score*' and '*Test Score*', respectively.

[3] https://machinelearningmastery.com/hyperparameters-for-classification-machine-learning-algorithms/



| Ticker | Classifier | X_shift | Best_params | Train Score | Test Score | Runtime | Date |
|---|---|---|---|---|---|---|---|
| UNH | RandomForestClassifier | 10 | {'max_features': 'log2', 'n_estimators': 1000} | 0.997021 | 0.736152 | 253.14 | 20230106_065612 |
| NVDA | RandomForestClassifier | 10 | {'max_features': 10, 'n_estimators': 1000} | 0.996824 | 0.743828 | 283.67 | 20230106_065612 |
| XOM | RandomForestClassifier | 10 | {'max_features': 20, 'n_estimators': 100} | 0.997120 | 0.713352 | 247.87 | 20230106_065612 |
| BRK-B | RandomForestClassifier | 10 | {'max_features': 10, 'n_estimators': 100} | 0.996679 | 0.727338 | 264.57 | 20230106_065612 |

Table 5: The best combination of X_shift, Classifier, and their respective hyperparameters.

Since the training score is better than the testing score, we could conclude that the model is overfitted to the training data.  The overall testing score is between 72 and 75%. The RandomForestClassifier delivers the best results, especially with a target shift of 10 days. At that point, the best hyperparameter combination was unclear, so we decided to attempt another fine-tuning.

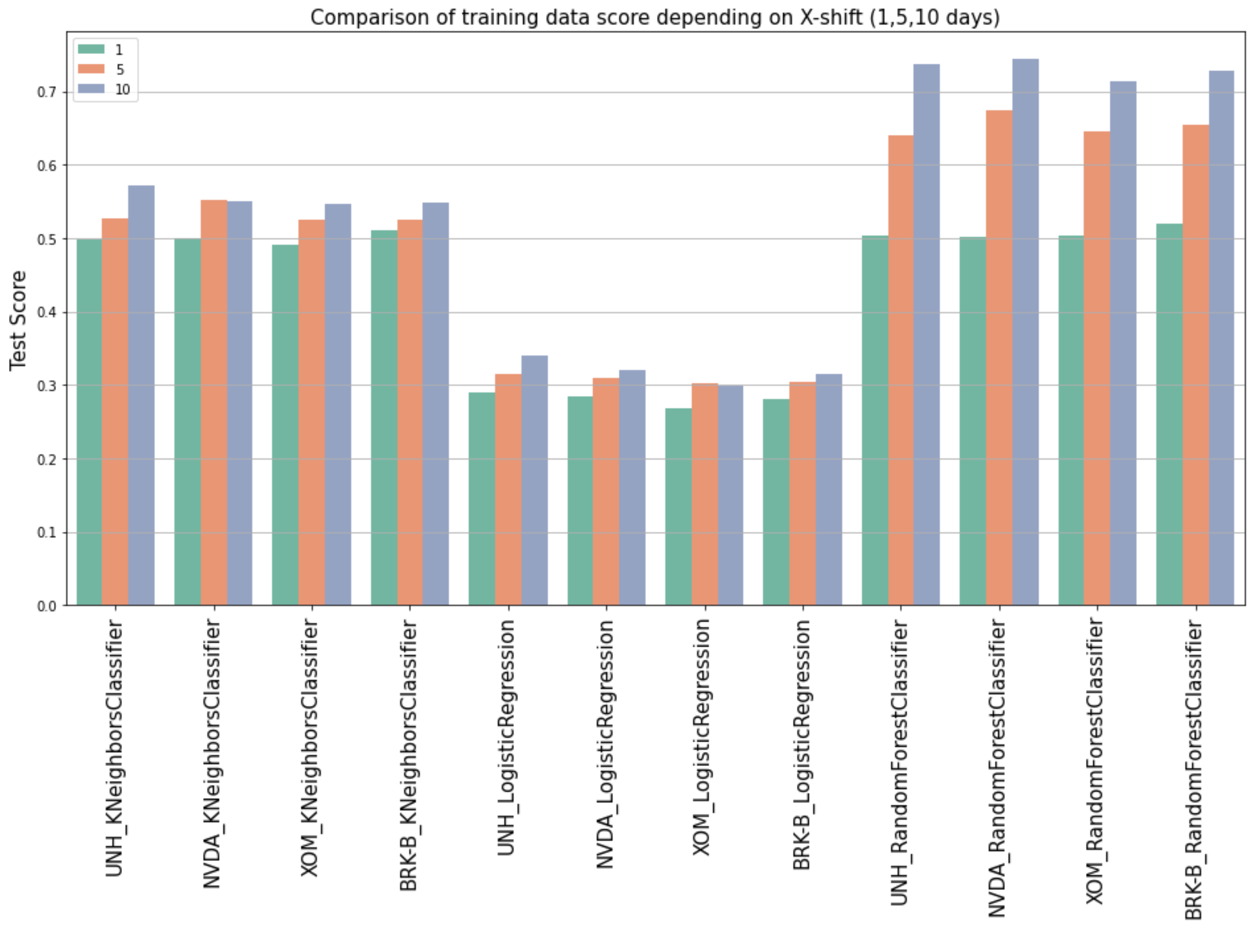


Fig. 8: Training Score over Classifier and X-shift (1,5 or 10 days)

After performing the grid search with the above initial values, we checked whether higher values for max_features would deliver better results. Since we have 39 features, we decided to consider the values 30 and 40 for max_features. Furthermore, we expanded the list of n_estimators by adding the value 500. You can see the tested hyperparameters in the table below:

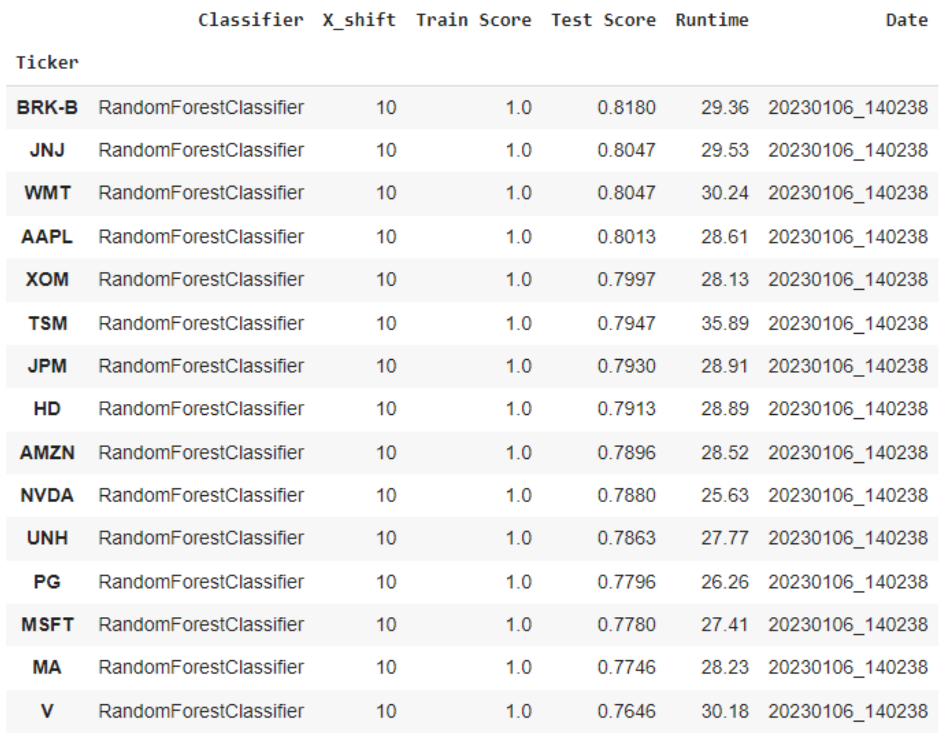
| Classifier | Hyperparameters | Range |
|---|---|---|
| RandomForestClassifier | n_estimators<br>max_features | [100,500,1000]<br>[20,30,40] |

Table 6: Expanding the hyperparameter values

The best hyperparameter combination n_estimators = 1000 and max_features = 20 still delivered the best score, so those parameters were fixed and applied on all stocks to create specific estimators.

## 6.4. Generating Estimators for Each Ticker

After determining the best classifier and its hyperparameters, we built a model for each ticker with a shift of 10 days(*'X_10'*). We saved the results with Joblib. The table below shows the results for each model. The test score is between 0.76 and 0.82 for all tickers.



| Ticker | Classifier | X_shift | Train Score | Test Score | Runtime | Date |
|---|---|---|---|---|---|---|
| BRK-B | RandomForestClassifier | 10 | 1.0 | 0.8180 | 29.36 | 20230106_140238 |
| JNJ | RandomForestClassifier | 10 | 1.0 | 0.8047 | 29.53 | 20230106_140238 |
| WMT | RandomForestClassifier | 10 | 1.0 | 0.8047 | 30.24 | 20230106_140238 |
| AAPL | RandomForestClassifier | 10 | 1.0 | 0.8013 | 28.61 | 20230106_140238 |
| XOM | RandomForestClassifier | 10 | 1.0 | 0.7997 | 28.13 | 20230106_140238 |
| TSM | RandomForestClassifier | 10 | 1.0 | 0.7947 | 35.89 | 20230106_140238 |
| JPM | RandomForestClassifier | 10 | 1.0 | 0.7930 | 28.91 | 20230106_140238 |
| HD | RandomForestClassifier | 10 | 1.0 | 0.7913 | 28.89 | 20230106_140238 |
| AMZN | RandomForestClassifier | 10 | 1.0 | 0.7896 | 28.52 | 20230106_140238 |
| NVDA | RandomForestClassifier | 10 | 1.0 | 0.7880 | 25.63 | 20230106_140238 |
| UNH | RandomForestClassifier | 10 | 1.0 | 0.7863 | 27.77 | 20230106_140238 |
| PG | RandomForestClassifier | 10 | 1.0 | 0.7796 | 26.26 | 20230106_140238 |
| MSFT | RandomForestClassifier | 10 | 1.0 | 0.7780 | 27.41 | 20230106_140238 |
| MA | RandomForestClassifier | 10 | 1.0 | 0.7746 | 28.23 | 20230106_140238 |
| V | RandomForestClassifier | 10 | 1.0 | 0.7646 | 30.18 | 20230106_140238 |

Table 7: Overview of all generated estimators' train and test scores

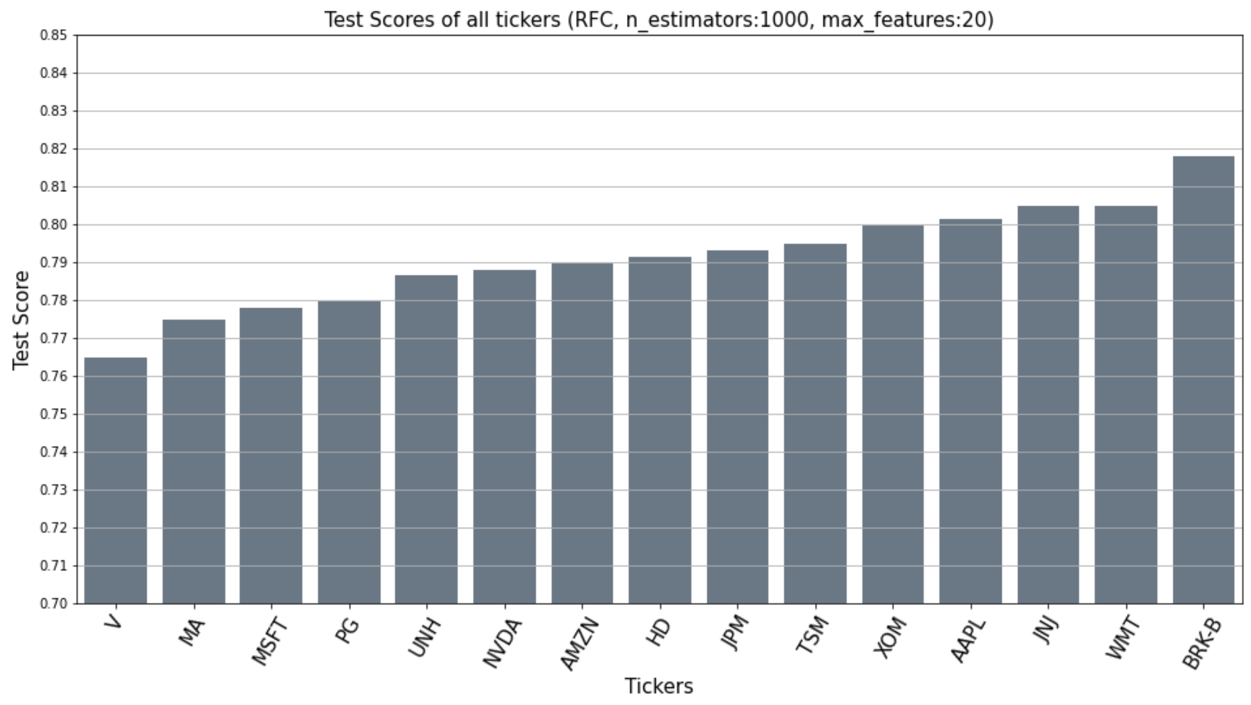


Fig. 9: Barplot of the train scores of all stocks

## 6.5. All Steps Implemented in the Final Model

- Initial classification for all tickers with different models
- Choosing 4 test tickers (UNH, NVDA, XOM, and BRK-B) depending on their performance and data analysis
- Performing a grid search with RandomForestClassifier, LogisticRegression, KNeighborClassifier with initial hyperparameter values
    - Iterating over the four tickers
    - performing a grid search with initial hyperparameter values for three different targets: *'X_1'*, *'X_5'* and *'X_10'.* (The target takes either the value 0 or 1)
    - Creating training and testing data
    - Performing 5-fold cross-validation
- Saving grid search results via joblib
- Creating a bar plot of all grid search results for the four tickers with the different classifiers for all three targets
- Finding the best combination: target = *'X_10',* RandomForestClassifier(n_estimators = 1000, max_features=20)
- Testing other hyperparameter values

- Training all 15 tickers with the best classifier and saving the results with joblib

## 7. Portfolio Manager Design and Working Principle

### 7.1. Object Generation

For portfolio management, a class called **'Portfolio'** was created. It takes the following parameters as input variables.

| | |
|---|---|
| data: | Raw data from YFinance - For performance reasons, the data is supplied by a pre-saved DataFrame. |
| tickers: | A list of tickers that can be traded and must be compatible with the YFinance denominations. The list remains constant throughout the whole investment period. |
| start_invest: | Amount of money that is available for the portfolio. |
| sum_pre_invest: | Purchase cap per trade |
| buy_threshold: | Hyperparameter that reacts on the predicted probability of a stock rise |
| sell_threshold: | Hyperparameter that reacts to the predicted probability of a stock fall |
| min_buy_signal: | Amount of consecutive buy signals that have to occur to allow a stock purchase |
| min_sell_signal: | Amount of consecutive sell signals that have to occur to allow a stock sale |
| silent_mode: | Toggles silent mode (less information) |

Example:

```
port_tickers=['HD','AAPL','MSFT']

port=Portfolio(data=get_yfin_data('2022-12-31',port_tickers,date_in='2022-
01-01'),

               buy_threshold=0.65,

               sell_threshold=0.4,

               tickers=port_tickers,

               buy_signal=3,

               sell_signal=1,

               sum_per_invest=1000,

               start_invest=10000,
                silent_mode=True
                )
```

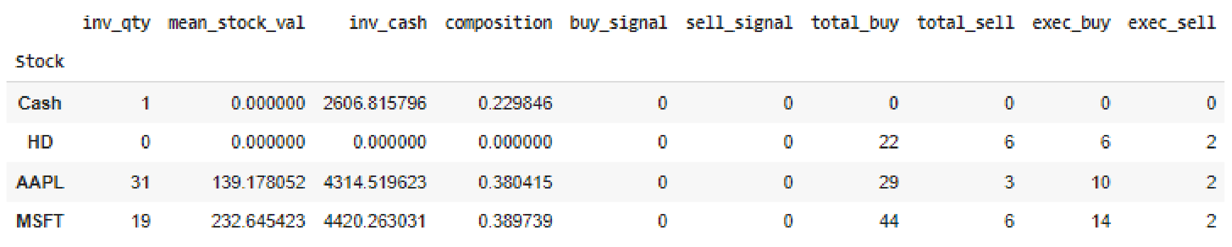
## 7.2. Overview of the Main Functions

The portfolio manager can run a complete investment cycle of any new given number of days supplied to the algorithm. A usual sequence works as follows:

- ➢ Generating the portfolio object
- ➢ Data preparation (**.prepare()**) Inline data preparation as mentioned above
- ➢ Creating a ticker_dictionary that contains all pre-fitted models of the parsed tickers (**.create_ticker_dict()**)
  The pre-fitted estimators are supplied from a separate folder and loaded via joblib according to the requested tickers
- ➢ Calculating the probability that target X is in the class 0 or 1 for each ticker (**.predict()**)
  The probability is calculated via predict_proba. For making buy/sell decisions, only the second column is used (y_pred_proba[1])
- ➢ Running the investment cycle starting from a specified date (**.invest(date)**)
  The .invest-function covers the most critical part of the portfolio management. It iterates through all given dates of the input dataset (data).
- ➢ Creating a report that shows investment outcomes as well as graphs (**.report()** & **.graphical_report()**)

## 7.3. 'Portfolio' DataFrame

When you execute the .invest-function, a sell or buy marker is assigned to each date and stock according to the corresponding prediction probability and the threshold set. The "buy-" or "sell-signals" are saved in an internal DataFrame called portfolio:



| Stock | inv_qty | mean_stock_val | inv_cash | composition | buy_signal | sell_signal | total_buy | total_sell | exec_buy | exec_sell |
|---|---|---|---|---|---|---|---|---|---|---|
| Cash | 1 | 0.000000 | 2606.815796 | 0.229846 | 0 | 0 | 0 | 0 | 0 | 0 |
| HD | 0 | 0.000000 | 0.000000 | 0.000000 | 0 | 0 | 22 | 6 | 6 | 2 |
| AAPL | 31 | 139.178052 | 4314.519623 | 0.380415 | 0 | 0 | 29 | 3 | 10 | 2 |
| MSFT | 19 | 232.645423 | 4420.263031 | 0.389739 | 0 | 0 | 44 | 6 | 14 | 2 |

Table 8: Example of the layout of the portfolio DataFrame

This DataFrame consists of one row for each traded stock, as well as an entry called 'Cash' that contains the amount of cash available for the portfolio. Each stock entry contains columns that make up the portfolio composition:

| | |
|---|---|
| ● inv_qty: | Current amount invested in the stock |
| ● mean_stock_val: | Mean stock value, adjusted at every purchase |
| ● inv_cash: | inv_qty * mean_stock_val |
| ● composition: | Composition of the portfolio. The maximum composition of the portfolio is limited for each stock (see 7.5) |
| ● buy_signal: | Increases if the predicted probability exceeds the buy_threshold. This value is reset if the purchase is executed. |
| ● sell_signal: | Sell Signal - It increases if the predicted probability falls below the sell_threshold. This value is reset after each sale. |
| ● total_buy: | Total buy signals |
| ● total_sell: | Total sell signals |
| ● exec_buy: | Number of executed purchases |
| ● exec_sell: | Number of executed sales |

The portfolio DataFrame is updated with each iteration.

## 7.4. 'Logfile' Dataframe

A log file DataFrame is generated that saves an entry for each day and stock and supports the graph generation after the investment function is executed.

| | Date | Stock | Buy_Signal | Sell_Signal | Action | inv_qty_cum | inv_cash_cum | mean_stock_val | Close | Port_total | Book_value |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | 2022-03-01 | MSFT | 1 | 0 | - | 0 | 0.000000 | 0.000000 | 292.816162 | 10000.00000 | 0.000000 |
| 1 | 2022-03-01 | HD | 0 | 0 | - | 0 | 0.000000 | 0.000000 | 312.390686 | 10000.00000 | 0.000000 |
| 2 | 2022-03-01 | AAPL | 0 | 0 | - | 0 | 0.000000 | 0.000000 | 162.465012 | 10000.00000 | 0.000000 |
| 3 | 2022-03-02 | MSFT | 2 | 0 | B | 3 | 894.054749 | 298.018250 | 298.018250 | 10000.00000 | 894.054749 |
| 4 | 2022-03-02 | AAPL | 0 | 0 | - | 0 | 0.000000 | 0.000000 | 165.809891 | 10000.00000 | 0.000000 |
| ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... | ... |
| 571 | 2022-11-30 | HD | 0 | 0 | - | 0 | 0.000000 | 0.000000 | 323.989990 | 11341.59845 | 0.000000 |
| 572 | 2022-11-30 | MSFT | 0 | 0 | - | 19 | 4420.263031 | 232.645423 | 255.139999 | 11341.59845 | 4847.659988 |
| 573 | 2022-12-01 | AAPL | 0 | 0 | - | 31 | 4314.519623 | 139.178052 | 148.309998 | 11341.59845 | 4597.609924 |
| 574 | 2022-12-01 | HD | 0 | 0 | - | 0 | 0.000000 | 0.000000 | 327.070007 | 11341.59845 | 0.000000 |
| 575 | 2022-12-01 | MSFT | 0 | 0 | - | 19 | 4420.263031 | 232.645423 | 254.690002 | 11341.59845 | 4839.110046 |

Table 9: Layout of the log file that keeps changes in the portfolio for each day and stock

Most importantly, the log file saves the cumulative data needed to show portfolio development:

| | |
|---|---|
| ● Action: | B: Buy, S: Sell, S10: Sell per 10% yield |
| ● inv_qty_cum: | Current stock quantity in the portfolio |
| ● Close: | Current closing price of that stock of that day |
| ● Port_total: | Total portfolio value |

- Book_value: Book value of that stock corresponding to the actual closing price of that day

## 7.5. Purchase and Sale Decisions

The model iterates over each day and sets a buy or sell marker for each stock. The buy and sell markers depend on the chosen threshold. If the signals occur consecutively, the model adds them up. The model resets them once a buy- or sell-signal streak breaks.

Suppose a buy/sell counter exceeds its minimum signal threshold (min_buy_signal & min_sell_signal as supplied). In this case, purchase or sale actions are conducted according to the following rules and limitations:

- A sale cannot exceed the composition limit of the portfolio:

| The total number of traded stocks | Composition Limit |
|---|---|
| 2 | 60% |
| 3 | 40% |
| >3 | 30% |

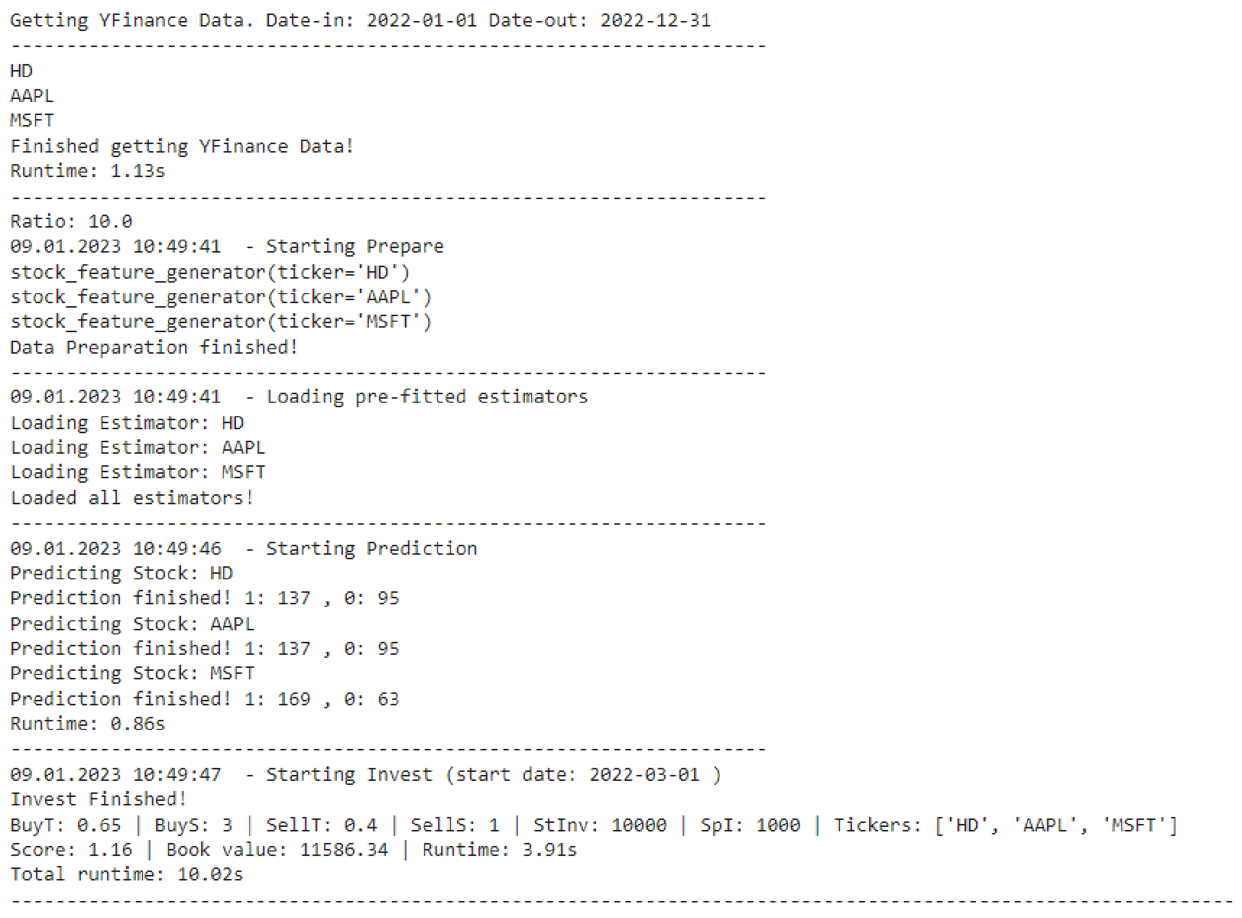
Table 10: Rules that moderate and stop the mass purchase of one stock to the disadvantage of other stocks

- If several purchases are planned, and there is not enough cash for all actions, choose the stock with the highest increase probability (y_proba).
- If there is insufficient cash to buy the total amount specified by the parameter sum_per_invest, purchase a smaller amount.
- If a sale signal for a specific stock in the portfolio occurs, sell the entire amount invested in that stock.
- If a stock value has increased by at least 10%, sell the invested amount.

## 7.6. Outputs and Graphical Reports

In this example, the stocks ‘HD’, ’AAPL & ‘MSFT’ were arbitrarily chosen but later kept for comparison reasons (see chapter ‘Portfolio Manager Hyperparameter Tuning & Analysis’).

This output is shown when the portfolio management is executed entirely with the parameters mentioned in section 7.1.



```
Getting YFinance Data. Date-in: 2022-01-01 Date-out: 2022-12-31
-------------------------------------------------------------------
HD
AAPL
MSFT
Finished getting YFinance Data!
Runtime: 1.13s
-------------------------------------------------------------------
Ratio: 10.0
09.01.2023 10:49:41  - Starting Prepare
stock_feature_generator(ticker='HD')
stock_feature_generator(ticker='AAPL')
stock_feature_generator(ticker='MSFT')
Data Preparation finished!
-------------------------------------------------------------------
09.01.2023 10:49:41  - Loading pre-fitted estimators
Loading Estimator: HD
Loading Estimator: AAPL
Loading Estimator: MSFT
Loaded all estimators!
-------------------------------------------------------------------
09.01.2023 10:49:46  - Starting Prediction
Predicting Stock: HD
Prediction finished! 1: 137 , 0: 95
Predicting Stock: AAPL
Prediction finished! 1: 137 , 0: 95
Predicting Stock: MSFT
Prediction finished! 1: 169 , 0: 63
Runtime: 0.86s
-------------------------------------------------------------------
09.01.2023 10:49:47  - Starting Invest (start date: 2022-03-01 )
Invest Finished!
BuyT: 0.65 | BuyS: 3 | SellT: 0.4 | SellS: 1 | StInv: 10000 | SpI: 1000 | Tickers: ['HD', 'AAPL', 'MSFT']
Score: 1.16 | Book value: 11586.34 | Runtime: 3.91s
Total runtime: 10.02s
-------------------------------------------------------------------------------------------------------------
```

Fig. 10: Output of the portfolio manager

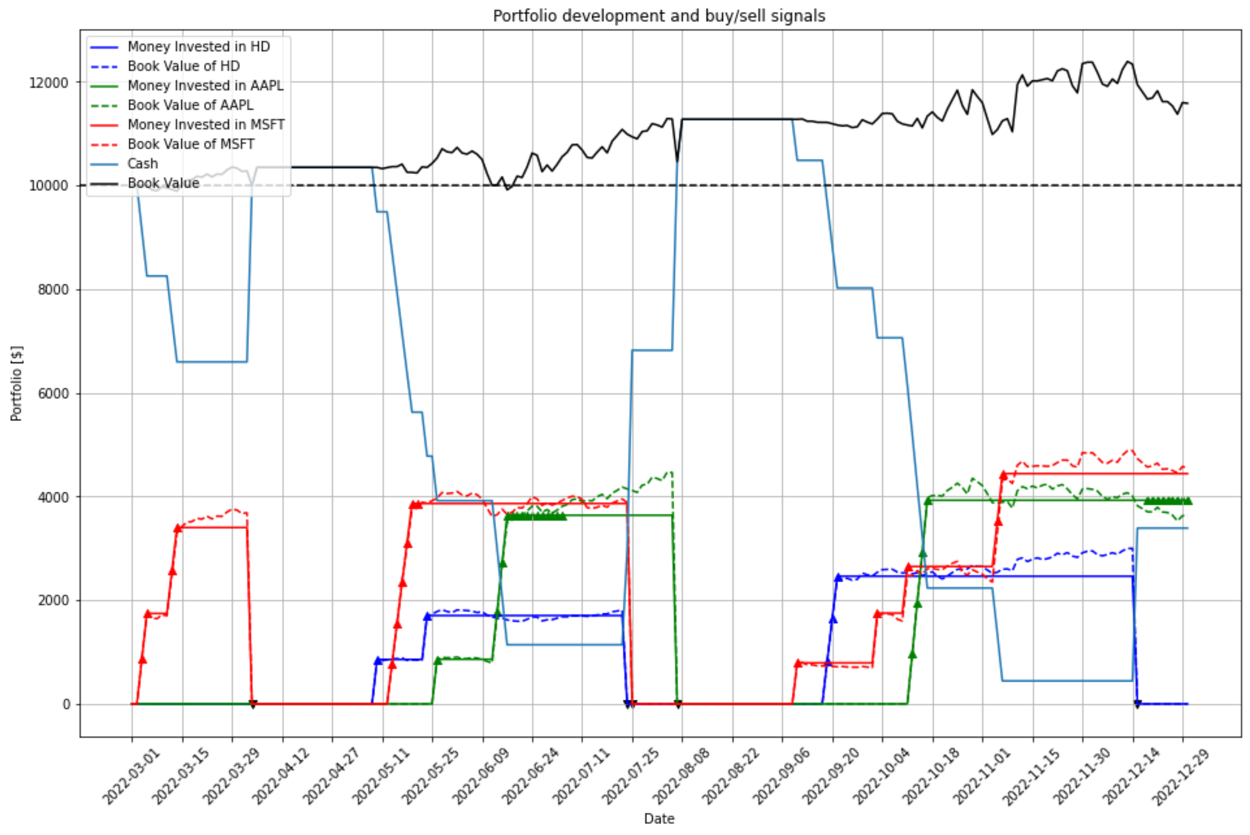


Fig. 11: Report Graph 1 showing the portfolio development, including buy and sell signals

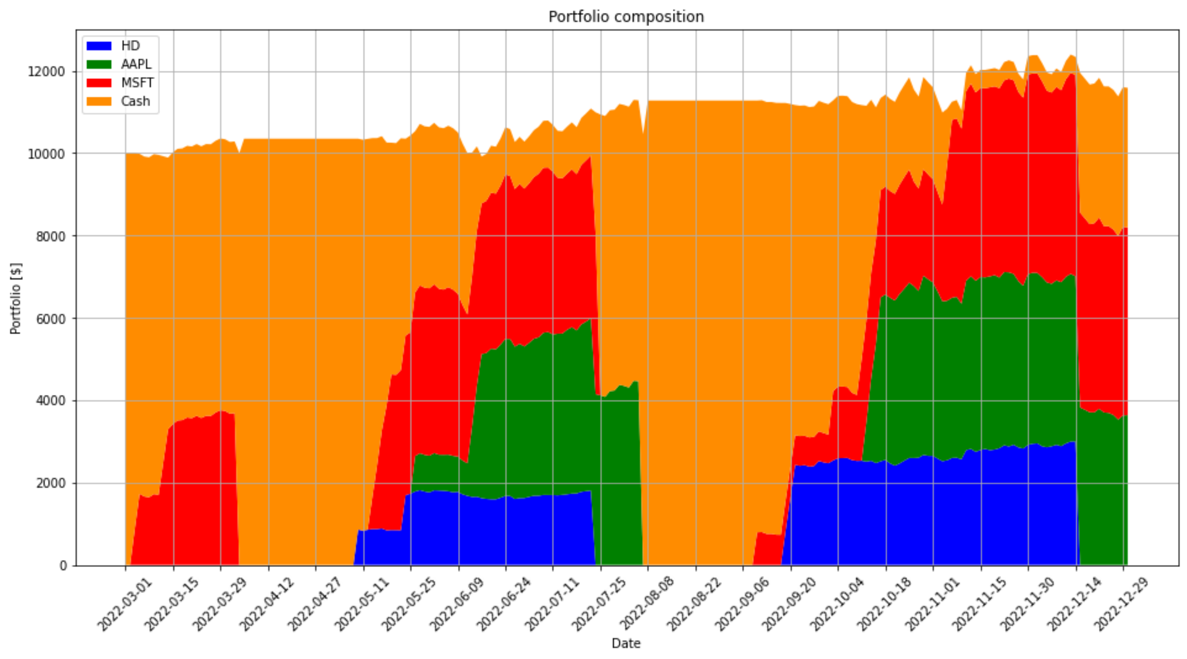


Fig. 12: Report Graph 2 showing the portfolio composition

# 8. Portfolio Manager Hyperparameter Tuning & Analysis

The portfolio manager shows strong dependencies and correlation with selected parameters like threshold, buy_signals, start_invest, and sum_per_invest parameters.

Therefore an attempt has been made to fine-tune the hyperparameters so the manager can use their full potential. The goal was to limit the manager to one set of parameters that shows the best overall performance. We determined the best parameters with a score: the ratio of invested money (start_invest) vs. final portfolio value.

| Parameters | Range |
|---|---|
| buy_threshold | [0.5,0.55,0.6,0.65] |
| sell_threshold | [0.3,0.35,0.4,0.45] |
| buy_signal | [1,2,3,4] |
| sell_signal | [1,2,3,4] |
| start_invest | [10000,15000,20000] |
| sum_per_invest | [500,1000,1500] |

Table 11: First set of hyperparameters to test the portfolio manager performance

These parameter combinations gave us a total of 2304 permutations to test. You can see the test results below.

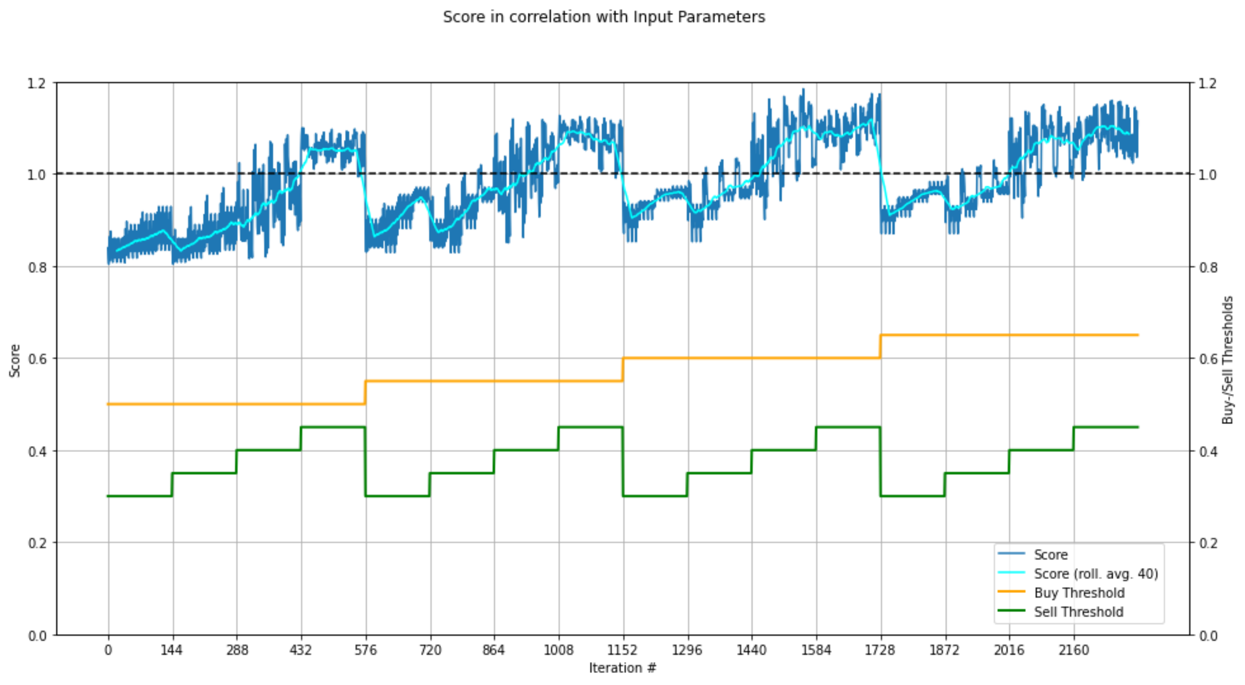


Fig. 13: Results of the first hyperparameter test (data sorted by Buy Threshold)

In the first graph, a correlation between the Buy threshold and the possible score can be observed, as well as an even stronger correlation with the Sell Threshold. Therefore another graph was generated where Sell Threshold sorts the source data:

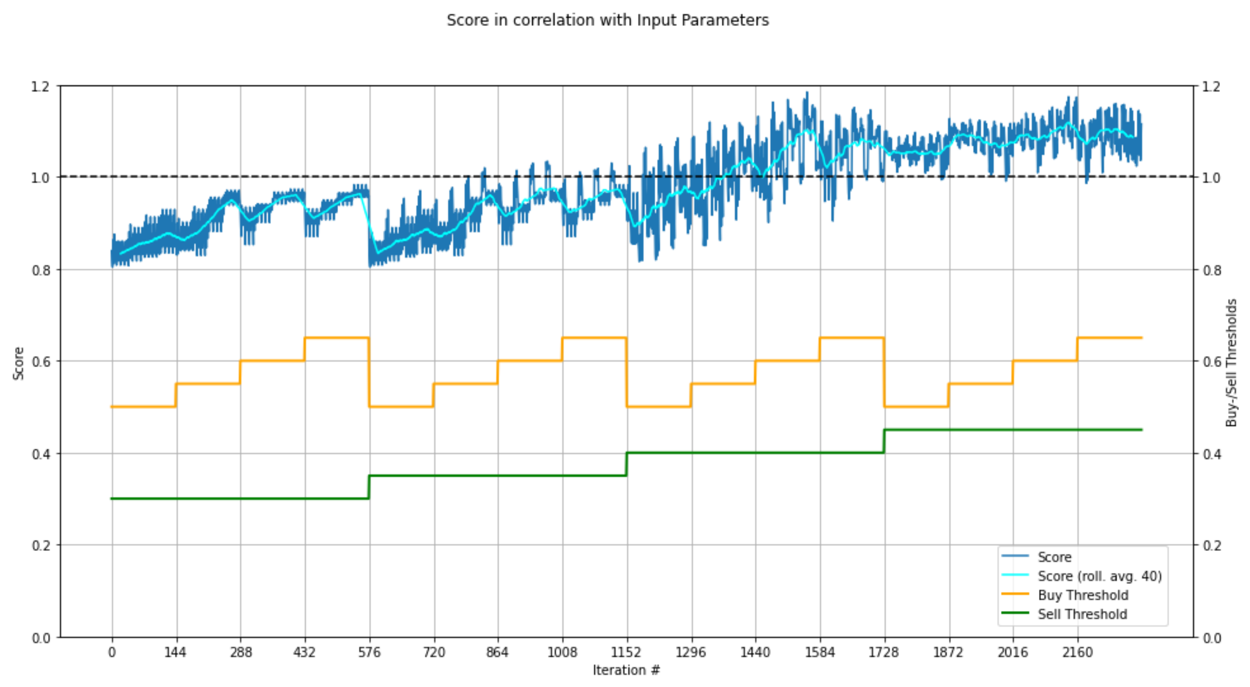


Fig. 14: Results of the first hyperparameter test (data sorted by Sell Threshold).

The Sell Threshold strongly influences the score. With a Sell Threshold of around 0.45, the score remains above 1 for nearly all cases.

We generated a third view to see a possible influence of the Buy or Sell Signals. It showed that the Sell Signal negatively influences the score, whereas the Buy Signal has a slightly positive influence.

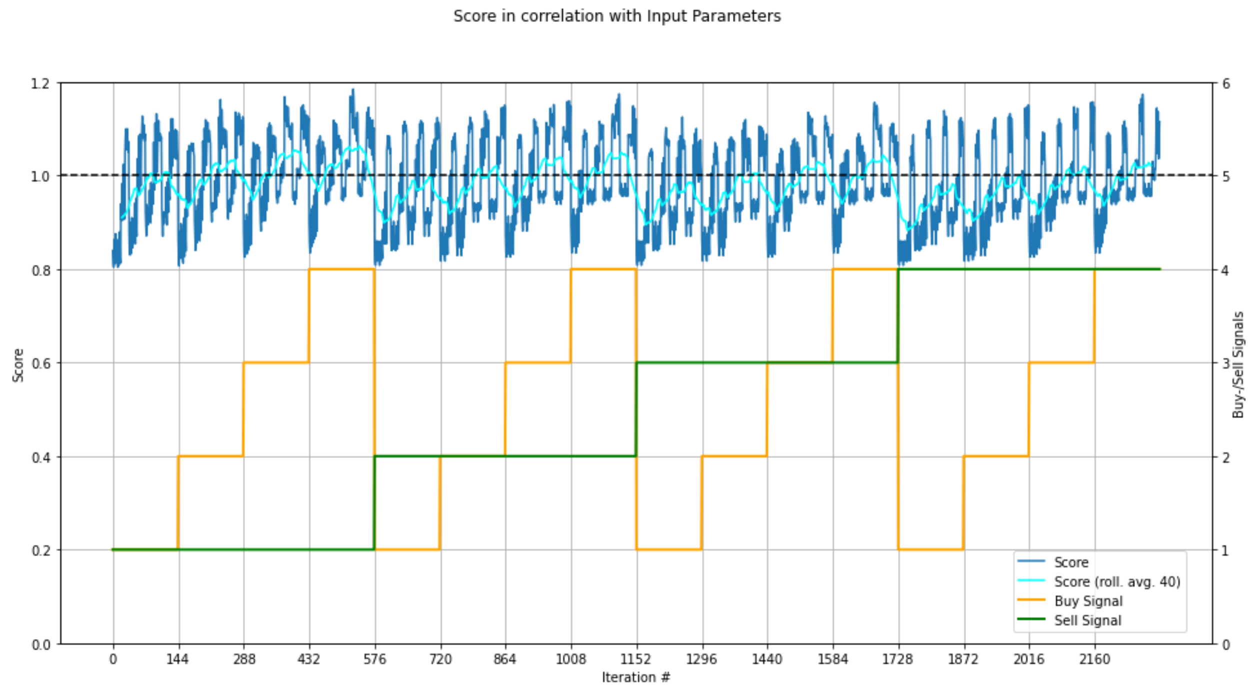


Fig. 15: The first hyperparameter test results (data sorted by Sell Signal)

Finally, we tested the influence of the investment ratio (start_invest: sum_per_invest). No evident influence was visible; however, a low ratio could generate better results in the long run.

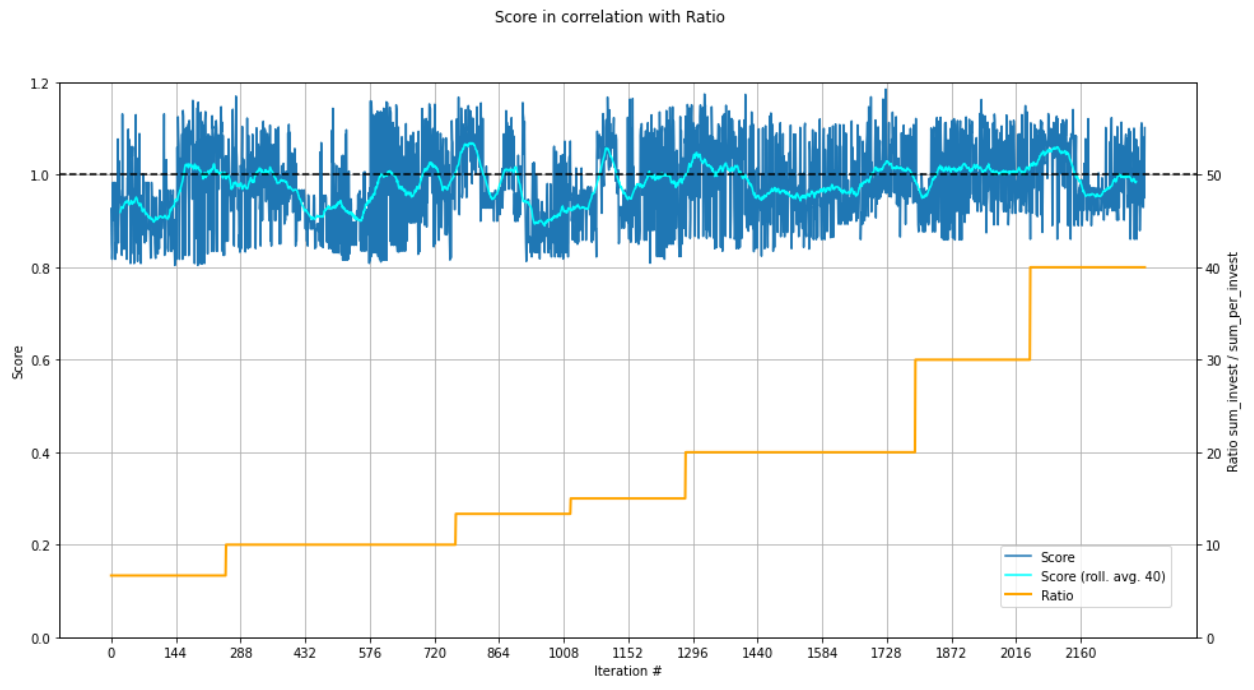


Fig. 16: Results of the first hyperparameter test (data sorted by Ratio)

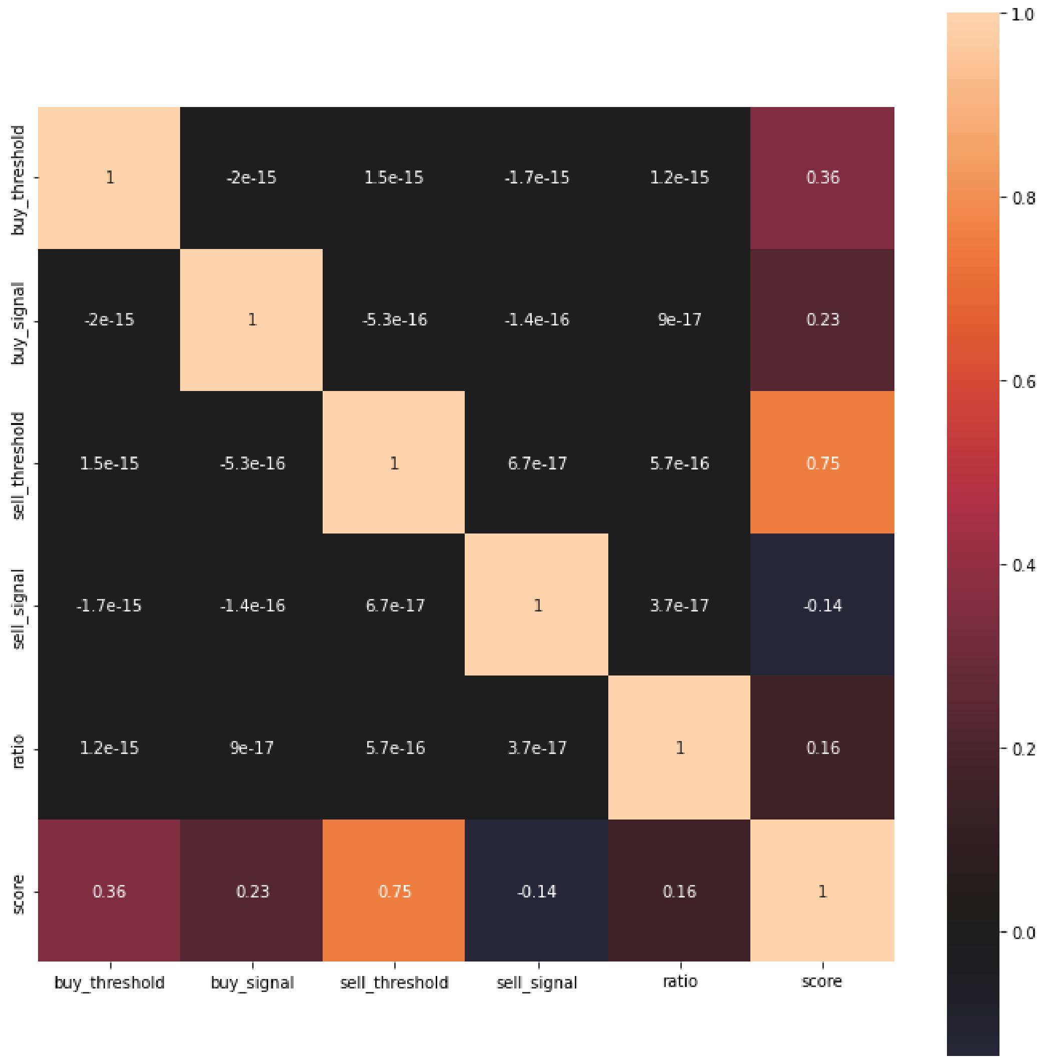


Fig. 17: Correlations of hyperparameters with score

Taking a look at the correlation heatmap of all parameters, our observations were verified, with the thresholds having the most substantial influence on the score.

After examining the examples, we narrowed down the hyperparameters and, in some cases, extended them to the following sets. We then conducted another calculation:

| Parameters | Range |
|---|---|
| buy_threshold | [0.6, 0.65] |
| sell_threshold | [0.4, 0.45] |
| buy_signal | [3, 4] |
| sell_signal | [1, 2, 3] |
| start_invest | [20000, 25000, 30000] |
| sum_per_invest | [1000, 1500, 2000] |

Table 12: Second set of hyperparameters

The evaluation shows that the parameters are narrowed down in a way that a correlation to the score is not immediately visible. More importantly, all scores were in a range above 1:

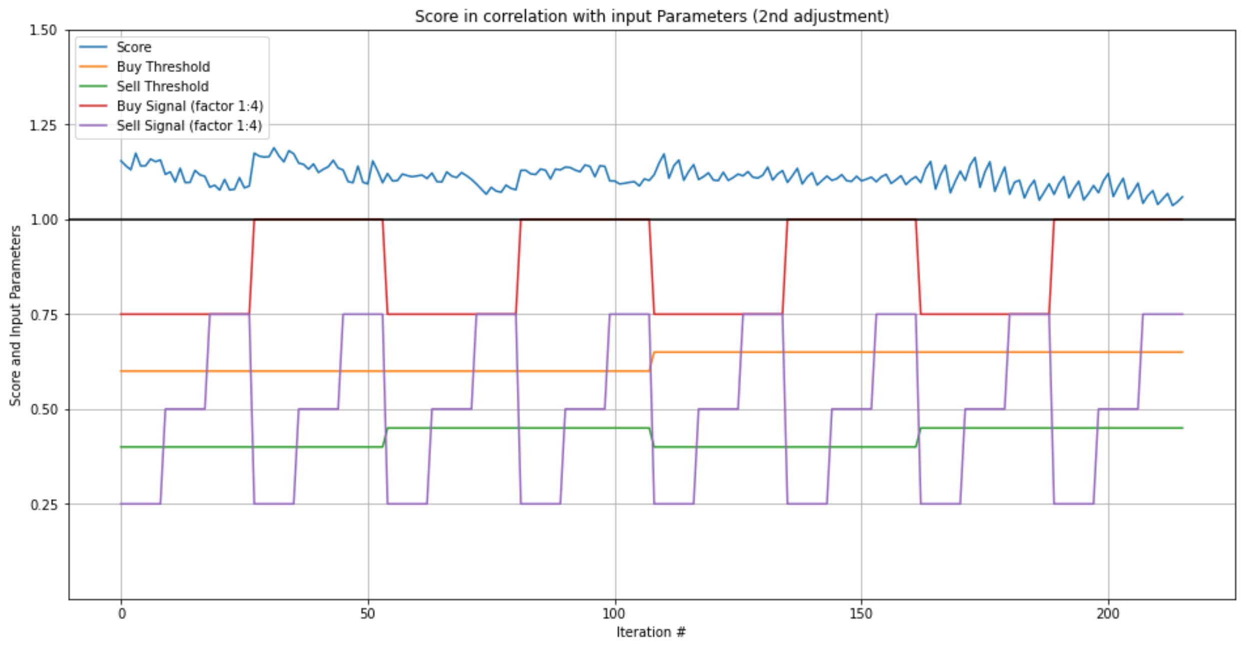
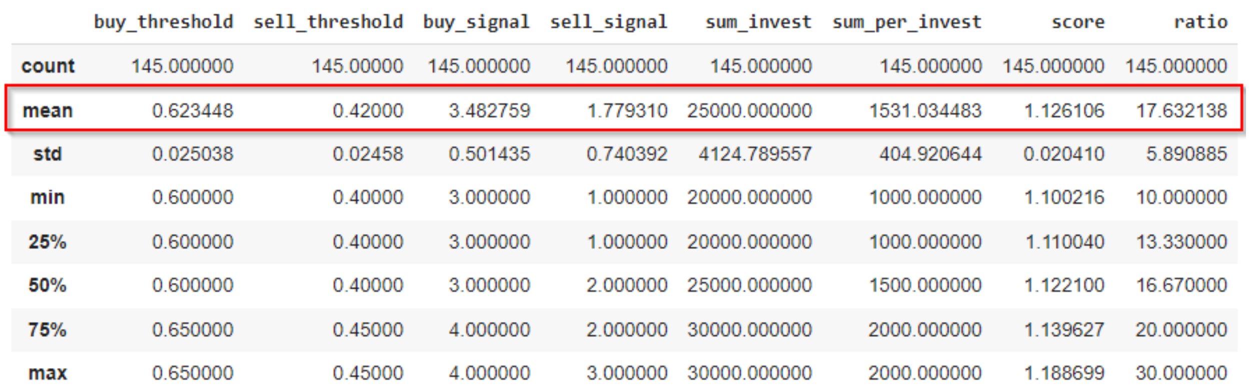


Fig. 18: Results of the second hyperparameter test

For all future calculations, the mean values of all parameter sets with a resulting score of >= 1.1 were displayed and fixed as the final parameter set:



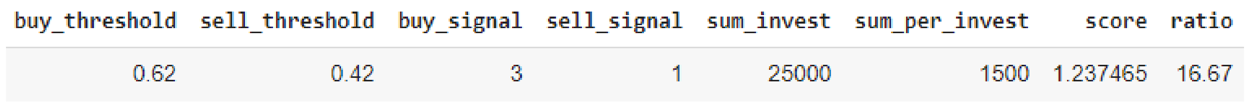
| | buy_threshold | sell_threshold | buy_signal | sell_signal | sum_invest | sum_per_invest | score | ratio |
|---|---|---|---|---|---|---|---|---|
| count | 145.000000 | 145.00000 | 145.000000 | 145.000000 | 145.000000 | 145.000000 | 145.000000 | 145.000000 |
| mean | 0.623448 | 0.42000 | 3.482759 | 1.779310 | 25000.000000 | 1531.034483 | 1.126106 | 17.632138 |
| std | 0.025038 | 0.02458 | 0.501435 | 0.740392 | 4124.789557 | 404.920644 | 0.020410 | 5.890885 |
| min | 0.600000 | 0.40000 | 3.000000 | 1.000000 | 20000.000000 | 1000.000000 | 1.100216 | 10.000000 |
| 25% | 0.600000 | 0.40000 | 3.000000 | 1.000000 | 20000.000000 | 1000.000000 | 1.110040 | 13.330000 |
| 50% | 0.600000 | 0.40000 | 3.000000 | 2.000000 | 25000.000000 | 1500.000000 | 1.122100 | 16.670000 |
| 75% | 0.650000 | 0.45000 | 4.000000 | 2.000000 | 30000.000000 | 2000.000000 | 1.139627 | 20.000000 |
| max | 0.650000 | 0.45000 | 4.000000 | 3.000000 | 30000.000000 | 2000.000000 | 1.188699 | 30.000000 |

Table 13: .describe of the DataFrame containing all scores >= 1.1

| Parameters | Range |
|---|---|
| buy_threshold | 0.62 |
| sell_threshold | 0.42 |
| min_buy_signal | 3 |
| min_sell_signal | 1 |
| start_invest | 25000 |
| sum_per_invest | 1500 |

Table 14: Final parameters set for the portfolio manager

For confirmation, we did a last portfolio test with our standard stock composition ('HD','AAPL','MSFT') and reached an excellent score of 1.24!



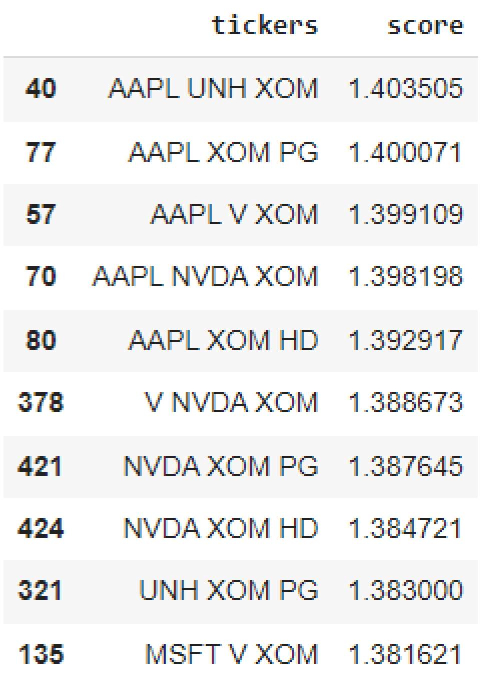
| buy_threshold | sell_threshold | buy_signal | sell_signal | sum_invest | sum_per_invest | score | ratio |
|---|---|---|---|---|---|---|---|
| 0.62 | 0.42 | 3 | 1 | 25000 | 1500 | 1.237465 | 16.67 |

Table 15: The result after fixing the hyperparameters of the portfolio manager

# 9. Testing and Analyzing the results

## 9.1. Results: Portfolio Composition Dependence

To determine if portfolio management works universally, we examined all 455 combinations of a 3-fold stock composition using the optimal parameters.

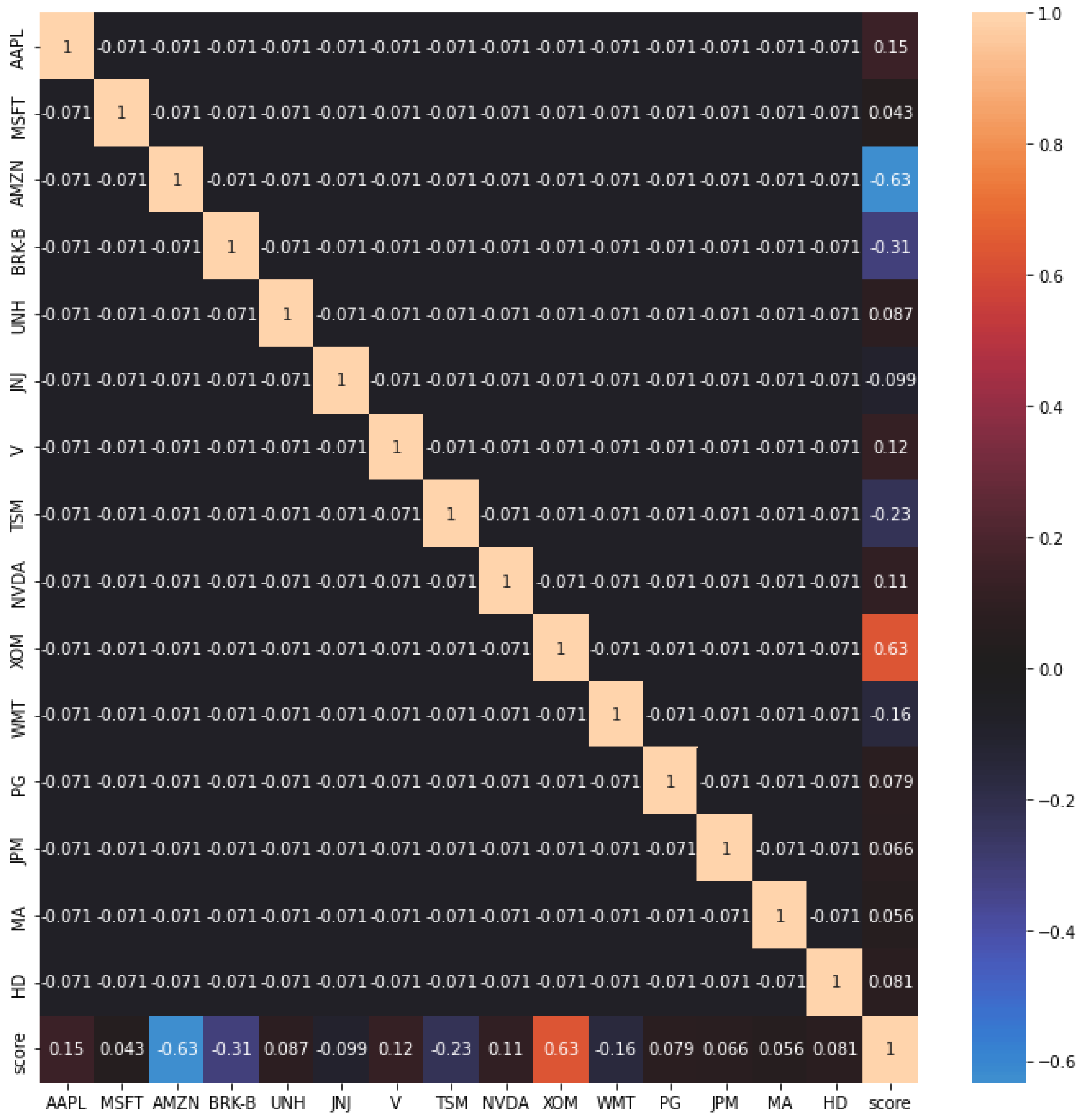


|  | tickers | score |
|---|---|---|
| 40 | AAPL UNH XOM | 1.403505 |
| 77 | AAPL XOM PG | 1.400071 |
| 57 | AAPL V XOM | 1.399109 |
| 70 | AAPL NVDA XOM | 1.398198 |
| 80 | AAPL XOM HD | 1.392917 |
| 378 | V NVDA XOM | 1.388673 |
| 421 | NVDA XOM PG | 1.387645 |
| 424 | NVDA XOM HD | 1.384721 |
| 321 | UNH XOM PG | 1.383000 |
| 135 | MSFT V XOM | 1.381621 |

Table 16: Results and Portfolio compositions of the 10 best scores

The table above shows the best scores, with the top scorer achieving a score as high as 1.40, meaning a 40% earning in a year.

Since stock selection is a big issue we discretized the ticker combination into distinct columns for each used ticker and generated a correlation heatmap.



|  | AAPL | MSFT | AMZN | BRK-B | UNH | JNJ | V | TSM | NVDA | XOM | WMT | PG | JPM | MA | HD | score |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| AAPL | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 0.15 |
| MSFT | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 0.043 |
| AMZN | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.63 |
| BRK-B | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.31 |
| UNH | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 0.087 |
| JNJ | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.099 |
| V | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 0.12 |
| TSM | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.23 |
| NVDA | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 0.11 |
| XOM | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 0.63 |
| WMT | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | -0.071 | -0.16 |
| PG | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | -0.071 | 0.079 |
| JPM | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | -0.071 | 0.066 |
| MA | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | -0.071 | 0.056 |
| HD | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | -0.071 | 1 | 0.081 |
| score | 0.15 | 0.043 | -0.63 | -0.31 | 0.087 | -0.099 | 0.12 | -0.23 | 0.11 | 0.63 | -0.16 | 0.079 | 0.066 | 0.056 | 0.081 | 1 |

Fig. 19: Correlation heatmap of the discretized stock compositions and the score

We could determine that the stock XOM (Exxon Mobile) has the strongest influence on the score. Looking at the top 20 scores and summing up the occurrence of all stocks, this could be verified, as XOM was present in all of those portfolios, with AAPL (Apple) in second place.

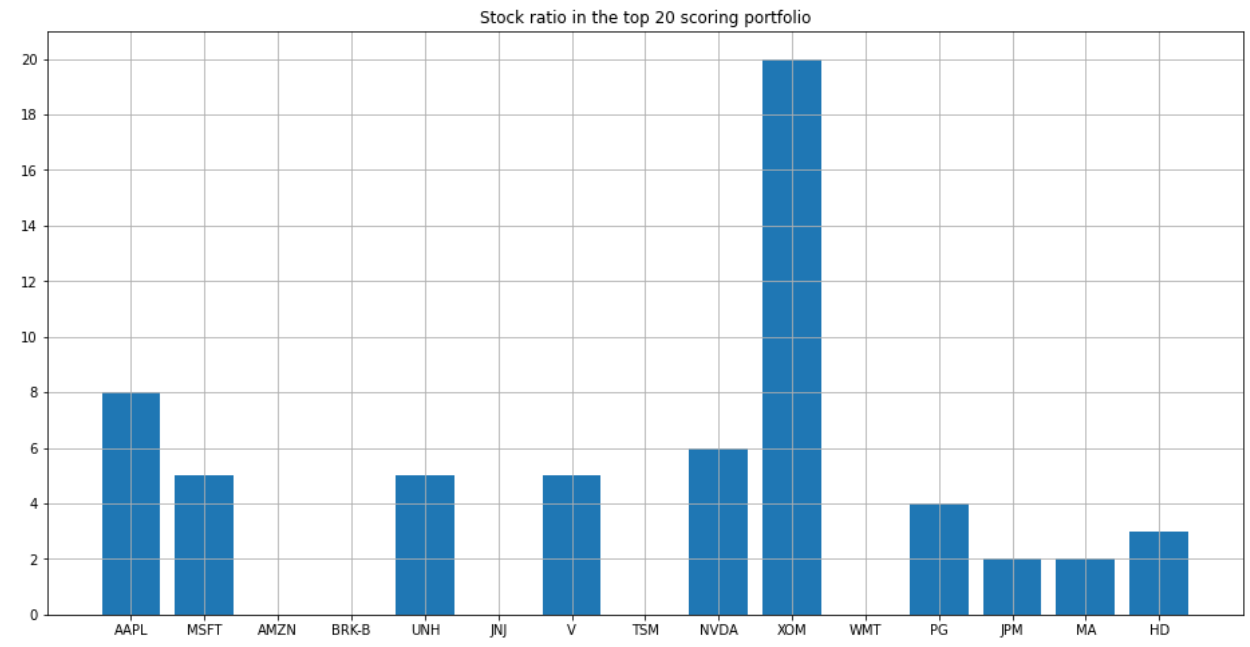
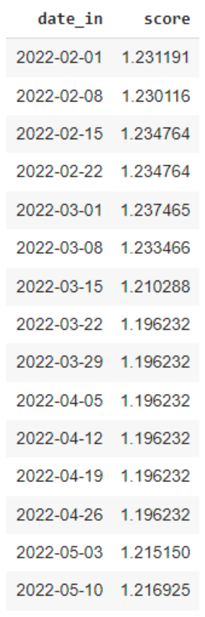


Fig. 20: Barplot showing the occurrence of stocks in the Top 20 scoring combinations

## 9.2. Results: Entry Point Selection Dependence

To show the dependency of the investment starting date, we set up another standard test that calculated scores with the start time going from 01.02.2022 to 10.05.2022 in weekly steps.



| date_in | score |
|---|---|
| 2022-02-01 | 1.231191 |
| 2022-02-08 | 1.230116 |
| 2022-02-15 | 1.234764 |
| 2022-02-22 | 1.234764 |
| 2022-03-01 | 1.237465 |
| 2022-03-08 | 1.233466 |
| 2022-03-15 | 1.210288 |
| 2022-03-22 | 1.196232 |
| 2022-03-29 | 1.196232 |
| 2022-04-05 | 1.196232 |
| 2022-04-12 | 1.196232 |
| 2022-04-19 | 1.196232 |
| 2022-04-26 | 1.196232 |
| 2022-05-03 | 1.215150 |
| 2022-05-10 | 1.216925 |

Table 17: Score as a function of investment starting date

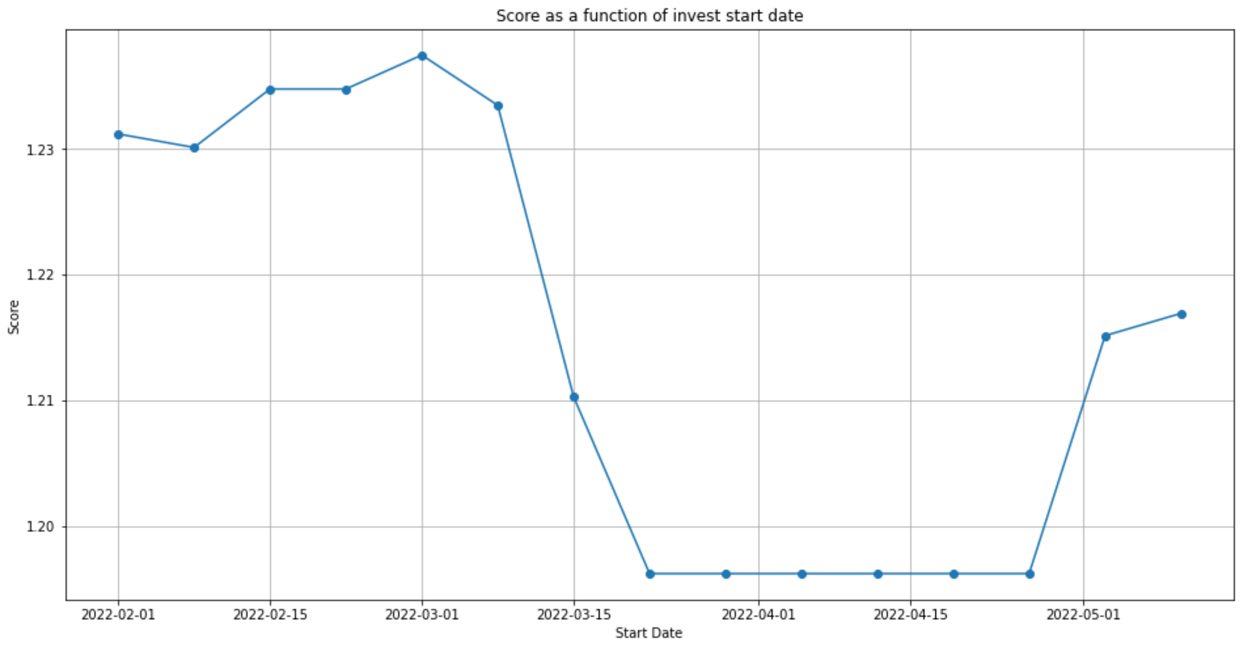


Fig. 21: Score as a function of the investment starting date

Particularly interesting in the graphical evaluation is that for 5 weeks, the starting date does not influence the resulting investment score.

## 9.3. Comparison Between Traditional Investing and Investing with our Model

To find out if the portfolio manager outperforms a "traditional" long-term investment, we set up a simulation where we invested the same amount in each of the tested stocks right at the beginning of the investment period and never sold any shares in the process again with the tickers 'HD', 'MSFT' and 'AAPL' and with an investment of $25000.

Without regulation, the portfolio reflected exactly the mean trend of the invested stocks and closed with a value of $21915,50, which is a loss of 12,34% (Fig. 23), whereas the managed portfolio closed with a positive outcome of $30936,62 or 23,75% surplus (Fig. 22). In direct comparison, the portfolio managed by our model, outperformed the non-managed portfolio by 41,16%.

```
BuyT: 0.62 | BuyS: 3 | SellT: 0.42 | SellS: 1 | StInv: 25000 | SpI: 1500 | Tickers: ['HD', 'AAPL', 'MSFT']
Score: 1.24 | Book value: 30936.62 | Runtime: 8.77s
Total runtime: 17.59s
-----------------------------------------------------------------------------------------------------------------
```

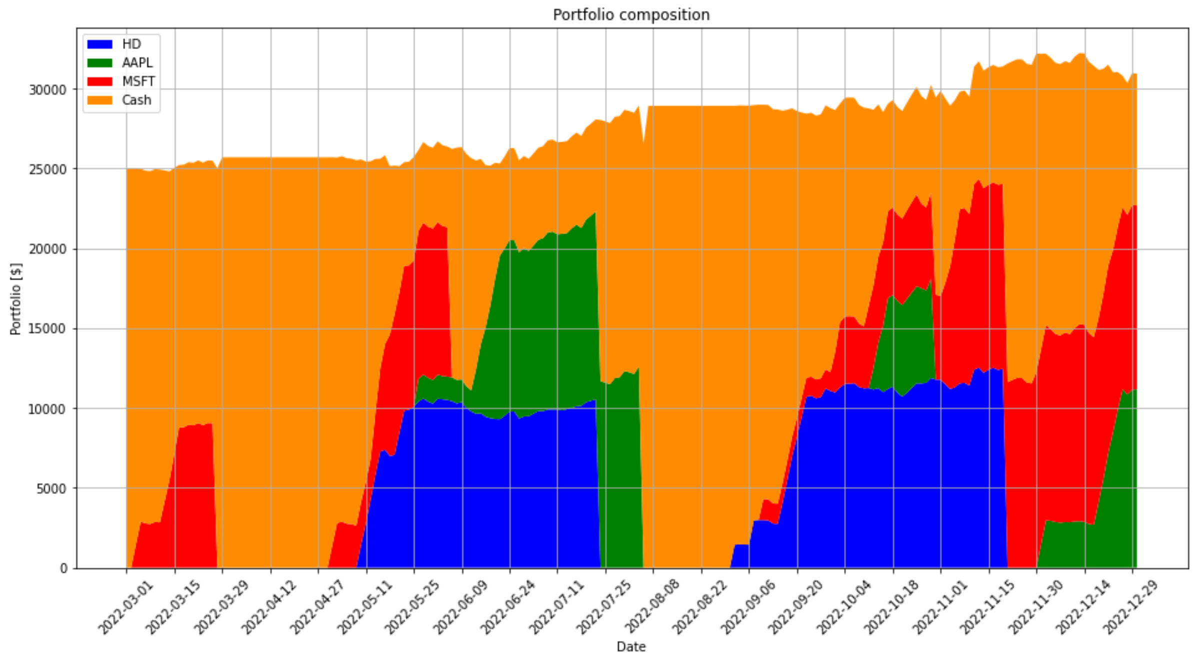


Fig. 22: Investment development  with the Portfolio Manager

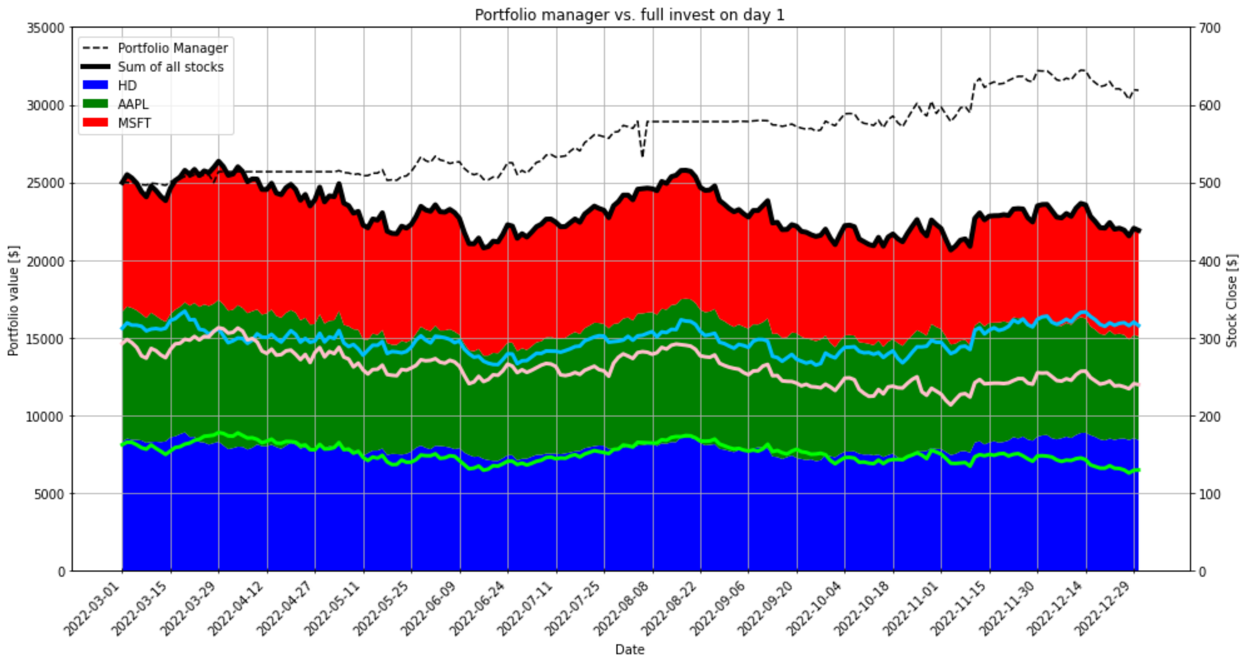


Fig. 23: “Traditional” investment compared to the result achieved by Portfolio Manager

## 9.4. Starting Investments in 2010

Out of curiosity, we ran the portfolio manager with a starting date of 04-01-2010, with interesting results. Since the trained estimators were severely overfitted, we assumed 100% correct predictions during that time period.

The invested money was returned over 18-fold in a 12-year period, which is the equivalent of a compounding interest rate of 27,37% p.a..

```
Invest Finished!
BuyT: 0.62 | BuyS: 3 | SellT: 0.42 | SellS: 1 | StInv: 25000 | SpI: 1500 | Tickers: ['HD', 'AAPL', 'MSFT']
Score: 18.24 | Book value: 455910.08 | Runtime: 77.34s
Total runtime: 92.74s
-----------------------------------------------------------------------------------------------------------
```

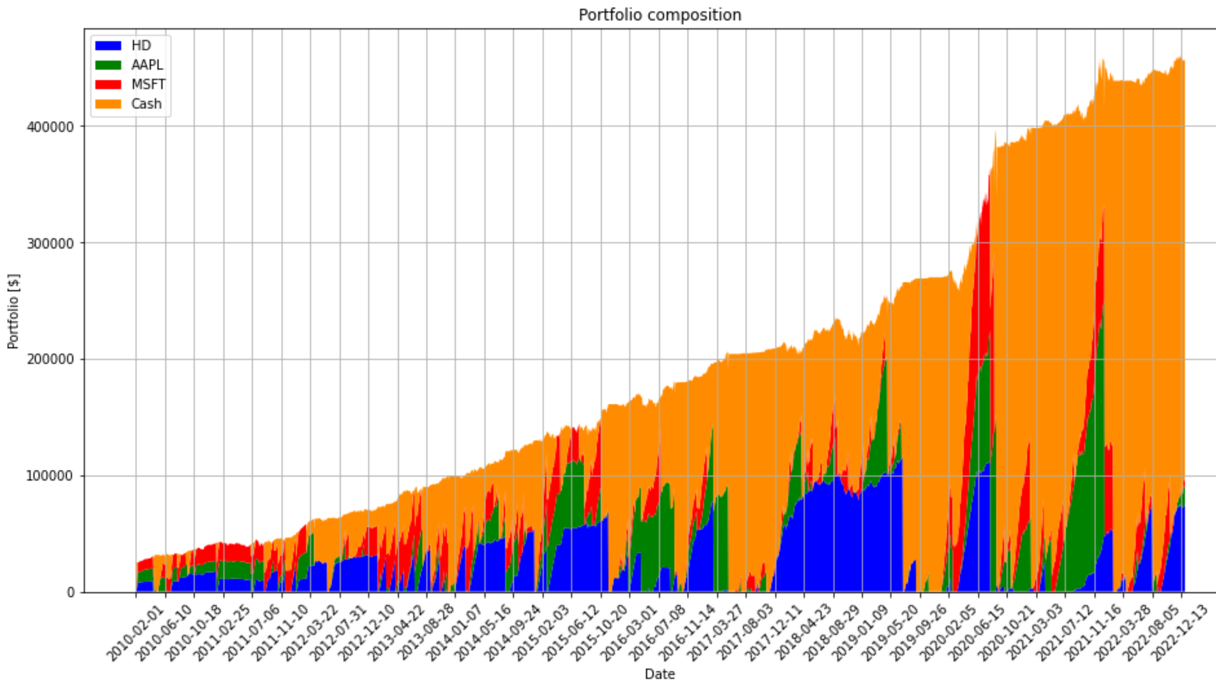


Fig. 24: Investment over 12 years

The next graph shows the period between the 4th and the 20th of March 2020, where the apparent outbreak of the COVID-19 pandemic led to an extensive loss in most traded stocks. The portfolio manager took advantage very well by selling a portion of owned stocks after a short period of losses, followed by a massive investment phase just as the markets bounced back to rake in huge profits. This led to the largest gain in a portfolio worth during that time.

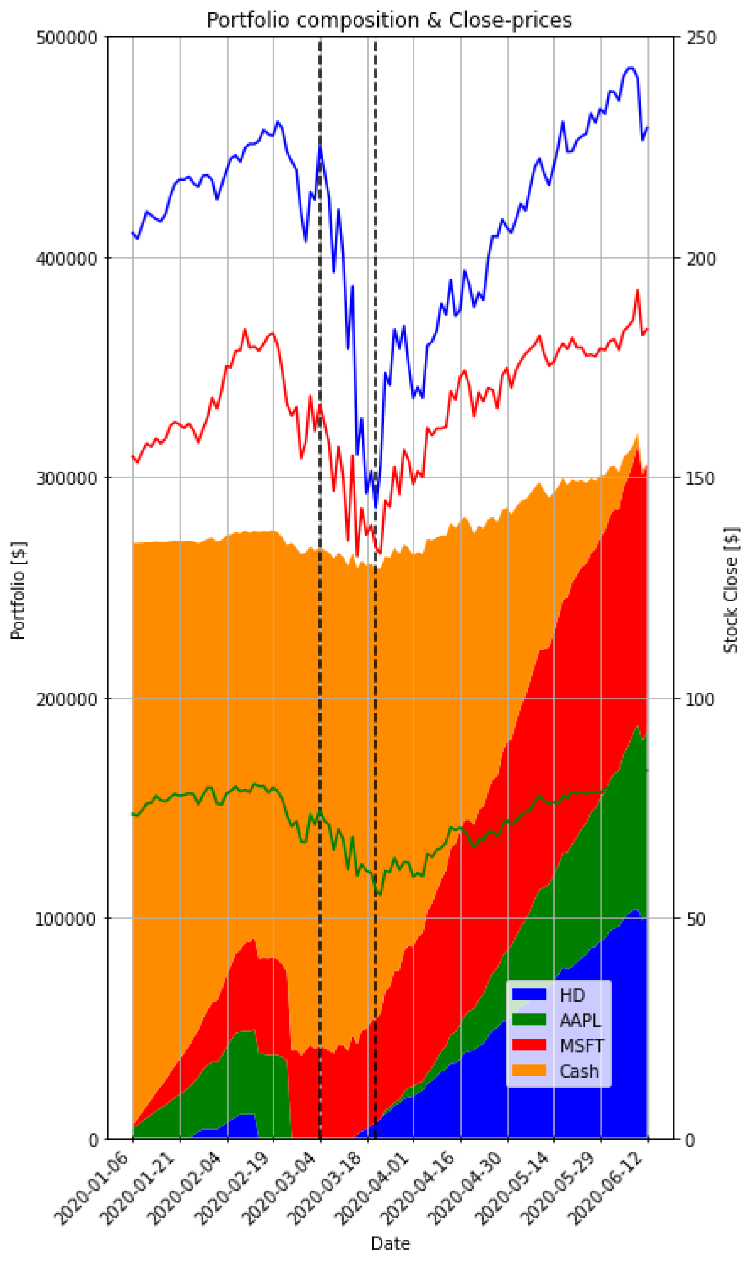


Fig. 25: 12-year portfolio during the market decline (03/2020)

However, we realized that the value for sum_per_invest (short: spi) was holding back more robust earnings. For example, an investment of $1000 in a $10000 portfolio has a different impact on the outcome than $1000 in an evolved $300.000 portfolio.

Realizing the need for an automated adjustment of the spi-parameter, we implemented a parameter for the portfolio manager called self_adjust. If this switch is set as TRUE, first, the ratio of start_invest and spi is calculated. During the investment phase, the spi-variable is adjusted every time a newly calculated new spi would exceed or under-shoot the current value by more than 5%. This self-adjustment guarantees that an optimum portfolio share is always used for investment.

With the self-adjustment switched on, the portfolio manager was able to increase the return on investment to a factor of 76.34, which is a compound interest rate of 43.35% p.a.

```
BuyT: 0.62 | BuyS: 3 | SellT: 0.42 | SellS: 1 | StInv: 25000 | SpI: 112117.01 | Tickers: ['HD', 'AAPL', 'MSFT']
Score: 76.34 | Book value: 1908526.68 | Runtime: 62.55s
Total runtime: 73.19s
-----------------------------------------------------------------------------------------------------
```

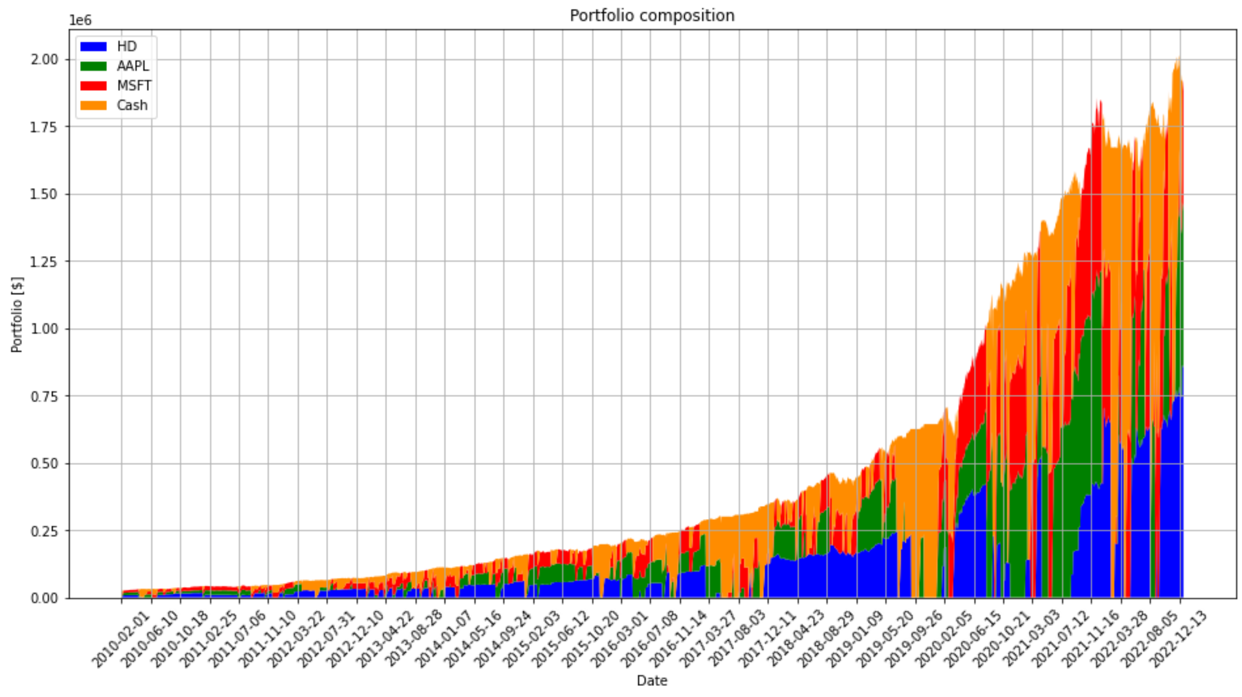


Fig. 26: 12-year portfolio with self-adjustment

Again,this was only a theoretical simulation as 91,66% of the dataset was used for the training of the estimators.

## 10. Conclusion

In this project, our goal was to create a model that decides which stocks to invest in for a given investment portfolio. To do so, we first chose certain stocks and tried different classifiers and hyperparameter combinations to decide whether the stock values would fall or rise in 1, 5, or 10 days. We created a portfolio with a fixed amount of cash and defined rules that would use the classification to decide whether we should buy or sell shares in a specific stock. The following section explains the steps we took and the results we achieved.

First, we chose certain stocks according to their cap value. We used data from YFinance to create a dataset consisting of basic information for each stock. After data analysis, we created new features in the dataset using the information already available (feature engineering). By calculating the ratio of closing prices of the present day and a day in the past period, we created a new feature that could determine whether the price has fallen or risen compared to the past days. We shifted this data into the future for each entry row. We then transformed the ratio into a binary label ('0' for a price fall and '1' for a rise).

We set the label as our target and performed a 5-fold cross-validation with three machine learning models and various hyperparameter combinations for four chosen stocks. In each step, we calculated the accuracy.

According to our tests, we could best predict the labels for each stock with a 10-day window using the RandomForestClassifier with 1000 estimators and 20 features at most. We then trained RandomForestClassifer with the data from all stocks and saved the results.

Afterward, we created a portfolio manager with a specified amount of cash and investment rules. The trained models were implemented into the portfolio manager. This way, given new data, the portfolio manager could determine whether the prices would fall or rise within 10 days and decide whether one should invest in the stock or sell their shares.

The portfolio manager has values set before the model execution, such as starting point and initial cash value. After running the portfolio manager a few times, we realized that the result depends on the investment starting point and stock composition. It is worth mentioning that stocks can be pre-selected by a scheme that shows the most potential.

During the tests, there was no case where we lost all the money, which we consider an accomplishment since no human intervention was necessary at all.

At this point, our model still has some limitations. For example, the model only reacts to the closing price after the markets have closed. This means we can buy on the next day when the price could already be significantly different. For simplification reasons, we considered no fee for the sale and purchase. This could be implemented in the future when a broker with a compatible API has been found.