Financial Big Data - Prof. Damien Challet

2019 Fall Semester Project

# Implementing a Financial Ratio Prediction-Based Investment Strategy

Allan Bellahsene, Jan Alexander Frogg

January 15, 2020

***Abstract*** **– Investing in public markets has always been about beating the market by gathering the most available information in order to get the best complete picture of a company and its financial returns. This information helps investors decide which companies to invest into. While this is true, stock prices and financial returns are extremely difficult to predict, even with a lot of publicly available information, for they are highly noisy and non-stationary. Accounting information as stock price movement predictors started to be investigated some decades ago. More specifically, changes in some financial ratios have proven to be highly correlated with the evolution of stock prices. In this work, we describe and implement a financial ratio prediction model in order to allocate capital into selected companies. We test time series models for predictions on monthly financial ratio data from 1971.01.31 to 2018.12.31 for 20'880 companies and we show that the ratio predictions yields satisfying results with simple yet effective methods and we base our work on the assumption that financial ratios are positively correlated with stock price movements.**

## I. Introduction

The most conventional way of investing in stocks would be to follow stock price or price returns trends and to buy or sell the stock accordingly. This means it would be of interest to investors to predict such data in the future, in order to know in advance which companies to invest into. If this would happen to exist today for all investors, there would be no way of making a profit by investing in the stock market. However, stock prices, and more specifically, stock returns present high noise and are highly non-stationary (as are all financial data), which makes them very difficult to process in terms of data. As an example, Figure 1 shows log-returns of stock prices for the S&P 500, from which we clearly see the noise we mentioned. Therefore, it is essential for us to either find another investing decision metric, or to go through the process of noise reduction for stock prices [1] before actually perform predictions.

The use of accounting data to explain changes in stock prices has been established, may it be earnings [2], operating cash flow information [3], [4], [5], impact factors [6] or others. Correlation of stock prices with financial ratios has also been explored. Some results give strong positive and significant relationships to stock price behaviour and trends. More specifically, we find that four specific financial ratios prove the most effective in this matter in the investment sector: the *Return on Equity* (ROE), *Return on Assets* (ROA), *Earnings per Share* (EPS), and *Price-to-Earnings* (P/E) ratios [7].

In this study, we implement an investment strategy based on a financial ratio prediction model. For this we use the four ratios mentioned above for their high potential to correlate with stock price movements, relative to other financial ratios. We explain through the following the data used for the entire analysis, as well as the financial ratio selection criteria and what prediction models we use.

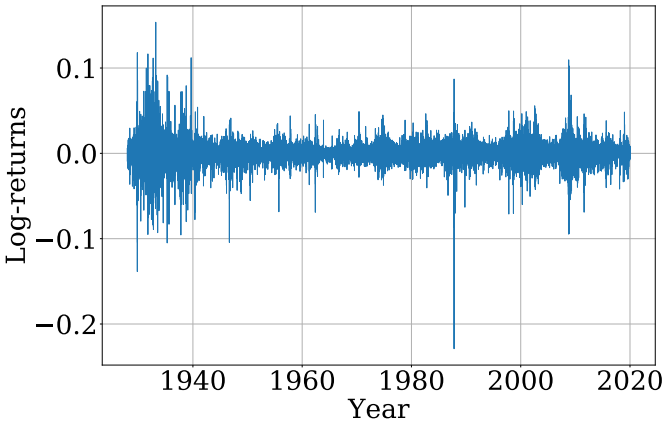


Figure 1: Time series plot of the the log-returns of the S&P 500 index price.

## II. Methods

In this section we describe the data set used for this work, as well as the investment strategy we use and other choices and assumptions we made.

***Financial ratio selection***: Firstly, our work is based on financial ratio prediction. For this, we had to choose relevant ratios to predict. While it would be possible to select them by taking a huge set of ratios and applying a dimension reduction technique such as *Principal Component Analysis* (PCA) in order to pick the most relevant ones, this would fail to include the movement of stock prices, which needs to be taken into account for our strategy. In order to cope with this, we will select specific ratios which have proved to be linked with stock price movements. The core assumption of our project is that four specific financial ratios will have an impact on stock prices: ROE, ROA, EPS and P/E [7], and therefore, we may choose to go with these four ratios for our work by keeping in mind they will be relevant for stock price movement prediction.

***Prediction models***: In order to make predictions of our selected financial ratios, we use the following standard time series models: *Autoregression* (AR), *Moving Average* (MA) and *Seasonal Autoregressive Integrated Moving-Average* (SARIMA). This is due to the time series nature of the ratios we use. It important to note that those models require the data to be stationary and to be autocorrelated.

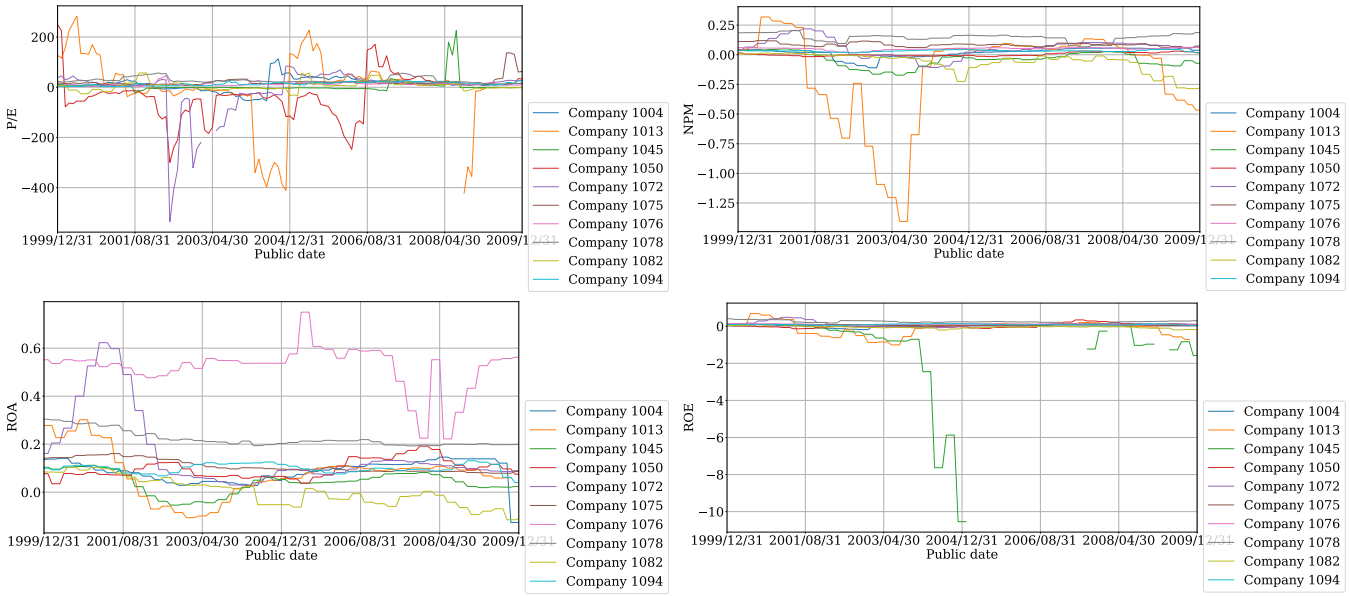


Figure 2: Plots of the four ratios over time between 1999 and 2009 for a subset of 10 companies identified by their Gvkeys.

***Dataset***: For the analysis presented in this work, we imported monthly data for the four ratios from 1971.01.31 to 2018.12.31 from the Wharton database[1]. However, it is important to note that due to the unfortunate unavailability of the EPS ratio on this database, we opted to replace it with the *Net Profit Margin* (NPM) ratio, which we assume to be similar to EPS, as it is a per share profit measure. We imported this data for all companies and dates contained in the database, which corresponds to a 2'744'771 observations by 8 quantities dataset. Table 1 shows some observations of the dataset. We may note that except for the P/E ratio, all ratios we are interested in change on a quarterly basis. This dataset contains information on 20'880 separate companies. In the following we describe how we clean and transform the data for it to be used later. We also note the difference in scale between P/E and the other ratios.

| Public date | Gvkey | Adate | Qdate | P/E | NPM | ROA | ROE |
|---|---|---|---|---|---|---|---|
| 1971/01/31 | 1000 | 1969/12/31 | 1970/09/30 | 13.433 | 0.060 | 0.287 | 0.287 |
| 1971/02/28 | 1000 | 1970/12/31 | 1970/12/31 | 65.441 | 0.041 | 0.173 | 0.181 |
| 1971/03/31 | 1000 | 1970/12/31 | 1970/12/31 | 57.353 | 0.041 | 0.173 | 0.181 |
| 1971/04/30 | 1000 | 1970/12/31 | 1970/12/31 | 63.235 | 0.041 | 0.173 | 0.181 |
| 1971/05/31 | 1000 | 1970/12/31 | 1971/03/31 | 50.735 | 0.036 | 0.173 | 0.137 |

Table 1: Five first observations of the raw dataset used in our work. The last four columns are the ratios we use for the prediction. Adate and Qdate refer to the year-end date and quarter-end date for which the variables were used for calculating the related financial ratios, respectively. The Public date is a rough estimate of when this information, i.e. the ratio value, becomes publicly available. The Gvkey is a key that identifies a company. There is a unique Gvkey per company.

Figure 2 shows all four ratios as a function of time of a given subset of 10 companies. As we can see, there are some large gaps due to missing data. We transformed the dataset in order to fill the gaps as follows: first, if the dataset of a company contains more than a certain quantity of missing values (what we define as the *maximum missing values threshold*), then the data of this company is simply removed and we will not use it, as we make the assumption that if there are more than a certain number of missing values, replacing artificially this quantity can have significant impact on our predictions later on. Else, we will treat the missing values. If there are at least two consecutive missing values, we erase both observations, as we believe linear interpolation would not be accurate since we are looking at monthly or quarterly data. If there are no consecutive missing values, then we replace single missing values by a linear interpolation between the previous and the next values. If the missing value is located at one of the edge of the dataset (first or last value), then it is replaced by the next or the previous value, respectively. Lastly, at each missing value replacement, we add a gaussian white noise (mean 0 and variance 1), in order to add some noise to our new artificial observation and make it more realistic. After applying this transformation, we convert our data to a quarterly frequency to use any one of our algorithms on these ratios, since all ratios (except P/E) change on a quarterly basis (Table 1). This leaves us with no more than 10'409 companies to analyse at this point, which is approximately half the companies we were dealing with in the raw imported data.

The features (ratios) in our dataset each have a different range. Therefore, we must normalise them in order to have a common scale among these features. This is needed to have reliable accuracies with the models we use - AR, MA, SARIMA in our case, but this can be extended

[1] `https://wrds-web.wharton.upenn.edu/wrds/ds/wrdsapps/finratiofirm/index.cfm?navId=401`

to more advanced Machine Learning (ML) models as well [8]. Such normalisation was done using a min-max scaling method - it rescales the data set such that all feature values are in the range $[0, 1]$. It is important to note that it is very sensitive to the presence of outliers, so we must take this into account when looking at the results, as Figure 2 shows there can be outliers in our data. After doing so, we remove all companies in our dataset for which we have less than 24 observations (i.e. 2 years of existence) in order to ensure a reasonable and consistent time window for our analysis. At this stage, we are left with 9'814 companies.

We must further transform our dataset in order to make it meet the requirements of the AR, MA and SARIMA models we test. That is, the dataset is assumed to be stationary - i.e., time has no effect on the data, as well as autocorrelated. We use the Augmented Dickey Fuller (ADF) test to check for stationarity [9]: the null hypothesis is that the data contains a unit root (hence, is non-stationary), while the alternative hypothesis is that the data is stationary. The preliminary test for stationarity on our data reveals that 64.6% (P/E), 89% (NPM) 82.3% (ROA), and 79% (ROE) of our company portfolio failed to reject the non-stationarity test at a 99% confidence interval (Table 2). Therefore, it is clear that we need to make our data stationary in order to apply the models we want. Since these models are to be tested on our whole dataset, we will transform it in its entirety to make it stationary, even for the time series that did not fail the test (the remaining 35.4%, 11%, 17.7% and 21%, respectively). To do this, we use the difference of the data in order to make it stationary. Figure 3 shows the number of companies that failed to reject the non-stationarity test at a 99% confidence interval, as a function of the difference order. As we can see, increasing the difference order improves the stationarity of a lot of companies, until we reach a minima - from 2 to 4 depending on the ratio, and after which the number of non-stationary companies increases again. A difference order of 2 for all ratio represents the "elbow" of the curve for approximately all four ratios, while at the same time keeping a reasonable computation time, which increases for each additional order. Table 2 summarises the fraction of companies which are non-stationary before and after having applied the $2^{nd}$ difference to each ratio's data. Hence, taking second order difference would allow us to have more companies to perform our strategy. However, finding a metric to assess whether our predictions are good or not would force us to be arbitrary, as the values would of small order of magnitude. If we were to use $1^{st}$ order differences instead, we would have less companies to make our predictions on, but the results would be immediately interpretable, as it would indicate whether a given ratio increased or decreased from a period to another. In this work we prioritise this interpretation. Therefore, we apply our models for predictions on the $1^{st}$ order differenced dataset.

| Ratio | Before $2^{nd}$ order difference [%] | After $2^{nd}$ order difference [%] |
|---|---|---|
| P/E | 64.6 | 5.75 |
| NPM | 89 | 17.7 |
| ROA | 82.3 | 12.8 |
| ROE | 79 | 17.7 |

Table 2: Fraction of the number of companies for which data is non-stationary against total companies, per ratio, before and after applying a difference of order 2.

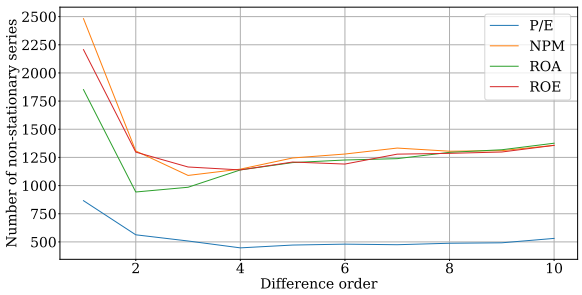


Figure 3: Number of companies that failed to reject the ADF non-stationarity test at a 99% confidence interval, as a function of the difference order, for each of the four ratios.

Figure 4 shows the autocorrelation of the data for all four ratios and for random companies. It appears that most of the time, there is indeed a presence of high autocorrelation. However, we have to keep in mind that the data we have here has been artificially edited in some cases, when it contained missing values. Nevertheless, we restricted ourselves to keep data that only had less than 10% of its values as missing values, and we dropped observations where there were at least 2 consecutive missing values.

That is the extent of our work in terms of data transformation and processing. This transformed dataset will now be used to perform predictions on the given set of ratio we analyse. We perform our data analysis with the remaining companies after our transformations: 9'250 for the P/E ratio, 6'124 for the NPM ratio, 6'460 for the ROA ratio, and 6'040 for the ROE ratio. While this dimension reduction means we test our model on a smaller amount of companies that we initially wished for, which is less representative, it has the advantage of making our following computations less costly.

***Investment strategy***: Our investment strategy is based on the four ratios we mentioned, for which we will make predictions using the aforementioned AR, MA and SARIMA time series models. Note that due to the transformations performed previously, our dataset is actually the $1^{st}$ order difference of our four ratios. Our strategy is as follows. We first pick a subset of companies in which we would like to invest. These companies are chosen as the ones that gave us minimal prediction error during a training phase, i.e. between two distinct dates referred to as *min date* and *split date*.

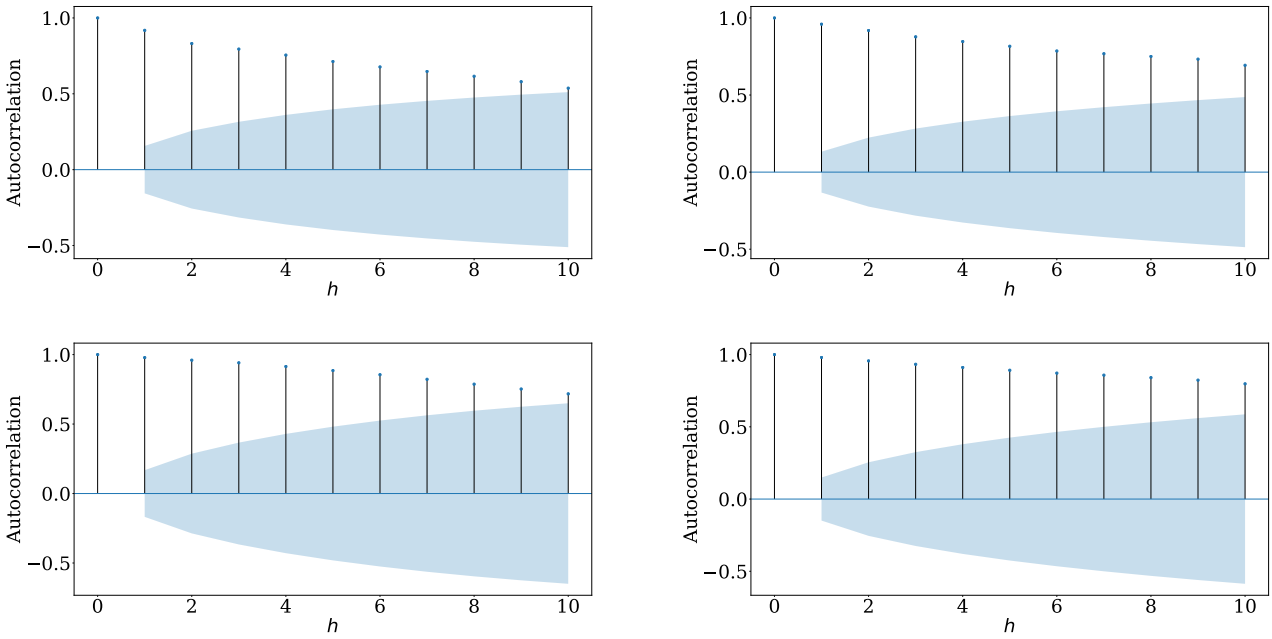


Figure 4: Autocorrelation plot for each financial ratio, for companies picked at random in the dataset: P/E (top-left), NPM (top-right), ROA (bottom-left) and ROE (bottom-right).

Then, we will invest in those companies between *split date* and what we refer to as *max date* in the following way: for each month, we use one model to predict how the ratio of the chosen companies will evolve. If the model predicts that the ratio of a company will rise during next month's announcement, then we make the decision to buy the stock. If, however, the model predicts the ratio will fall, we interpret this as a loss in stock value and therefore we make the decision to sell the stock. When doing so, we essentially mean we would have made the decision to buy or sell right before the announcement, supposing the stock price of a company is indeed influenced right after the results of the company are announced; i.e., the stock price will increase if the results are good and decrease if the results are bad. By good and bad, we imply that the variation of a given ratio between two periods is positive or negative.

The prediction for each ratio is computed by performing time series cross-validation on the data as follows: first, the *train data* will contain every observations except the last one, then every observations except the two last ones, and so on, until it contains every observations except the 10 last ones. Each time we use a different train set, we compute the error: so per company, we compute $n$ errors, where $n$ is the number of training sets: the total error per company is thus simply the average over those $n$ errors. Note that we used the *Mean Absolute Error* (MAE) in order to keep our error in the same order of magnitude as our predictions. The way we performed a time series cross-validation is depicted in Figure 5.

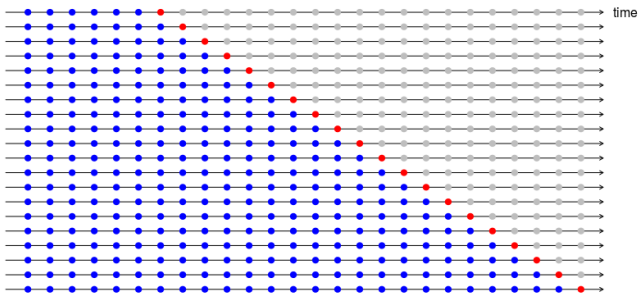


Figure 5: Schematic representation of the time series cross-validation performed on our data to compute the prediction error - "the forecast accuracy". Source: `https://robjhyndman.com/hyndsight/tscv/`.

## III. Results

This section describes the obtained results for our investment strategy using the prediction of the change in the four financial ratios we study in this work. As mentioned previously, this strategy consists of making a positive investment decision (buy stock), should we predict the ratios to increase, or else making a negative investment decision (sell stock). We remind here that our core assumption is that the stock price of the company is positively correlated with the announced value of the ratio at hand [7]. Additionally, we compare the predicted value with the real (historical) ratio value, in order to evaluate the performance of our model.

Our performance metric is described as follows: the goal is to predict the variation of the ratio, as we make the assumption that this variation is positively correlated

with stock price movements. Hence, we only wish to predict the correct sign of the variation of the ratio. And since our dataset is already differenced, we use the following decision rule:

$$K = \begin{cases} +1, & \text{if } sign(\Delta \hat{r}_{i,n}) = sign(\Delta r_{i,n}) \\ -1, & \text{otherwise} \end{cases}$$

where $\Delta \hat{r}_{i,n}, \Delta r_{i,n}$ correspond to the i-th predicted and historical ratio for the n-th company, respectively.

This is defined as the performance metric for our prediction. Hence, a positive $K$ means a good prediction and a negative $K$ means a poor prediction. We choose to make the prediction for a given subset of companies, that we take accordingly to the minimal error they yield for each model. As an example, and to give a preliminary idea of the relative performance of the models we use, Table 3 lists the prediction errors for all three models used in this work for averaged over a subset of companies and for each ratio. As we can see, the SARIMA model seems to yield the best average performance in terms of prediction, particularly for the P/E and ROE ratios. In comparison, the AR model seems to perform more poorly and the MA model seems to behave similarly to, although a bit worse than SARIMA.

| **Ratio** | **AR Error** | **MA Error** | **SARIMA Error** |
|---|---|---|---|
| P/E | 0.103604 | 0.076667 | 0.050920 |
| NPM | 0.176339 | 0.112392 | 0.074674 |
| ROA | 0.319941 | 0.072236 | 0.058367 |
| ROE | 0.134052 | 0.092549 | 0.089948 |

Table 3: Prediction errors evaluated on a subset of companies for all ratios and for each time series model used on the transformed dataset. The AR error was computed over 100 companies and the MA and SARIMA error over 10 companies for computational run time purposes.

As it would be interesting to see what happens when we increase the number of companies, we test the performance of each model as a function of the number of companies used for computing the error. This is depicted in Figures 6, 7 and 8. As we can see, the average prediction error for AR decreases in an exponential fashion when increasing the number of companies on which to compute the average, until it reaches some plateau for each ratio. This shows that we can take a large number of companies for computing the average prediction error with the AR model. For MA however, the error is overall contained in $[0.05, 0.15]$, and converges to around 0.1 for all ratio when we reach 60 companies. After that point, the error disperses again but stays within the same limits. Moreover, Table 4 lists the run time for the computation of the results in Figures 6, 7 and 8. One clearly observe that the AR model is much faster for this computation than both MA and SARIMA models, for more companies (note that for AR we tested with 200 companies, for MA with 100, and for SARIMA with 60 and we already have a significant difference in run time).

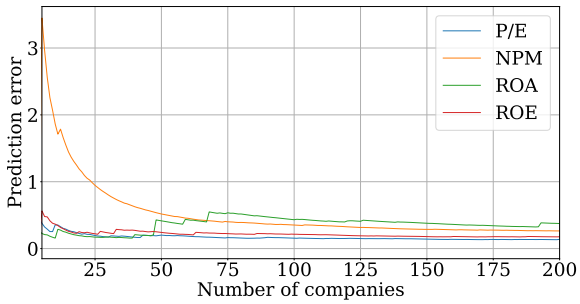


Figure 6: Average prediction error as a function of the number of companies studied for the AR model on the transformed dataset.

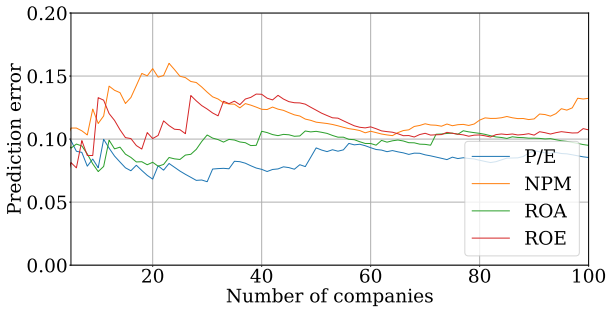


Figure 7: Average prediction error as a function of the number of companies studied for the MA model on the transformed dataset.

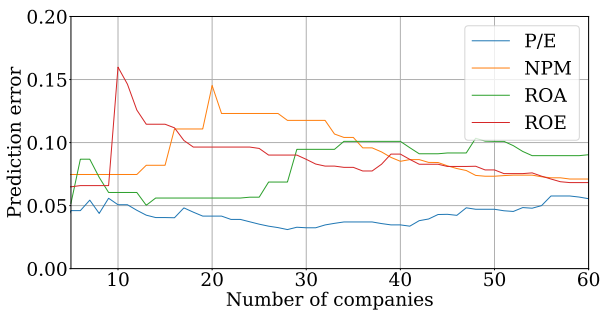


Figure 8: Average prediction error as a function of the number of companies studied for the SARIMA model on the transformed dataset.

| Model | Run time |
|---|---|
| **AR** | 10min 31s |
| **MA** | 1h 57min 19s |
| **SARIMA** | 1h 59min 51s |

Table 4: Computational run time for the average prediction error computation for each model. Computed for AR with 200 companies, for MA with 100 companies, and for SARIMA with 60 companies.

Based on those results, we conclude here that with the error computation staying relatively small even for a small number of companies with MA and SARIMA, we could allow ourselves to take a subset of 60 companies to compute it with these two models, in order to save computation time. For AR however, taking the whole set of companies is acceptable, both in terms of computation cost and prediction error. As a whole, it seems like SARIMA is the model that yields the best predictions, as the error is the smallest for each ratios compared to the two other models.

The second part of the analysis is to actually run the allocation strategy and to evaluate its performance. As mentioned above, the portfolio of companies is constructed by taking the first $N$ companies which give the minimal average error. We perform the prediction in a rolling window fashion, predicting next month's ratio each time. Figures 9 and 10 show the performance metric $K$ as a function of the number of companies which give minimal average prediction error for both AR and MA models. As we can see in the AR case, the performance is relatively constant amongst the four ratios for a low number of minimal-prediction error companies. However, all curves clearly set apart from each other from around 80 companies. The prediction performance is very good for ROE and NPM compared to ROA and, even more so, P/E. The same is true in the case of MA in terms of relative performance. However, Figure 10 shows that the prediction performance hovers around 0 for the best two predictions: ROE and NPM. One can therefore conclude that it would be best to use the AR model in order to predict financial ratio changes and invest accordingly. More specifically, ROE and NPM seem like the most accurate ratios in terms of prediction. Lastly, we note the computational run time for AR and MA predictions are similar in scale, due to the fact that we chose to compute the average prediction error on 60 companies for MA. We also note the graph becoming flat from 60 companies for all ratios. This is normal, as we select minimal-prediction error companies for 60 out 60 of them from this point on. For SARIMA, we listed the results for 100 companies giving minimal-prediction error only in Table 5. As we can see, the performance, as well as run time is quite terrible for the P/E ratio relative to the other three. In this case, NPM carries the best prediction performance. For SARIMA, we performed parallel computation through all 8 of our machine's processing cores, as the computation for this model is much more important than in the case of AR and MA.

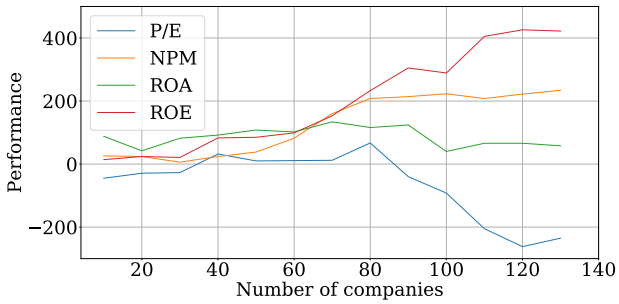


Figure 9: Prediction performance for the AR model as a function of the number of companies which give the minimal average prediction error. The error was computed using all companies. *start date* = 1980.01.01, *split date* = 1990.01.01, *stop date* = 2000.01.01. Computational run time: 11min 18s.

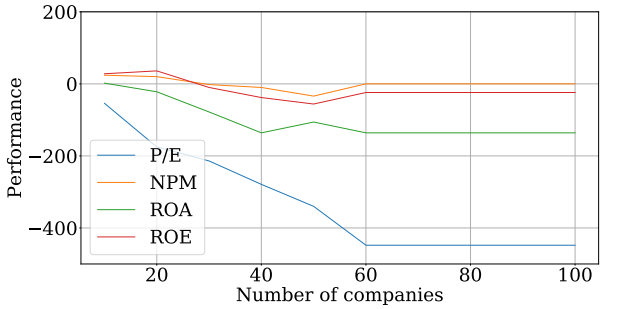


Figure 10: Prediction performance for the MA model as a function of the number of companies which give the minimal average prediction error. The error was computed using 60 companies. *start date* = 1980.01.01, *split date* = 1990.01.01, *stop date* = 2000.01.01. Computational run time: 37min 26s.

| Ratio | Performance $K$ | Computational Run time |
|---|---|---|
| **P/E** | -484 | 1h 32min 33s |
| **NPM** | 160 | 12min 42s |
| **ROA** | 93 | 11min 53s |
| **ROE** | 14 | 22min 4s |

Table 5: Prediction performance for the SARIMA model for 100 companies which give the minimal average prediction error. The error was computed using 60 companies. *start date* = 1980.01.01, *split date* = 1990.01.01, *stop date* = 2000.01.01.

## IV. Discussion

In this letter we summarise the methods, and discuss the results we obtained, as well as future perspectives for our work. We selected the following financial ratios based on their potential to best explain the movement of stock prices amongst other ratios: P/E, NPM, ROA and ROE

[7]. Our prediction models used here are three simple yet effective time series models. The AR, MA and SARIMA models are used in this work and they require the data to be both stationary and autocorrelated. Our dataset contains monthly data for the selected ratios from 1971.01.31 to 2018.12.31 for 20'880 companies (Table 1). In order to meet the models' requirements, the dataset (positively tested for non-stationarity using the ADF test) first had to be made stationary. This was done using the $1^{st}$ order difference. Although an additional difference computation makes the data of more companies stationary (Table 2), this result cannot be interpreted in the context of our allocation strategy. For this reason, we use only the $1^{st}$ order difference. We made additional transformations to the data in order to remove or replace missing values in a reasonable manner and to normalise it. In essence, we cleaned, processed and prepared the data to be ready as input to our models. Our portfolio allocation is summarised as follows: each month, we predict if the value of the ratio of a company that will be announced the next month will drop or rise, by first training the model on previous dates. If our model predicts that the ratio of next month will rise, then we buy the share of the company right before the announcement (i.e. sufficiently soon so that the market does not anticipate that the ratio will indeed rise, but not too soon to prevent from external shocks). Then, if the ratio indeed rises, then we suppose - and this is our main assumption, that the stock price of the company will rise following the announcement, and so we have made a profit. We operate similarly, in the opposite way, if we predict that the value of the ratio will fall. We use a counting value $K$ in order to evaluate the performance of our model; a positive $K$ translates into a good predictions, and a negative $K$ into a bad one. Of course, it would have been more appropriate to assess our results using the prices of the chosen companies to test the real performance of our strategy, but unfortunately we could not import prices of the chosen companies as the companies in our dataset are identified through Gvkeys and this identification is not available to import prices. However, we believe that this work can at least give a good intuition on how fundamental analysis can be used in investment decisions.

In terms of results, we first observed that each model carried a different prediction error to one another for all four studied ratios. Although the MA and SARIMA models show better performance than AR for computing the prediction error, AR is significantly faster to compute. Therefore, there is a tradeoff to be made between computation cost and performance, and AR is still interesting as a solution to perform financial ratio prediction (Tables 3, 4). Moreover, the performance of our models (Figures 9, 10) show that although MA has proven to be yielding a smaller prediction error, the prediction performance is better for AR. More generally, the performance is better for ROE and NPM ratios, in both cases. The performance results for SARIMA were not shown as a function of the companies, as is the case for AR and MA, due to unfortunate problems faced during the fitting of the data with the rolling window method. Due to a lack of time, we were able to resolve this issue only for the case mentioned in Table 5. Lastly, we note that the MA prediction performance result comes from the fact that we computed the prediction error on 60 companies instead of all of them, for previously mentioned computation cost reasons. This perhaps had an effect on the performance, as not all companies are considered for the error computation. A step forward would be to compute the error on all companies for MA. This would require parallel calculations and/or external additional cores, as it would be quite an extensive computation. This was done for SARIMA in the case of 60 companies to compute the prediction error and 100 companies to compute the performance. The computational run times are inferior to what we had before (same kind of computation was run in Figure 4), except for the case of the P/E ratio. We are not sure why this is, but we suspect the fitting of the data to be the cause here, as lots of warnings on maximum likelihood optimisation were generated. While the results for NPM and ROA in this case are somewhat satisfying, it is safe to conclude that, against our initial assumptions, AR seems to be the best predictor for the variation of the ratios in this study. This is pleasing, as it is also the faster model to run. More generally, the P/E ratio was predicted with a poor performance in all cases, relative to the others. We suspect this to be due to the presence of outliers in this data, which may have impacted negatively the normalisation process.

Future perspectives include several factors in this context. We note that this study is a basis for future and more complex models to be used for the prediction of financial ratios. Firstly, we chose the P/E, NPM, ROA and ROE ratios for their higher correlation with stock price evolution. However, it would be interesting to include a lot more ratios and to perform PCA to compare the output in terms of ratio relevance. In this work, we have chosen different numbers of companies for computing the average prediction error for each time series model to implement our allocation strategy for run time purposes. We would like to see how would the models compare when using the same number of companies for the error computation. Also, one further development of this work could be to parametrise each model to its specific optimal number of companies to compute the error. Moreover, the main assumption in this work is that the four financial ratios we use are correlated to stock price evolution. This is why we compare our predictions with the actual historical values of the ratio in order to make an investment decision. It would be interesting however, to validate this prediction by comparing the change from one prediction to the next one in time, with the change in the stock price for the corresponding company. Moreover, the use of this work combined with a definition of financial ratios for private companies, which present less publicly available information by nature, could be achieved in order to predict their performance and therefore it could open the door for

more investors (in the private equity sector for example) to benefit from this strategy. Lastly, we may wish to investigate alternative analyses such as social media data analysis (more advanced: Twitter sentiment analysis [10]) in order to broaden our model to lower the element of surprise from certain ratio announcements and therefore, improve our prediction performance.

## V. Conclusion

In this work we presented a set of methods for implementing a financial ratio prediction-based investment strategy using a monthly dataset of the P/E, NPM, ROA and ROE financial ratios for 20'880 initial companies, as well as for transforming this data for it to be used as input to our chosen time series models. Results seem satisfying for one out of the three methods implemented. However, such methods are subject to further developments, as this work serves as a basis for more complex computational tools. All data processing and analysis was done using Python 3.7, and the parallelised computations were done using the *multiprocessing* python package.

## References


[1] H. Hassani, A. Dionisio, and M. Ghodsi, "The effect of noise reduction in measuring the linear and nonlinear dependency of financial markets," *Nonlinear Analysis: Real World Applications*, vol. 11, pp. 492–502, Feb. 2010.

[2] R. Ball and P. Brown, "An empirical evaluation of accounting income numbers," *Journal of accounting research*, pp. 159–178, 1968.

[3] R. G. Sloan, "Do stock prices fully reflect information in accruals and cash flows about future earnings?," *Accounting review*, pp. 289–315, 1996.

[4] V. L. Bernard, "The nature and amount of information reflected in cash flows and accruals," 1989.

[5] P. M. Dechow, "Accounting earnings and cash flows as measures of firm performance: The role of accounting accruals," *Journal of accounting and economics*, vol. 18, no. 1, pp. 3–42, 1994.

[6] P. Chen and G. Zhang, "How do accounting variables explain stock price movements? theory and evidence," *Journal of accounting and economics*, vol. 43, no. 2-3, pp. 219–244, 2007.

[7] T. Arkan *et al.*, "The importance of financial ratios in predicting stock price trends: A case study in emerging markets," *Finanse, Rynki Finansowe, Ubezpieczenia*, vol. 79, no. 1, pp. 13–26, 2016.

[8] K. Doherty, R. Adams, and N. Davey, "Unsupervised learning with normalised data and non-euclidean norms," *Applied Soft Computing*, vol. 7, no. 1, pp. 203–210, 2007.

[9] R. Mushtaq, "Augmented dickey fuller test," 2011.

[10] J. Bollen, H. Mao, and X.-J. Zeng, "Twitter mood predicts the stock market," *Journal of Computational Science*, vol. 2, pp. 1–8, Mar. 2011. arXiv: 1010.3003.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import random
import statsmodels.api as sm

def data_import(path):
    import pandas as pd
    data = pd.read_csv(path)
    data['index'] = data.index
    data=data.sort_values(by=['gvkey', 'index']) #We sort the data so that it is
                                                    ordered by company.
    data.set_index('public_date', inplace=True)
    data['public_date'] = data.index
    data=data.drop(columns=['index'])
    return data

def sort_data(data): #sort data in a list such that each element of the list corresponds
                        to the ratios time series of a company
    np_data = data.to_numpy()
    unique, counts = np.unique(np_data[:,0], return_counts=True)
    gvkeys = unique.reshape((len(unique),1)) #contains all the company gvkeys
    sorted_data=[]
    for i in range(0, len(gvkeys)):
        sorted_data.append(data[data['gvkey']==int(gvkeys[i])])
    return sorted_data


def data_3m(data): #convert the monthly data to a trimestrial data
    data3m = data.copy()
    for i in range(0, len(data3m)):
        data3m[i]=data3m[i].drop_duplicates(subset='qdate') # Get trimestrial data by
                                        dropping ratio duplicates wrt qdate column
    return data3m

def select_by_date(data, date_column, min_date, max_date): #select companies by years of
                                                                existence
    selected_companies=[]
    for i in range(len(data)):
        if str(data[i][date_column].iloc[0]) <= min_date and str(data[i][date_column].iloc[-1]) >=
    max_date:
            selected_companies.append(data[i])
    return selected_companies

def gvkeys(data): #This function returns a list containing each gvkey of the dataset
    gvkeys_=[]
    for i in range(len(data)):
        gvkeys_.append(data[i]['gvkey'].iloc[0])
    return gvkeys_

def select_by_gvkey(database, gvkey, ratio=None): #return ratios of a company based on the gvkey of
    the company. If ratio argument not specified, returns all ratios of the company.

    for i in range(len(database)):
        if database[i]['gvkey'].iloc[0]==gvkey:
            if not ratio:
                selected_data=database[i]
            else:
                selected_data=database[i][ratio]
    return selected_data

def sub_sample(sample, min_date, max_date):
    subsample = []
    for k in range(len(sample)):
        for i in range(len(sample[k])):
            if sample[k]['public_date'].iloc[i] == min_date:
                idx_min=i
        for j in range(idx_min, len(sample[k])):
            if sample[k]['public_date'].iloc[j] == max_date:
                idx_max=j
```

```
        subsample.append(sample[k].iloc[idx_min:idx_max+1])
    return subsample


def plot_ratios(data, date_column, start_date, end_date, n_companies, ratio): #n_companies is the number of companies for which we want to plot ratios, ratio specifies the ratio to plot.
    sample = select_by_date(data, date_column, min_date=start_date, max_date=end_date)
    subsample = sub_sample(sample, min_date=start_date, max_date=end_date)
    plt.figure()
    if n_companies <= len(subsample):
        for companies in range(0, n_companies):
            if len(subsample[companies])>1:
                subsample[companies][ratio].plot(label='Company {}'.format(subsample[companies]['gvkey'][0]))
            else:
                print('The dataset of company with index ' + str(companies), 'is too small to be plotted')
    else:
        print('The number of companies ratios to plot is larger than the number of companies that possess these attributes.               max(n_companies)=' + str(len(subsample)))
    plt.xlabel('Public date')
    if ratio=='pe_inc':
        plt.ylabel('P/E')
    elif ratio=='npm':
        plt.ylabel('NPM')
    elif ratio=='roa':
        plt.ylabel('ROA')
    elif ratio=='roe':
        plt.ylabel('ROE')
    plt.legend(bbox_to_anchor=(1,.6))
    plt.grid()


def autocorrelation_plot(data, gvkey, ratio, n_lags): #return the autocorrelation figure of a ratio time series of a specific company (defined by its gvkey) for a desired number of lags n_lags.
    import statsmodels.api as sm
    data_to_plot = select_by_gvkey(data, gvkey, ratio)
    sm.graphics.tsa.plot_acf(data_to_plot, lags=n_lags)
    plt.show()
    print('gvkey: ' + str(gvkey))


def count_nan(data_company, ratio, gvkey=None): #count nan values of a company's time series ratio, data_company is a time series (an output of sort_data(raw_data))
    if not gvkey:
        count = len(data_company[ratio]) - data_company[ratio].count()
    else:
        data_to_count = select_by_gvkey(data_company, ratio, gvkey)
        count = len(data_to_count) - data_to_count.count()
    return count


def nan_per_company(data, ratio): #This function prints the number of nans of each company in the dataset for a desired ratio.
    from colorama import Fore
    count=np.zeros((len(data), 1))
    print('The companies with NaN values are:')
    threshold = 0.1 #Companies that present a ratio of nan values above this ratio are printed in red.
    for i in range(0,len(data)):
        count[i] = count_nan(data[i], ratio)
        if count[i]==0:
            print(Fore.GREEN + 'index:' + str(i), ', gvkey:' + str(data[i]['gvkey'][0]), ', number of NaNs: 0')
        if count[i]>0:
            if count[i]/len(data[i])<threshold:
                print(Fore.BLACK + 'index:' + str(i), ', gvkey:' + str(data[i]['gvkey'][0]), ', number of NaNs:' +                                    str(int(count[i])), ', ratio of NaNs:' + str(count[i]/len(data[i])))
            if count[i]/len(data[i])>threshold:
                print(Fore.RED + 'index:' + str(i), ', gvkey:' + str(data[i]['gvkey'][0]), ', number of NaNs:' +                                    str(int(count[i])), ', ratio of NaNs:' + str(
```

```
    count[i]/len(data[i])))

def where_nan(data, ratio, gvkey=None): #returns the index of the NaN values for a company's time series ratio
    if not gvkey:
        index = np.argwhere(np.isnan(data[ratio].to_numpy()))
    else:
        data_to_search = select_by_gvkey(data, gvkey, ratio)
        index = np.argwhere(np.isnan(data_to_search.to_numpy()))

    return index

def consecutive_nan(data, ratio): #returns the maximum number of consecutive NaNs in a dataset.

    idx = where_nan(data, ratio) #index of the NaN value

    if count_nan(data, ratio) == 0 or count_nan(data, ratio) == 1: #no nans or 1 nan
        cons = 0

    else: #more than one nan
        cons_ = []
        count = 1
        for i in range(len(idx)-1):
            if int(idx[i]+1) == int(idx[i+1]):
                count+=1
            else:
                cons_.append(count)
                count = 1
        cons_.append(count)
        cons = max(cons_)
        #print('List of consecutive nans ' + str(cons_), ', with indices' + str(idx) )
    return cons

def nan_correction(data, ratio, max_nan_threshold): #(example: data = sort_data(original_data), ratio = 'pe_inc') max_nan_threshold is the maximum ratio of nan allowed in the dataset of a company

    import pandas as pd
    import numpy as np
    pd.options.mode.chained_assignment = None

    data[:] = [x for x in data if count_nan(x, ratio)/len(x) < max_nan_threshold] #keep only the datasets of companies that have nan ratio below a certain threshold



    for i in range(len(data)):
        idx = where_nan(data[i], ratio)

        if count_nan(data[i], ratio) == 1: #The data only contains one nan value
            if int(idx) == 0: #If the only nan value is located on the first observation of the data
                data[i][ratio].iloc[int(idx)] = data[i][ratio].iloc[int(idx)+1] + np.random.normal(0,1) #replace the nan value by its next value if the nan is located on the first observation + some gaussian noise
            if int(idx) > 0 and int(idx) < len(data[i][ratio]) - 1:
                data[i][ratio].iloc[int(idx)] = (data[i][ratio].iloc[int(idx)-1]+data[i][ratio].iloc[int(idx)+1])/2 + np.random.normal(0,1) #if the nan value is between two non-nan values, replace the nan value by the mean of these two values + some gaussian noise
            if int(idx) == len(data[i]) - 1:
                data[i][ratio].iloc[int(idx)] = data[i][ratio].iloc[int(idx)-1] + np.random.normal(0,1) #if the nan value is the final value of the data, replace it by the previous value + some gaussian noise


        if count_nan(data[i], ratio) > 1: #The data contains more than one nan value
            threshold = 2
            if consecutive_nan(data[i], ratio) >= threshold:
                data[i][ratio].dropna(inplace=True) #if there are at least 2 consecutive NaNs, drop the observations
            else: #if there are no consecutive nans in the dataset, the function treats each nan value similarly as above.
                for indices in idx:
```

```
                    if int(indices) == 0:
                        data[i][ratio].iloc[int(indices)] = data[i][ratio].iloc[int(indices)+1] + np.random.normal(0,1)
                    if int(indices) > 0 and int(indices) < len(data[i])-1:
                        data[i][ratio].iloc[int(indices)] = ( data[i][ratio].iloc[int(indices)-1] + data[i][ratio].iloc[int(indices)+1] ) / 2 + np.random.normal(0,1)
                    if int(indices) == len(data[i]) - 1:
                        data[i][ratio].iloc[int(indices)] = data[i][ratio].iloc[int(indices)-1] + np.random.normal(0,1)


    return data

def data_clean(clean_dataset, ratio):
    clean_ratios=[]
    gvkeys = []
    concat=[]
    for i in range(len(clean_dataset)):
        clean_ratio = clean_dataset[i][ratio]
        gvkey = clean_dataset[i]['gvkey']
        clean_ratios.append(clean_ratio)
        gvkeys.append(gvkey)
        concat_data = pd.concat([clean_ratio, gvkey], axis=1, join='inner')
        concat.append(concat_data)
    return concat

def normalize(data, ratio):
    import pandas as pd
    from sklearn import preprocessing

    x = data[ratio].values
    x = x.reshape(-1,1)
    min_max_scaler = preprocessing.MinMaxScaler()
    x_scaled = min_max_scaler.fit_transform(x)
    x_scaled = x_scaled.ravel()
    data[ratio].iloc[:] = x_scaled

    return data

def normalize_dataset(data, ratio):
    norm_dataset = []
    norm_ratio = []
    gvkeys = []
    for i in range(len(data)):
        norm_dataset.append(normalize(data[i], ratio))

    #for i in range(len(norm_dataset)):
        #norm_ratio.append(norm_dataset[i].iloc[:,0])
        #gvkeys.append(norm_dataset[i].iloc[:,1])

    return norm_dataset

def check_stationarity(data, ratio, text=True):
    from statsmodels.tsa.stattools import adfuller
    count = 0
    for i in range(len(data)):
        critical_values = []
        test = adfuller(data[i][ratio])
        for key, value in test[4].items():
            critical_values.append(value)
        if critical_values[0] < test[0]: #critical_values[0] is the critical value at a 99% confidence interval: if the value returned by the test (test[0]) is larger than the critical value, then the null hypothesis cannot be rejected at a 99% confidence interval.
            if text:
                print('The data with index ' + str(i), 'has failed to reject the non-stationarity test at a 99% confidence interval')
            count+=1
        elif critical_values[1] < test[0]: #similarly as above, but for a 95% confidence interval
            if text:
                print('The data with index ' + str(i), 'has failed to reject the non-stationarity test at a 95% confidence interval')
            count+=1
```

```
        elif critical_values[2] < test[0]: #similarly as above, but for a 90% confidence interval
            if text:
                print('The data with index ' + str(i), 'has failed to reject the non-stationarity test at a 90% confidence interval')
            count+=1
    print('{} companies do not have stationary data'.format(count))
    return count



def stationarity_test(data, ratio): #returns 1 if the time serie fails the non-stationarity test (at 99%, 95% or 90%), and 0 otherwise.
    from statsmodels.tsa.stattools import adfuller
    test = adfuller(data[ratio])
    critical_values=[]
    for key, value in test[4].items():
        critical_values.append(value)

    if critical_values[0] < test[0] or critical_values[1] < test[0] or critical_values[2] < test[0]:
        adf = 1 #Non stationary time series
    else:
        adf = 0
    return adf

def first_difference(data, ratio):
    for i in range(len(data)):
        data[i][ratio] = data[i][ratio].diff()
        data[i] = data[i].dropna()
    return data




###### PREDICTION FUNCTIONS ######

def autoreg(ratio): #one step ahead AR prediction (only predicts the t+1 ratio)
    from statsmodels.tsa.ar_model import AR
    if type(ratio) == list:
        model = AR(ratio)
        model_fit = model.fit(maxlag=10)
        ratio_pred = model_fit.predict(len(ratio), len(ratio))
    else:
        model = AR(ratio.tolist())
        model_fit = model.fit(maxlag=10)
        ratio_pred = model_fit.predict(len(ratio), len(ratio))
    return ratio_pred

def movingavg(ratio):
    from statsmodels.tsa.arima_model import ARMA
    model = ARMA(ratio.tolist(), order=(0,1))
    model_fit = model.fit(disp=False)
    ratio_pred = model_fit.predict(len(ratio), len(ratio))
    return ratio_pred

def ts_cross_validation(data, algorithm): #performs a predictive algorithm on a time series dataset while doing cross validation.
    predictions = []
    errors = []
    test = []
    for i in range(1, 10):
        train_data = data[0:-1 -i] # first the train data will contain every observations except the last one, then every observations except the two last ones, etc., until it contains every observations except the 10 last ones. This is a way to do cross validation with time series (to see how our algorithm performs while estimating the parameters with a different train set each time. If the the error largely changes depending on the train set, then this might indicate our algorithm is not robust.)
        ratio_pred = algorithm(train_data) #the algorithm uses data from t=0 to t=T-i (where T is the time of the last observation, i.e. -1 in python language) to predict the ratio at time (T-i)+1
        predictions.append(ratio_pred)
        test_data = data[-1 -i +1] #actual observation at time (T-i)+1
        test.append(test_data)
```

```
        error = np.abs(ratio_pred - test_data  ) #absolute error between the prediction and the actual observation

        errors.append(error) #Returns the mean absolute error

    predictions.reverse()
    test.reverse()


    return np.mean(errors), predictions, test #returns the mean absolute error, the predictions and the actual observations (both are displayed in lists)



def average_error(data, sample, algorithm): #returns the mean absolute error of a sample of companies for a prediction algorithm for a given ratio (given by the variable data)

    errors=[]
    for i in range(sample):
        try:
            error = ts_cross_validation(data[i], algorithm)[0]
            errors.append(error)
        except:
            pass
    return np.mean(errors)

def sarima(ratio):
    from statsmodels.tsa.statespace.sarimax import SARIMAX
    model = SARIMAX(ratio.tolist(), order=(1, 1, 1), seasonal_order=(1, 1, 1, 1))
    model_fit = model.fit(disp=False)
    ratio_pred = model_fit.predict(len(ratio), len(ratio))
    return ratio_pred

def binary(diff_data): #transforms first order differentiated ratios into binary data
    x = diff_data
    x[x<0] = int(0) #the observation becomes 0 if it is initially negative
    x[x>0] = int(1) #1 otherwise
    return x

def binary_dataset(data):
    binary_data = []
    for i in range(len(data)):
        binary_data.append(binary(data[i]))
    return binary_data


def classification(data, algorithm):
    predictions = []
    test = []
    threshold = 0.5
    for i in range(1, 10):
        train_data = data[0:-1 -i]
        ratio_pred = algorithm(train_data)
        predictions.append(ratio_pred)
        test_data = data[-1 -i +1] #actual observation at time (T-i)+1
        test.append(test_data)



    predictions.reverse()
    predictions=[0 if i <=threshold else i for i in predictions]
    predictions=[1 if i>threshold else i for i in predictions]
    test.reverse()
    test=[int(x) for x in test]

    count=0
    for i in range(len(predictions)):
        if predictions[i] == test[i]:
            count +=1

    accuracy = count/len(predictions)
    return accuracy, predictions, test
```

```
384
385 def average_accuracy(ratio, sample, algorithm): #returns the mean absolute error of a sample of
    companies for a prediction algorithm for a given ratio (given by the variable data)
386
387
388     accuracies=[]
389     for i in range(sample):
390         try:
391             accuracy = classification(data[i], algorithm)[0]
392             accuracies.append(accuracy)
393         except:
394             pass
395     return np.mean(accuracies)
396
397 def best_companies(data, ratio, n_min_errors, algorithm): #return the gvkeys of the n_min_errors
    companies that have the minimum error for a given algorithm
398     errors = []
399     gvkeys_error=[]
400     for i in range(len(data)):
401         try:
402
403             error = ts_cross_validation(data[i][ratio], algorithm)[0]
404             errors.append(error)
405             gvkey = data[i]['gvkey'][0]
406             gvkeys_error.append(gvkey)
407         except:
408             pass
409
410     idx = []
411     err_val = []
412     for j in range(n_min_errors):
413         m = min(errors)
414         err_val.append(m)
415         for i, x in enumerate(errors):
416             if x == m:
417                 idx.append(i)
418                 del errors[i]
419
420     best_comp = []
421     for indices in idx:
422         best_comp.append(gvkeys_error[indices])
423
424     return best_comp, err_val
425
426 def select_by_dates2(data, min_date, max_date): #select companies by years of existence
427     selected_companies=[]
428     for i in range(len(data)):
429         if str(data[i].index[0]) <= min_date and str(data[i].index[-1]) >= max_date:
430             selected_companies.append(data[i])
431     return selected_companies
432
433 def split_data(start_date, split_date, end_date, data):
434     train_data = data.copy()
435     test_data = data.copy()
436     for i in range(len(data)):
437         train_data[i] = train_data[i][train_data[i].index>=start_date]
438         train_data[i] = train_data[i][train_data[i].index<=split_date]
439         test_data[i] = test_data[i][test_data[i].index> split_date]
440         test_data[i] = test_data[i][test_data[i].index<= end_date]
441
442     return train_data, test_data
443
444 def allocation_strategy(ratio_data, min_date, split_date, max_date, ratio,
445                         n_companies, algorithm):
446
447     selected_companies = select_by_dates2(ratio_data, min_date, max_date)
448     train, test = split_data(min_date, split_date, max_date, selected_companies)
449     companies_to_invest = best_companies(train, ratio, n_companies, algorithm)
450
451     train_data = []
452     test_data = []
453
```

```
    for keys in companies_to_invest[0]:
        train_sets = select_by_gvkey(ratio_data, gvkey=keys, ratio=ratio)
        train_sets = train_sets[train_sets.index>min_date]
        train_sets = train_sets[train_sets.index<=split_date]
        train_data.append(train_sets)
        test_sets = select_by_gvkey(ratio_data, gvkey=keys, ratio=ratio)
        test_sets = test_sets[test_sets.index>split_date]
        test_sets = test_sets[test_sets.index<=max_date]
        test_data.append(test_sets)

    performance=[]
    predictions = []
    rolling_train = []

    for j in range(n_companies):

        for i in range(0, len(test_data[j])):
            rolling_test = test_data[j][0:i]
            rolling_train = pd.concat((train_data[j], rolling_test))
            pred = algorithm(rolling_train)
            obs = test_data[j][i]
            if np.sign(pred) == np.sign(obs):
                performance.append(1)
            else:
                performance.append(-1)
            predictions.append(pred)
        print(np.sum(performance))
        results.append(np.sum(performance))
        performance=[]
    total_profit = np.sum(results)


    return total_profit


def differences(data, ratio, length=2):
    y = []
    x = []
    if ratio=='pe_inc':
        name = 'P/E'
    elif ratio=='npm':
        name = 'NPM'
    elif ratio=='roa':
        name = 'ROA'
    elif ratio=='roe':
        name = 'ROE'

    for i in range(length):
        x.append(i+1)
        data = first_difference(data, ratio)
        y.append(check_stationarity(data, ratio, text=False))

    return x,y,name

############################################################
### THE FOLLOWING ARE MODIFIED VERSIONS OF SOME ########
### FUNCTIONS FOUND ABOVE, ADAPTED FOR SARIMA ##########
### AND FOR PARALLEL COMPUTING - USE IF NEEDED #########
############################################################

def best_companies_COPY(data, ratio, n_min_errors, algorithm, sample=0): #return the gvkeys of the
    n_min_errors companies that have the minimum error for a given algorithm
    errors = []
    gvkeys_error=[]
    if sample==0:
        count = n_min_errors
    else:
        count = sample
    if n_min_errors > count:
        n_min_errors = count
        print('Number of companies higher than sample -> this number is set to the sample size')
    for i in range(count):
```

```
        try:
            gvkey = data[i]['gvkey'][0]
            gvkeys_error.append(gvkey)
            error = ts_cross_validation(data[i][ratio], algorithm)[0]
            errors.append(error)
        except:
            pass

    idx = []
    err_val = []
    try:
        for j in range(len(errors)):
            m = min(errors)
            err_val.append(m)
            for i, x in enumerate(errors):
                if x == m:
                    idx.append(i)
                    del errors[i]
        best_comp = []
        for indices in idx:
            best_comp.append(gvkeys_error[indices])
    except:
        best_comp = []

    return best_comp, err_val


def allocation_strategy_COPY(k,ratio_data, min_date, split_date, max_date, ratio,
                  n_companies, algorithm,sample=0):

print('n = {}'.format(int(n_companies)))

selected_companies = select_by_dates2(ratio_data, min_date, max_date)
train, test = split_data(min_date, split_date, max_date, selected_companies)
companies_to_invest = best_companies_COPY(train, ratio, n_companies, algorithm, sample)

if companies_to_invest==[]:
    companies_to_invest = [ratio_data[g]['gvkey'][0] for g in np.arange(10)]

    train_data = []
    test_data = []
    results = []

    for keys in companies_to_invest[0]:
        train_sets = select_by_gvkey(ratio_data, gvkey=keys, ratio=ratio)
        train_sets = train_sets[train_sets.index>min_date]
        train_sets = train_sets[train_sets.index<=split_date]
        train_data.append(train_sets)
        test_sets = select_by_gvkey(ratio_data, gvkey=keys, ratio=ratio)
        test_sets = test_sets[test_sets.index>split_date]
        test_sets = test_sets[test_sets.index<=max_date]
        test_data.append(test_sets)

    performance=[]
    predictions = []
    rolling_train = []

    for j in range(len(companies_to_invest[0])):
        #pdb.set_trace()
        for i in range(len(test_data[j])):
            rolling_test = test_data[j][0:i]
            rolling_train = pd.concat((train_data[j], rolling_test))
            try:
                pred = algorithm(rolling_train)
                obs = test_data[j][i]
                if np.sign(pred) == np.sign(obs):
                    performance.append(1)
                else:
                    performance.append(-1)
                predictions.append(pred)
            except:
                pass
```

```
597         results.append(np.sum(performance))
598         performance=[]
599     total_profit = np.sum(results)
600 
601 
602     return total_profit,k
```