Point72 Take-Home Project

Giulio Caputi

February 21, 2025

Contents

1 Summary

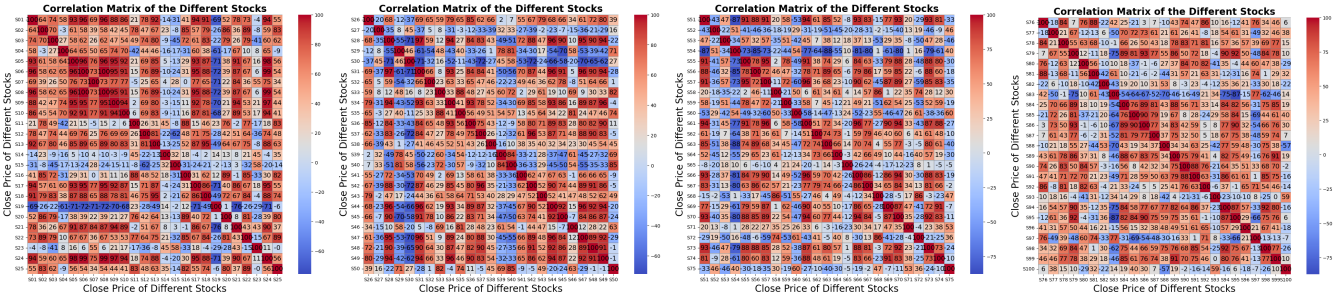
With this work, I analyse and attempt to exploit the dynamics influencing stock price trends. The objective is to generate daily end-of-day (EOD) trading positions in [–1,1] for a set of 100 stocks over a four-year period (2000-2003) using a combination of historical market data and analyst estimates. After performing various sanity checks to clean and validate the data, I identify several pairs of highly correlated securities. I then create two types of features: technical indicators derived from prices and volumes, and sentiment scores based on analyst estimates. Constructing the latter requires making specific assumptions on the provided data. Instead of predicting raw stock returns, I define a categorical target variable, `Position`, which spots imminent and current market trends (–1 for bearish, 0 for neutral, 1 for bullish). I test two methodologies for creating this target variable. The first uses linear regressions on future prices with significance tests, the second computes a modified Kendall's tau statistic on future prices. I adopt the latter because it yields a more stable and meaningful `Position` variable, and I then apply a smoothing function to reduce erratic changes in the target variable and make it more stable. Next, I undertake a thorough process to identify the most relevant features and build an optimal machine learning (ML) model. This process involves correlation analysis, mixed-effects models to assess each variable's statistical association with `Position`, feature-importance ranking with likelihood ratio tests and iterative LASSO (for a multinomial logistic regression) and with Gini impurity reduction (for a random forest classifier). After these analyses, I select random forest as the final predictive model, as it outperforms the other models in out-of-sample accuracy. I proceed to generate the EOD positions for all stocks on all days in a batch-wise fashion. Specifically, the dataset is split into sequential 40-day batches; for each batch, the model is trained on all prior batches and used to predict EOD positions for the current batch, preventing data leakage from the future. The final EOD positions are computed as the expected values of `Position` (using the predicted class probabilities), yielding a continuous number in [–1, 1]. To evaluate the model's practical application, I backtest a trading strategy that trades according to these predicted EOD positions. Using the actual `Position` variable yields unrealistically high returns, as this variable is computed by looking at future prices. This confirms the validity of my approach for defining `Position`. When instead the strategy is tested on the predicted EOD positions, it achieves a decent PnL and a positive Sharpe ratio (albeit below 1), for certain parameter settings. In an effort to improve performance further, I then construct a dynamic strategy aiming at investing in uncorrelated and hopefully predictable stocks, and that iteratively updates the stocks it focuses on. Unfortunately, probably due to the high degree of randomness in these stocks (which causes the correlations between them to be highly non-stationary, especially in certain market periods), this more complex approach does not outperform the simpler strategy. I conclude by discussing the main contributions and limitations of this analysis, along with some possible improvements.

2 Preliminary Checks

After reading the 4 provided datasets as **market**, **est1**, **est2**, and **est3** and converting the data in the `dates` column to datetime format, I perform some sanity checks.

2.1 Checks on Market Data

The **market** dataset contains null values only in a single row (row 81090), specifically in the `High`, `Low`, and `Open` columns. I use linear interpolation to impute these values, preserving continuity and minimising distortion. Next, I verify that the dataset (i) does not contain any negative values, (ii) is sorted first by ticker and then by date in ascending order, and (iii) has the same dates for every ticker, with no trading day skipped from the start (2000-01-04) to the end (2003-12-31) of the period. I then plot the close price of all 100 stocks one by one, noticing no apparent anomalies that might indicate the presence of entry errors. I notice that weeks go from Tuesday to Saturday, suggesting that dates here might be shifted by one day, but this is irrelevant for my analysis. Lastly, I check the correlation between the close prices of different stocks. To get an approximate idea of the general correlations between tickers, instead of computing the correlation between all possible $\binom{100}{2} = 4950$ pairs of companies, I calculate correlations in groups of 25 stocks. The resulting correlation matrices, shown in Figure 1, reveal many pairs of highly correlated tickers, with some being essentially identical (e.g., S06 and S08, S08 and S24, S05 and S24, S44 and S47, or S37 and S42). Instead of removing one stock from each highly correlated pair, or applying a dimensionality-reduction algorithm to eliminate some stocks, I opt for keeping all stocks in the dataset, dealing with their correlations in Section 8. This decision is also justified by the fact that I lack crucial information on the stocks (primarily their names), and so I cannot establish whether these high correlations represent data repetitions or simply spurious associations (which, as shown here, would not be unrealistic, especially since two stocks might be influenced by the same market dynamics for a certain period of time). Anyway, these high correlations will have important implications later.



(a) S01 - S25 (b) S26 - S50 (c) S51 - S75 (d) S76 - S100

Figure 1: Correlation between close prices of different stocks, 25 per image. Overall, stock prices appear quite correlated, with some stocks being essentially identical.

2.2 Checks on Estimate Data

The datasets **est1**, **est2**, and **est3** do not contain any null or negative values and are sorted first by ticker and then by date. Their maximum values are 23, 38, and 10, respectively, which are low enough not to suggest the presence of entry errors. They differ in length and none of them covers all 100 tickers.

3 Feature Engineering

3.1 Technical Indicators

To spot market trends, momentum, and potential overbought/oversold conditions, I compute the variables below, all functions of available price and volume data, and I later use them for prediction. Because I want my predictive models to work on any stock, independently of its price level, these predictors are all in a form that makes them comparable across different tickers. All of these variables are clearly computed by ticker. Many of these indicators normally require a warm-up period, which means that their first values are null by default. I decide to imputed all nulls (below I explain how I do it for each variable), though when training my predictive models, I exclude rows with these imputed values because they are not particularly reliable.

- `Shares_traded` → Approximate number of shares traded on a given day, computed as the ratio between dollar volume and close price.
- `Return` → Day-by-day percentage change of the close price. The first value (originally null) is replaced with 0.
- `Volatility` → 20-day exponentially weighted rolling standard deviation of returns. The first value (which is null) is substituted with 0.
- `EMA_diff` → Difference between a short-term (10-day) Exponential Moving Average (EMA) and a longer-term (30-day) EMA of close prices, divided by the close price to make it comparable across different stocks. The first null values (9 and 29, respectively, for each EMA) are filled by computing an EMA of all close prices up to that point (1 price for the first value, 2 for the second one, and so on).
- `MACD_momentum` → Categorical variable capturing the strength and direction of momentum. First the algorithm computes the Moving Average Convergence Divergence (MACD) line and the signal line. The MACD line is the difference between the fast (12-day) and slow (24-day) EMAs of the close prices. The signal line is the 9-day EMA of the MACD line. Null values are filled as for `EMA_diff`. When the MACD line is above the Signal line, we have bullish momentum (and vice versa when the opposite happens). When the difference between the MACD line and the Signal line is increasing, momentum is strengthening (and vice versa when this difference is decreasing). The categorical variable `MACD_momentum` is 2 when momentum is both bullish and increasing, 1 when momentum is bullish but decreasing, –1 when it is bearish but increasing, –2 when it is bearish and decreasing, and 0 otherwise.
- `RSI` (Relative strength index) → Measure of the speed and change of price movements, helpful to identify overbought and oversold conditions. It is computed as $RSI = 100 - \left(\frac{100}{1+\frac{average\_gain_{14}}{average\_loss_{14}}}\right)$, and it is bounded between 0 and 100. The average gain and loss are computed over the previous 14 days, and the resulting null values (which are the first 13) are filled by computing the average gain and loss over all previous days.

- `Bol_up` and `Bol_down` → Binary variables measuring price extremity relative to recent volatility. Specifically, `Bol_up` is set to 1 if the close price is above the upper Bollinger band (which is a 20-day EMA of `Close` plus 2 standard deviations), while `Bol_down` is set to 1 if the close price is below the lower Bollinger band (which is a 20-day EMA of `Close` minus 2 standard deviations). Null values are filled as for `EMA_diff`.

- `Stoch_osc_signal` (Stochastic oscillator signal) → Momentum indicator computed as the difference between the stochastic oscillator (3-day average of $\%K = \frac{close - lowest\_low_{14}}{highest\_high_{14} - lowest\_low_{14}} * 100$, always between 0 and 100) and its signal line ($\%D$, which is a moving average of the smoothed $\%K$ line over the same 3-day period). So this variable is bounded in [–100, 100]. Here the lowest low and highest high are computed over the previous 14 days, except for the first 13 days, in which they are computed over all available previous days, to avoid nulls.

- `CCI` (Commodity channel index) → Another momentum-based oscillator that attempts to show when a stock has been overbought or oversold relative to its typical price range. It is computed as $CCI = \frac{typical\_price - SMA}{0.015 * mean\_deviation}$, where *typical_price* at time t is $\frac{High_t + Low_t + Close_t}{3}$, SMA is a 20-day simple moving average of *typical_price*, *mean_deviation* is the average absolute difference between *typical_price* and SMA over 20 days, and 0.015 is a scaling constant chosen to make `CCI` more comparable across different stocks. This variable is not bounded.

- `OBV` (On-balance volume) → Normalised cumulative measure of volume that tries to show when volume is flowing into or out of a security. It increases when the price closes higher than the previous close and decreases when the price closes lower. On day t, the raw on-balance volume is $ROBV_t = \begin{cases} ROBV_{t-1} + Volume_t & \text{if } Close_t > Close_{t-1} \\ ROBV_{t-1} - Volume_t & \text{if } Close_t < Close_{t-1} \\ ROBV_{t-1} & \text{if } Close_t = Close_{t-1} \end{cases}$ (with $ROBV_0 = Volume_0$), and the predictor `OBV` is obtained by dividing this raw version by the volume, to make it comparable across different stocks. This variable is not bounded.

- `MFI` (Money flow index) → Momentum indicator that uses price and volume to identify overbought or oversold conditions. The Python function first computes the typical price as $TP_t = \frac{high_t + low_t + close_t}{3}$, then the raw money flow as $RMF_t = TP_t * volume_t$, then the money flow ratio as $MFR_t = \frac{\sum_{i=t-13}^{t} RMF_i * I(TP_i > TP_{i-1})}{\sum_{i=t-13}^{t} RMF_i * I(TP_i < TP_{i-1})}$, where $I()$ is the indicator function, and finally the money flow index as $MFI_t = 100 - \left(\frac{100}{1 + MFR_t}\right)$. This value ranges from 0 to 100, and, essentially, combines price and volume to measure the strength of money flowing into or out of an asset. Null values are filled as usual.

- `PSAR` (Parabolic stop and reverse) → A trend-following indicator computed recursively as $PSAR_{t+1} = PSAR_t + \alpha(EP - PSAR_t)$. Here, EP (extreme point) is the highest high (in an uptrend) or lowest low (in a downtrend) observed so far in the trend, and α is an acceleration factor that starts at 0.02 and increases by 0.02 with each new high (if $Close > PSAR$) or new low (if $Close < PSAR$), up to a maximum of 0.20. I divide the parabolic SAR by the close price to make it more comparable across different stocks.

Importantly, all of these variables are available at the end of each day, so they are suitable to compute the EOD positions (that are, by definition, computed after the market close), but the EOD position of any given date can only be used for trading on the following trading day. This is considered in Sections 7 and 8.

3.2 Sentiment Scores

In this subsection, I explain how I derive sentiment scores from the **est*** datasets. First, I make the following assumptions:

- The 3 datasets contain no entry errors. Without this assumption, any interpretation of the data could be justified.

- The 3 datasets correspond to 3 different groups of analysts (possibly from different companies).

- Each group has a certain "view" on every stock on each day, and the only changes which occurred to this view over the studied period are the ones implied by the 3 **est*** datasets.

- All 3 groups have a neutral outlook on all 100 stocks at the start of the studied period.

- Each of the 3 groups is itself composed of 3 subgroups. Specifically, the columns `NumEstLowered1W` and `NumEstRaised1W` correspond to one of these subgroups, the columns `NumEstLowered4W` and `NumEstRaised4W` correspond to another subgroup, and the columns `NumEstLowered1M` and `NumEstRaised1M` correspond to the

third subgroup. So we have a total of 9 subgroups, 3 in **est1**, 3 in **est2**, 3 in **est3**. I make this assumption because there are several rows where, for instance, `NumEstLowered1W` > `NumEstLowered4W` (e.g., lines 975, 1018, 1019, or 1316 for **est1**), or `NumEstRaised1W` > `NumEstRaised1M` (e.g., lines 149, 509, 734, or 754 for **est3**). Without assuming the existence of 3 subgroups per analyst group, these occurrences would be errors as, for instance, the past week is clearly a subset of the past 4 weeks. Additionally, even if I assume "month" refers to the calendar month (i.e., from January to December) and "week" refers to the week according to the ISO 8601 standard (i.e., from 1 to 52), inconsistencies still remain. For instance, days 2000-03-30 and 2000-03-31 for ticker S02 in **est2** have a `NumEstRaised4W` value of 2 and 1, respectively, which should not be possible under this interpretation, as both days belong to the same ISO 8601 week (week 13), but the number of estimates raised at the more recent date is smaller. Moreover, `NumEstLowered1W`, `NumEstLowered4W`, and `NumEstLowered1M` are defined merely as "Number of estimates revised downward in the last 1 week / 4 weeks / 1 month", without specifying whether they were generated by the same group of people, thus the 3-subgroup interpretation is coherent with the provided descriptions of these variables.

With the above assumptions in mind, I define a function called *sentiment_score*() that takes as input the portion of an **est*** dataset corresponding to a single ticker, then expands it so that it includes all dates from 2000-01-04 to 2003-12-31, thus creating plenty of null values. Then, for each `NumEstLowered*` and `NumEstRaised*` column, null values are replaced with 0s (since no change in the group's estimates occurred other than those implied my these datasets, no estimate was lowered or raised on missing dates). Subsequently, for each subgroup, the function computes the cumulative sums of lowered/raised estimates, called *negative_score* and *positive_score*. These are used to compute a score for each subgroup on each day. Specifically, for each date t, I check if $positive\_score_t + negative\_score_t$ is non-zero. If it is, I compute the fraction $\frac{positive\_score_t - negative\_score_t}{positive\_score_t + negative\_score_t}$ (which is a measure of the sentiment of the subgroup on the stock at time t, always between –1 and 1). If instead $positive\_score_t + negative\_score_t$ is zero, I simply set the subgroup's estimate for that day on that stock to 0. This way I compute a daily score for each of the 3 subgroups. These variables are called `1WScore`, `4WScore`, and `1MScore`. To get the final score for the overall group of analysts, I compute the mean of these 3 columns. Doing this for each ticker of **est1**, **est2**, and **est3** results, respectively, in the columns `Score1`, `Score2`, and `Score3`, which I add to the **market** dataset. For this procedure to work properly, all tickers must be present in all **est*** datasets, thus before running the *sentiment_score*() function, I add the missing tickers to the 3 estimate datasets, clearly ensuring the order is correct and that the resulting estimate score over the whole period is 0 for missing tickers (since I have no data on these tickers for a particular group of analysts). All of this ignores the `NumEst*` columns. Indeed, I experimented with multiplying the so-obtained scores by the relatively `NumEst` value, to give more weight to a score which results from many analysts. However, this could have dangerous consequences (e.g., a normalised score of 0.1 resulting from 1000 estimates would turn into a final score much higher than a 0.9 normalised score resulting from 10 estimates). If I had information on exactly how many analysts were bullish, bearish and neutral on each day for each stock, my approach for creating these sentiment scores would change.

4 Creating the Target Variable

The goal of this project is to generate EOD positions for every ticker on every day. After ruling out the idea of predicting daily returns (given their high degree of randomness), I decide that my target variable, named `Position`, will be categorical with classes –1, 0, and 1. My ML models will thus predict a value in $\{-1, 0, 1\}$ for every observation. They will also assign a certain probability for the observation to have a `Position` of –1 (p_{-1}), 0 (p_0) and 1 (p_1), with $p_{-1} + p_0 + p_1 = 1$. The corresponding EOD position is defined as the expected value of the label, i.e., $EOD\_position = -1p_{-1} + 0p_0 + 1p_1 = p_1 - p_{-1}$, a quantity always between –1 and 1, which effectively reflects the level of conviction for short (–1), neutral (0), or long (+1) positions. If *EOD_position* is strongly negative, the model is more confident in the –1 class, and vice versa for strongly positive *EOD_position*. If *EOD_position* is close to 0, the model is leaning towards the 0 class or is generally uncertain. To accomplish this, the first step is defining the correct positions, i.e., the target variable that my classification algorithms will learn to predict, stored in the `Position` column. This represents the "ideal position" to have on a certain stock at a certain date, and is therefore computed by looking at future prices. Notice that this does not introduce any look-ahead bias, because to predict `Position` for stock x at time t I will only use information available for stock x at time t. I experiment with two ways to define `Position`, which are described below.

4.1 First Approach: Linear Regression with Significance Test

Let *lookahead* be a positive integer. The *compute_position_lr*() function fits a linear regression on the next *lookahead* close prices to classify the future trend as bullish (+1), bearish (–1), or flat (0). Specifically, for

each day t apart from the last *lookahead* ones, I fit a linear regression with $X = [0, 1, 2, ..., lookahead - 1]$ and $Y = [close_t, close_{t+1}, ..., close_{t+lookahead-1}]$, I compute the slope of the fitted line, its standard error, and the t-statistic for its significance, along with the associated p-value of a two-sided test (using a t distribution with $lookahead - 2$ degrees of freedom). Observations with a positive and significant slope are assigned a *Position* value of 1 (indicating a bullish imminent/current trend), observations with a negative and significant slope are assigned a *Position* of –1 (indicating a bearish trend), and the other observations are assigned a *Position* of 0 (indicating an unclear trend). For the last *lookahead* days, *Position* is set to 0. After several tests, I opt for a value of 20 for *lookahead* and 0.05 for the significance level. The resulting `Position` column is later smoothed with the *remove_short_runs*() function, described in Section 4.3.

4.2 Second Approach: Modified Kendall's Tau

The *compute_position_kt*() function also assigns a position in $\{-1, 0, 1\}$ to every stock on every day. It does so by looking ahead a fixed number of days (*lookahead*) and measuring whether price movements over that window tend to be predominantly upward or downward, while also weighing these movements by their magnitude. Specifically, for each day t apart from the last *lookahead* ones, it takes the next *lookahead* close prices and compares all possible pairs of days (i, j) within that window, with $i < j$. If the later day j has a higher price than the earlier day i, the difference $price_j - price_i$ is added to a variable C, which is a weighted count of concordant pairs (i.e., pairs where a later time has a higher price than an earlier time). Conversely, if the later day has a lower price, the difference $price_i - price_j$ is added to a variable D, which is a weighted count of discordant pairs (i.e., pairs where a later time has a lower price than an earlier time). So C and D are both non-negative. After considering all pairs, the function calculates a weighted Kendall-like statistic $\tau = \frac{C-D}{C+D}$, which is positive if large upward movements dominate, and negative if large downward movements dominate, and always between –1 and 1. The function sets the position to 1 if τ exceeds *threshold*, sets it to –1 if τ is below the negative of that threshold, and otherwise sets it to 0. The value I choose for *threshold* is 0.4, implying that I consider a trend to be statistically significant only if $\frac{C}{C+D} \geq 0.7$, or $\frac{D}{C+D} \geq 0.7$, meaning that I require at least 70% of the total price movement magnitude to be in one direction (either upward or downward). As with the *compute_position_lr*() function, the last *lookahead* days in the data do not have enough forward prices for this analysis, and thus default to `Position` 0. Moreover, the resulting `Position` column is again smoothed with the *remove_short_runs*() function, described below.

4.3 Smoothing

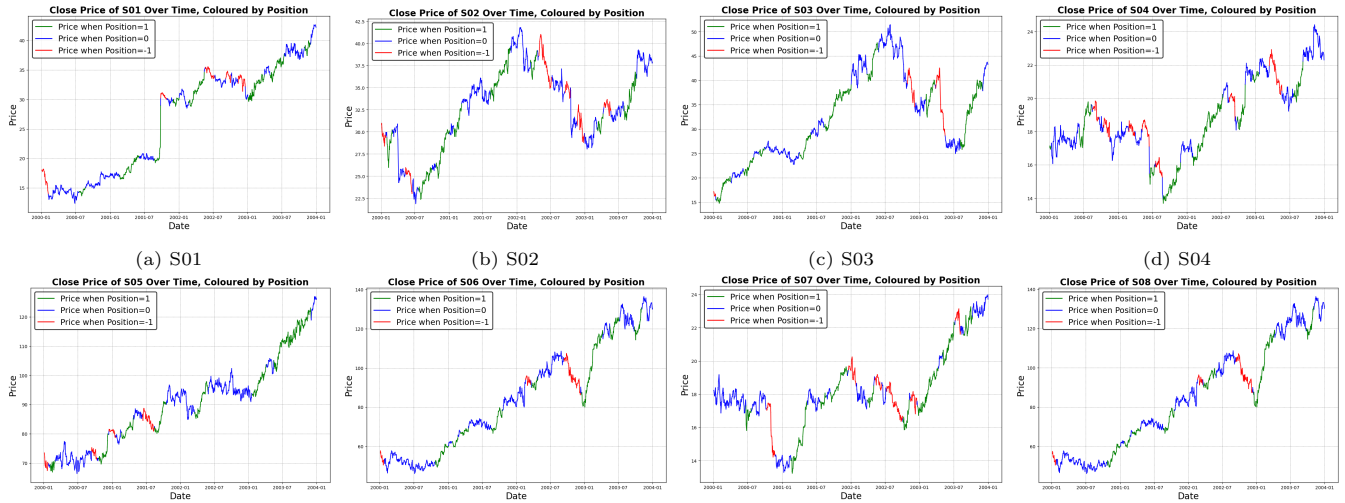
To increase the smoothness (and so, hopefully, both the reliability and the predictability) of the `Position` variable, I define the *remove_short_runs*() helper function, which smooths the `Position` series by merging any short, isolated runs of a position with the surrounding runs if they share the same value. First, it identifies each consecutive "run" of the same position value, storing it as (*start_index*, *end_index*, *value*). Then, for each run whose length is shorter than *min_run_length* (an input of the function), the function checks if it is sandwiched between two runs of the same value. If so, it merges the short run into those longer runs, effectively "flattening out" brief spikes in `Position`. This process essentially prevents `Position` from quickly flipping up and down due to transient or minimal changes in the data. This should also make the estimate scores `Score1`, `Score2`, and `Score3` more useful for predicting `Position`, because for many stocks these estimates remain stable over long periods of time, implying that analysts' views on many stocks do not change rapidly.

4.4 Using a Lagged Modified Kendall's Tau as a Predictor

To develop stronger predictive models, I considered adding to the dataset a kind of "lagged position" variable, corresponding to the true value of `Position` some days back. Specifically, I considered adding a 20-day lagged position since, at first glance, on day t I should indeed have access to the true `Position` value of day $t - 20$, as `Position` is computed by looking at the future 20 daily prices. This would indeed be the case if I did not apply the smoothing function, but since I do use *remove_short_runs*(), `Position` at time t could depend not just on all `Position` values up to time $t + 20$, but also on values further in the future, as `Position` at time $t + 20$ itself depends directly on values up to time $t + 40$. So, using a lagged position would necessarily introduce a look-ahead bias. Since `Position` is computed using a modified Kendall's tau statistic, as outlined in Section 4.2, I add a variable called `Lagged_tau` as a predictor. This variable, at any day t, computes the τ statistic on the previous 20 days (from $t - 19$ to t). For the first 19 days per stock, which do not have enough past days to compute the tau statistic, I set `Lagged_tau` to 0. This variable, as every other predictor in this study, relies only on information available on day t, and indeed the EOD positions I will create using these variables will be used for trading on the following day, as they are available after the market has closed.

4.5 Comments on the Target Variable

The `Position` variable should be meaningful (i.e., it should be +1 right before and during a bullish trend, –1 right before and during a bearish trend, and 0 when the price is flat or lacks a clear direction). It should also have a certain structure, in order to be predictable by an ML algorithm. After several tests, I found that *compute_position_kt*() generates a `Position` column satisfying these two criteria better than *compute_position_lr*(), specifically with values of $lookahead = 20$, $threshold = 0.4$, and $min\_run\_length = 20$. Figure 2 shows the close price of the first 8 stocks over time, coloured by `Position`. When `Position` is +1, the price line is green; when `Position` is 0, the price line is blue; and when `Position` is –1, the price line is red. From these plots, it is evident that `Position` is meaningful (since it identifies trends well), and stable (since it does not flip frequently from one value to another). Similar behaviour holds across all 100 stocks. It is worth mentioning that the `Position` variable is relatively unbalanced, with 56% of observations being 0, 29% being +1, and 15% of –1. This was expected, since many stocks spend significant time in no clear short-term trend.



(a) S01 (b) S02 (c) S03 (d) S04

(e) S05 (f) S06 (g) S07 (h) S08

Figure 2: Close prices of the first 8 stocks over time, coloured by `Position`. It is clear that `Position` genuinely reflects trends in the prices. Notice that the last 20 days default to `Position` 0.

5 Variable and Model Selection

To perform variable selection and train my ML models, I first remove from **market** the first 29 and the last 20 rows for each ticker, storing the result in a dataset called **df**, indexed by `dates`. Indeed, the first 29 rows per ticker contain imputed values (as explained in Section 3.1) and are thus not entirely reliable, and the last 20 rows have a `Position` value of 0 by default since, as explained in Sections 4.1 and 4.2, they do not have enough forward prices to accurately compute `Position`. Additionally, I have a non-binary categorical predictor, namely `MACD_momentum`, which I leave as is rather than using one-hot encoding, because it has a natural ordering that conveys meaningful information (e.g., `MACD_momentum` values of –2 and –1 are more similar to each other than to +2). Lastly, after performing tests with and without price variables (`High`, `Low`, `Open`, or `Close`), I decide not to employ them to predict `Position`, as (i) the essential information in prices is already reflected by my technical indicators, and (ii) I want my ML models to be applicable to any stock, regardless of its baseline price level.

5.1 Correlation Between Predictors

Figure 3 shows the correlation matrix of **df**. From this, we see that the dependent variable `Position` has weak correlations with most features. This indicates that none of the individual features has a strong linear relationship with it, as the highest correlations are 0.26 (with `RSI`), 0.25 (with `CCI`), and 0.22 (with `MFI`). However, given that `Position` is categorical, its correlation with predictors is not the most appropriate measure of association. Looking instead at the correlations between pairs of predictors, we see that the 4 price variables have a correlation of 1, which was expected. Also, since `Shares_traded` is a transformed version of `volumeUSD` that retains the same information but in a normalised form (i.e., dividing it by `Close`), this variable is more comparable across different stocks, and makes

`volumeUSD` redundant, also given the 57% correlation between them, so I eliminate `volumeUSD` from my analysis. Additionally, the 3 predictors derived from analysts' estimates are relatively correlated with one other (especially `Score1` and `Score2`, with a 63% correlation), indicating that the opinions of one group of analysts on these stocks were often not particularly dissimilar from the opinions of another group over the studied period. This also reinforces the validity of my approach for calculating these variables. Furthermore, some pairs of variables are quite correlated, such as `CCI` and `RSI` (0.86), `Return` and `Stoch_osc_signal` (0.7), `CCI` and `PSAR` (-0.73), or `EMA_diff` and `Lagged_tau` (0.71), suggesting they capture similar aspects of market behaviour. Consequently, to mitigate multicollinearity and reduce unnecessary model complexity, I opt to remove certain variables, as detailed in the following sections.

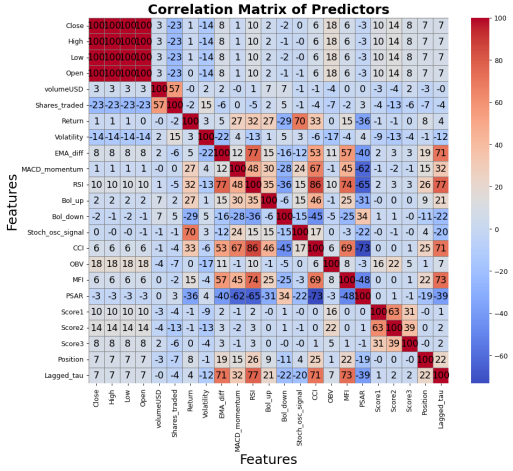


Figure 3: Correlation matrix of the predictors.

5.2 Mixed-Effects Models for Significance Tests

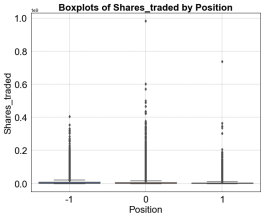
Now I want to test whether each continuous variable, by itself, is significantly associated with `Position`, while properly accounting for the hierarchical (panel-like) structure of the data (i.e., multiple stocks, each with time-series observations). By fitting a separate mixed-effects model for each variable, I essentially treat `Position` as a fixed effect, examining how the mean of each variable differs across the three classes (–1, 0, 1), while simultaneously including a random intercept for each ticker. This random intercept captures stock-specific baseline shifts, helping prevent spurious findings that would arise if I simply pooled all data without recognising that observations from the same stock are linked. Consequently, standardising the data is not strictly necessary here, because (i) the random intercept already absorbs differences in overall mean levels among tickers, and (ii) all variables are either bounded, or anyway in a form that makes them comparable across different stocks, as noted in Section 3.1. Note, however, that this approach does not address possible differences in variance across stocks, a detail that I consider uncritical at this stage. For each continuous predictor X, I fit the following model:

$$X_i = \beta_0 + \beta_1 \cdot \mathbb{I}[\text{Position}_i = 0] + \beta_2 \cdot \mathbb{I}[\text{Position}_i = 1] + b_{\text{ticker}(i)} + \varepsilon_i$$

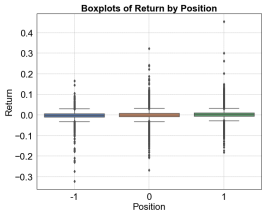
with $b_{\text{ticker}(i)} \sim \mathcal{N}(0, \sigma_b^2)$ and $\varepsilon_i \sim \mathcal{N}(0, \sigma^2)$.
Notice that the original predictors are, in this model, the targets. In each model's output, I look at the estimated fixed-effect coefficients and their statistical significance for the three `Position` categories. A significant coefficient suggests that differences in `Position` are associated with meaningful changes in that particular variable, meaning that this variable is a valid candidate for further analysis or inclusion in a predictive model. This step is just a first-pass, univariate check. Notice that no train-test split is required here, as I am trying to answer a purely descriptive question ("Does a certain predictor differ across the 3 `Position` categories?").
Carrying out this analysis separately for each continuous variable gave interesting results. Nearly all models showed very low p-values for the `Position` coefficients, except for the model of `Shares_traded`, `Score2`, and `Score3`. This shows that the mean of all variables except for these 3 significantly differs across different `Position` classes, suggesting that all covariates apart from these 3 are strongly associated with `Position`, at least when modelled alone. However, as Section 5.7 will clearly show, it is possible for a variable to be non-significant alone yet improve prediction when combined with other variables. In addition, for most features, the group-level random-effect variance is extremely low (often near zero). This indicates that, after accounting for `Position`, given the data and the model, the differences between tickers do not explain significant variability in these predictors (i.e., the relation between each of these predictors and `Position` is essentially the same for all tickers). This furthers validates my choice of developing a single model that works on every stock, without considering `ticker` (or any function of it) as a predictor. This is also consistent with the fact that many stocks are highly correlated, as discussed in Section 2.1, and helps avoid unnecessary complexity in the model. By contrast, the variables `CCI`, `OBV`, and especially `Shares_traded` show moderate-to-high random-effect variance, implying that certain stocks have notably different baseline levels for those predictors. This fact will reveal crucial in Sections 5.4 and (especially) 5.5.

5.3 Visual Exploration of Features by Position

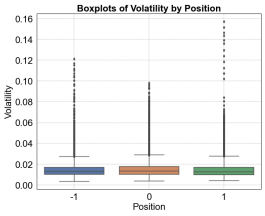
To better visualise the relation between each of my covariates and the target variable `Position`, I plot, for each continuous variable, 3 boxplots based on the values of `Position`, and for each categorical variable, its value counts again based on `Position` classes. These images are shown in Figure 4 below. From it, I observe that (i) the continuous variables are significantly spread-out (which was expected, since financial variables often exhibit fat-tail distributions), (ii) certain predictors, such as `RSI`, `CCI`, `MFI`, and `Lagged_tau`, show noticeable trends related to `Position`, (iii) others, like `Shares_traded`, `Return`, `Stoch_osc_signal`, and `OBV`, exhibit no clear differences in their first, second, and third quartiles across `Position` values, and (iv) `Bol_up` and `Bol_down` are predominantly 0. The main limitation of the univariate approaches in Sections 5.2 and 5.3 is that they examine each variable in isolation, without considering potential interactions among variables, which Section 5.1 shows are often relevant. Hence, a multivariate analysis follows.



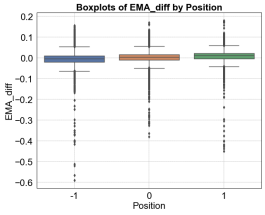
(a) `Shares_traded`



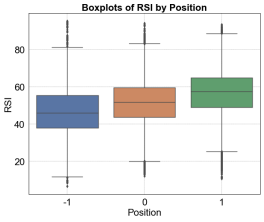
(b) `Return`



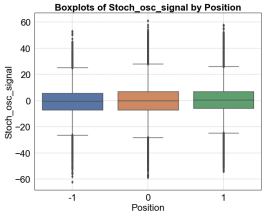
(c) `Volatility`



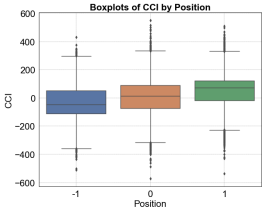
(d) `Ema_diff`



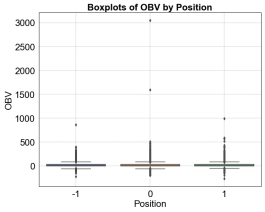
(e) `RSI`



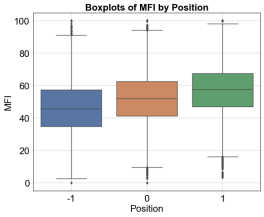
(f) `Stoch_osc_signal`



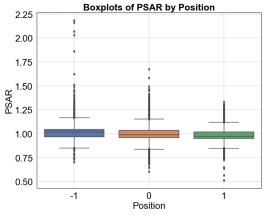
(g) `CCI`



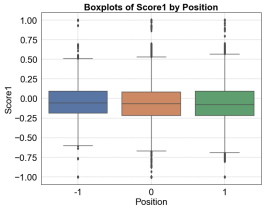
(h) `OBV`



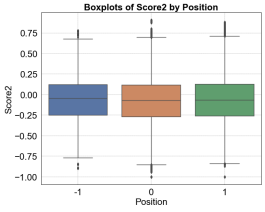
(i) `MFI`



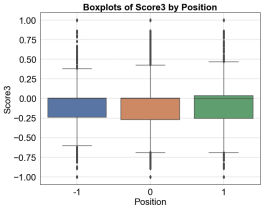
(j) `PSAR`



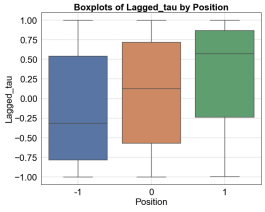
(k) `Score1`



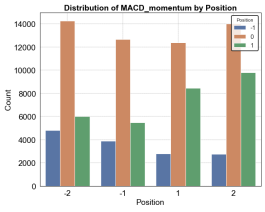
(l) `Score2`



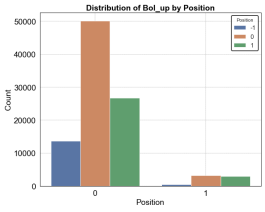
(m) `Score3`



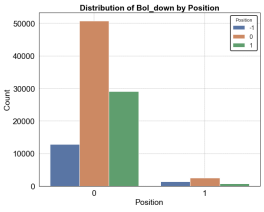
(n) `Lagged_tau`



(o) `MACD_momentum`



(p) `Bol_up`



(q) `Bol_down`

Figure 4: Distribution of predictors based on the value of `Position`. Boxplots for continuous variables, counts for categorical variables.

5.4 Likelihood Ratio Tests with Multinomial Logistic Regression

Next, I perform likelihood ratio tests (LRTs) to assess the contribution of each predictor in a multinomial logistic regression. I start by fitting a full multinomial logistic regression model, and then, in turn, I fit a reduced model that excludes a single predictor, and I compute the associated LRT statistic

$$LRT = 2[log(L_{full}) - log(L_{reduced})]$$

where L_{full} and $L_{reduced}$ are the likelihoods of the full and reduced model, respectively. I then compare this statistic to a χ^2_2 random variable to get a p-value for the significance of each predictor's effect. Because the target variable `Position` has 3 categories, each predictor contributes 2 parameters, as one category is taken as the baseline. So, when I remove a single predictor, I eliminate 2 parameters from the model, and this is why I use a χ^2 distribution with 2 degrees of freedom. Even though standardisation is often not strictly required for LRTs, I anyway standardise the variables `Shares_traded`, `CCI`, and `OBV` before running these tests, to improve numerical stability. I choose to only standardise these 3 predictors because (i) their values are not bounded (as noted in Section 3.1), and (ii) when each of them was used as the target variable with `Position` as the predictor in the mixed-effects models of Section 5.2, these variables showed moderate-to-high random-effect variance, implying that certain stocks have notably different baseline levels for them (especially `Shares_traded`). To avoid look-ahead bias (i.e., knowing, at any point, the mean and standard deviation over the whole time period), I first compute the by-ticker mean and standard deviation of each of these 3 variables over the first 2 years of data, then I use these statistics to standardise the values of the remaining 2 years, and I only use the last 2 years of data to run my LRTs. Clearly this results in features which do not have a mean and standard deviation of exactly 0 and 1, respectively, but (i) it is probably the best possible approach to avoid look-ahead bias and (ii) what is important here is that variables are on the same scale. Moreover, this standardisation is performed within each ticker rather than on the whole dataset, because different stocks have different price levels, trading volumes, and volatilities, so mixing data from all stocks would cause distortions (stocks with higher baseline levels would disproportionately influence the mean and standard deviation of variables, leading to misleading standardisation). The results of this analysis are summarised in Table 1, which sorts rows based on the p-value of the LRT. From this, many variables appear strongly relevant to predict `Position`, even when added to a model with all the other variables. On the other hand, `Lagged_tau`, `CCI`, `Stoch_osc_signal`, and `Return` have a relatively high p-value, and `Bol_up` and `MACD_momentum` are not significant at any acceptable confidence level, which indicates they provide little to no explanatory power for predicting `Position` when added to a logistic regression model with all the other covariates. Of course, the significance of any predictor depends on the model used (as I will show in Section 5.7, a more complex model is able to capture more intricate relationships among the variables).

Variable	LR Statistic	p-value	Significance
Score1	56.23	6.160578e-13	***
Volatility	50.96	8.597027e-12	***
RSI	49.56	1.731278e-11	***
PSAR	48.56	2.855287e-11	***
const	42.87	4.904944e-10	***
EMA_diff	36.34	1.288063e-08	***
Score2	34.56	3.122272e-08	***
OBV	33.56	5.170181e-08	***
Bol_down	32.21	1.015111e-07	***
Score3	19.91	4.748603e-05	***
Shares_traded	14.51	7.061139e-04	***
MFI	12.81	1.650431e-03	**
Lagged_tau	7.41	2.461651e-02	*
CCI	6.85	3.259268e-02	*
Stoch_osc_signal	6.51	3.862947e-02	*
Return	4.90	8.610995e-02	.
Bol_up	1.45	4.852751e-01	
MACD_momentum	0.65	7.228641e-01	

Table 1: Results of likelihood ratio tests. Significance codes: 0 '***' , 0.001 '**', 0.01 '*', 0.05 '.', 0.1 ' '.

5.5 Variable Ranking with Iterative LASSO (again with Multinomial Logistic Regression)

To ensure the findings of Section 5.4 are reliable, I perform a further variable ranking procedure using Least Absolute Shrinkage and Selection Operator (LASSO). Because `Position` classes are relatively imbalanced, I begin by undersampling my data, i.e., eliminating observations from the dominant classes until each class is represented equally. This step mitigates bias (without it, the model would default too often to predicting 0, the dominant class), although it clearly "throws away" important information (as the model is trained on fewer data). Even though this is time-series data, removing some samples from the majority class does not undermine the overall validity of the approach, but I acknowledge that random undersampling disrupts the temporal sequence to a degree. Next, I fit a multinomial logistic regression with L1 regularisation and a high penalty term, which means that no variable is added to the model. Then, I gradually decrease the penalty term, re-fit the model, and track the order in which predictors are added to the logistic regression. As the penalty term decreases, more variables are added to the model, until eventually all of them are, and features added earlier are considered more influential. Since the LASSO penalty term is applied to all predictors equally, it is essential to ensure they share the same scale, otherwise variables with large baseline values (e.g., `Shares_traded`) would appear more relevant than they actually are. Therefore, I train these models on the last 2 years of data, standardised following the procedure in Section 5.4 (i.e., with the by-ticker mean and standard deviation of the previous 2 years of data), but this time I standardise every continuous feature, not just `Shares_traded`, `CCI`, and `OBV`, as now it is crucial for the magnitudes of all predictors to be as similar to each other as possible. This process results in the following list, called $logit\_sorted\_features$, where elements are sorted according to when they were added to the model in the above procedure: [`MACD_momentum`, `OBV`, `CCI`, `RSI`, `EMA_diff`, `PSAR`, `Volatility`, `Score2`, `MFI`, `Score3`, `Shares_traded`, `Stoch_osc_signal`, `Bol_down`, `Lagged_tau`, `Return`, `Score1`,

Bol_up] (notice that the regularisation is not applied to the constant). This ranking is not too dissimilar to the one in Table 1, given by the LRTs (e.g., Stoch_osc_signal, Return, Lagged_tau, and Bol_up, which were considered not particularly relevant by the LRTs, are also at the bottom of the *logit_sorted_features* list). This reinforces the idea that both approaches have validity. The second approach is more reliable, as the p-values of the LRTs can be misleading in the presence of multicollinearity (variables that share explanatory power may appear less significant), while LASSO selects variables sequentially based on their true contribution to the model's predictive power. So, I consider the ranking of *logit_sorted_features* to be the actual order of features by importance for the logistic regression model.

5.6 Optimal Number of Predictors in Multinomial Logistic Regression

To decide how many features to keep in the logistic model, I split my data between a train set (first 3 years) and a test set (last year), both containing all the stocks, I again standardise both sets using the by-ticker means and standard deviations computed on the train set, I rebalance the train set with undersampling, and train 17 models. At each i^{th} step (for $i \in \{1, ..., 17\}$), I train on the train set a model with the most important i features, and I compute its accuracy on the test set. Figure 5 plots this out-of-sample accuracy as a function of the number of predictors used. From this, we see that having more than 7 predictors (MACD_momentum, OBV, CCI, RSI, EMA_diff, PSAR, and Volatility) results in no improvement in out-of-sample performance. From this plot we also notice that the maximum out-of-sample prediction accuracy is only 0.33, essentially equal to what I would have with random predictions and balanced classes. Moreover, none of the sentiment score variables are included in the final logistic regression model. It is clear that either (i) this data is highly unpredictable, or (ii) a more powerful model is necessary. I now try to improve this out-of-sample performance with a more complex model, namely a random forest.

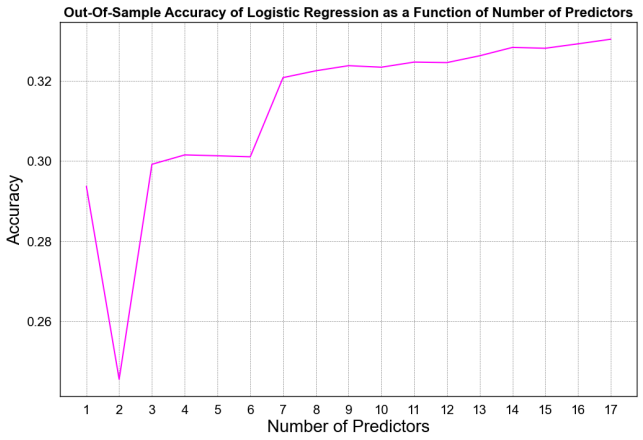


Figure 5: Out-of-sample prediction accuracy of the multinomial logistic regression model, as a function of the number of variables used.

5.7 Similar Analysis with Random Forest

Here I work directly on the non-standardised data because random forests are trained by creating a split at each node of each tree, which essentially compares the value of a certain feature to a threshold. This makes the scale of the variables almost irrelevant, as random forests do not compute any distance measures or gradients, so large or small values do not inherently dominate a split. In order not to disregard important information, I also do not perform undersampling, which is admissible as random forests are trained on multiple bootstrapped samples of the original dataset, which implies that different trees see different class distributions during training. Moreover, the distribution of Position is not massively skewed.
With the above in mind, I train a random forest model on all available data, and I derive a measure of feature relevance based on how much each feature reduces the Gini impurity at each node where it is used and how frequently it is used. The results are summarised in Table 2, which sorts features by importance. Then, to identify the optimal number of predictors, I proceed as I did for the logistic regression model in Section 5.6, training a model with the i most important variables on the first 3 years of data and testing it on the remaining year, for $i \in \{1, ..., 17\}$, obtaining the results displayed in Figure 6.

Feature	Importance
Score2	0.098365
Score1	0.094841
OBV	0.074778
Shares_traded	0.074238
Volatility	0.074157
EMA_diff	0.073856
Lagged_tau	0.069735
RSI	0.068883
Score3	0.067028
MFI	0.064742
PSAR	0.064601
CCI	0.063256
Stoch_osc_signal	0.049398
Return	0.047838
MACD_momentum	0.011498
Bol_up	0.001576
Bol_down	0.001209

Table 2: Feature importance for the random forest model.

This plot shows a massive improvement compared to the results of the logit model, as the out-of-sample accuracy now reaches 52% (with 10 predictors), which justifies the use of a less interpretable ML algorithm (a random forest)

compared to a logistic regression, as the former likely spots patterns in the data that were missed by the logistic regression (and indeed considers more predictors as relevant). The features I choose to employ for prediction, stored in a list called $rf\_features$, are the 10 most important ones for the random forest model, namely `Score2`, `Score1`, `OBV`, `Shares_traded`, `Volatility`, `EMA_diff`, `Lagged_tau`, `RSI`, `Score3`, and `MFI`. The simpler logit model could not exploit the predictive power of the sentiment score variables, while with the random forest all three `Score*` predictors are deemed relevant, and this further strengthens the validity of my approach for defining them. Importantly, even though it is true that random forests perform an "implicit feature selection" by ignoring unhelpful variables, this analysis is not superfluous, as random forests could nevertheless overfit (even though in this case the model does not, as out-of-sample accuracy does not decrease when all predictors are used). I carried out a similar analysis both with an extreme gradient boosting model and with a neural network, and the out-of-sample accuracy I obtained was not particularly different to the one of the random forest. My model of choice is therefore the random forest. Importantly, the fact that I have used the data for the whole period to come up with this model and this list of relevant predictors, and then I use them to predict the EOD positions on the same data, might seem problematic at first glance. However, consider that:

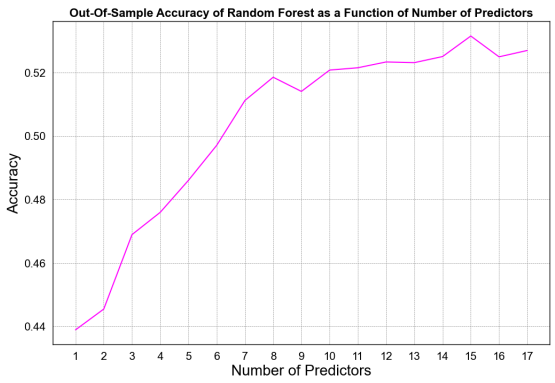


Figure 6: Out-of-sample prediction accuracy of the random forest model, as a function of the number of variables used.

- As Figure 6 clearly shows, the random forest does not overfit this data. Therefore, if I only cared about generating the EOD positions and backtesting a trading strategy using them (i.e., what I do in Sections 6, 7, and 8), I could have simply skipped Section 5 and used all variables for predictions. Moreover, the performance of the random forest is very similar to what I obtained with other ML algorithms (neural networks and gradient boostings), so the results of the following sections are not particularly dependent on the choice of the random forest as a model.

- If I wanted to be extremely precise, it would be impossible to meet the project's requirement (generating EOD positions for all stocks on each day). Indeed, it is true that I could have used an initial part of the data to perform model and variable selection, and I could have then predicted the EOD positions on the other part of the data. However, to perform a meaningful model and variable selection, as it is clear throughout Section 5, a huge amount of data is needed. So, performing variable and model selection on, say, 3 years of data, to then obtain EOD positions only on the last year (and thus setting the EOD positions of the first 3 years to 0 by default), would have avoided the problem described here, but it would have also not met the specific task of generating the EOD positions for each stock on each day. Nevertheless, I consider it admissible to initialise the first few EOD positions to 0, as I indeed do in Section 6.

- A crucial part of this work is evaluating how the model performs in monetary terms, which is done in Sections 7 and 8. If I only had the EOD positions of the last year, I would have missed many of the insights I discover in those two sections.

6 Creating the End-Of-Day Positions

After having identified my model of choice (a random forest classifier) and the variables to employ for prediction (stored in the $rf\_features$ list), I actually create the EOD positions as outlined in Section 4. Therefore, each prediction is the expected value of `Position` as implied by the predicted class probabilities, i.e., $EOD\_position = -1p_{-1} + 0p_0 + 1p_1 = p_1 - p_{-1}$, with p_{-1}, p_0, and p_1 being the probabilities of the observation to have `Position` of –1, 0, and 1, respectively, as predicted by the model. Now, I define *unique_dates* as an array that contains all unique dates in the **market** dataset, except for the first 29 days. As mentioned at the beginning of Section 5, the first 29 days are excluded from training, as they contain some artificially imputed variables that were originally null. Their EOD positions will default to 0. For this task I proceed in batches of length 40. Specifically, I set the EOD positions of the first 40 days in *unique_dates* for all stocks to 0, as they do not have enough past reliable data to accurately train the model. In other words, the first 29+40=69 days of the **market** dataset have an `EOD_position` value of 0 by default. Then I use the first 40 days in *unique_dates* to train a random forest, and I predict the EOD positions of the following 40 days with this model. At this point, I re-train the model on the first 80 days in *unique_dates*, and I use it to predict the EOD positions of the following 40 days. This process goes on until every stock has an EOD position on every day. Notice that this procedure never trains on the first 29 days nor on the last 20 days of **market** (that have a `Position` value

of 0 by default, and are thus not reliable). It is obvious that earlier EOD positions are less accurate, because they are obtained with a model trained on little data, but employing a different approach would risk introducing look-ahead bias. Indeed, another possibility could have been to train the model on a certain number of stocks, and then use it to predict the EOD positions of the remaining stocks, in a kind of leave-k-out fashion. The problem with this approach is that predictions would be made for the same period as the training data, so the model could leverage patterns learned by looking at the whole time period to make predictions for observations in the past, therefore risking to introduce a look-ahead bias. This is why this methodology is not implemented.
The procedure outline above results in a column, `EOD_position`, containing the required EOD positions for each stock on each day. So-computed EOD positions appear "rounded" to the nearest 0.01. The reason is that, when making predictions, each decision tree in the forest outputs a class prediction for a given sample, and the final probability estimates for each class are the fractions of trees that predict each class. Since I use 100 trees in my random forest models, estimated class probabilities (and thus EOD positions) have a precision of 0.01.

7 Backtesting a Trading Strategy

In this section I implement a trading strategy. For each stock, the strategy I propose starts with no open trade at the beginning of the studied period, and looks at the `EOD_position` value from the previous day. Indeed, since `EOD_position` at time t is only available at the end of the trading day, it can only be used for trading on the next day. Therefore, the `EOD_position` column is shifted by one day, so that on every day t the strategy uses day $t-1$'s `EOD_position` value. This strategy considers a signal to be positive if the EOD position is above a certain $threshold \in (0,1)$, negative if the EOD position is below the negative of *threshold*, and 0 otherwise. This "discretised" and shifted variable is first encoded in a column named `Signal` $\in \{-1,0,1\}$ (which clearly depends on *threshold*), and it is then used for trading. The strategy follows each stock separately. Whenever `Signal` changes from the previous day, the strategy first closes any existing trade at the day's open price, then opens the new trade if `Signal` is non-zero, investing a fixed fraction of the available capital ($capital\_fraction$). The number of shares traded is an integer, based on how much capital is allocated divided by the current stock price, so that no fractional shares are permitted. Whenever the strategy trades (either to open or close), it pays a transaction cost of $transaction\_cost$*100%. At the end of each trading day, my algorithm computes the total equity using the closing price. This logic, encoded in the *backtest_single_ticker*() function, is applied to each ticker separately with the *backtest_all_tickers*() function. The backtests are run by allocating \$10,000 for each stock, by investing all available capital at each trade (so that $capital\_fraction = 1$) and by assuming transaction costs of 0.1% on each trade. It is possible to experiment with leverage by setting $capital\_fraction > 1$. When this strategy is tested on the actual `Position` column (the target variable itself) it performs astonishingly well, with a total PnL of 233.98%, annualised PnL of 35.19%, annualised volatility of 5.3%, a Sharpe ratio of 6.07 (assuming a 3% risk-free rate from January 2000 to December 2003), and only minimal drawdowns. These results, shown in Figure 7, were expected, since `Position` is computed by looking at future prices, and are clearly unrealistic for a live trading strategy, but they are anyway a confirmation of the validity of my approach for defining the target variable.

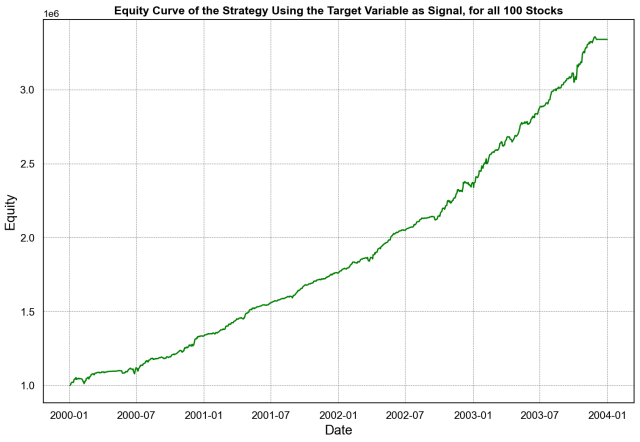
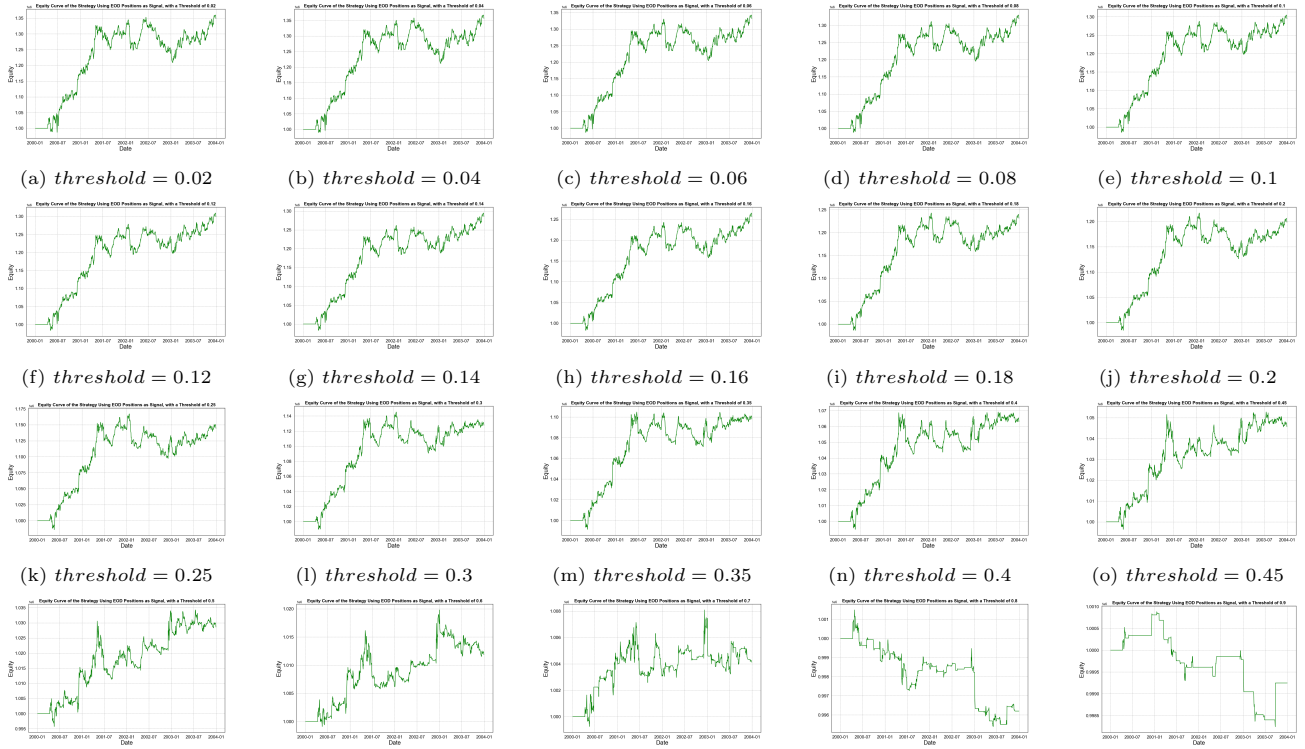


Figure 7: Equity curve of the strategy backtested using the actual target variable `Position` as the signal, for all 100 stocks. This corresponds to the performance of a predictive algorithm with 100% out-of-sample accuracy. These results are not meant to prove the validity of my predictive models, but just of the way I have created the target variable `Position`.

A backtest of my strategy on the `EOD_position` column, obtained as explained in Section 6, yields instead the results in Figure 8 and Table 3, for 20 different *threshold* values (0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.18, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.6, 0.7, 0.8, and 0.9). The strategy was again tested by allocating \$10,000 per stock. For small *threshold* values, the strategy obtains a positive Sharpe ratio (although never higher than 0.61). Notably, as *threshold* increases, the PnL and the Sharpe ratio tend to decrease. To understand why this makes sense, it is important to consider that `EOD_position` is usually quite close to 0 (if, e.g., $p_{-1} = 0.35$, $p_0 = 0.25$, and $p_1 = 0.4$, the random forest algorithm will predict a `Position` of 1, but an `EOD_position` of only $0.4 - 0.35 = 0.05$, reflecting the high uncertainty implied by the estimated class probabilities). This shows why even small deviations of `EOD_position` from 0 can be meaningful. As *threshold* increases, the algorithm clearly performs less trades, which implies the portfolio volatility

decreases (with a low *threshold* value, it is reasonable to assume that the strategy has money invested in all stocks at almost every time). This implies that, if *threshold* is high, once the strategy opens a trade, it will be difficult to close it, as a high degree of certainty in predictions will be required. This could be one of the reasons why smaller *threshold* values tend to yield higher PnL (they close losing trades quickly). The highest 4-year PnL (35.61%) and Sharpe ratio (0.61) are obtained with a *threshold* of 0.04, so this is the value I deem optimal.



(a) $threshold = 0.02$ (b) $threshold = 0.04$ (c) $threshold = 0.06$ (d) $threshold = 0.08$ (e) $threshold = 0.1$

(f) $threshold = 0.12$ (g) $threshold = 0.14$ (h) $threshold = 0.16$ (i) $threshold = 0.18$ (j) $threshold = 0.2$

(k) $threshold = 0.25$ (l) $threshold = 0.3$ (m) $threshold = 0.35$ (n) $threshold = 0.4$ (o) $threshold = 0.45$

(p) $threshold = 0.5$ (q) $threshold = 0.6$ (r) $threshold = 0.7$ (s) $threshold = 0.8$ (t) $threshold = 0.9$

Figure 8: Equity curve of the strategy backtested using `EOD_position`. Each plot corresponds to a different *threshold* value.

Threshold	Total PnL	Ann. PnL	Ann. volatility	Sharpe ratio
0.02	34.85%	7.76%	8.3%	0.57
0.04	35.61%	7.91%	8.06%	0.61
0.06	33.13%	7.42%	7.56%	0.58
0.08	31.87%	7.16%	7.19%	0.58
0.10	29.46%	6.67%	6.74%	0.54
0.12	29.91%	6.76%	6.49%	0.58
0.14	28.53%	6.48%	6.02%	0.58
0.16	25.63%	5.87%	5.59%	0.51
0.18	23.13%	5.34%	5.23%	0.45
0.20	19.87%	4.64%	4.89%	0.33
0.25	14.45%	3.43%	4.09%	0.11
0.30	12.77%	3.05%	3.46%	0.01
0.35	9.8%	2.36%	2.81%	-0.23
0.40	6.3%	1.54%	2.2%	-0.66
0.45	4.57%	1.12%	1.77%	-1.06
0.50	2.86%	0.71%	1.36%	-1.68
0.60	1.19%	0.3%	0.8%	-3.39
0.70	0.41%	0.1%	0.43%	-6.81
0.80	-0.38%	-0.1%	0.19%	-16.43
0.90	-0.08%	-0.02%	0.08%	-37.27

Table 3: Performance metrics of the strategy backtested using `EOD_position`. Each row corresponds to a different *threshold* value.

8 Constructing a Dynamic Portfolio

8.1 Summary, Motivations, and Assumptions of the Dynamic Strategy

Since, as seen in Section 2.1, some stock pairs are highly correlated while others often move in opposite directions, investing the same amount of capital in every stock is not ideal. In this section, I explore an effective and efficient capital allocation strategy that leverages the similarity among certain stocks. As outlined in Section 6, the **market** dataset includes a column named `EOD_position`, which is used for backtesting the trading strategy. In what follows, I divide the dataset into consecutive 40-day segments and, within each segment, cluster stocks based on their closing price correlations. This ensures that stocks within the same cluster at segment t experienced similar price movements over that period. After analysing and commenting on the obtained clusters, I backtest an updated version of my trading strategy. Specifically, for each 40-day segment (starting from the second), I assess how my strategy would have performed on each stock over the previous segment. Then, using the clusters of the previous period, I select the best-performing stock from each cluster, choosing only among the stocks on which my strategy had a positive PnL. Therefore, at each period starting from the second one, I have a list, called $best\_performers$, which is composed of stocks that, over the previous period, had 2 important characteristics: (i) they were relatively uncorrelated with one another (since they belonged to different clusters); and (ii) if my strategy had been implemented on them, it would have yielded relatively high returns over the previous period. Importantly, at any period, the list of best performers is created using only data from the previous period (both for the PnL calculations and the clusters), ensuring that no look-ahead bias is introduced. At each period from the second onward, I then distribute my available capital equally among the selected best performers, and track the returns of my strategy over that period. This approach aims at investing in uncorrelated stocks on which my strategy would have yielded high returns in the previous period. It is important to note that the stocks in $best\_performers$ are not necessarily those whose prices increased the most over the previous period. Instead, they are the ones on which my strategy, leveraging the previously computed EOD positions, would have been the most profitable over the previous period.
A crucial motivation behind this frequent re-clustering of stocks is the well-documented fact that stock correlations are non-stationary. Hopefully, updating the clusters every 40 days strikes a balance between flexibility (the strategy adapts to the latest correlation structure) and stability (updating clusters too frequently could introduce excessive noise, as short-term fluctuations in correlations may not reflect meaningful relationships). Another important motivation is diversification. Indeed, picking a maximum of one stock per cluster aims at reducing exposure to redundant risk, as two stocks in the same cluster are necessarily highly correlated (at least over that cluster's period), so every time I pick a stock from a cluster, I am automatically deciding not to invest in other stocks correlated to it (in the previous period). However, as I will explain below, it is still possible for two stocks in two different clusters to be highly correlated, although my approach should sharply reduces the correlation between the assets I invest in compared to the simple strategy of Section 7. A further motivation is that I want my strategy to dynamically adapt to different market regimes. Indeed, in periods in which correlations between stocks are high, I obtain fewer distinct clusters, and thus I invest in fewer stocks. On the other hand, in times when stocks are less correlated, this strategy naturally invests in more securities.
An important assumption here is that, although correlations between stocks are known to be non-stationary, they do not change drastically from one period to the next. This assumption allows me to use clusters from the previous period when selecting stocks for investment in the current period. A further assumption I make is the one of predictability. Specifically, at any 40-day period, I only trade on the stocks on which my strategy would have been the most profitable over the previous period. This involves the assumption that some stocks are more predictable and/or tradable than others, meaning that stocks on which the strategy performed well in the past are more likely to continue performing well in the near future, due to certain patterns in their movements. Clearly I do not assume this predictability to last any longer than two consecutive periods. This is a practical compromise between full predictability (which is unrealistic) and complete randomness (which would disregard useful information from past data).
Each step of this strategy is now explained in detail.

8.2 Identifying Groups of Correlated Stocks

The first step is to cluster together similar stocks. As I did in Section 2.1, I work with a dataset called **market_by_ticker**, which is now indexed by dates. This dataset has 100 columns named after the stocks (i.e., "S01", "S02", ..., "S100"), each containing the daily close prices of a different stock. Every 40 trading days, I perform the following steps:

> I cluster stocks based on their close price correlation, ensuring that each stock has a minimum correlation of 75% with every other stock in the same cluster, while aiming to minimise the number of clusters (notice that reaching the global optimum is an NP-hard problem). To do this, I first compute the correlation

matrix of stock prices over the current 40-day period, which is a symmetric 100×100 matrix where each entry represents the correlation between two stocks. Next, I iterate through the list of stocks and assign each unclustered stock to a new cluster. I then attempt to add more stocks to this cluster, ensuring that each added stock satisfies the minimum correlation constraint with every other stock in the same cluster. This greedy process continues until no more stocks can be added, so that clusters are formed as early as possible. I subsequently implement a refinement phase in which I iteratively adjust clusters by allowing stocks to move between them. In this phase, I iterate through the clusters using a for-loop, moving stocks to a different cluster if (i) the move still satisfies the minimum correlation constraint, and (ii) the stock is moved to a cluster with at least as many elements as the current one. This step is designed to merge smaller clusters into larger ones while maintaining the correlation constraint. The for-loop terminates immediately after moving a stock, ensuring that each stock is only evaluated once per iteration. This prevents excessive reassignments and allows for more controlled convergence. The refinement process is governed by a while-loop that continues until convergence, with a safety limit of 10,000 maximum iterations to prevent infinite loops. The algorithm stops whenever a certain iteration of the while-loop results in no changes compared to the previous one, indicating that a stable clustering configuration has been reached, thus reaching convergence. This method ensures that the clustering process is both efficient and stable. The refinement steps also mitigate the order dependence of the greedy phase, reducing the sensitivity of the clusters to the order in which stocks are processed.

As a result, every 40 days, I get a different set of clusters. Indeed, stock correlations are known to be non-stationary, as the fact that two stocks were correlated in the past does not necessarily imply that they will remain correlated in the future, so it makes sense to iteratively modify the set of "similar stocks". Despite working with a dataset covering 100 stocks over a 4-year period, the clustering process is surprisingly efficient in terms of computational time.

8.3 Comments on the Stock Clusters

Table 4 provides interesting insights into the obtained clusters. Although the total number of clusters in each 40-day period is quite high, there are always a significant number of single-stock clusters. This means that, if we exclude these stocks, the remaining ones form only a handful of groups. This is summarised in the `Notes` column of Table 4. For instance, in the first period, 87 stocks are grouped into just 18 clusters, meaning that every two stocks within the same cluster exhibit a correlation of at least 75%, which is quite remarkable. As explained in Section 8.1, this implies that, at every 40-day period, the strategy only invests in a (hopefully uncorrelated) subset of stocks.

Period	Num clusters	Single-stock clusters	Notes
(2000-01-04, 2000-02-26)	31	13	Can group 87 stocks in 18 clusters
(2000-02-29, 2000-04-26)	43	20	Can group 80 stocks in 23 clusters
(2000-04-27, 2000-06-21)	31	13	Can group 87 stocks in 18 clusters
(2000-06-22, 2000-08-16)	23	6	Can group 94 stocks in 17 clusters
(2000-08-17, 2000-10-11)	37	12	Can group 88 stocks in 25 clusters
(2000-10-12, 2000-12-06)	34	18	Can group 82 stocks in 16 clusters
(2000-12-07, 2001-02-01)	38	18	Can group 82 stocks in 20 clusters
(2001-02-02, 2001-03-29)	37	20	Can group 80 stocks in 17 clusters
(2001-03-30, 2001-05-29)	25	17	Can group 83 stocks in 8 clusters
(2001-05-30, 2001-07-24)	41	20	Can group 80 stocks in 21 clusters
(2001-07-25, 2001-09-18)	40	17	Can group 83 stocks in 23 clusters
(2001-09-19, 2001-11-13)	46	26	Can group 74 stocks in 20 clusters
(2001-11-14, 2002-01-11)	45	25	Can group 75 stocks in 20 clusters
(2002-01-12, 2002-03-08)	24	7	Can group 93 stocks in 17 clusters
(2002-03-09, 2002-05-08)	37	22	Can group 78 stocks in 15 clusters
(2002-05-09, 2002-07-03)	41	23	Can group 77 stocks in 18 clusters
(2002-07-04, 2002-08-28)	41	20	Can group 80 stocks in 21 clusters
(2002-08-29, 2002-10-23)	34	19	Can group 81 stocks in 15 clusters
(2002-10-24, 2002-12-18)	37	21	Can group 79 stocks in 16 clusters
(2002-12-19, 2003-02-15)	25	11	Can group 89 stocks in 14 clusters
(2003-02-18, 2003-04-12)	40	22	Can group 78 stocks in 18 clusters
(2003-04-15, 2003-06-12)	40	22	Can group 78 stocks in 18 clusters
(2003-06-13, 2003-08-07)	38	18	Can group 82 stocks in 20 clusters
(2003-08-08, 2003-10-02)	33	20	Can group 80 stocks in 13 clusters
(2003-10-03, 2003-11-27)	35	20	Can group 80 stocks in 15 clusters

Table 4: Summary of the stock clusters, one for each 40-day period.

Figure 9 visually represents the correlation structure among the 100 stocks across all 25 40-day periods. Each node in these graph corresponds to a different stock, and stocks of the same colour belong to the same cluster (meaning that all stocks of the same colour exhibit a correlation of at least 0.75 with each other over that 40-day period). Two stocks are linked by an edge if their correlation is at least 0.75. So, same-colour (i.e., same-cluster) stocks are all connected by edges, but two connected stocks do not necessarily belong to the same cluster. As expected, strongly correlated stocks tend to cluster together, forming groups of interconnected nodes. In almost all networks, the central region is densely populated with highly interlinked stocks, indicating that these stocks exhibit strong mutual correlations. These stocks might belong to the same sector, or anyway be influenced by the same market dynamics. On the other hand, each network also contains isolated peripheral stocks that form small, independent clusters and are uncorrelated with most other stocks. They may represent niche industries and/or idiosyncratic movers. The presence of these peripheral stocks highlights why clustering all 100 stocks in every 40-day period requires a relatively large number of groups. However, if these stocks are excluded, the required number of clusters drops dramatically, as reflected in the `Notes` column of Table 4. One key feature of these networks is the very low stationarity of the clusters: indeed, in some cases, even from one period to the next one, the groups of correlated stocks are quite different. This means the data on these stocks has a high degree of randomness, and similar results are obtained with different period lengths. In the section below, I try to leverage these clusters to construct properly diversified portfolios.

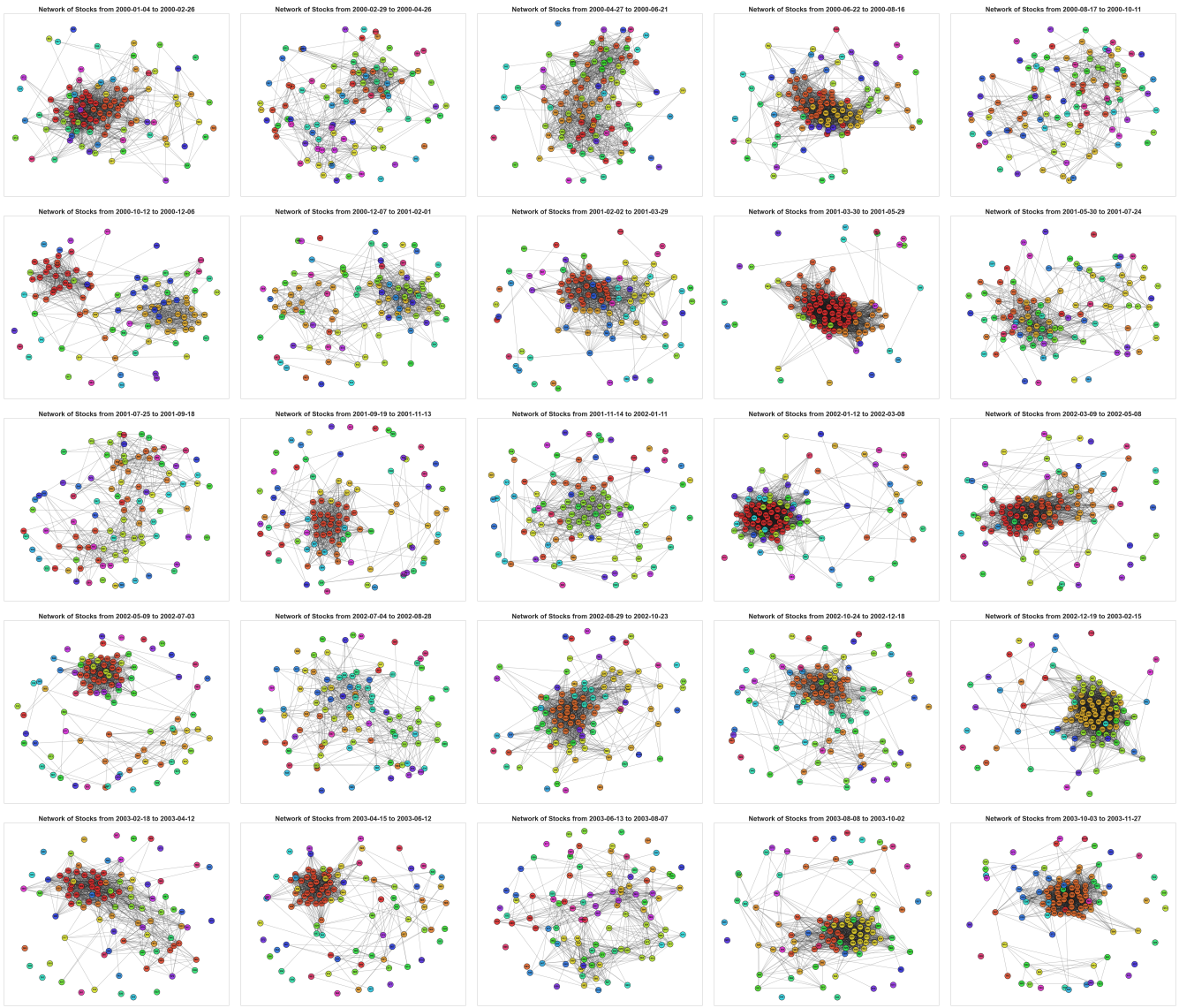


Figure 9: In these networks, each node corresponds to a stock and each colour to a cluster. Two stocks are connected with a link if their correlation is at least 75%.

8.4 Backtest of the Dynamic Strategy

Now I actually implement the enhanced trading strategy. As explained at the beginning of Section 8, I iterate over the 40-day periods starting from the second onward. At each period, I create a dictionary, called *ticker_pnl*, whose keys are the 100 stocks ("S01", "S02", ... "S100"), and whose values are the PnL's of my simple strategy if applied to a certain stock over the previous 40-day period. Then, using the clusters of the previous period, I select the best-performing stock from each cluster, choosing only among the stocks on which my strategy had a positive PnL. This results in a list called $best\_performers$. Clearly, if all the previous period's clusters contain at least one stock on which my simple strategy would have been profitable over the previous period, the length of $best\_performers$ is the total number of clusters in the previous period. If instead none of the previous period's clusters contain a stock on which my simple strategy would have been profitable over the previous period, the length of $best\_performers$ is 0. This happens at the first iteration, since the `EOD_position` column has all 0's at the beginning of the period. At each 40-day period, the $best\_performers$ list is created by using only data from the previous period (both for the PnL calculations and the clusters), ensuring that no look-ahead bias is introduced. At each period from the second onward, I then distribute all of my available capital equally among the selected best performers, and track the returns of my strategy over that period. Here I make a conservative move that (i) biases downwards the return of this backtest, and (ii) allows me to simplify the code. Specifically, I assume that, at the end of any 40-day period, I pay a transaction cost equal to 0.1% of my current capital. In reality, this would only happen in the (extremely unlikely) worst-case scenario, the one in which, at the end of all periods, I have money invested in all stocks in $best\_performers$ (meaning that the `Signal` column for all of them was not 0), and all the stocks in the next-period's $best\_performers$ are not in the current-period's $best\_performers$. Only in this case I would have to close a number of positions equal to the length of $best\_performers$ at the end of every 40-day period, and thus pay transaction costs of 0.1% of my current total equity. To be precise, I would have to keep track, at the end of every 40-day period, of which stock's `Signal` columns are 0, and of the stocks that, between one period and the following one, enter and exit from the $best\_performers$. To avoid these complications, I act as just described, which makes the final PnL slightly lower than it would be without this approximation (anyway, without considering at all the transaction costs at the end of each 40-day period, the strategy results change by a very small amount).
Figure 10 plots the evolution of my capital over time as this strategy is implemented. The results are worse than those of Section 7. During the first 2 years, the strategy performs quite well, which indicates that, during this period, the correlations between stocks are more stable, and/or that the predictability condition is met (i.e., stock that would have yielded high returns with my strategy in the close past, also do so in the present). During the last 2 years, however, at least one of these conditions seems to break, as the strategy loses money. Over the whole 975-day period (24 40-day periods, as I exclude the first), the total PnL is 15.9% (annualised at 3.89%), the annualised volatility is 9.16%, and the Sharpe ratio is 0.1.

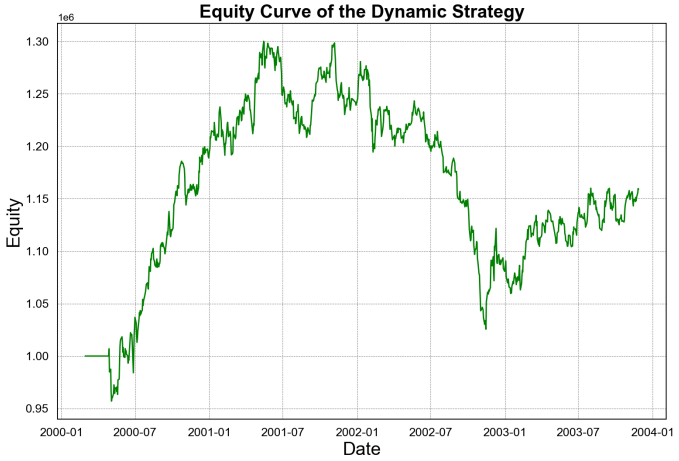


Figure 10: Equity curve of the dynamic strategy.

9 Main Achievements, Limitations and Possible Improvements

This work involves a creative procedure for creating EOD positions and testing a trading strategy using them as a signal. Among the main achievements of this analysis, I can list the thorough data checks, the comprehensive feature engineering, the definition of a stable and powerful target variable, the rigorous variable and model selection process, and the implementation of a simple and a dynamic trading strategy. All of this is achieved by avoiding look-ahead bias.
This analysis also suffers from a few limitations that, if addressed, have the potential to greatly enhance its reliability.

- First of all, knowing the stocks' name or their sector would aid in the clustering phase, ensuring each group really represents stocks that are expected to move together in the near future. This would probably increase the PnL of the dynamic trading strategy. This would also allow for the use of a categorical predictor named `Sector`, which could further improve my models.

- Additionally, my trading strategies could benefit from a few modifications, such as adding a (maybe trailing) stop-loss. This would most likely increase the PnL of the strategy with high *threshold* values, as when *threshold* is high, losing positions are usually held for too much time.

- Furthermore, my predictive models are trained by optimising predictive performance (the categorical cross-entropy for the multinomial logistic regression and the Gini impurity for the random forest), not backtest results. Since a few misclassifications could lead to large losses, maximising predictive performance can in principle significantly differ from maximising PnL, so a possible improvement to my approach is to train the ML models with a customised loss function that penalises a misclassification based on the resulting amount of money lost. Although it is also possible to assign different weights to different classes (so that a misclassifiation of the 0 class results in a lower penalty than a misclassification of the other two classes), I have experimented with this approach, without improving my backtests results.

- Then, to predict the EOD positions, the predictive models could assign more weight to more recent observations. Instead, in the current implementation, when the random forest is trained, the same importance is given to all observations, regardless of their temporal distance from the prediction point.

- Also, one could experiment with dynamic variable selection, choosing which predictors to use for any time period based on which variables would have been relevant during the previous period.

- Moreover, this data is from the early 2000's, so many of the patterns that my models have learned to identify may not be valid any more today.

- Additionally, I have framed the problem as a three-label classification one, with ML algorithms predicting `Position` $\in \{-1, 0, 1\}$, but it is also possible to instead have a binary target variable with classes "long" $(+1)$ and "short" (-1), and decide to trade only if one class is predicted with a certain minimum certainty. The resulting EOD positions would still be the expected values of the labels, $p_1 - p_{-1} \in [-1, 1]$. This would simplify the predictive models, and possibly also improve the performance of the trading strategy.

- Also, there are several possible interpretations of the **est*** datasets, and one different from the one I have used (outlined in Section 3.2) might generate sentiment scores with more predictive power.

- Lastly, I have only experimented with "traditional" ML approaches which are not specifically designed for sequential data, as they do not inherently account for the order of data points. Instead, a transformer model excels at capturing complex temporal relationships in sequential data, as it uses a self-attention mechanism to focus on different parts of a time series simultaneously. Also, its attention weights can offer interpretability by highlighting which historical points are most influential for a given prediction (those points may correspond to, e.g., earnings releases or huge changes in the **est*** datasets). So, employing a transformer to predict the target variable could, in principle, enhance performance.