uc3m | Universidad Carlos III de Madrid

University Degree in Finance and Accounting
2021-2022

Bachelor Thesis

"Portfolio managers' skills or randomness during bull and bear markets"

Iván Carrillo García

Pedro José Agudo González

Getafe, June 2022

ABSTRACT

This thesis aims to identify the ability of professional portfolio managers to produce extra risk-adjusted returns over the market, and to distinguish if it is produced by ability or randomness.

To do so, historical data is compiled from a database of United States mutual funds. To establish a comparison with random portfolios, equity data is also gathered and processed to form portfolios by randomness, simulating thousands of investment decisions. In order to evaluate the data, several statistical tests are applied, including machine learning techniques, and a new measurement metric is proposed.

Results show that portfolio managers are not worth the cost, as they are not able to persistently beat the market, especially during uptrends. Furthermore, portfolios formed by randomness show slightly better results, putting the portfolio manager figure in jeopardy.

Key words: *financial markets, alpha, randomness, active management, passive funds, machine learning*

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

1. INTRODUCTION.

1.1. Motivation

For many years, investors have been trying to beat the market using a wide range of tools like fundamental analysis, technical analysis and more recently, quantitative techniques including the likes of machine learning. All these methods, heavily time-consuming – and therefore money-consuming – seem to have worked the best for the sell-side, including fund managers, rather than for the investors that hoped their savings would be better managed by a professional. With this thesis, my main motivation is to distinguish between the supposed skills that these fund managers have – and are handsomely paid for – and the randomness of financial returns, to assess if they are worth the cost or not.

1.2. Objectives

The main objective for this thesis is the quantification of the skills of fund managers through a series of statistical tests. Another objective is the analysis of their returns against the ones of completely random traders to be able to compare their performance and explain the differences. Furthermore, it is of interest to have a look at the biases that may explain the preference of investors to hand their savings to fund managers rather than buying the whole market through an equity index fund.

1.3 Methodology

To achieve the objectives, historical equity data from several sources is used. This data is processed with a self-developed software written in Python. The software allows for a complete statistical analysis, including innovative machine learning techniques, and the visualization of the results.

2. STATE OF THE ART.

2.1. Current situation of active and passive funds

First, it is needed to acknowledge the current situation regarding active and passive funds, focusing on the United States. According to data compiled by Bloomberg, domestic equity funds stood at 11,6 trillion dollars at the end of 2020, **54% of it belonging to passive funds**. The trend favouring passive funds started more than 20 years ago, and crossed the 50% milestone in August 2018.

On the other hand, global equity funds – which this thesis will not cover extensively – remained at four trillion dollars, with 41.5% composed of passive funds and a trend that stopped during 2019, coinciding with the emergence of ARK's active funds. Globally, slightly more than half the U.S. equity assets are hold by passive funds. (Bloomberg Intelligence, 2021)

(Bloomberg Intelligence, 2021)

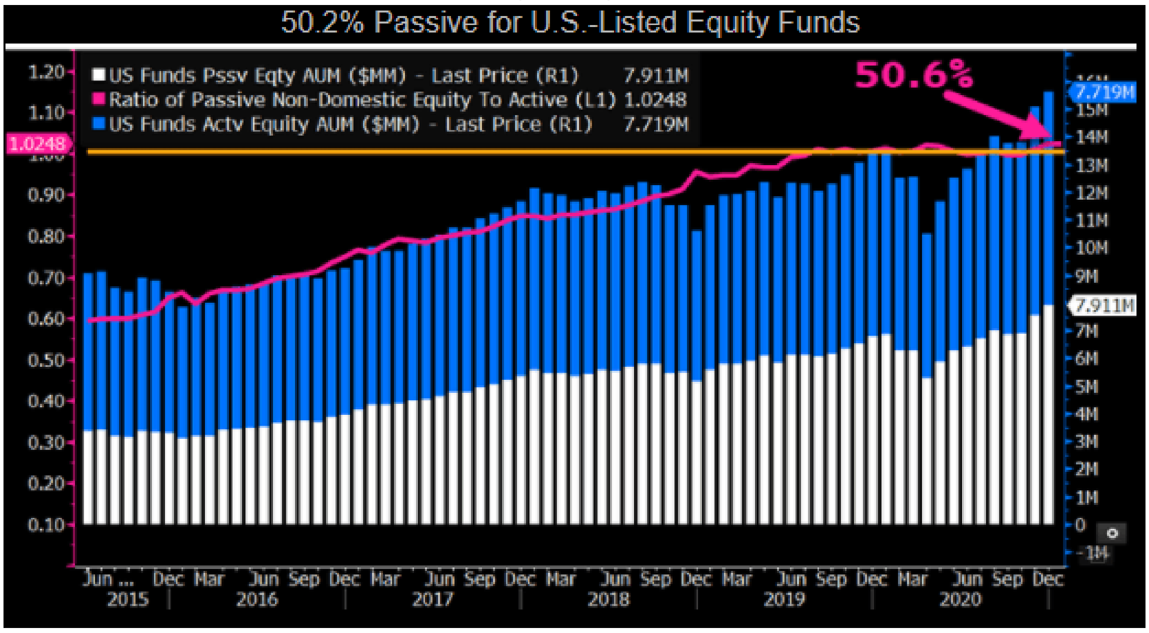


Figure 1 - Evolution of active and passive Assets Under Management in the United States - Source: Bloomberg Intelligence

Regarding returns, it is widely known that most active funds struggle to outperform market indices on a yearly basis, and very few are successful consistently over periods of more than five or ten years. For instance, according to research by S&P Global, in 2021 less than 20% of United States outperformed the S&P Composite 1500, a proxy used for the overall United States stock market. Looking at previous years and rolling returns, the results are not bright, with the majority of funds consistently underperforming over three-year periods, irrespectively of their investment style. For example**, 90.30% of all domestic funds underperformed** during the 20 years ended in December 2021 in absolute terms, and 95.39% when it comes to risk-adjusted returns.

Interestingly enough, the longer the time-period, the higher the percentage of under-performers, in line with the idea that it is difficult to beat the market consistently. In fact, only 62% of domestic funds survived the last 10 years, defined by a secular bull market, and just 30% the last 20 years, which include the Great Financial Crisis (2007-2009). Also, the majority of funds (59%) changed their investment style through the last 20 years, probably in a bid to survive. (S&P Global, 2021)

2.2. Mean-variance framework.

In 1952, Harry Markowitz introduced the mean-variance framework in his "Portfolio Selection" thesis, worth a Nobel Memorial Prize. This framework weighs both the expected return, a forward-looking expectation, and the variance, a backward-looking realization, and considers that investors, as risk-averse individuals, will always select the portfolio with the lowest variance given the same expected return. This framework formalizes the idea that for an extra level of risk (variance) to be taken, an additional expected return would be requested. (Markowitz, 1952)

Markowitz also stated the first innings of portfolio optimization. Using as inputs the weights and variance of the portfolios' securities, and the correlations between them, one can determine the variance of the overall portfolio. This variance will always be lower than the variance of individual securities, as long as the backward-looking correlations are lower than one.

To optimize the investor decision, Markowitz stated the concept of the "**efficient frontier**." Given a basket of individual securities, the efficient frontier collects the different portfolios that can be created out of the available securities, and that satisfy the condition that no other portfolio exists that with its level of risk, achieves a higher rate of return. This concept, combined with the existence of a risk-free asset, creates the "tangent portfolio," which is the portfolio with the highest risk/reward ratio and the capital allocation line, which is defined by the following formula:

$$E(R_i) = R_f + \sigma_i \frac{E(Rp) - R_f}{\sigma_p} \tag{2.1}$$

Of course, this framework deals with several nuances. In first place, returns are a critical input, but they are expected returns, and the probability that they end up being exactly as predicted is really thin, especially in the short term. On the other hand, the covariance matrix that is used for correlations and variances is based on historical data that may not be significant going into the future, especially if the data is collected on a specific regime, i.e., a bull market.

It is no surprise, then, that the own Markowitz preferred to use a simple heuristic when it came to managing his own portfolio. According to an interview he conceded, to minimize his potential future regret, **Markowitz decided to be an equal-weight investor**, splitting his money evenly between equities and bonds. No skin in the game for Harry. (Bower, 2011)

Later, in 1964, William Sharpe published one of the most influential papers in finance. Reflecting on previous research by Markowitz, Sharpe proposed the foundations of the Capital Asset Pricing Model (CAPM), core for the developments in portfolio construction for the rest of the century. This model relates the performance of a stock with its sensitivity to the market risk premia, and the risk-free rate. (Sharpe, 1964)

$$E(R_i) = R_f + \beta_i\big(E(Rm) - R_f\big) \tag{2.2}$$

According to this equation, the return of a stock, R_i, should be fully explained by two exogenous variables: Rm, the return of the market, and R_f, the risk-free rate. This way, the only risk that an investor is being rewarded for is the market risk. In the real world, it is known that this is not entirely true, and an error term should be added.

$$E(R_i) = R_f + \beta_i\big(E(Rm) - R_f\big) + \alpha \qquad (2.3)$$

This term, alpha, reflects the performance of the stock that cannot be explained by the equilibrium model, and that may be attributed to manager's skills, or randomness in the data.

2.3. Efficient markets.

Eugene Fama published in 1970 a key paper on market efficiency, which he defined as "a market in which prices always fully reflect available information" (Fama, Efficient Capital Markets: A Review of Theory and Empirical Work, 1970). This led to the formulation of the efficient market hypothesis (EMH). According to this hypothesis, **no investor should generate extra risk-adjusted returns**, as every stock price should have incorporated its associated risk. Furthermore, only added information should drive the price changes, and by definition that added information should be unpredictable and hence random. However, the efficiency of a market cannot be proved, as it is a state that can only be approached.

On his paper, Fama formalized an empirical approach to evaluate the efficient market hypothesis. As he acknowledged, to assess market efficiency, two different tests were needed: a test of the EMH, and a test of the market equilibrium model. As seen in the CAPM, the only risk that should reward investors is the market risk, so that to generate an extra return an investor should stand a proportionate increase in risk. If some kind of stocks violate this model, the market efficiency should be deemed as false. But, as it was proved later, it was not the efficiency of the market which was false, but **the equilibrium model which was flawed**.

In 1981, Rolf Banz demonstrated that the **small firms produced higher risk adjusted returns that larger firms** on average, proving wrong the CAPM. However, he could not assess if the outperformance were due to the size, or a combination of other factors correlated with size. (Banz, 1981) Few years later, Barr Rosenberg, Kenneth Reid and Ronald Lanstein published a paper where they found that the **companies with a high ratio of book value to market price also produced higher risk adjusted return** relative to companies with a low ratio. (Rosenberg, Reid, & Lanstein, 1985)

To address these developments, Eugene Fama alongside Kenneth French published in 1992 a paper where they showed their progress in market efficiency. They found a number of stock variables that consistently violated the EMH under the CAPM equilibrium model. These variables included size, leverage, earnings/price, and book-to market equity. Used alone, all of them had explanatory power, but in combinations size and book-to-market absorbed the roles of leverage and earnings/price. This led to the creation of the Fama-French three-factor model (FF3FM), an extension of the CAPM. (Fama & French, 1993)

$$r = R_f + \beta(R_m - R_f) + b_s * SMB + b_v * HML + \alpha \tag{2.4}$$

This model identifies size (SMB, small (capitalization) minus big) and book-to-market (HML, high (book-to-market ratio) minus low) as independent risk factors that should reward investors. In order to these factors to be accepted as independent, they should have a rational explanation of why should generate extra returns. Fama-French justify size as a factor on the grounds of the longer periods that small firms can suffer earnings depressions, and book-to-market, commonly known as value, on its relation to earnings that suggests that profitability is the source of a common risk factor. With these two additional factors, any alpha proved by the CAPM should be eliminated and incorporated into either size and/or value factors. This way, the **EMH can be restored through a change in the equilibrium model**, that properly accounts for the risks an investor can be rewarded for in the market.

Later, in 2014, Fama and French expanded their model into a five-factor model (FF5FM), with the inclusion of profitability (RMW, robust minus weak) and investment (CMA, conservative minus aggressive) factors. (Fama & French, 2015)

$$r = R_f + \beta(R_m - R_f) + b_s * SMB + b_v * HML + b_p * RMW + b_i * CMA + \alpha \tag{2.5}$$

These factors are related to the higher returns associated with a robust profitability and firms with low (conservative) investment levels, and all together should be able to explain the difference in returns over two diversified portfolios.

2.4. Behavioural Finance

Another relevant topic, which could be the object of a whole different thesis, is what makes people take poor financial decisions, like engaging in active stock-picking either through a fund or individually. Here, lots of human biases influence the decisions and usually lead to bad outcomes. For instance, confirmation bias may lead to an individual to invest in the best fund of the moment, when as seen before it is likely that that fund will produce inferior performance in the future. Also, emotional biases like herding lead to irrational decisions, like investing in an asset just because of its popularity.

Both biases exposed, although there are a lot more, can be displayed with the recent case of the ARK Innovation ETF managed by Cathie Wood. This actively managed investment vehicle, **beat by a factor of two the S&P 500 benchmark index** for the two-year period coming into the COVID crisis in March 2020, gaining popularity among investors. From the trough of the March 2020 bear market, this tech-focused ETF achieved a whopping **324% return** while the S&P 500 "only" returned 60%.[1]

During this period, the popularity of Cathie Wood skyrocketed and even the investment style of legendary investors like Warren Buffett was questioned (Bowman, 2020). Not surprisingly, returns since then have been poor and let many investors holding shares

[1] Own calculation with data from Yahoo Finance.

bought at the top of the bubble. To be precise, from the top in February 2021 to the provisional trough in June 2022, **ARK Innovation ETF has plunged 76%, while the S&P 500 decreased 4% or Warren Buffett's Berkshire Hathaway returned a positive 16%**.[2]

This field has been subject of substantive studies in the last decades as well as its implementation on financial markets. For instance, George Soros used the concept of reflexivity to explain how the biases affect markets (Soros, 2009). He argued that positive feedback loops push investors to form expectations that diverge from economic fundamentals, which may cause persistent bull markets. He adds that this process is self-reinforcing and cannot last forever as it would reach a moment when market participants view expectations are unrealistic.

While this fits perfectly with the case of Cathie Wood's fund, it goes against the efficient market hypothesis. Soros said that instead of equilibrium, "*we are faced with a dynamic disequilibrium or what may be described as far-from-equilibrium conditions*." These conditions create the environment for a reversal movement when the bubble pops as the divergence between expectations and fundamentals is maximum.

[2] Own calculation with data from Yahoo Finance.

3. FIELD OF STUDY

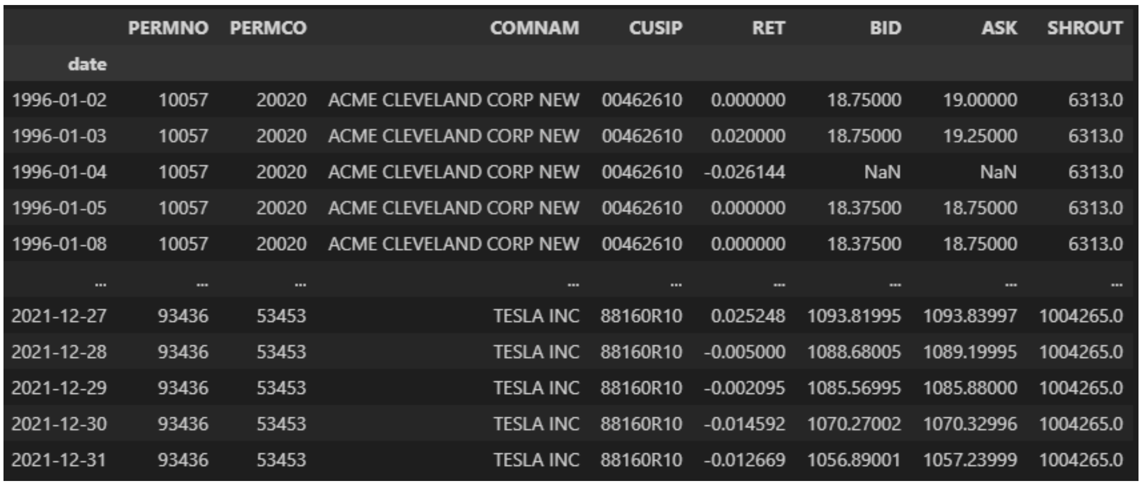
3.1. Criteria for company selection

For the selection of the data used in this research, it was decided to remain with United States as the only country of study, with the main reason being the availability of data and its reliability. To replicate the behaviour of a large, domestic equity mutual fund, it was deemed reasonable to narrow the investable universe to S&P 500 historical components, since small-cap stocks may be illiquid for large funds as well as less accessible for retail investors.

As data for historical components of S&P 500 index, accounting for entries and exclusions, is difficult to obtain, an alternative approach was followed. A list of historical components was obtained (Historical Lists of S&P 500 components) and no adjustments for index exclusion were done. Instead, the list of tickers was fed on Wharton Research Data Services (WRDS) website and historical data since the beginning of 1996 to the end of 2021 was downloaded.

3.2. Description of data

The following data was retrieved from the Center for Research in Security Prices database (CSRP) within the WRDS website.



	PERMNO	PERMCO	COMNAM	CUSIP	RET	BID	ASK	SHROUT
date								
1996-01-02	10057	20020	ACME CLEVELAND CORP NEW	00462610	0.000000	18.75000	19.00000	6313.0
1996-01-03	10057	20020	ACME CLEVELAND CORP NEW	00462610	0.020000	18.75000	19.25000	6313.0
1996-01-04	10057	20020	ACME CLEVELAND CORP NEW	00462610	-0.026144	NaN	NaN	6313.0
1996-01-05	10057	20020	ACME CLEVELAND CORP NEW	00462610	0.000000	18.37500	18.75000	6313.0
1996-01-08	10057	20020	ACME CLEVELAND CORP NEW	00462610	0.000000	18.37500	18.75000	6313.0
...	...	...	...	...	...	...	...	...
2021-12-27	93436	53453	TESLA INC	88160R10	0.025248	1093.81995	1093.83997	1004265.0
2021-12-28	93436	53453	TESLA INC	88160R10	-0.005000	1088.68005	1089.19995	1004265.0
2021-12-29	93436	53453	TESLA INC	88160R10	-0.002095	1085.56995	1085.88000	1004265.0
2021-12-30	93436	53453	TESLA INC	88160R10	-0.014592	1070.27002	1070.32996	1004265.0
2021-12-31	93436	53453	TESLA INC	88160R10	-0.012669	1056.89001	1057.23999	1004265.0

Figure 2 - Preview of data downloaded from CSRP database - Own elaboration

PERMNO – Permanent identifier assigned by CSRP to every security.

PERMCO – Permanent identifier assigned by CSRP to every company.

COMNAM – Company name.

CUSIP – Eight-character security identifier.

RET – Return for every day, accounting for corporate actions like dividends or splits, assuming midpoint between bid and ask.

BID – Closing bid price at every day.

ASK – Closing ask price at every day.

SHROUT – Number of publicly held shares, in thousands.

3.3. Data processing

To clean the data and let it ready for the simulations, several manipulations were needed. First, three days were removed as they were included with some data in the sample although they were not trading days on the United States stock market. Then, using the variable PERMNO, an iterative process was done to find the last name for every security, to account for the name changes or acquisitions that were performed during the period. After that, using PERMCO a filter was used to be left only with companies. Nevertheless, some investment trusts were left in the sample, so they were manually removed.

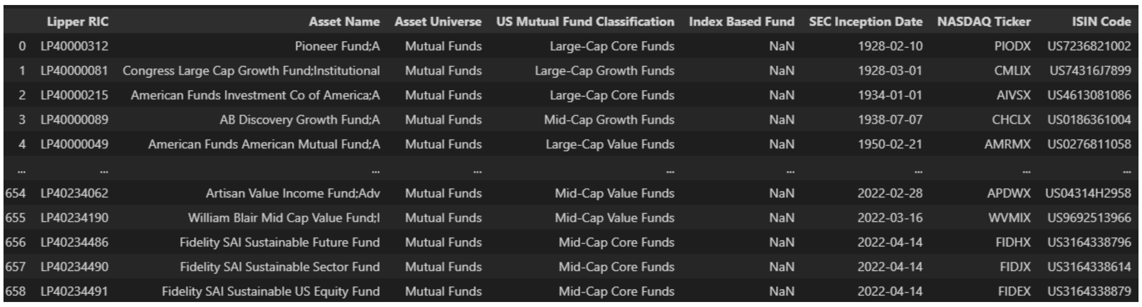
Ask price data was used together with share outstanding data to compute the market capitalization of every company. As a filter to ensure reliability data and adhere to the requisite of not including small caps, the market cap of every company at the end of every month was computed to ensure that a company was only investable if it had a market cap of at least 5 billion dollars, slightly below the usual minimum capitalization to be included in the S&P 500 index.

Lastly, a price index was computed for every company, compounding their returns to monthly periods, and letting the data ready for the simulations.

3.4. Mutual fund data

To compare the simulations with real-world funds, historical data of mutual funds was retrieved. To ensure the maximum similarity with the simulations, the following criteria was used.

- In first place, only funds domiciled in the United States were eligible.
- Secondly, only mid-cap or large-cap focused funds were elected, to match the size of the companies the simulations can invest in.
- Finally, index funds were filtered out to ensure only active funds remained in the selection. This screening was done on the Thomson Reuters Eikon Terminal, using Refinitiv Lipper data. After this screening, 659 mutual funds were found, and the ticker from each one was downloaded.



	Lipper RIC	Asset Name	Asset Universe	US Mutual Fund Classification	Index Based Fund	SEC Inception Date	NASDAQ Ticker	ISIN Code
0	LP40000312	Pioneer Fund;A	Mutual Funds	Large-Cap Core Funds	NaN	1928-02-10	PIODX	US7236821002
1	LP40000081	Congress Large Cap Growth Fund;Institutional	Mutual Funds	Large-Cap Growth Funds	NaN	1928-03-01	CMLIX	US74316J7899
2	LP40000215	American Funds Investment Co of America;A	Mutual Funds	Large-Cap Core Funds	NaN	1934-01-01	AIVSX	US4613081086
3	LP40000089	AB Discovery Growth Fund;A	Mutual Funds	Mid-Cap Growth Funds	NaN	1938-07-07	CHCLX	US0186361004
4	LP40000049	American Funds American Mutual Fund;A	Mutual Funds	Large-Cap Value Funds	NaN	1950-02-21	AMRMX	US0276811058
...	...	...	...	...	...	...	...	...
654	LP40234062	Artisan Value Income Fund;Adv	Mutual Funds	Mid-Cap Value Funds	NaN	2022-02-28	APDWX	US04314H2958
655	LP40234190	William Blair Mid Cap Value Fund;I	Mutual Funds	Mid-Cap Value Funds	NaN	2022-03-16	WVMIX	US9692513966
656	LP40234486	Fidelity SAI Sustainable Future Fund	Mutual Funds	Mid-Cap Core Funds	NaN	2022-04-14	FIDHX	US3164338796
657	LP40234490	Fidelity SAI Sustainable Sector Fund	Mutual Funds	Mid-Cap Core Funds	NaN	2022-04-14	FIDJX	US3164338614
658	LP40234491	Fidelity SAI Sustainable US Equity Fund	Mutual Funds	Mid-Cap Core Funds	NaN	2022-04-14	FIDEX	US3164338879

Figure 3 - Preview of data downloaded from Eikon Terminal - Own elaboration

As the price data was difficult to obtain from the Eikon Terminal, the tickers were used to download historical data from Yahoo Finance using a Python API. Unfortunately, a couple of problems arose with the use of mutual funds. First, it was not possible to retrieve a survivorship bias-free dataset, as it was not possible to find inactive funds with a reliable historical price data. Secondly, not every fund started their operations at the beginning of the date range used in this study, so a reduced number of funds is available in the first periods.

4. RANDOM PORTFOLIOS AND MUTUAL FUNDS

4.1. Random portfolios.

4.1.1 How the simulations were done.

With the data downloaded and curated, it is now ready to perform the simulations to compute the random portfolios. The entire process was developed using Python, and its development was uploaded to a GitHub repository. A function was created to generate the desired number of portfolios with the preferred number of companies in each one. To compute the portfolios, a Python library called "bt" was used. This library allows for a very customizable strategy back-test. In this case the following characteristics were used:

- **Rebalance frequency** – Monthly. As developments can happen quickly in the stock market, any manager may drop a holding in any given day. To account for that, it was deemed appropriate to set a monthly rebalance frequency, meaning that every month the portfolio can (completely) change.
- **Stock-picking method** – Random with constrains. Every period, the portfolio picks randomly a number of stocks, given that they are active in the previous period (this controls for acquired companies, among other events) and that they had a market capitalization of at least five billion dollars.
- **Number of holdings** – For the simulations, a number of holdings of twenty was used. Many studies have reflected about the number of stocks that makes a portfolio diversified. As some authors compiled, the number ranges from eight to more than three hundred. (Zaimovic, Omanovic, & Arnaut-Berilo, 2021) Twenty was deemed enough given the rebalancing frequency and in consonance with the next characteristic.
- **Security weighting** – As the own Markowitz confessed, an equally weighted portfolio may be the best ex-ante decision for weighting the holdings, and the least biased. This way, every company is given the same chance to contribute to the overall performance, and the volatility of the portfolio is maximally reduced without incurring into bias. For example, if an alternative, inverse volatility method was used, value and size factors would have been exaggerated (Novy-Marx, 2014)
- **Number of simulations** – Ten thousand portfolios were generated.
- **Time period** – As the first data in the sample belongs to January 1996, the simulation first waits to the end of January to filter market capitalizations and active stocks. Then, at the end of February 1996 the portfolios trade for the first time, so the simulation period goes from March 1996 through December 2021.

4.1.2 Graphs

To visualize the portfolios, a couple of graphs were generated.

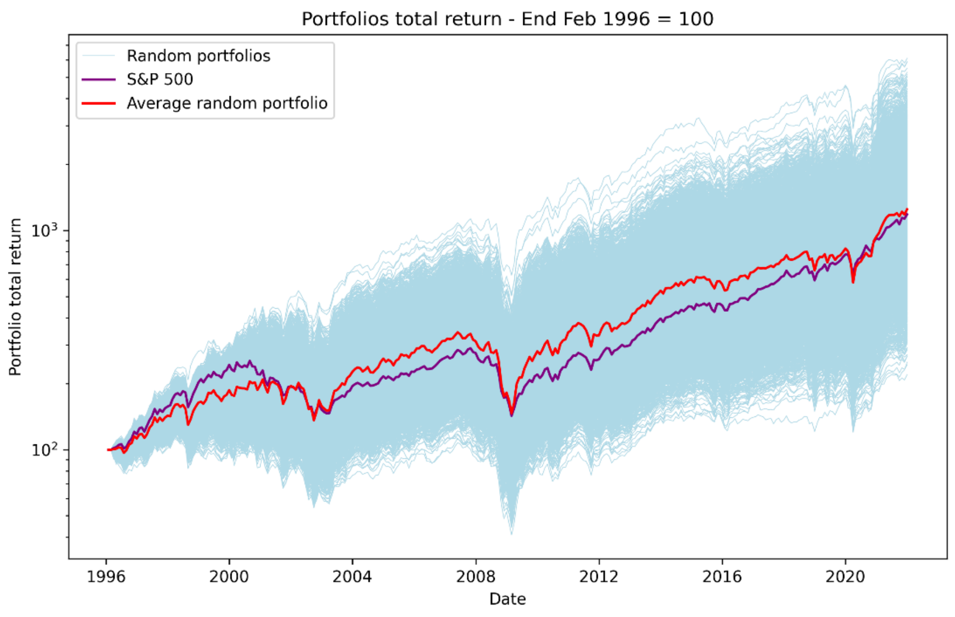


Figure 4 - Evolution of ten thousand simulated portfolios from March 1996 to December 2021 - Own elaboration

In the first graph, the evolution of the ten thousand simulated portfolios is depicted, along with the average random portfolio and the S&P 500 index as the benchmark. In order to account for the compounding effect, the y-axis is in log scale. Despite this modification, a great dispersion can be seen, especially during high volatility periods.

As expected, the average portfolio tracks closely the S&P 500, although it has periods of over and under performance relative to this benchmark, that can be explained mainly through the exposure to the size factor. Returns are extraordinarily polarized, with nine portfolios multiplying the initial capital by more than fifty times and twelve doing so less than 3 times, while the benchmark saw an appreciation of almost twelve times the invested capital.

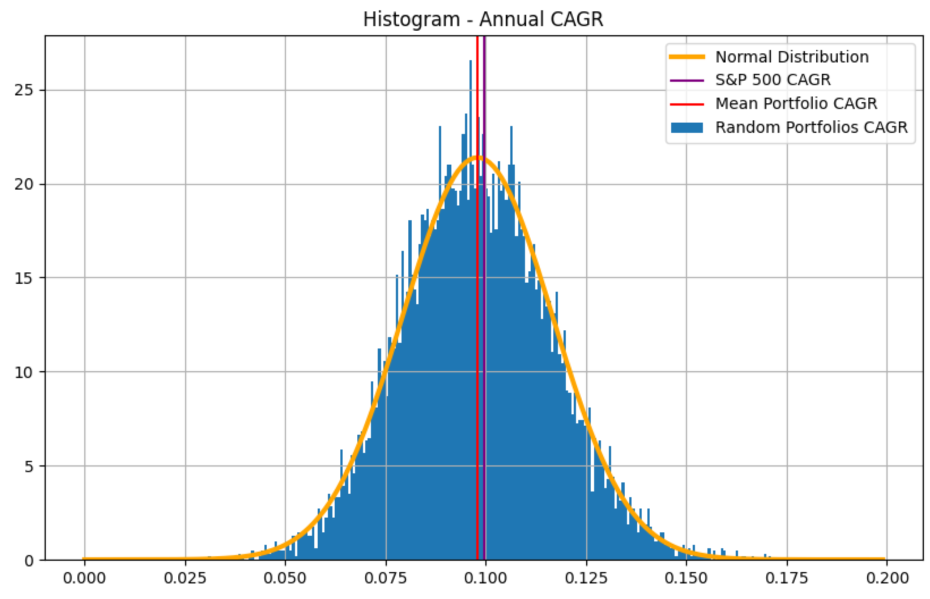


Figure 5 - Random portfolios' CAGR histogram – Own elaboration

Translating the previous graph to a histogram of the compounded annual growth rate (CAGR), a distribution close to the Gaussian can be seen. But as it is usual in finance, fat tails are present, and the normal distribution is a poor fit no matter the number of observations. Thus, portfolios of close to 18% CAGR appear in the sample, as well of just 3%, in a period where the S&P 500 yielded almost 10% annually.

Just for the sake of assessing the fit of the normal distribution, the highest CAGR had a 0.0043% probability, or 1 in 23210, of appearing on the sample under this distribution. This is another proof that an extremely cautious approach should be used when dealing with distributions in finance, as extreme cases are often under-estimated, even in cases like this when diversified portfolios with frequent rebalance, that mitigate risk, are used.

4.2. Mutual funds

4.2.1 Description

After downloading the data using the Yahoo Finance API, prices were converted to monthly periods to compute returns with this frequency. First, the data was inspected to check the availability of data through the years.

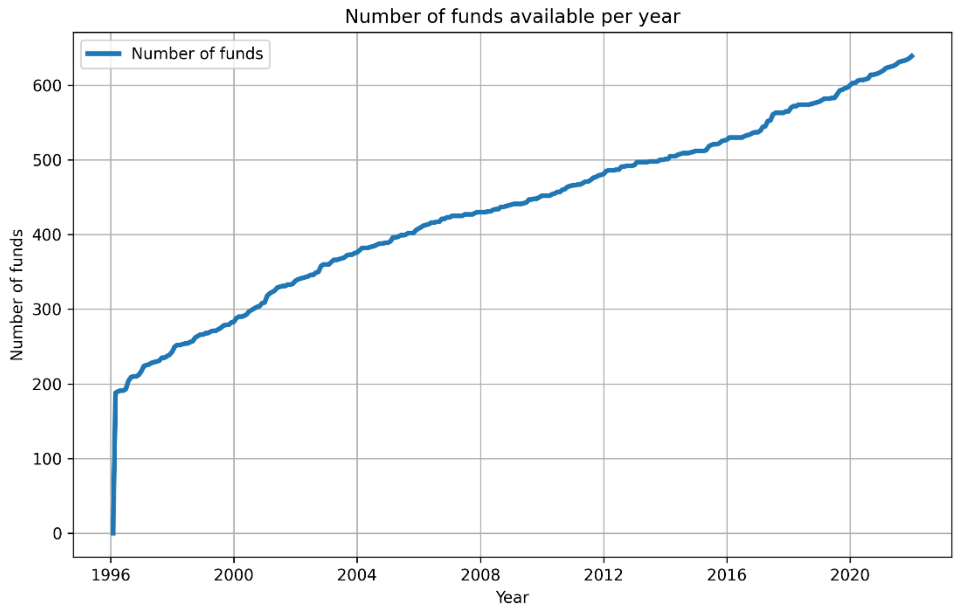


Figure 6 - Evolution of mutual fund availability data - Own elaboration

As can be seen in the graph, less than half of the funds are available at the beginning of the period, but the number evolves linearly throughout the years. Notoriously, this evolution does not seem to be impacted by relevant financial events like the Great Financial Crisis (2007-2009).

4.3. Volatility regimes

In order to establish fair comparisons across portfolios and funds, a separation by time and volatility was made. This means that five different time periods inside the time range were selected, according to their volatility regimes.

To identify volatility regimes, the benchmark S&P 500 index was used. Computing the mean yearly rolling volatility for every year in the period, volatility regimes can easily be spotted.

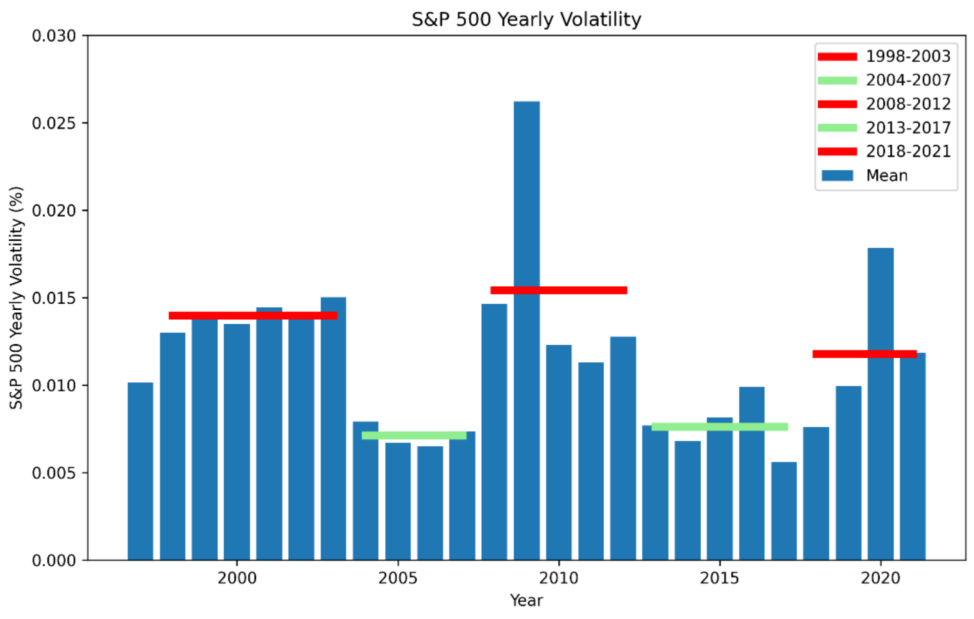


Figure 7 - S&P 500 Yearly Volatility - Own elaboration

Specifically, five periods were determined after classifying them into low or high volatility regimes. 1998-2003, 2008-2012 and 2018-2021 made it into the high volatility regimes and their mean can be seen in red, while 2004-2007 and 2013-2017 classified for the low volatility ones and their mean appear in green.

This classification can also be seen as a separation between bull markets, characterized by low volatility, and bear markets, with high volatility. As seen before in Figure 4, equity indices usually show a low realized volatility during uptrends, that may last several years, for example between 2013 and 2017. In fact, 2017 had by the far the lowest volatility in the sample, a situation that led to the popularity of short-volatility funds which ultimately resulted in a surge in volatility from 2018 onwards (Franck, 2018). This quick change between regimes is a key aspect in financial markets and it is important to precisely delimit them to be able to do a fair analysis.

5. TESTS

5.1. Alpha measurement

With all the data in place, everything is ready to start testing whether the portfolio managers have an ability to outperform the market or not.

The first computation to be done was the alpha according to the previously exposed Fama-French five factors model. Starting with the simulated portfolios, different computations were done to each period, and the summarized results can be seen below.

	First period 1998-2003 Bear	Second period 2004-2007 Bull	Third period 2008-2012 Bear	Fourth period 2013-2017 Bull	Fifth period 2018-2021 Bear
25%	-0,189%	-0,308%	-0,016%	0,364%	-0,375%
Mean	0,084%	-0,091%	0,234%	-0,173%	-0,079%
75%	0,357%	0,136%	0,484%	0,021%	0,122%

Table 1 - Summary of alpha for random portfolios

For instance, in the bear market of 2008-2012, the mean of the alpha of the random portfolios, is 0,234%, meaning that, on average, random portfolios produced 0.234% alpha per year. It is especially remarkable the difference in the behaviour of alpha depending on the market regime. Alpha tends to be higher in bear markets than in bull markets, as can be seen in the graph below.

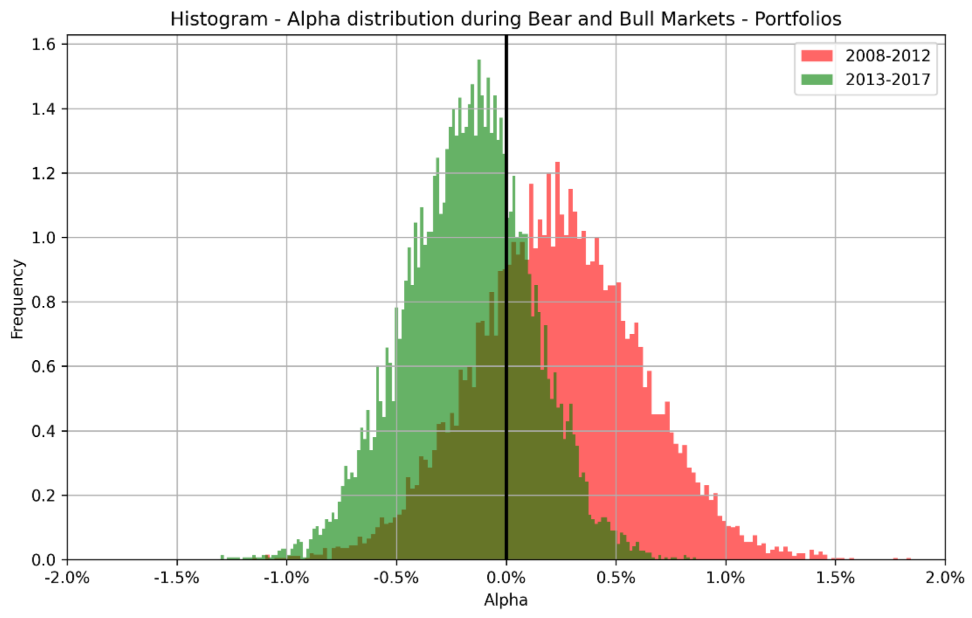


Figure 8 - Alpha distribution during Bear and Bull markets for random portfolios - Own elaboration

	2008-2012	2013-2017
Mean	0.234%	-0,173%
Standard Deviation	0.375%	0.288%

Table 2 - Mean and standard deviation for the alpha of the FF 5-factors model during Bear and Bull markets for random portfolios

The distribution shifts to the right and becomes wider during the bear market provoked by the Great Financial Crisis (2007-2009) than during the subsequent recovery in financial markets. This suggests that the ability of portfolio managers to generate excess risk-adjusted returns may be restricted to periods of market declines, although the wider distribution imply that this ability is very volatile and uncertain. This has been corroborated in similar research by Vanguard (Felix, 2018).

To try to explain this different behaviour between bear and bull markets, the following graphs were produced.

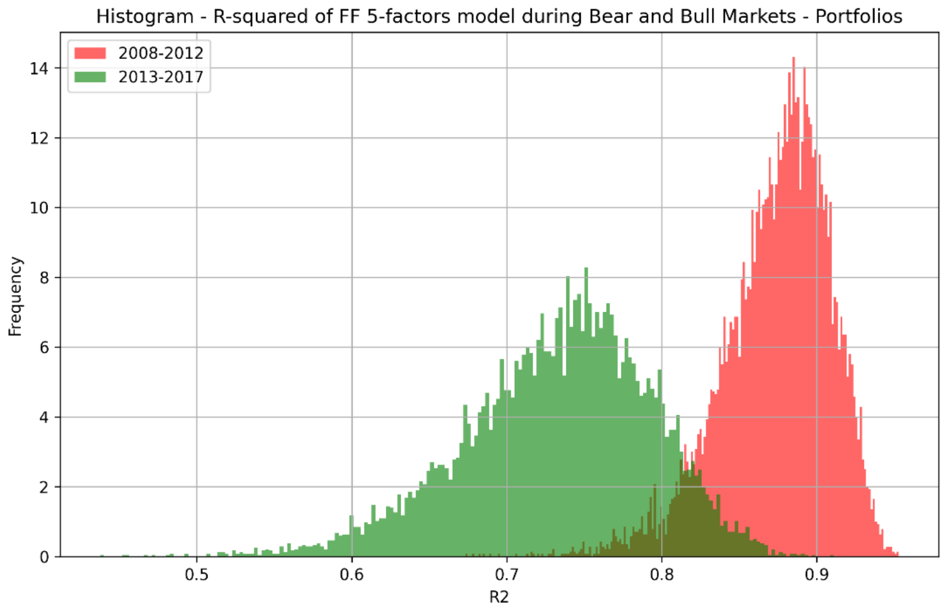


Figure 9 - R-squared of the FF 5-factors model during Bear and Bull markets for random portfolios - Own elaboration

	2008-2012	2013-2017
Mean	0.8740	0.7330
Standard Deviation	0.0349	0.0650

Table 3 - Mean and standard deviation for the R-squared of the FF 5-factors model during Bear and Bull markets for random portfolios

First, the R-squared of the model, which is the amount of variance that the model is able to explain in each portfolio, is displayed as a histogram. As it can be seen, the distributions are completely different. The models are much better fit during bear markets than during bull markets. This may be a cause of the difference in alpha, but further analysis is needed.

Another factor that should be considered for the alpha computation is the correlation. Below, the 12-month rolling correlation with the S&P 500 benchmarked is plotted.

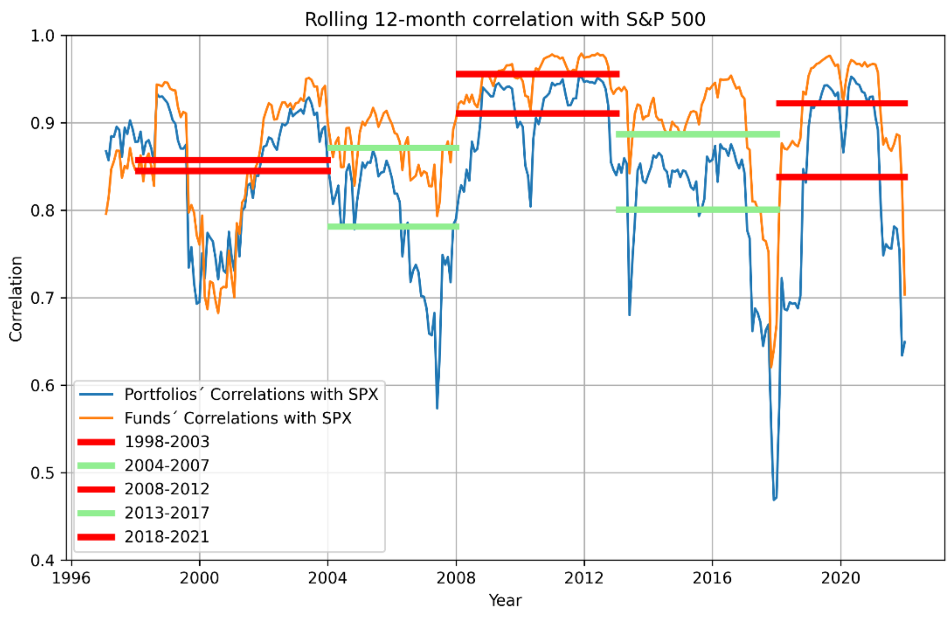


Figure 10 - Rolling 12-month correlation of random portfolios and mutual funds with respect to the S&P 500 – Own elaboration

As it can be seen, correlation is generally higher during bear markets, when usually every sector of the market is brought down, compared to bull markets when sector's performance is more disperse. This may be the explanation as to why the models are much better fit during bear markets, as the portfolios are much more similar between them as well as relative to the benchmark regarding their performance. This also highlights the ability of the portfolio managers to protect their returns during the downturns, although as the wider distribution suggests, this ability is more uncertain.

Now, it is the turn for the analysis of the mutual funds. It is important to note, that as not every fund is available at the beginning of the sample, each period has an increasing number of funds available. Also, the implicit survivorship bias should be kept in mind.

	First period 1998-2003 Bear	Second period 2004-2007 Bull	Third period 2008-2012 Bear	Fourth period 2013-2017 Bull	Fifth period 2018-2021 Bear
n	251	381	432	499	572
25%	-0,210%	-0,110%	-0,125%	-0,184%	-0,190%
Mean	0,042%	0,050%	0,013%	-0,097%	-0,031%
75%	0,264%	0,176%	0,124%	-0,001%	0,094%

Table 4 - Summary of alpha for mutual funds

In this case, in the bear market of 2008-2012, the mean of the alpha of the mutual funds is 0,013%, lower than the 0,234% of the random portfolios. Also, the quartiles are twice as closer. For the first periods especially, it could be assumed that the alpha is overstated due to biases, but the behaviour of alpha is similar to the one seen with the random portfolios, being in general higher during bear markets.

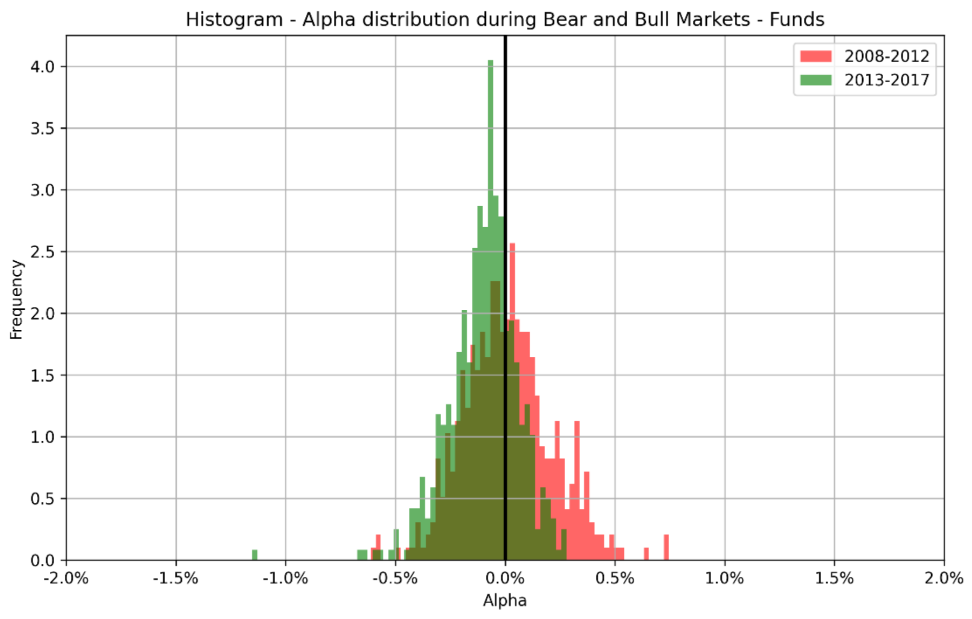


Figure 11 - Alpha distribution during Bear and Bull markets for mutual funds - Own elaboration

	2008-2012	2013-2017
Mean	0,013%	-0,097%
Standard Deviation	0.203%	0.159%

Table 5 - Mean and standard deviation for the alpha of the FF 5-factors model during Bear and Bull markets for mutual funds

Comparing the same periods of bull and bear markets than with the random portfolios, a similar distribution is observed. During bull markets the distribution becomes more leptokurtic, and when the market turns south the distribution shifts to the right and adopts a more platykurtic shape.

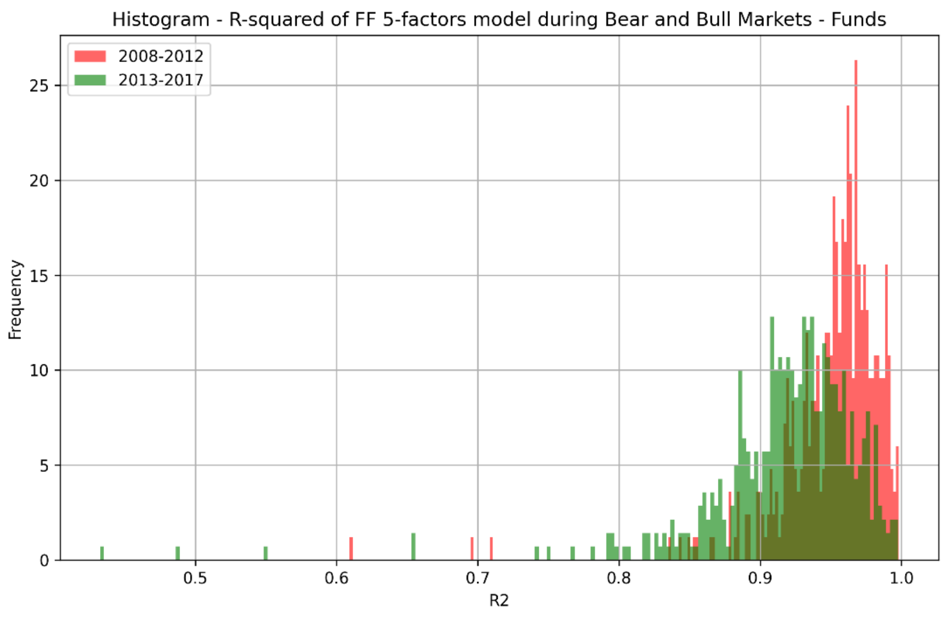


Figure 12 - R-squared of the FF 5-factors model during Bear and Bull markets for mutual funds - Own elaboration

	2008-2012	2013-2017
Mean	0.9514	0.9179
Standard Deviation	0.0372	0.0543

Table 6 - Mean and standard deviation for the R-squared of the FF 5-factors model during Bear and Bull markets for mutual funds

Also, the distribution for the goodness of the fit is similar to the simulated one, although in this case the mean of the bull market distribution is much close to the bear market one. Looking at the previous correlation graph, this could be explained by the higher correlation to the market the funds have relative to the simulated portfolios.

The conclusion with the first test is that alpha on aggregate is not present when controlling for the correct risk factors, which is the Fama-French Five Factor Model. However, this alpha may be significant depending on the market regime.

In order not to over-complicate the thesis, from this point only the third (2008-2012) and fourth (2013-2017) are going to be considered for the tests. The reason is that these periods are likely to be the most representative of future bear and bull markets behaviour, and have a decent amount of data available. For instance, the period of 2018 onwards, is distorted by the March 2020 crash, which could be considered as a black swan event, and the case of GameStop stock, whose rapid and irrational increase in price – of more than 1700% (Phillips & Lorenz, 2021) - supposed a good amount of alpha for those portfolios that were lucky to be invested in it during the surge.

5.2. Probability of finding a fund/portfolio with positive alpha

To compare the probabilities of finding positive alpha, rolling regressions were computed to calculate the alpha achieved in the last twelve months. Then the proportion of portfolios and funds with positive alpha was calculated to be able to compare to each other fairly. In order to also consider the riskiness of the portfolio relative to the market, the proportion of portfolios with alpha was classified according to their beta.

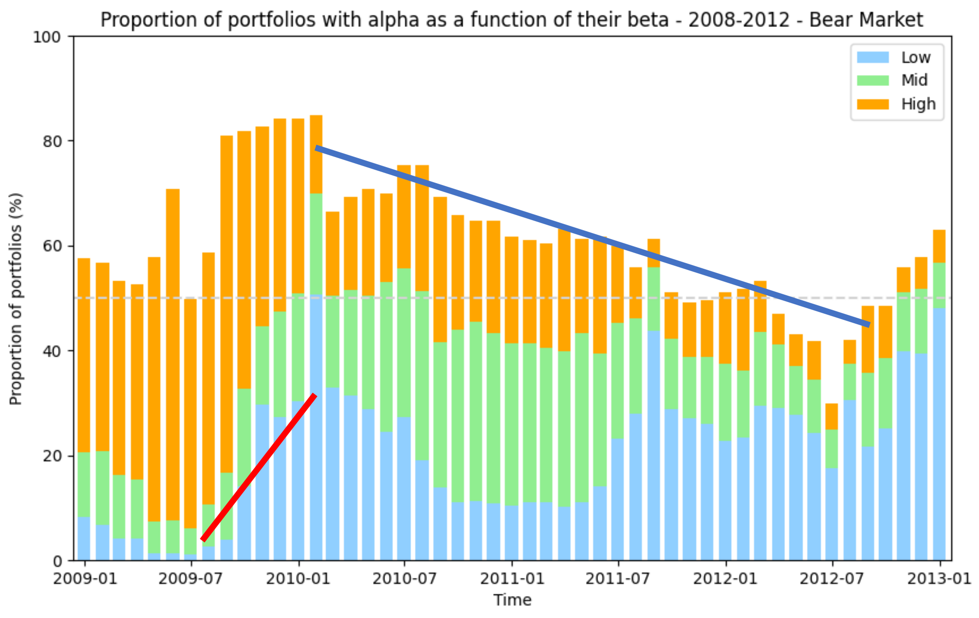


Figure 13 - Proportion of random portfolios with alpha as a function of their beta for a Bear market

First, plotting the evolution during the bear market it can be seen that random portfolios are superior to managed funds. On average, the probability of having a random portfolio with alpha in the past 12 months is slightly higher than a coin toss, whereas for mutual funds it is lower. It can be noted that the proportion becomes higher for both portfolios and funds while the market sold off during 2009, especially in the low beta bracket – the threshold for low beta is lower than 0.9, for high beta higher than 1.1 – while they turn lower and lower when the market starts to recover.

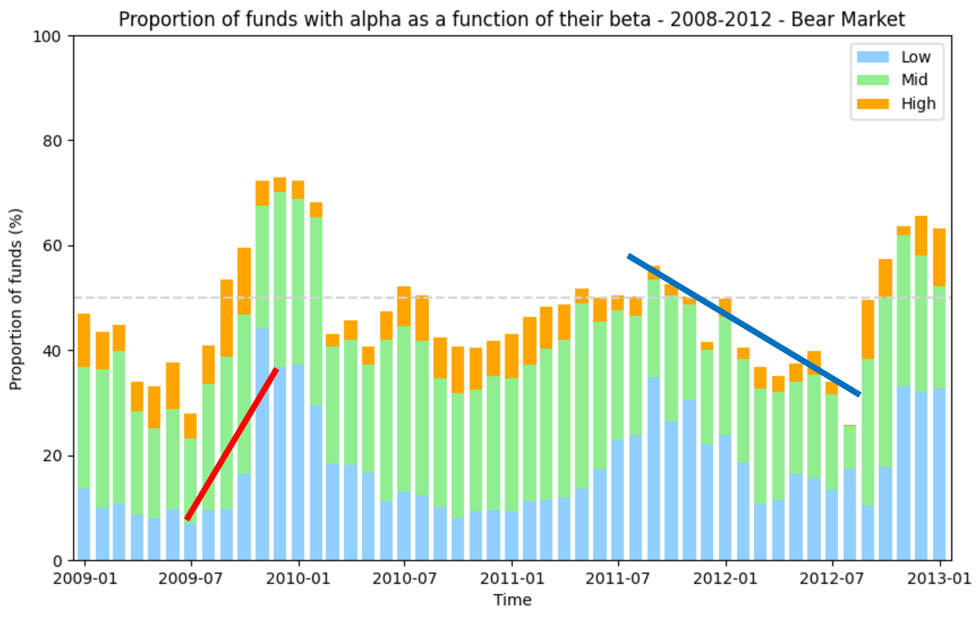


Figure 14 - Proportion of mutual funds with alpha as a function of their beta for a Bear market

Looking specifically to funds, it is shown that for most of the time the majority of the funds have negative alpha during this period. It is important to note that despite having on aggregate positive alpha during the entire period, as seen before, the consistency is low and for most of the time one would hold a negative alpha portfolio even during a period that is supposed to be more benign to actively managed portfolios relative to an index fund.

Now the attention turns to the bull market that reigned for most of the 10s decade.

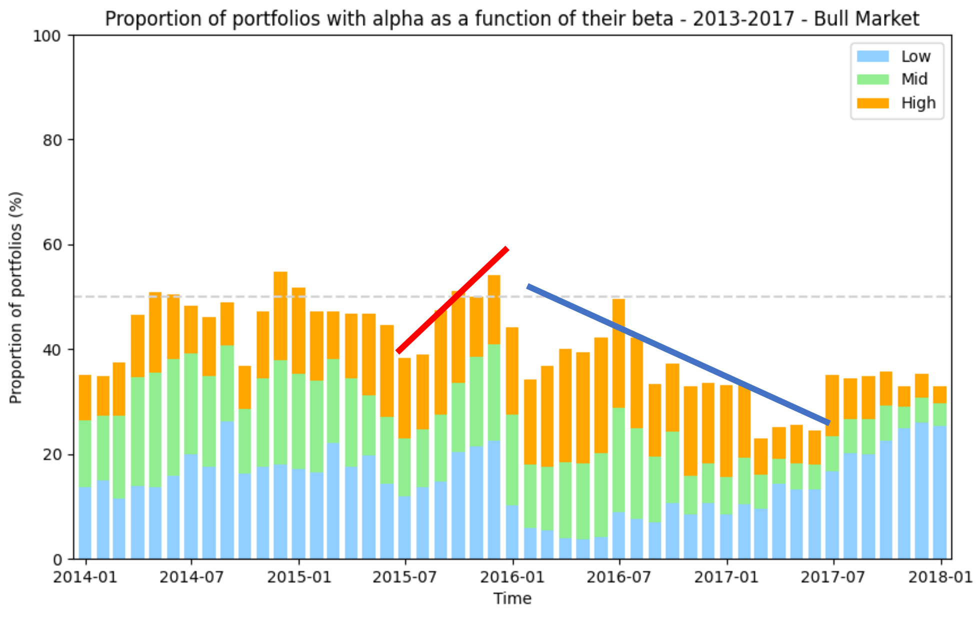


Figure 15 - Proportion of random portfolios with alpha as a function of their beta for a Bull market

Here can be seen that the proportion of random portfolios with positive alpha during the previous year is much lower, and barely reaches 50% at any time. Notoriously the only time when it is close is during 2016, when an economic deceleration threatened with a recession.

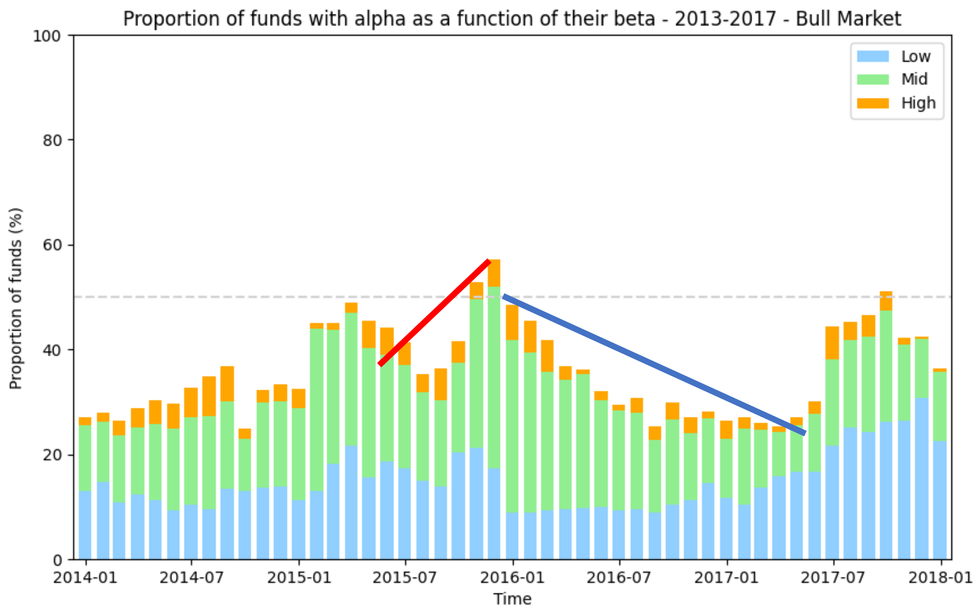


Figure 16 - Proportion of mutual funds with alpha as a function of their beta for a Bull market

When it comes to mutual funds, a similar proportion is depicted although with even lower proportions especially during low volatility periods like 2017. This supports the idea previously presented. When low volatility – therefore bull market – is present, stock correlations relative to the benchmark are much lower and fund managers struggle to select the best performers for their funds, resulting in better chances for the random portfolios.

5.3. Alpha persistence.

To be able to quantify the ability of portfolio managers to maintain the alpha they achieved, a concept known as persistence, the "Alpha Decay Line" (ADL) is introduced. For every month in the period, the number of funds – or random portfolios – that show positive alpha is computed, and it is searched the proportion of them that continue to show alpha in the following 24 months. Then the mean for each of the 24 months is computed to produce the "Alpha Decay Line".

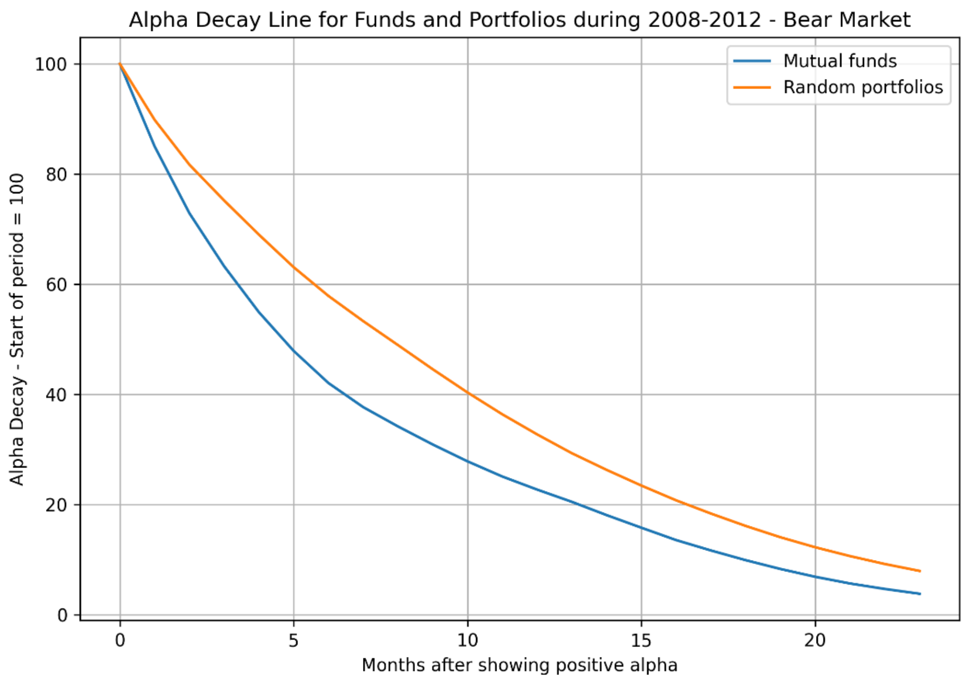


Figure 17 - Alpha Decay Line for mutual funds and random portfolios during a Bear market

As it is shown in the first graph, during the bear market funds struggle to show consistency at much faster pace than random portfolios. In only five months the majority of funds lose their alpha, proving that while managers show better ability during bear markets, this is ephemeral and loses ground quickly. Here, many psychological biases may have an impact, like Soros explained, as well as circumstances that may present during bear markets like liquidations.

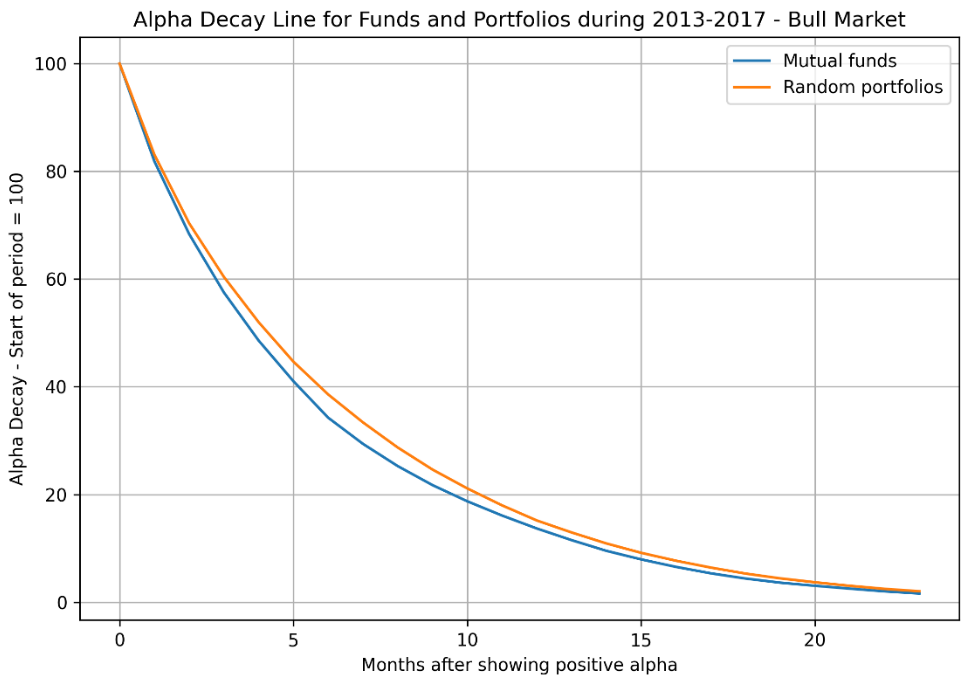


Figure 18 - Alpha Decay Line for mutual funds and random portfolios during a Bull market

However, during bull markets funds and portfolios are much closer. Here it is noticed that both discretionary and random stock picking methods lose alpha very quickly. For instance, during the bear market, 40% of the portfolios retained alpha ten months before, while the figure comes down to just 20% during the bull market.

This is yet another representation of the difficulty to generate alpha in times of low volatility and correlations.

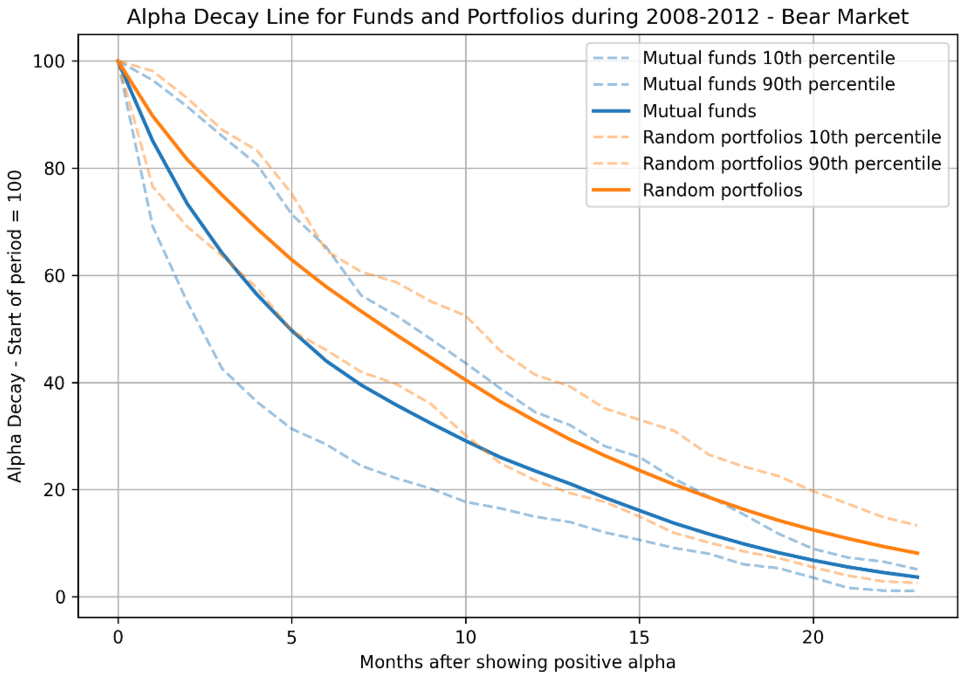


Figure 19 - Alpha Decay Line and percentiles for mutual funds and random portfolios during a Bear market

If alongside the "Alpha Decay Line" the 10th and 90th percentile lines are plotted, the dispersion from the mean can be appreciated. For mutual funds, this dispersion is far greater, especially from the second month onwards when the alpha persistence for funds falls off a cliff. However, for the 90th percentile, first half a year is similar between funds and random portfolios, meaning that on the best scenario cases both perform equally regarding alpha persistence. This highlights the ability of some active managers to preserve capital during market downturns, although as it can be seen this ability is not maintained for more than six months. This ability in some managers explains the wider alpha distribution during the bear market.

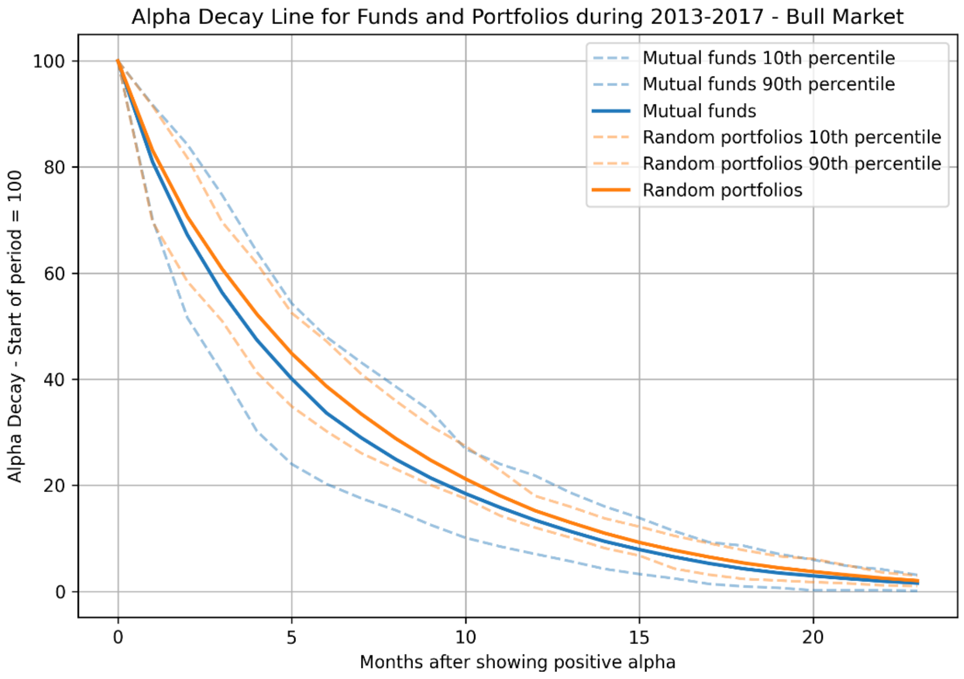


Figure 20 - Alpha Decay Line and percentiles for mutual funds and random portfolios during a Bull market

Contrasting with the bull market, the dispersion is much tighter. Some characteristics from the bear market chart remain, like the ephemeral persistence for the worst funds. However, the best funds keep pace with the best portfolios in terms of alpha persistence, which suggests there is a random component in the ability of these managers to maintain their alpha, which in any case decreases very quickly.

5.4. Machine learning classification models

For the last round of tests to determine whether managers are skilled or not, classification statistical models were used. The idea is to let the models differentiate if a portfolio is random or not, based on the factors obtained through the Five Factor Fama-French model. If models are not able to differentiate them, the original idea that managers are not able to outperform randomness could be accepted.

To do so, in first place, several models were used. Also, in order to be consistent with the previous tests, in this occasion the data was also split according to the volatility periods previously defined.

To evaluate the accuracy of the models, a confusion matrix is used. Confusion matrices allow for an overview of the number of observations that the model is able to predict accurately, as well as type one and two errors. Type one errors occur when a true observation is deemed as false, and type two errors happen when a false observation is predicted as true. For this analysis, both types or errors are of interest as the focus remains in the ability of the models to differentiate between random portfolios and mutual funds.

To do so, a key metric is the class precision. This indicator computes the proportion of times that a class is correctly classified. For instance, if the class is “random portfolios,” its precision is the percentage of times that when the observation actually is “random portfolios,” the model has classified it correctly.

5.4.1 K-means and Support Vector Machines

K-means. This clustering algorithm divides observations into a given number of clusters, in this case the number of interest is two, one for mutual funds and the other for random portfolios. This unsupervised machine learning algorithm tries to find homogeneous groups whose members are more similar between them than the others. In this case, the features that the algorithm tries to group are the alpha, beta, and the rest of factor of the Five-Factor model, that is, size, value, profitability, and investment.

The results from this model are gathered in the following confusion matrix.

Bear market.

Accuracy = 51,96%	Predicted - Random Portfolios	Predicted - Mutual Funds
Actual - Random portfolios	48,1%	47,8%
Actual - Mutual funds	0,2%	3,9%

Table 7 - K-means confusion matrix - Bear Market

The model has a poor accuracy, as it barely classifies correctly more than half of the observations. What results specially interesting is that it confounds random portfolios for mutual funds more than the other way around. To be precise, the class precision for mutual funds is 94.41%, but this percentage decreases to 50.14% for random portfolios.

Bull market.

Accuracy = 52,71%	Predicted - Random Portfolios	Predicted - Mutual Funds
Actual - Random portfolios	48,3%	46,9%
Actual - Mutual funds	0,4%	4,4%

Table 8 - K-means confusion matrix - Bull Market

During the subsequent bull market, the results are similar. In this case the precision predicting mutual funds slightly decreases to 92,55%, while regarding random portfolios increases to 50,74%. This suggests that a fair amount of the actively managed funds is likely to have no additional value added than a random portfolio as it is likely that its behaviour is similar.

Support Vector Machines. This more advanced algorithm is a supervised learning classification method rather than an unsupervised clustering one like k-means. Given a set of labelled data, the algorithm is trained and tries to classify them into two groups with the highest possible margin between them. In this case, as the data has more than two dimensions, the algorithm uses more complex kernel functions to deal with non-linearity.

The results from this model are the following.

Bear market.

Accuracy = 95,97%	Predicted - Random Portfolios	Predicted - Mutual Funds
Actual - Random portfolios	95,93%	0,00%
Actual - Mutual funds	4,03%	0,04%

Table 9 - Support Vector machines confusion matrix - Bear Market

Here, the results are different compared to the k-means algorithm. The accuracy is much higher, but it is biased due to the number of random portfolios. Looking at class precision, it is 100% for random portfolios as it correctly classifies all of them. The problem comes with mutual funds, as the precision plummets to 0,94%.

Bull market.

Accuracy = 95,29%	Predicted - Random Portfolios	Predicted - Mutual Funds
Actual - Random portfolios	95,29%	0,00%
Actual - Mutual funds	4,71%	0,00%

Table 10 - Support Vector machines confusion matrix - Bull Market

During the bull market the difference is even more extreme, with a 100% class precision for random portfolios and 0% for mutual funds. Here, the results are completely different to k-means algorithm. The class with high precision is random portfolios rather than mutual funds. However, for this analysis the same idea can be drawn in both cases, mutual funds and random portfolios exhibit similar behaviour that cannot be differentiated by a classification algorithm, especially during bull markets.

5.4.2 Decision tree and gradient boosted trees

Now, more sophisticated, and innovative classification models are used to analyse and compare their results. For these more advanced models, RapidMiner statistical software was used, where the labelled data was fed, and several models proposed by this tool where tested. Out of them, these two models had the best accuracy and were therefore more thoroughly examined.

Decision Tree. This model, as the support vector machines, is a supervised learning method, which can be used for regression or classification. In this case classification trees are be used. Decision trees also take sets of labelled data, and models a tree where the data is classified according to the value it takes on the different variables that were fed in the model. It provides a straightforward way to visualize the criteria that makes the model classify each observation into a group.

Bear market.

Accuracy = 97,8%	Predicted - Random Portfolios	Predicted - Mutual Funds
Actual - Random portfolios	95,74%	0,17%
Actual - Mutual funds	2,08%	2,01%

Table 11 - Decision Tree confusion matrix - Bear Market

This model increases the accuracy to almost 98%, with a class precision for random portfolios of 99.82% and 49.18% for mutual funds. The model is particularly good at detecting the random portfolios but struggles more with the funds, although half of them is correctly assessed.

Bull market.

Accuracy = 96,2%	Predicted - Random Portfolios	Predicted - Mutual Funds
Actual -Random portfolios	95,13%	0,10%
Actual - Mutual funds	3,67%	1,10%

Table 12 - Decision Tree confusion matrix - Bull Market

Comparing the results with the bull market, the class precision in random portfolios increases marginally to 99.89%, but the precision with mutual funds decreases significantly to 23.07%. This result is line with the results from previous tests. During a bull market, a fund is more likely to behave just like a random portfolio would do, achieving no alpha. However, during bear markets managers have a better chance, on average, to beat markets returns on a risk-adjusted basis.

Gradient Boosted Trees. This algorithm is similar to decision trees. Several trees are iteratively computed, and progressively adjusted with the goal of minimizing the loss function. The gradient is the incremental adjustment made during each iteration and boosting is a method of accelerating the improvement in predictive accuracy. (C3ai, s.f.)

Bear market.

Accuracy = 98,3%	Predicted - Random Portfolios	Predicted - Mutual Funds
Actual - Random portfolios	95,84%	0,03%
Actual - Mutual funds	1,64%	2,48%

Table 13 - Gradient Boosted Trees confusion matrix - Bear Market

Similar to the decision tree results, accuracy is remarkably high, with a random portfolios precision of 99.96% and mutual funds precision of 60.16%.

Bull market.

Accuracy = 97,2%	Predicted - Random Portfolios	Predicted - Mutual Funds
Actual -Random portfolios	95,17%	0,10%
Actual - Mutual funds	2,70%	2,03%

Table 14 - Gradient Boosted Trees confusion matrix - Bull Market

Comparing the results with the bull market, random portfolios precision barely decreases to 99.89% and mutual fund precision falls to 42.96%. This model, although more accurate, gives similar indications than the decision trees one. During bull markets it is more difficult to differentiate between funds and random portfolios, suggesting that the ability of the managers is limited.

5.4.3 Conclusions from machine learning classification models.

From the classification models, several conclusions can be made. In first place, it is seen that the only unsupervised model, k-means has the by far the lowest accuracy, therefore its results must have less weight. In fact, looking at the rest of the models, all of them supervised models that go through a period of training, they have a common characteristic opposed to k-means. This means, the supervised models score a high accuracy regarding the random portfolios class. This means that all of them learnt how to tell if a portfolio is random rather than a managed one. On the other hand, they struggle to distinguish how many of the mutual funds are actually managed and not random.

Translating this to the argument of this thesis, the models are as well not able to differentiate if a mutual fund is actually managed by a professional or if it is just a random process. The precision for mutual funds is generally lower than 50% – except for k-means – which is a poor figure. The results are consistent with previous tests, as the precision is in any case higher during bear markets, which again suggests that the portfolio managers have more room to show their skills during bear markets.

6. CONCLUSIONS

The thesis started with the question of whether active managers are worth or not. After retrieving equity and mutual fund historical data, and processing it with self-developed software, enough data was available to conduct a series of tests that allow to answer the question. The main findings are the following:

- Alpha under FF5FM distributions is statistically different depending on the market being in a bull or bear market regime. During bear markets the distribution have on average a positive mean, while it is negative during bull markets. This holds for both random portfolios and mutual funds.
- Volatility not only defines the regime in which the market is, but also have a role in the shape of the distributions, making it wider during downturns.
- During bear markets, both random portfolios and mutual funds have, on average, better risk-adjusted returns than the market compared to bull markets, but the random managers significantly outperform actual ones. In comparison, during bull markets they are much closer.
- The introduced "Alpha Decay Line" suggests as well that random portfolios are better than mutual funds during bear markets, and are close during bull ones, and allows to see how little persistence they have. In about seven months alpha is lost on average during bear markets, and as soon as four months during bull ones.
- Classification models add another statistical perspective, suggesting that it is difficult to distinguish the most mutual funds whether they are random or not, especially during bull markets.

These findings, which are coherent and consistent with each other, suggest that portfolio managers are not worth the cost. For an individual investor, the probability of finding a fund that is going to produce better risk-adjusted returns than the market over the years is really low, as every analysis has suggested. On the other hand, risking the savings on random stocks does not produce worse outcomes that handing the money to a professional, which show the low ability that managers have. Therefore, in any case the best decision for a non-professional is to buy the market index through a cheap index fund and let the returns compound over time.

7. BIBLIOGRAPHY

Banz, R. W. (1981). The relationship between return and market value of common stocks. *Journal of Financial Economics, 9*(1), 3-18. Retrieved from https://doi.org/10.1016/0304-405X(81)90018-0

Bloomberg Intelligence. (2021, March 11). *Bloomberg*. Retrieved from Bloomberg Web site: https://www.bloomberg.com/professional/blog/passive-likely-overtakes-active-by-2026-earlier-if-bear-market/

Bloomberg Intelligence. (2021, March 11). Passive likely overtakes active by 2026, earlier if bear market. *Bloomberg*. Retrieved from https://www.bloomberg.com/professional/blog/passive-likely-overtakes-active-by-2026-earlier-if-bear-market/

Bower, B. (2011). Simple heresy: Rules of thumb challenge complex financial analyses. *Science News, 179*(12), 26-29. Retrieved from https://doi.org/10.1002/scin.5591791221

Bowman, J. (2020, December 30). *Move Over, Warren Buffett: This Is the Star Investor You Should Be Following*. Retrieved from Nasdaq: https://www.nasdaq.com/articles/move-over-warren-buffett%3A-this-is-the-star-investor-you-should-be-following-2020-12-31

C3ai. (n.d.). *Gradient-Boosted Decision Trees (GBDT)*. Retrieved from https://c3.ai/glossary/data-science/gradient-boosted-decision-trees-gbdt/

Fama, E. F. (1970). Efficient Capital Markets: A Review of Theory and Empirical Work. *The Journal of Finance, 25*(2), 383-417. doi:10.2307/2325486

Fama, E. F., & French, K. R. (1993). Common risk factors in the returns on stocks and bonds. *Journal of Financial Economics, 33*(1), 3-56. Retrieved from https://doi.org/10.1016/0304-405X(93)90023-5

Fama, E. F., & French, K. R. (2015). A five-factor asset pricing model. *Journal of Financial Economics, 116*(1), 1-22. Retrieved from https://doi.org/10.1016/j.jfineco.2014.10.010

Felix, B. (2018, January 18). Do Active Managers Protect Your Downside? | Common Sense Investing. Retrieved from https://youtu.be/KmXOvj_kRLA?t=194

Franck, T. (2018, February 6). *CNBC*. Retrieved from CNBC Web site: https://www.cnbc.com/2018/02/06/the-obscure-volatility-security-thats-become-the-focus-of-this-sell-off-is-halted-after-an-80-percent-plunge.html

Historical Lists of S&P 500 components. (n.d.). Retrieved from https://github.com/fja05680/sp500

Markowitz, H. (1952, March). Portfolio Selection. *The Journal of Finance, 7*(1), 77-91. doi:10.2307/2975974

Novy-Marx, R. (2014, October). Understanding Defensive Equity. *, NBER Working Paper No. 20591*. Retrieved from https://www.nber.org/system/files/working_papers/w20591/w20591.pdf

Phillips, M., & Lorenz, T. (2021, January 27). 'Dumb Money' Is on GameStop, and It's Beating Wall Street at Its Own Game. *The New York Times*. Retrieved from https://www.nytimes.com/2021/01/27/business/gamestop-wall-street-bets.html

Rosenberg, B., Reid, K., & Lanstein, R. (1985). Persuasive evidence of market inefficiency. *The Journal of Portfolio Management, 11*(3), 9-16. Retrieved from https://doi.org/10.3905/jpm.1985.409007

S&P Global. (2021). *SPIVA® U.S. Scorecard.* S&P Global. Retrieved from https://www.spglobal.com/spdji/en/documents/spiva/spiva-us-year-end-2021.pdf

Sharpe, W. F. (1964, September). CAPITAL ASSET PRICES: A THEORY OF MARKET EQUILIBRIUM UNDER CONDITIONS OF RISK*. *The Journal of Finance, 19*(3), 425-442. Retrieved from https://doi.org/10.1111/j.1540-6261.1964.tb02865.x

Soros, G. (2009, October 27). Soros: General Theory of Reflexivity. *Financial Times*. Retrieved from https://www.ft.com/content/0ca06172-bfe9-11de-aed2-00144feab49a

Zaimovic, A., Omanovic, A., & Arnaut-Berilo, A. (2021, November). How Many Stocks Are Sufficient for Equity Portfolio Diversification? A Review of the Literature. *Journal of Risk and Financial Management, 14*(11), 1-30. Retrieved from https://ideas.repec.org/a/gam/jjrfmx/v14y2021i11p551-d679488.html

DECLARATION OF ORIGINALITY

Hereby I, Iván Carrillo García, certify that the Bachelor Thesis titled “Portfolio managers’ skills or randomness during bull and bear markets” is totally original mine, that has not been presented in any other university as a Bachelor Thesis and that all sources that have been used have been properly cited and appear in the references.

Getafe, June 22, 2022

Signature: **Iván Carrillo García**

ANNEX A. GLOSARY

CAPM	*Capital Asset Pricing Model*
EMH	*Efficient Market Hypothesis*
FF3FM	*Fama-French three-factor model*
FF5FM	*Fama-French five-factor model*
SMB	*Small minus Big*
HML	*High minus Low*
RMW	*Robust minus Weak*
CMA	*Conservative minus Aggressive*
ETF	*Exchange-traded fund*
WRDS	*Wharton Research Data Services*
CRSP	*Center for Research in Security Prices*
CAGR	*Compounded Annual Growth Rate*
ADL	*Alpha Decay Line*

ANNEX B. PYTHON CODE

1_Data_processing

June 22, 2022

1 TFG

1.1 In the first part of the code, the data is processed and let ready for the computations.

1.1.1 First the data from WRDS is loaded and cleaned.

[2]:
```
import pandas as pd
import numpy as np
```

[ ]:
```
data = pd.read_csv("data_wrds.csv",engine="python")

data.index = data["date"]
del data["date"]

data = data.drop('29/10/2012')
data = data.drop('30/09/2017')
data = data.drop('23/04/2005')

data.index = pd.to_datetime(data.index,dayfirst=True)
```

1.1.2 Return data is cleaned

[2]:
```
returns = pd.DataFrame(index=data.index,columns=["RETURNS"])

for i in range(0,len(data["RET"])):

    if data["RET"][i] == "C":

        pass

    elif data["RET"][i] == "B":

        pass

    else:
        returns.iloc[i] = float(data["RET"][i])
```

```
data["RETURNS"] = returns
```

```
[2]:             PERMNO                  COMNAM  PERMCO     CUSIP       RET  \
date
1996-01-02   10057  ACME CLEVELAND CORP NEW   20020  00462610  0.000000
1996-01-03   10057  ACME CLEVELAND CORP NEW   20020  00462610  0.020000
1996-01-04   10057  ACME CLEVELAND CORP NEW   20020  00462610 -0.026144
1996-01-05   10057  ACME CLEVELAND CORP NEW   20020  00462610  0.000000
1996-01-08   10057  ACME CLEVELAND CORP NEW   20020  00462610  0.000000
…              …                       …     …        …         …
2021-12-27   93436               TESLA INC   53453  88160R10  0.025248
2021-12-28   93436               TESLA INC   53453  88160R10 -0.005000
2021-12-29   93436               TESLA INC   53453  88160R10 -0.002095
2021-12-30   93436               TESLA INC   53453  88160R10 -0.014592
2021-12-31   93436               TESLA INC   53453  88160R10 -0.012669

                   BID         ASK     SHROUT   RETURNS
date
1996-01-02    18.75000    19.00000     6313.0       0.0
1996-01-03    18.75000    19.25000     6313.0      0.02
1996-01-04         NaN         NaN     6313.0 -0.026144
1996-01-05    18.37500    18.75000     6313.0       0.0
1996-01-08    18.37500    18.75000     6313.0       0.0
…                  …          …         …        …
2021-12-27  1093.81995  1093.83997  1004265.0  0.025248
2021-12-28  1088.68005  1089.19995  1004265.0    -0.005
2021-12-29  1085.56995  1085.88000  1004265.0 -0.002095
2021-12-30  1070.27002  1070.32996  1004265.0 -0.014592
2021-12-31  1056.89001  1057.23999  1004265.0 -0.012669

[5726276 rows x 9 columns]
```

1.1.3 A list for the name of every security is created, so every ticker is labelled for the last name its security had.

```
[3]: names = pd.DataFrame(index=data["PERMNO"].unique(),columns=["Name"])

for i in data["PERMNO"].unique():
    names.loc[i] = data[data["PERMNO"]==i]["COMNAM"].unique()[-1]
```

1.1.4 The same is done for company names, so we only remain with companies. Also, ETFs and Trusts on the sample are removed.

```
[4]: pre_permco_list = list(data["PERMCO"].unique())

delete_permcos = [22100,29010,29548,29941,30421,32030,37493,39147,44072,45684,
                  45874,46673,50699,50846,51187,52902,52996,53171,54681,54917,
                  56624,57105,21148,21584,22043,41593,46673,53120,57105]

permco_list = [permco for permco in pre_permco_list if permco not in␣
 ↪delete_permcos]

names_company = pd.DataFrame(index=permco_list,columns=["Name"])

for i in permco_list:
    names_company.loc[i] = data[data["PERMCO"]==i]["COMNAM"].unique()[-1]
```

1.1.5 A data frame for ask price, shares outstanding and returns data from every company is created

[5]:
```
ask = pd.DataFrame(index=data.index.unique(),columns=data["PERMNO"].unique())

for i in ask.columns:

   ask[i] = data[data["PERMNO"]==i]["ASK"]

ask.columns = names["Name"]

ask = ask[list(names_company["Name"])]

ask = ask.loc[:,~ask.columns.duplicated()]
```

[6]:
```
shares = pd.DataFrame(index=data.index.unique(),columns=data["PERMNO"].unique())

for i in shares.columns:

   shares[i] = data[data["PERMNO"]==i]["SHROUT"]

shares.columns = names["Name"]

shares = shares[list(names_company["Name"])]

shares = shares.loc[:,~shares.columns.duplicated()]

### million of shares outstanding

shares = shares/1000000
```

[ ]:
```
returns = pd.DataFrame(index=data.index.unique(),columns=data["PERMNO"].
 ↪unique())
```

```
for i in returns.columns:

   returns[i] = data[data["PERMNO"]==i]["RETURNS"]

returns.columns = names["Name"]

returns = returns[list(names_company["Name"])]

returns = returns.loc[:,~returns.columns.duplicated()]
```

1.1.6 Market cap is calculated and the eligility of every company is determined

[8]:
```
market_cap = ask*shares

eom_market_cap = market_cap.resample("M").pad()

eligibility = pd.DataFrame(index=eom_market_cap.index,columns=eom_market_cap.
 ↪columns)

for m in range(0,len(eom_market_cap)):

    for cap in range(0,len(eom_market_cap.iloc[m])):

        if eom_market_cap.iloc[m,cap] > 5:

            eligibility.iloc[m,cap] = 1

        else:
            eligibility.iloc[m,cap] = np.nan
```

1.1.7 Return data is filtered so it only contains companies that are elegible

[10]:
```
comp_filtered_by_cap = pd.DataFrame(eligibility.sum().
 ↪sort_values(),columns=["Months wirh marketcap > 5"])

comp_filtered_by_cap = comp_filtered_by_cap[comp_filtered_by_cap==0].dropna().
 ↪index

large_caps = [col for col in returns.columns if col not in␣
 ↪list(comp_filtered_by_cap)]

returns = returns[large_caps]
```

1.1.8 Returns are compounder to monthly frequency, and are controlled for companies that are no longer trading

[11]:
```
monthly_returns = returns.resample("M").agg(lambda x: (x + 1).prod() - 1)

active = pd.DataFrame(index=monthly_returns.index,columns=monthly_returns.
 ↪columns)

for m in range(0,len(monthly_returns)):

    for stock in range(0,len(monthly_returns.iloc[m])):

        if monthly_returns.iloc[m,stock] == 0:

            active.iloc[m,stock] = 0

        else:
            active.iloc[m,stock] = 1
```

1.1.9 A price index is constructed

[12]:
```
price_index_ = (1+monthly_returns).cumprod()

price_index = eligibility[large_caps].shift().ffill().fillna(0) * active *␣
 ↪price_index_
```

1.1.10 Data is saved

[ ]:
```
monthly_returns.to_excel("monthly_returns.xlsx")
price_index.to_excel("price_index.xlsx")
```

2_Simulations

June 22, 2022

1 TFG

1.1 With the data ready, portfolios are simulated

[2]:
```
import bt
import yfinance as yf
import pandas as pd
import numpy as np
```

[24]:
```
monthly_returns = pd.read_excel('monthly_returns.
↪xlsx',parse_dates=True,index_col=0)
price_index = pd.read_excel('price_index.xlsx',parse_dates=True,index_col=0)
```

1.1.1 A function is created to simulate portfolios

[19]:
```
def generate_random_portfolios(number_of_portfolios=5,number_of_companies=20):

    random_portfolios = pd.DataFrame(index=monthly_returns.index)

    for portfolio in range(0,number_of_portfolios):

        s = bt.Strategy('s1', [bt.algos.RunMonthly(),
                       bt.algos.SelectRandomly(number_of_companies),
                       bt.algos.WeighEqually(),
                       bt.algos.Rebalance()])

        test = bt.Backtest(s, price_index,initial_capital=10000000)
        res = bt.run(test)

        random_portfolios[portfolio] = res.prices

    return random_portfolios
```

1.1.2 Ten thousand portfolios are simulated

This computations is computationally expensive, and may take more than 12 hours

[27]:
```
number_of_portfolios = 10000

results = generate_random_portfolios(number_of_portfolios=number_of_portfolios)
```

1.1.3 S&P 500 index data is added for reference

[ ]:
```
spy = yf.download("SPY")["Adj Close"].pct_change()

spy = spy["1996":]

spy = spy["1996":].resample("M").agg(lambda x: (x + 1).prod() - 1)

spy[0] = 0
spy[1] = 0

results["SPX"] = (1+spy).cumprod()*100

results.to_excel("Simulations.xlsx")
```

1.1.4 Data is saved for future access

[44]:
```
results = pd.read_excel("Simulations.xlsx",index_col=0)
```

3_Data_testing

June 22, 2022

1 TFG

1.1 In this notebook, the data is used to perform several tests

[1]:
```
import bt
from scipy.stats import norm
import yfinance as yf
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm
import datetime
%matplotlib inline
plt.style.use("default")
import warnings
warnings.filterwarnings("ignore")
```

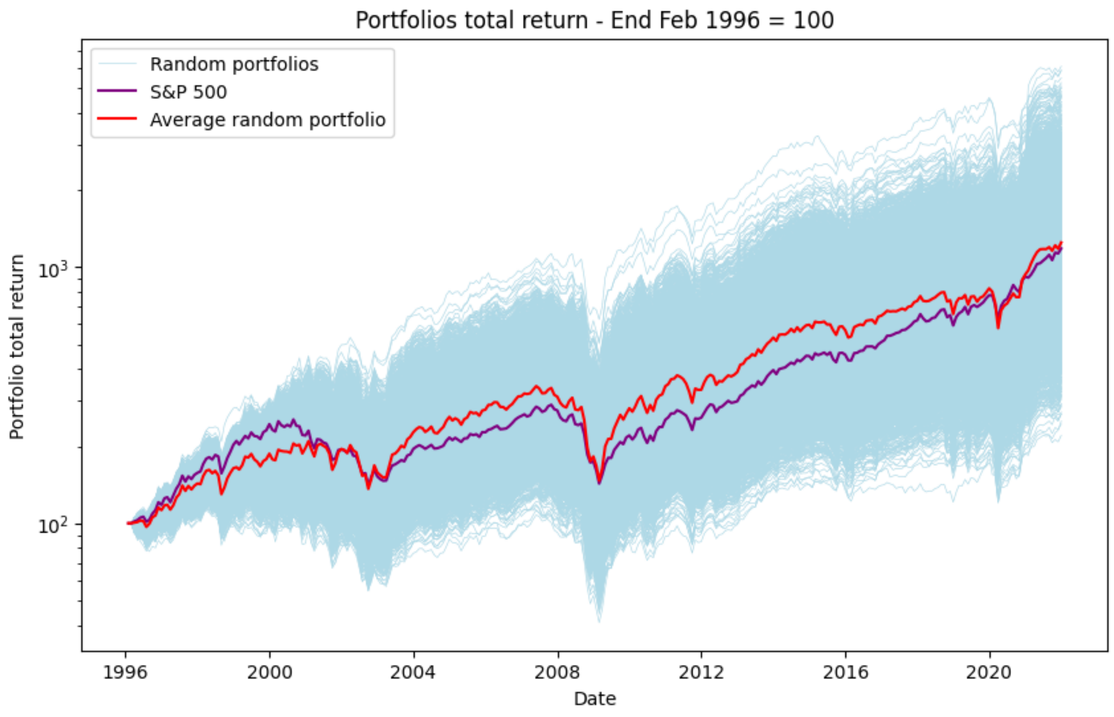
1.1.1 Data processed in the previous notebooks is loaded

[2]:
```
prices = pd.read_excel("price_index.xlsx",index_col=0)
company_returns = pd.read_excel("company_returns.xlsx",index_col=0)

index_portfolios = pd.read_excel("Simulations.xlsx",index_col=0)
index_portfolios["AVERAGE"] = index_portfolios.mean(axis=1)

returns = index_portfolios.pct_change()
```

1.1.2 Below we can see a graph with the evolution of every random portfolio, as well as the average and the S&P 500 index for reference

[13]:
```
fig,ax1 = plt.subplots(1,figsize=(10,6))
ax1.plot(index_portfolios.iloc[:,0],linewidth=0.
↪45,color="lightblue",label="Random portfolios");
ax1.plot(index_portfolios.iloc[:,1:-2],linewidth=0.45,color="lightblue");
ax1.plot(index_portfolios["SPX"],linewidth=1.5,color="purple",label="S&P 500");
ax1.plot(index_portfolios["AVERAGE"],linewidth=1.5,color="red", label="Average␣
↪random portfolio");
```

```
ax1.set_yscale('log')
ax1.set_title("Portfolios total return - End Feb 1996 = 100");
ax1.legend()

ax1.set_xlabel("Date");
ax1.set_ylabel("Portfolio total return");

plt.savefig("Graphs/Portfolios_total_return.png",dpi=300);
```

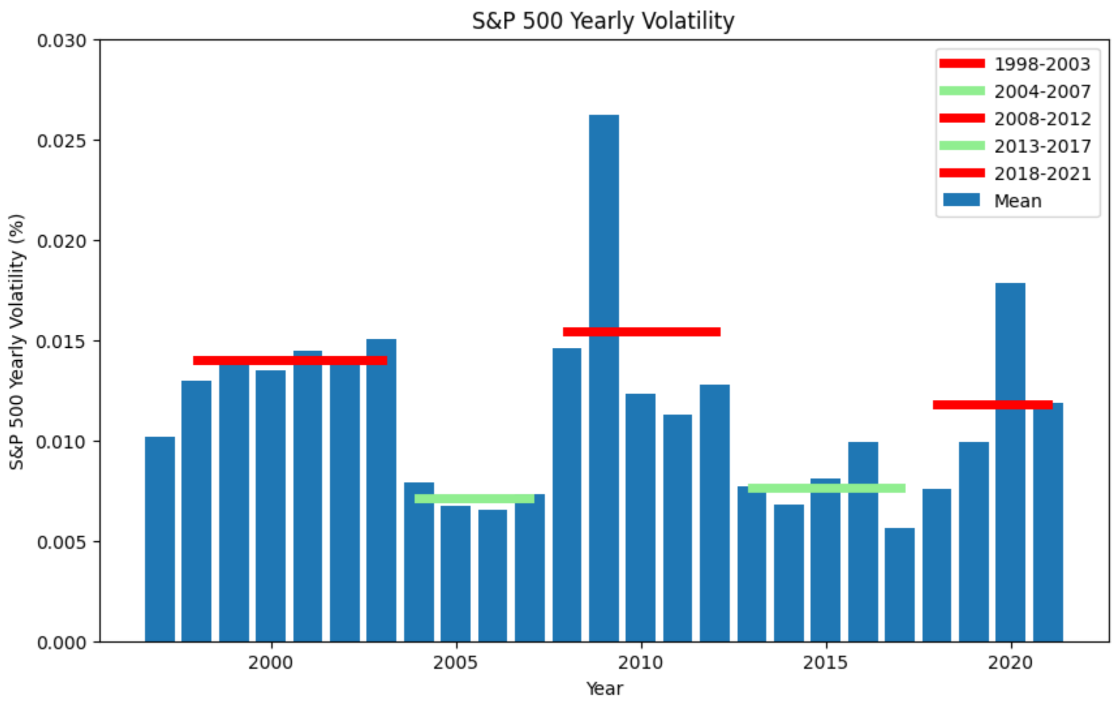


1.1.3 To divide the period into subperiods, yearly volatility is computed to identify regimes

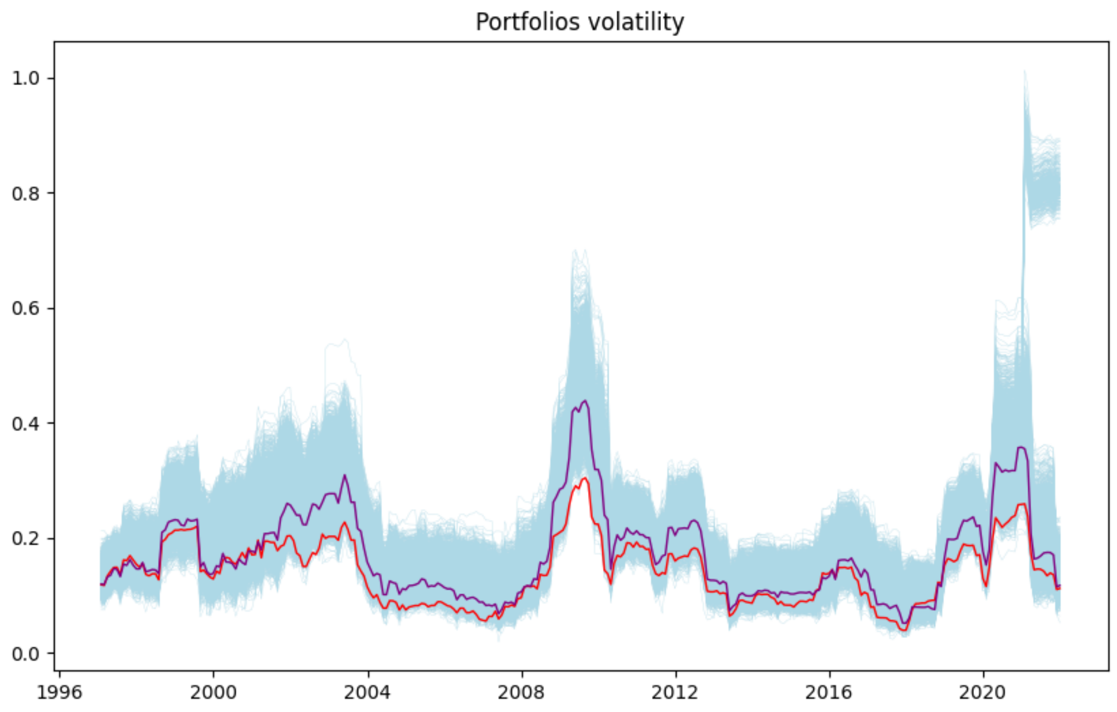
[3]:
```
spxdata = yf.
↪download("SPY",start="1996-03-01",end="2021-12-31",progress=False)["Adj␣
↪Close"].pct_change().dropna()

spxvol = pd.DataFrame(spxdata.rolling(252).std())
spxyearvol = spxvol.groupby(lambda x: x.year)['Adj Close'].agg(['mean'])

spxyearvol["P1"] = np.nan
spxyearvol["P1"].loc[1998:2003]= spxyearvol.loc[1998:2003].mean().values[0]
spxyearvol["P2"] = np.nan
spxyearvol["P2"].loc[2004:2007]= spxyearvol.loc[2004:2007].mean().values[0]
```

```
spxyearvol["P3"] = np.nan
spxyearvol["P3"].loc[2008:2012]= spxyearvol.loc[2008:2012].mean().values[0]
spxyearvol["P4"] = np.nan
spxyearvol["P4"].loc[2013:2017]= spxyearvol.loc[2013:2017].mean().values[0]
spxyearvol["P5"] = np.nan
spxyearvol["P5"].loc[2018:2021]= spxyearvol.loc[2018:2021].mean().values[0]
```

[4]:
```
######## plot spx yearly volatility

fig,ax1 = plt.subplots(1,figsize=(10,6))
ax1.bar(list(spxyearvol.index),spxyearvol["mean"].values,label="Mean");
ax1.plot(spxyearvol.index,spxyearvol["P1"].
 ↪values,label="1998-2003",linewidth=5,color="red");
ax1.plot(spxyearvol.index,spxyearvol["P2"].
 ↪values,label="2004-2007",linewidth=5,color="lightgreen");
ax1.plot(spxyearvol.index,spxyearvol["P3"].
 ↪values,label="2008-2012",linewidth=5,color="red");
ax1.plot(spxyearvol.index,spxyearvol["P4"].
 ↪values,label="2013-2017",linewidth=5,color="lightgreen");
ax1.plot(spxyearvol.index,spxyearvol["P5"].
 ↪values,label="2018-2021",linewidth=5,color="red");
ax1.set_title("S&P 500 Yearly Volatility");
ax1.legend()

ax1.set_ylim(0,0.03)
ax1.set_xlabel("Year");
ax1.set_ylabel("S&P 500 Yearly Volatility (%)");

plt.savefig("Graphs/S&P_yearly_volatility.png",dpi=300);
```



[5]:
```
### volatilidades de los portfolios en rolling periods de 12 meses

fig,ax1 = plt.subplots(1,figsize=(10,6))
ax1.plot(((returns.rolling(12).std() * np.sqrt(12)).iloc[:,:-2]),linewidth=0.
 ↪2,color="lightblue");
ax1.plot(((returns.rolling(12).std() * np.
 ↪sqrt(12))["SPX"]),linewidth=1,color="red");
ax1.plot(((returns.rolling(12).std() * np.
 ↪sqrt(12))["AVERAGE"]),linewidth=1,color="purple");
plt.title("Portfolios volatility");
```

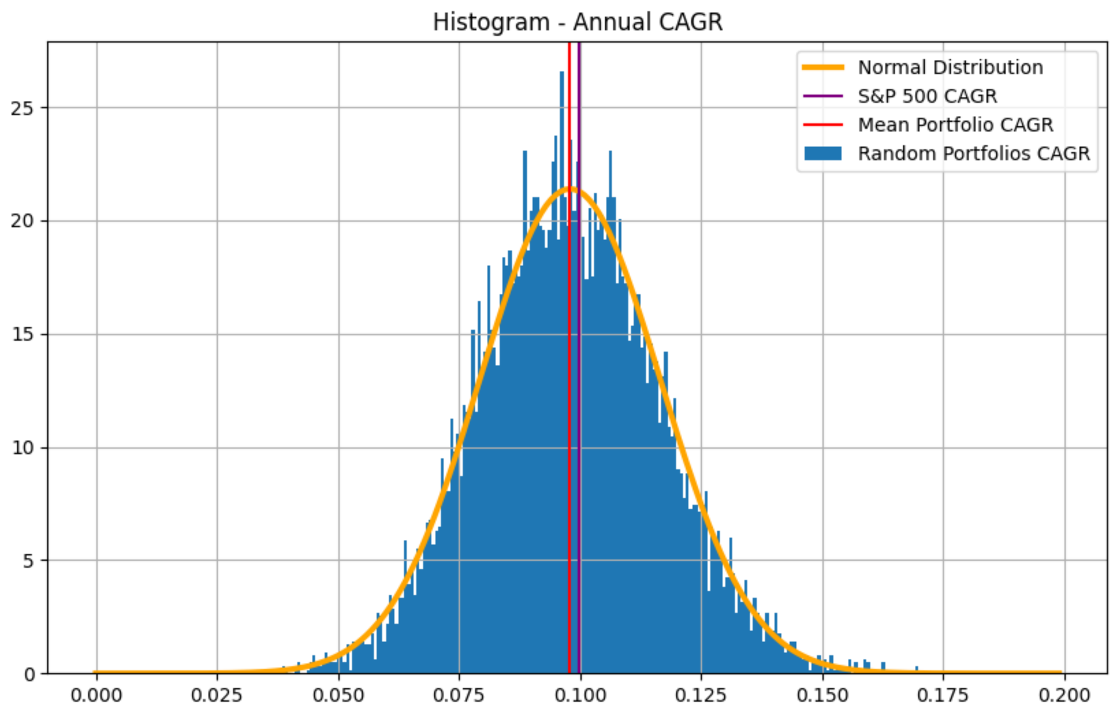


Several metrics like the volatility and CAGR

[6]:
```
ann_rets = (index_portfolios.iloc[-1] /  index_portfolios.iloc[0])**(12/
↪(12*26)) - 1

metrics = pd.DataFrame(ann_rets,columns=["Annual Return"])

metrics["Annual Volatility"] = index_portfolios.pct_change().dropna().std() *␣
↪np.sqrt(12)

metrics["Return / Volatility"] = metrics["Annual Return"] / metrics["Annual␣
↪Volatility"]

return_position_spx = metrics["Annual Return"].sort_values(ascending=False).
↪index.get_loc("SPX")

volatility_position_spx = metrics["Annual Volatility"].
↪sort_values(ascending=False).index.get_loc("SPX")

return_vol_position_spx = metrics["Return / Volatility"].
↪sort_values(ascending=False).index.get_loc("SPX")
```

1.1.4 Histogram of the portfolios' CAGR

[19]:
```
fig,ax1 = plt.subplots(1,figsize=(10,6))

x = np.arange(0, 0.2, 0.001)

mu, std = norm.fit(metrics["Annual Return"])

ax1.plot(x, norm.pdf(x, mu, std),color="orange",linewidth=3,label="Normal␣
↪Distribution")

ax1.hist(metrics["Annual␣
↪Return"],bins=222,cumulative=False,density=True,label="Random Portfolios␣
↪CAGR");
ax1.grid(True)
ax1.axvline(metrics["Annual Return"]["SPX"],color="purple",linewidth=1.
↪5,label="S&P 500 CAGR");
ax1.axvline(metrics["Annual Return"][0:10000].mean(),color="red",linewidth=1.
↪5,label="Mean Portfolio CAGR");
ax1.set_title("Histogram - Annual CAGR");
ax1.legend();

plt.savefig("Graphs/Histogram_CAGR.png")
```



1.1.5 Now, Fama-French data is loaded and processed

[3]:
```
ff = pd.read_csv("F-F_Research_Data_5_Factors_2x3.
↪CSV",skiprows=3,index_col=0,skipfooter=63,engine="python",
                 parse_dates=True,infer_datetime_format=True)

ff.index = pd.to_datetime(ff.index, format= '%Y%m')
ff.index = ff.index + pd.offsets.MonthEnd()
ff = ff["1996-03":"2021"]
ff = ff/100

ff.rename(columns={"Mkt-RF":"mkt_excess"}, inplace=True)
```

1.1.6 Regressions are performed for every random portfolio, for every period to compute every factor

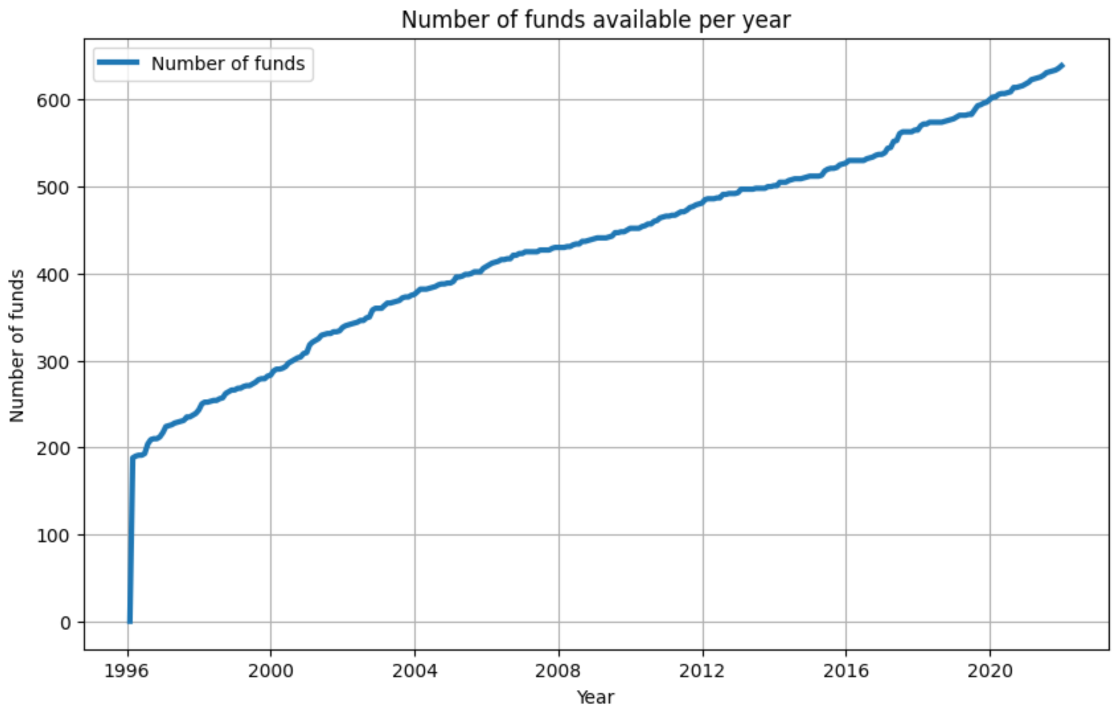
[ ]:
```
from statsmodels.regression.rolling import RollingOLS

factorsp1 = pd.DataFrame(columns=returns[range(0,10000)].
↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


for i in returns[range(0,10000)].columns:

        endog = returns[i]["1998":"2003"].dropna() - ff["RF"]["1998":"2003"]
        exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["1998":"2003"]
        exog = sm.add_constant(exog)

        mod = sm.regression.linear_model.OLS(endog,exog)
        rres = mod.fit()

        factorsp1.loc["Alpha"][i] = rres.params["const"]
        factorsp1.loc["Beta"][i] = rres.params["mkt_excess"]
        factorsp1.loc["SMB"][i] = rres.params["SMB"]
        factorsp1.loc["HML"][i] = rres.params["HML"]
        factorsp1.loc["RMW"][i] = rres.params["RMW"]
        factorsp1.loc["CMA"][i] = rres.params["CMA"]
        factorsp1.loc["R2"][i] = rres.rsquared
        factorsp1.loc["P-value Alpha"][i] = rres.pvalues["const"]

factorsp2 = pd.DataFrame(columns=returns[range(0,10000)].
↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


for i in returns[range(0,10000)].columns:

        endog = returns[i]["2004":"2007"].dropna() - ff["RF"]["2004":"2007"]
        exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["2004":"2007"]
        exog = sm.add_constant(exog)

        mod = sm.regression.linear_model.OLS(endog,exog)
        rres = mod.fit()

        factorsp2.loc["Alpha"][i] = rres.params["const"]
        factorsp2.loc["Beta"][i] = rres.params["mkt_excess"]
        factorsp2.loc["SMB"][i] = rres.params["SMB"]
        factorsp2.loc["HML"][i] = rres.params["HML"]
        factorsp2.loc["RMW"][i] = rres.params["RMW"]
        factorsp2.loc["CMA"][i] = rres.params["CMA"]
        factorsp2.loc["R2"][i] = rres.rsquared
        factorsp2.loc["P-value Alpha"][i] = rres.pvalues["const"]


factorsp3 = pd.DataFrame(columns=returns[range(0,10000)].
 ↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


for i in returns[range(0,10000)].columns:

        endog = returns[i]["2008":"2012"].dropna() - ff["RF"]["2008":"2012"]
        exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["2008":"2012"]
        exog = sm.add_constant(exog)

        mod = sm.regression.linear_model.OLS(endog,exog)
        rres = mod.fit()

        factorsp3.loc["Alpha"][i] = rres.params["const"]
        factorsp3.loc["Beta"][i] = rres.params["mkt_excess"]
        factorsp3.loc["SMB"][i] = rres.params["SMB"]
        factorsp3.loc["HML"][i] = rres.params["HML"]
        factorsp3.loc["RMW"][i] = rres.params["RMW"]
        factorsp3.loc["CMA"][i] = rres.params["CMA"]
        factorsp3.loc["R2"][i] = rres.rsquared
        factorsp3.loc["P-value Alpha"][i] = rres.pvalues["const"]


factorsp4 = pd.DataFrame(columns=returns[range(0,10000)].
 ↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


for i in returns[range(0,10000)].columns:

        endog = returns[i]["2013":"2017"].dropna() - ff["RF"]["2013":"2017"]
        exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["2013":"2017"]
        exog = sm.add_constant(exog)

        mod = sm.regression.linear_model.OLS(endog,exog)
        rres = mod.fit()

        factorsp4.loc["Alpha"][i] = rres.params["const"]
        factorsp4.loc["Beta"][i] = rres.params["mkt_excess"]
        factorsp4.loc["SMB"][i] = rres.params["SMB"]
        factorsp4.loc["HML"][i] = rres.params["HML"]
        factorsp4.loc["RMW"][i] = rres.params["RMW"]
        factorsp4.loc["CMA"][i] = rres.params["CMA"]
        factorsp4.loc["R2"][i] = rres.rsquared
        factorsp4.loc["P-value Alpha"][i] = rres.pvalues["const"]


factorsp5 = pd.DataFrame(columns=returns[range(0,10000)].
 ↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


for i in returns[range(0,10000)].columns:

        endog = returns[i]["2018":"2021"].dropna() - ff["RF"]["2018":"2021"]
        exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["2018":"2021"]
        exog = sm.add_constant(exog)

        mod = sm.regression.linear_model.OLS(endog,exog)
        rres = mod.fit()

        factorsp5.loc["Alpha"][i] = rres.params["const"]
        factorsp5.loc["Beta"][i] = rres.params["mkt_excess"]
        factorsp5.loc["SMB"][i] = rres.params["SMB"]
        factorsp5.loc["HML"][i] = rres.params["HML"]
        factorsp5.loc["RMW"][i] = rres.params["RMW"]
        factorsp5.loc["CMA"][i] = rres.params["CMA"]
        factorsp5.loc["R2"][i] = rres.rsquared
        factorsp5.loc["P-value Alpha"][i] = rres.pvalues["const"]

factorsp1.to_csv("Computations/factorsp1.csv")
factorsp2.to_csv("Computations/factorsp2.csv")
factorsp3.to_csv("Computations/factorsp3.csv")
factorsp4.to_csv("Computations/factorsp4.csv")
factorsp5.to_csv("Computations/factorsp5.csv")
```

1.1.7 Mutual fund data is loaded

```
[ ]: funds = pd.read_excel("Reuters Mutual Fund Data.xlsx")

     tickers = list(funds[funds["Index Based Fund"]!=1]["NASDAQ Ticker"].dropna())

     data_f = yf.download(tickers,start="1995-12-31",end="2022-01-01",
                          progress=False,threads=4,periods="1mo")["Adj Close"]
     data_f = data_f.resample("M").pad().pct_change()
```

1.1.8 To inspect the data, the number of funds available is computed

```
[81]: fig,ax1 = plt.subplots(1,figsize=(10,6))

      ax1.plot(data_f.count(axis=1),label="Number of funds",linewidth=3)
      ax1.set_xlabel("Year");
      ax1.set_ylabel("Number of funds");
      ax1.grid(True);
      ax1.legend();
      ax1.set_title("Number of funds available per year");
      plt.savefig("Graphs/numfunds.png",dpi=300);
```



1.1.9 As done before, regressions are performed

```
[ ]: factorsfp1 = pd.DataFrame(columns=data_f.
      ↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


     for i in data_f.columns:

             endog = data_f[i]["1998":"2003"].dropna() - ff["RF"]["1998":"2003"]
             exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["1998":"2003"]
             exog = sm.add_constant(exog)

             mod = sm.regression.linear_model.OLS(endog,exog)
             rres = mod.fit()

             factorsfp1.loc["Alpha"][i]  = rres.params["const"]
             factorsfp1.loc["Beta"][i]  = rres.params["mkt_excess"]
             factorsfp1.loc["SMB"][i]  = rres.params["SMB"]
             factorsfp1.loc["HML"][i]  = rres.params["HML"]
             factorsfp1.loc["RMW"][i]  = rres.params["RMW"]
             factorsfp1.loc["CMA"][i]  = rres.params["CMA"]
             factorsfp1.loc["R2"][i]  = rres.rsquared
             factorsfp1.loc["P-value Alpha"][i]  = rres.pvalues["const"]

     factorsfp2 = pd.DataFrame(columns=data_f.
      ↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


     for i in data_f.columns:

             endog = data_f[i]["2004":"2007"].dropna() - ff["RF"]["2004":"2007"]
             exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["2004":"2007"]
             exog = sm.add_constant(exog)

             mod = sm.regression.linear_model.OLS(endog,exog)
             rres = mod.fit()

             factorsfp2.loc["Alpha"][i]  = rres.params["const"]
             factorsfp2.loc["Beta"][i]  = rres.params["mkt_excess"]
             factorsfp2.loc["SMB"][i]  = rres.params["SMB"]
             factorsfp2.loc["HML"][i]  = rres.params["HML"]
             factorsfp2.loc["RMW"][i]  = rres.params["RMW"]
             factorsfp2.loc["CMA"][i]  = rres.params["CMA"]
             factorsfp2.loc["R2"][i]  = rres.rsquared
             factorsfp2.loc["P-value Alpha"][i]  = rres.pvalues["const"]
```

```
factorsfp3 = pd.DataFrame(columns=data_f.
 ↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


for i in data_f.columns:

        endog = data_f[i]["2008":"2012"].dropna() - ff["RF"]["2008":"2012"]
        exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["2008":"2012"]
        exog = sm.add_constant(exog)

        mod = sm.regression.linear_model.OLS(endog,exog)
        rres = mod.fit()

        factorsfp3.loc["Alpha"][i] = rres.params["const"]
        factorsfp3.loc["Beta"][i] = rres.params["mkt_excess"]
        factorsfp3.loc["SMB"][i] = rres.params["SMB"]
        factorsfp3.loc["HML"][i] = rres.params["HML"]
        factorsfp3.loc["RMW"][i] = rres.params["RMW"]
        factorsfp3.loc["CMA"][i] = rres.params["CMA"]
        factorsfp3.loc["R2"][i] = rres.rsquared
        factorsfp3.loc["P-value Alpha"][i] = rres.pvalues["const"]


factorsfp4 = pd.DataFrame(columns=data_f.
 ↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


for i in data_f.columns:

        endog = data_f[i]["2013":"2017"].dropna() - ff["RF"]["2013":"2017"]
        exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["2013":"2017"]
        exog = sm.add_constant(exog)

        mod = sm.regression.linear_model.OLS(endog,exog)
        rres = mod.fit()

        factorsfp4.loc["Alpha"][i] = rres.params["const"]
        factorsfp4.loc["Beta"][i] = rres.params["mkt_excess"]
        factorsfp4.loc["SMB"][i] = rres.params["SMB"]
        factorsfp4.loc["HML"][i] = rres.params["HML"]
        factorsfp4.loc["RMW"][i] = rres.params["RMW"]
        factorsfp4.loc["CMA"][i] = rres.params["CMA"]
        factorsfp4.loc["R2"][i] = rres.rsquared
        factorsfp4.loc["P-value Alpha"][i] = rres.pvalues["const"]
```

```
factorsfp5 = pd.DataFrame(columns=data_f.
 ↪columns,index=["Alpha","Beta","SMB","HML","RMW","CMA","R2","P-value Alpha"])


for i in data_f.columns:

        endog = data_f[i]["2018":"2021"].dropna() - ff["RF"]["2018":"2021"]
        exog = ff[["mkt_excess","SMB","HML","RMW","CMA"]]["2018":"2021"]
        exog = sm.add_constant(exog)

        mod = sm.regression.linear_model.OLS(endog,exog)
        rres = mod.fit()

        factorsfp5.loc["Alpha"][i] = rres.params["const"]
        factorsfp5.loc["Beta"][i] = rres.params["mkt_excess"]
        factorsfp5.loc["SMB"][i] = rres.params["SMB"]
        factorsfp5.loc["HML"][i] = rres.params["HML"]
        factorsfp5.loc["RMW"][i] = rres.params["RMW"]
        factorsfp5.loc["CMA"][i] = rres.params["CMA"]
        factorsfp5.loc["R2"][i] = rres.rsquared
        factorsfp5.loc["P-value Alpha"][i] = rres.pvalues["const"]


factorsfp1.to_csv("Computations/factorsfp1.csv")
factorsfp2.to_csv("Computations/factorsfp2.csv")
factorsfp3.to_csv("Computations/factorsfp3.csv")
factorsfp4.to_csv("Computations/factorsfp4.csv")
factorsfp5.to_csv("Computations/factorsfp5.csv")
```

1.1.10 To avoid repeting this computationally expensive regressions, we can just load the previously processed data

```
[3]: factorsfp1 = pd.read_csv("Computations/factorsfp1.
 ↪csv",index_col=0,parse_dates=True)
factorsfp2 = pd.read_csv("Computations/factorsfp2.
 ↪csv",index_col=0,parse_dates=True)
factorsfp3 = pd.read_csv("Computations/factorsfp3.
 ↪csv",index_col=0,parse_dates=True)
factorsfp4 = pd.read_csv("Computations/factorsfp4.
 ↪csv",index_col=0,parse_dates=True)
factorsfp5 = pd.read_csv("Computations/factorsfp5.
 ↪csv",index_col=0,parse_dates=True)


factorsp1 = pd.read_csv("Computations/factorsp1.csv",index_col=0)
factorsp2 = pd.read_csv("Computations/factorsp2.csv",index_col=0)
```

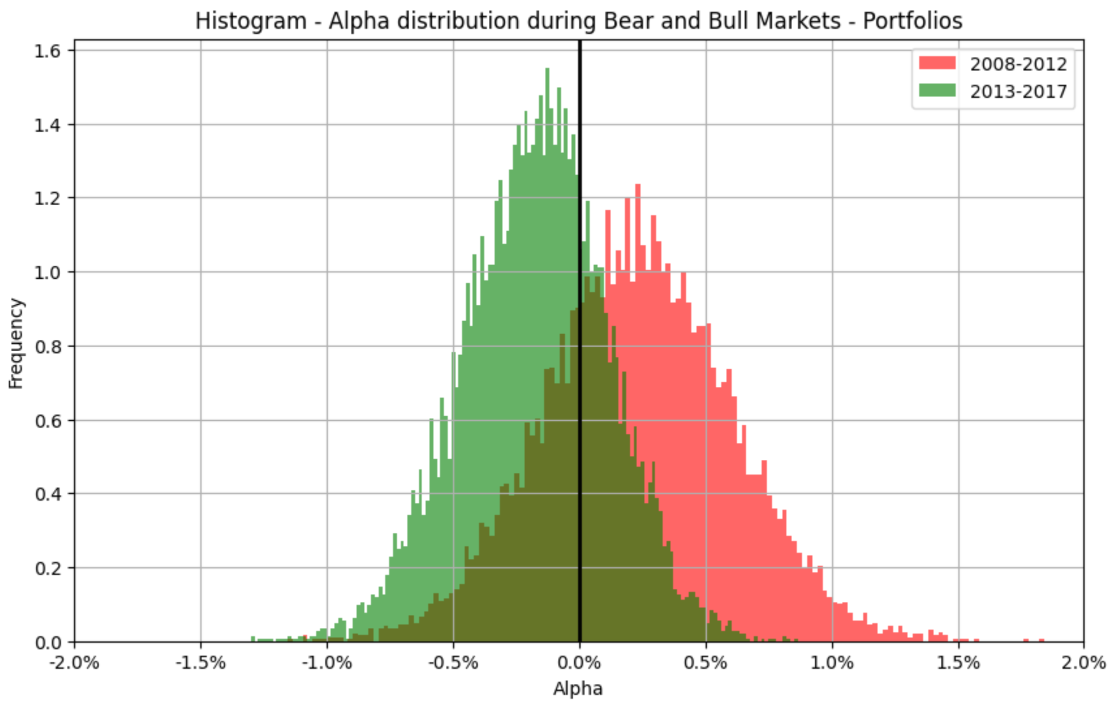
```
factorsp3 = pd.read_csv("Computations/factorsp3.csv",index_col=0)
factorsp4 = pd.read_csv("Computations/factorsp4.csv",index_col=0)
factorsp5 = pd.read_csv("Computations/factorsp5.csv",index_col=0)
```

1.2 Now, we can start with the tests

1.2.1 First we plot the alphas depending of the market regime, for portfolios and funds

After looking at the data, we decide to focus on the bear market caused by the Great Financial Crisis, and the bull market of its subsequent recovery, so from now on those are going to be the periods studied.

[9]:
```
import matplotlib.ticker as mtick

fig,ax1 = plt.subplots(1,figsize=(10,6))

plt.hist(factorsp3.loc["Alpha"]*100,bins=150,alpha=0.
 ↪6,label="2008-2012",color="red",density=True);
plt.hist(factorsp4.loc["Alpha"]*100,bins=150,alpha=0.
 ↪6,label="2013-2017",color="green",density=True);
plt.grid(True);
plt.legend();
plt.xlim((-2,2))
plt.axvline(0,color="black",linewidth=2);
plt.title("Histogram - Alpha distribution during Bear and Bull Markets -␣
 ↪Portfolios");
plt.xlabel("Alpha");
plt.ylabel("Frequency");

fmt = '%.1f%%' # Format you want the ticks, e.g. '40%'
xticks = mtick.FormatStrFormatter(fmt)
ax1.xaxis.set_major_formatter(xticks)

plt.savefig("Graphs/hist_alpha_portfolios.png",dpi=300);
```

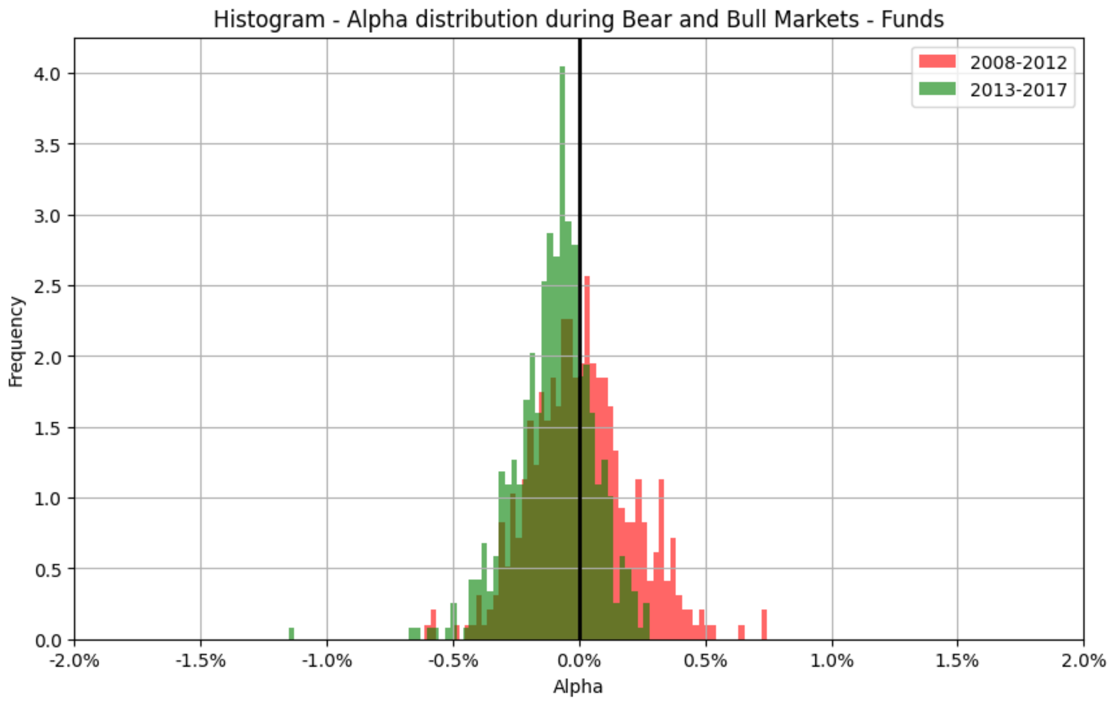


Descriptive table of mean and std of the previous graph

```
[7]: describe_histogram_r2 = pd.
      ↪DataFrame(index=["mean","std"],columns=["2008-2012","2013-2017"])
     describe_histogram_r2["2008-2012"] = factorsfp3.loc["Alpha"].
      ↪describe()[["mean","std"]]
     describe_histogram_r2["2013-2017"] = factorsfp4.loc["Alpha"].
      ↪describe()[["mean","std"]]
     describe_histogram_r2 = describe_histogram_r2.round(5) * 100
     describe_histogram_r2
```

```
[7]:       2008-2012  2013-2017
     mean      0.012     -0.099
     std       0.203      0.159
```

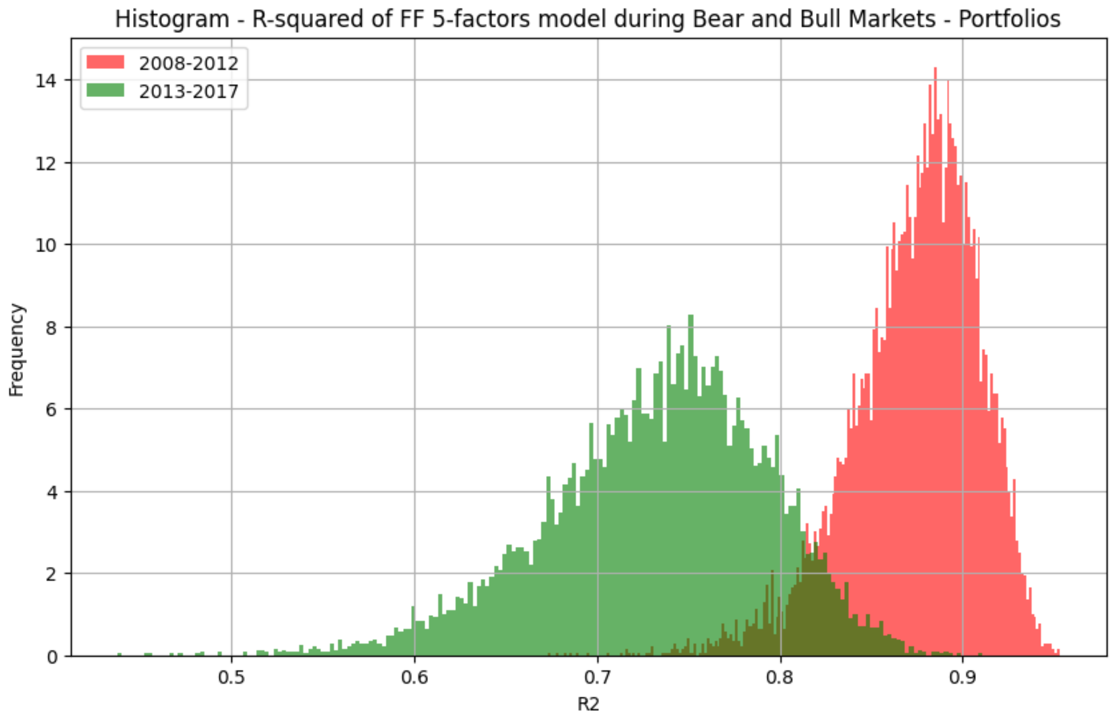
```
[10]: fig,ax1 = plt.subplots(1,figsize=(10,6))

      plt.hist(factorsfp3.loc["Alpha"]*100,bins=60,alpha=0.
       ↪6,label="2008-2012",color="red",density=True);
      plt.hist(factorsfp4.loc["Alpha"]*100,bins=60,alpha=0.
       ↪6,label="2013-2017",color="green",density=True);
      plt.grid(True);
      plt.legend();
      plt.xlim((-2,2))
      plt.axvline(0,color="black",linewidth=2);
      plt.title("Histogram - Alpha distribution during Bear and Bull Markets -␣
       ↪Funds");
      plt.xlabel("Alpha");
      plt.ylabel("Frequency");


      fmt = '%.1f%%'
      xticks = mtick.FormatStrFormatter(fmt)
      ax1.xaxis.set_major_formatter(xticks)

      plt.savefig("Graphs/hist_alpha_funds.png",dpi=300);
```



1.2.2 To try to explain the differences, the R-Squared is also plotted

[67]:
```
fig,ax1 = plt.subplots(1,figsize=(10,6))

plt.hist(factorsp3.loc["R2"],bins=200,alpha=0.
 ↪6,label="2008-2012",color="red",density=True);
plt.hist(factorsp4.loc["R2"],bins=200,alpha=0.
 ↪6,label="2013-2017",color="green",density=True);
plt.grid(True);
plt.legend();
ax1.set_ylabel("Frequency");
```

```
ax1.set_xlabel("R2");
plt.title("Histogram - R-squared of FF 5-factors model during Bear and Bull␣
↪Markets - Portfolios");
plt.savefig("Graphs/hist_r2_ff_portfolios.png",dpi=300);
```



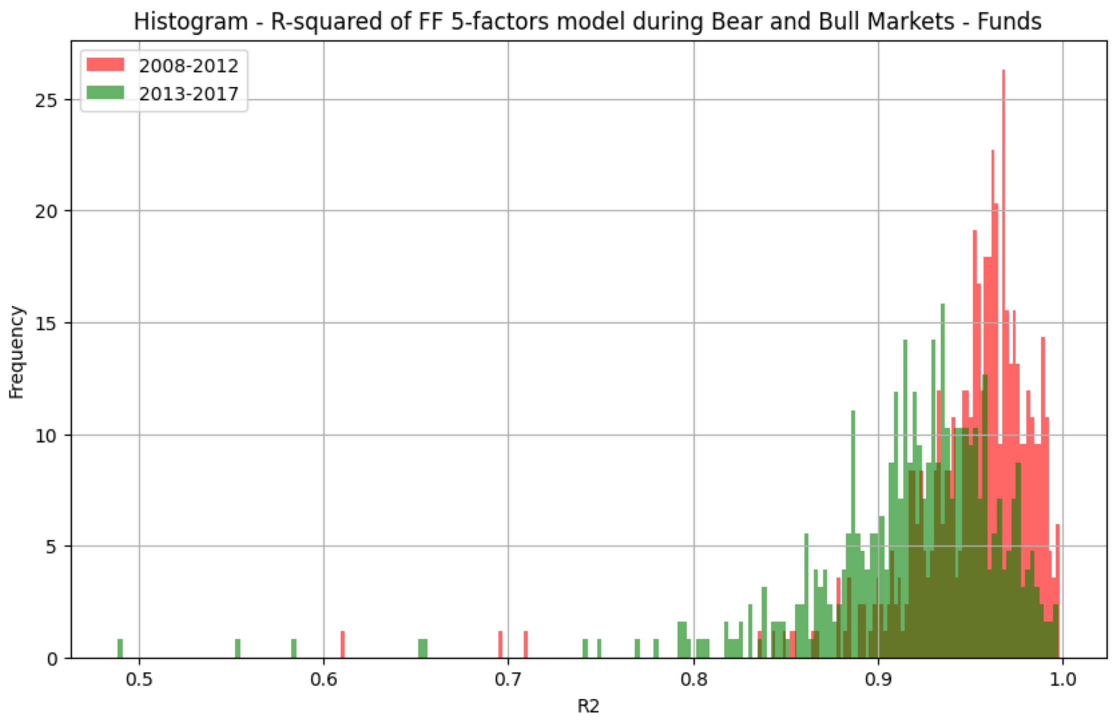
Descriptive table of mean and std of the previous graph

[28]:
```
describe_histogram_r2 = pd.
↪DataFrame(index=["mean","std"],columns=["2008-2012","2013-2017"])
describe_histogram_r2["2008-2012"] = factorsp3.loc["R2"].
↪describe()[["mean","std"]]
describe_histogram_r2["2013-2017"] = factorsp4.loc["R2"].
↪describe()[["mean","std"]]
describe_histogram_r2 = describe_histogram_r2.round(4)
describe_histogram_r2
```

[28]:
```
      2008-2012  2013-2017
mean     0.8740     0.7330
std      0.0349     0.0605
```

[68]:
```
fig,ax1 = plt.subplots(1,figsize=(10,6))

plt.hist(factorsfp3.loc["R2"],bins=200,alpha=0.
↪6,label="2008-2012",color="red",density=True);
```

```
plt.hist(factorsfp4.loc["R2"],bins=200,alpha=0.
 ↪6,label="2013-2017",color="green",density=True);
plt.grid(True);
plt.legend();
ax1.set_ylabel("Frequency");
ax1.set_xlabel("R2");
#plt.xlim((-0.02,0.02))
#plt.axvline(0,color="black",linewidth=2);
plt.title("Histogram - R-squared of FF 5-factors model during Bear and Bull␣
 ↪Markets - Funds");
plt.savefig("Graphs/hist_r2_ff_funds.png",dpi=300);
```



[29]:
```
describe_histogram_r2 = pd.
 ↪DataFrame(index=["mean","std"],columns=["2008-2012","2013-2017"])
describe_histogram_r2["2008-2012"] = factorsfp3.loc["R2"].
 ↪describe()[["mean","std"]]
describe_histogram_r2["2013-2017"] = factorsfp4.loc["R2"].
 ↪describe()[["mean","std"]]
describe_histogram_r2 = describe_histogram_r2.round(4)
describe_histogram_r2
```

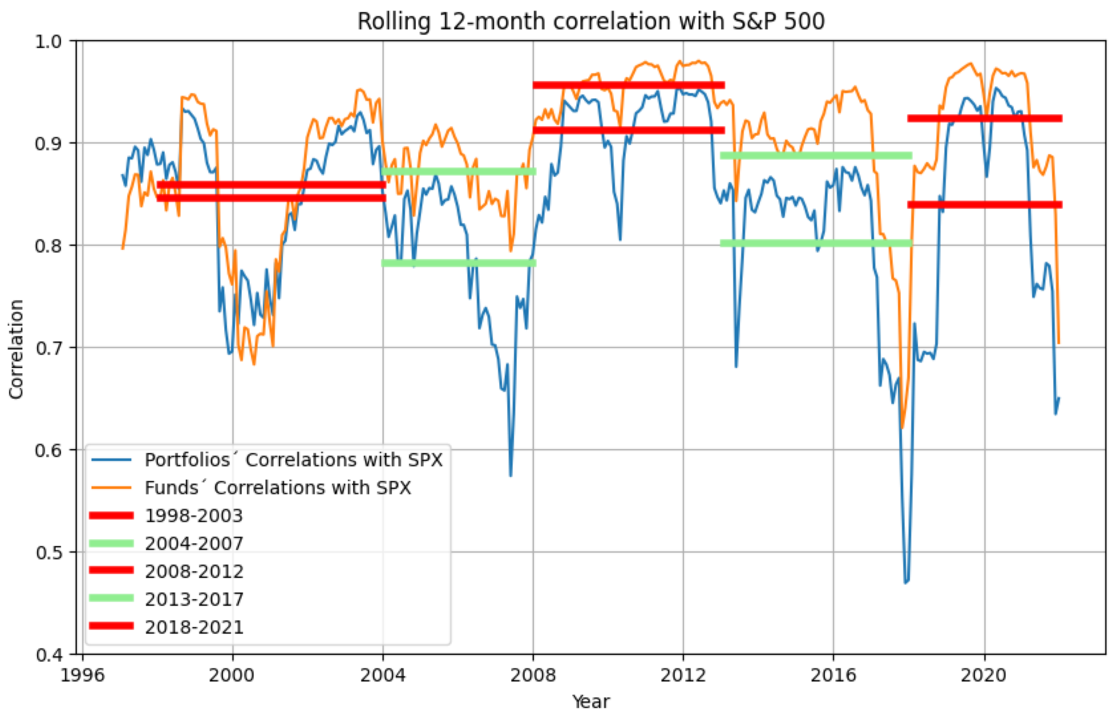
[29]:
```
      2008-2012  2013-2017
mean     0.9514     0.9179
```

```
std      0.0372     0.0543
```

[13]:
```
#### calculo correlaciones de carteras simuladas y fondos con sp500

correlations_with_spx = returns.iloc[:,:-2].rolling(12).corr(returns.iloc[:,-2])


correlations_with_spx["P1"] = np.nan
correlations_with_spx["P1"]["1998":"2003"]= correlations_with_spx["1998":
 ↪"2003"].mean(axis=1).mean()
correlations_with_spx["P2"] = np.nan
correlations_with_spx["P2"]["2004":"2007"]= correlations_with_spx["2004":
 ↪"2007"].mean(axis=1).mean()
correlations_with_spx["P3"] = np.nan
correlations_with_spx["P3"]["2008":"2012"]= correlations_with_spx["2008":
 ↪"2012"].mean(axis=1).mean()
correlations_with_spx["P4"] = np.nan
correlations_with_spx["P4"]["2013":"2017"]= correlations_with_spx["2013":
 ↪"2017"].mean(axis=1).mean()
correlations_with_spx["P5"] = np.nan
correlations_with_spx["P5"]["2018":"2021"]= correlations_with_spx["2018":
 ↪"2021"].mean(axis=1).mean()

correlations_with_spx = correlations_with_spx.dropna(axis=1,how="all")


data_f_corr = data_f
data_f_corr["SPX"] = returns["SPX"]
data_f_corr


correlations_with_spx_funds = data_f_corr.iloc[:,:-1].rolling(12).
 ↪corr(data_f_corr.iloc[:,-1])


correlations_with_spx_funds["P1"] = np.nan
correlations_with_spx_funds["P1"]["1998":"2003"]=␣
 ↪correlations_with_spx_funds["1998":"2003"].mean(axis=1).mean()
correlations_with_spx_funds["P2"] = np.nan
correlations_with_spx_funds["P2"]["2004":"2007"]=␣
 ↪correlations_with_spx_funds["2004":"2007"].mean(axis=1).mean()
correlations_with_spx_funds["P3"] = np.nan
correlations_with_spx_funds["P3"]["2008":"2012"]=␣
 ↪correlations_with_spx_funds["2008":"2012"].mean(axis=1).mean()
correlations_with_spx_funds["P4"] = np.nan
```

```
correlations_with_spx_funds["P4"]["2013":"2017"]=␣
↪correlations_with_spx_funds["2013":"2017"].mean(axis=1).mean()
correlations_with_spx_funds["P5"] = np.nan
correlations_with_spx_funds["P5"]["2018":"2021"]=␣
↪correlations_with_spx_funds["2018":"2021"].mean(axis=1).mean()

correlations_with_spx_funds = correlations_with_spx_funds.
↪dropna(axis=1,how="all")


fig,ax1 = plt.subplots(1,figsize=(10,6))
ax1.plot(correlations_with_spx.mean(axis=1),label="Portfolios´ Correlations␣
↪with SPX");
ax1.plot(correlations_with_spx_funds.mean(axis=1),label="Funds´ Correlations␣
↪with SPX");
ax1.set_title("Rolling 12-month correlation with S&P 500")


ax1.plot(correlations_with_spx.index,correlations_with_spx["P1"].
↪values,linewidth=4,color="red");
ax1.plot(correlations_with_spx.index,correlations_with_spx["P2"].
↪values,linewidth=4,color="lightgreen");
ax1.plot(correlations_with_spx.index,correlations_with_spx["P3"].
↪values,linewidth=4,color="red");
ax1.plot(correlations_with_spx.index,correlations_with_spx["P4"].
↪values,linewidth=4,color="lightgreen");
ax1.plot(correlations_with_spx.index,correlations_with_spx["P5"].
↪values,linewidth=4,color="red");

ax1.plot(correlations_with_spx_funds.index,correlations_with_spx_funds["P1"].
↪values,label="1998-2003",linewidth=4,color="red");
ax1.plot(correlations_with_spx_funds.index,correlations_with_spx_funds["P2"].
↪values,label="2004-2007",linewidth=4,color="lightgreen");
ax1.plot(correlations_with_spx_funds.index,correlations_with_spx_funds["P3"].
↪values,label="2008-2012",linewidth=4,color="red");
ax1.plot(correlations_with_spx_funds.index,correlations_with_spx_funds["P4"].
↪values,label="2013-2017",linewidth=4,color="lightgreen");
ax1.plot(correlations_with_spx_funds.index,correlations_with_spx_funds["P5"].
↪values,label="2018-2021",linewidth=4,color="red");

ax1.legend();
ax1.set_ylim((0.4,1))

ax1.set_xlabel("Year")
ax1.set_ylabel("Correlation")
ax1.grid(True)
```

```
plt.savefig("Graphs/correlation_with_spx.png",dpi=300)
```



1.2.3 Now, regressions are again computed, but in this case split by periods, for funds and portfolios

```
[ ]: alpha5f_portfoliosp1 = pd.DataFrame(columns=returns["1998":"2003"].
     ↪dropna(axis=1).columns,index=returns.index)["1998":"2003"]
     beta5f_portfoliosp1 = pd.DataFrame(columns=returns["1998":"2003"].
     ↪dropna(axis=1).columns,index=returns.index)["1998":"2003"]

     for i in alpha5f_portfoliosp1.columns:
             ff_ = ff.copy()["1998":"2003"]
             ff_[i] = returns[i]["1998":"2003"]
             ff_["Excess"] = ff_[i] - ff_["RF"]
             mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
     ↪CMA",window=12,data=ff_)
             rres = mod.fit()
             alpha5f_portfoliosp1[i] = rres.params["Intercept"]
             beta5f_portfoliosp1[i] = rres.params["mkt_excess"]


     alpha5f_portfoliosp2 = pd.DataFrame(columns=returns["2004":"2007"].
     ↪dropna(axis=1).columns,index=returns.index)["2004":"2007"]
```

```
beta5f_portfoliosp2 = pd.DataFrame(columns=returns["2004":"2007"].
↪dropna(axis=1).columns,index=returns.index)["2004":"2007"]

for i in alpha5f_portfoliosp2.columns:
        ff_ = ff.copy()["2004":"2007"]
        ff_[i] = returns[i]["2004":"2007"]
        ff_["Excess"] = ff_[i] - ff_["RF"]
        mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
↪CMA",window=12,data=ff_)
        rres = mod.fit()
        alpha5f_portfoliosp2[i] = rres.params["Intercept"]
        beta5f_portfoliosp2[i] = rres.params["mkt_excess"]

alpha5f_portfoliosp3 = pd.DataFrame(columns=returns["2008":"2012"].
↪dropna(axis=1).columns,index=returns.index)["2008":"2012"]
beta5f_portfoliosp3 = pd.DataFrame(columns=returns["2008":"2012"].
↪dropna(axis=1).columns,index=returns.index)["2008":"2012"]

for i in alpha5f_portfoliosp3.columns:
        ff_ = ff.copy()["2008":"2012"]
        ff_[i] = returns[i]["2008":"2012"]
        ff_["Excess"] = ff_[i] - ff_["RF"]
        mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
↪CMA",window=12,data=ff_)
        rres = mod.fit()
        alpha5f_portfoliosp3[i] = rres.params["Intercept"]
        beta5f_portfoliosp3[i] = rres.params["mkt_excess"]

alpha5f_portfoliosp4 = pd.DataFrame(columns=returns["2013":"2017"].
↪dropna(axis=1).columns,index=returns.index)["2013":"2017"]
beta5f_portfoliosp4 = pd.DataFrame(columns=returns["2013":"2017"].
↪dropna(axis=1).columns,index=returns.index)["2013":"2017"]

for i in alpha5f_portfoliosp4.columns:
        ff_ = ff.copy()["2013":"2017"]
        ff_[i] = returns[i]["2013":"2017"]
        ff_["Excess"] = ff_[i] - ff_["RF"]
        mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
↪CMA",window=12,data=ff_)
        rres = mod.fit()
        alpha5f_portfoliosp4[i] = rres.params["Intercept"]
        beta5f_portfoliosp4[i] = rres.params["mkt_excess"]

alpha5f_portfoliosp5 = pd.DataFrame(columns=returns["2018":"2021"].
↪dropna(axis=1).columns,index=returns.index)["2018":"2021"]
```

```
beta5f_portfoliosp5 = pd.DataFrame(columns=returns["2018":"2021"].
↪dropna(axis=1).columns,index=returns.index)["2018":"2021"]

for i in alpha5f_portfoliosp5.columns:
        ff_ = ff.copy()["2018":"2021"]
        ff_[i] = returns[i]["2018":"2021"]
        ff_["Excess"] = ff_[i] - ff_["RF"]
        mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
↪CMA",window=12,data=ff_)
        rres = mod.fit()
        alpha5f_portfoliosp5[i] = rres.params["Intercept"]
        beta5f_portfoliosp5[i] = rres.params["mkt_excess"]


alpha5f_portfoliosp1.to_csv("Computations/alpha5f_portfoliosp1.csv")
alpha5f_portfoliosp2.to_csv("Computations/alpha5f_portfoliosp2.csv")
alpha5f_portfoliosp3.to_csv("Computations/alpha5f_portfoliosp3.csv")
alpha5f_portfoliosp4.to_csv("Computations/alpha5f_portfoliosp4.csv")
alpha5f_portfoliosp5.to_csv("Computations/alpha5f_portfoliosp5.csv")

beta5f_portfoliosp1.to_csv("Computations/beta5f_portfoliosp1.csv")
beta5f_portfoliosp2.to_csv("Computations/beta5f_portfoliosp2.csv")
beta5f_portfoliosp3.to_csv("Computations/beta5f_portfoliosp3.csv")
beta5f_portfoliosp4.to_csv("Computations/beta5f_portfoliosp4.csv")
beta5f_portfoliosp5.to_csv("Computations/beta5f_portfoliosp5.csv")
```

```
[ ]: alpha5f_fundsp1 = pd.DataFrame(columns=data_f["1998":"2003"].dropna(axis=1).
     ↪columns,index=data_f.index)["1998":"2003"]
     beta5f_fundsp1 = pd.DataFrame(columns=data_f["1998":"2003"].dropna(axis=1).
     ↪columns,index=data_f.index)["1998":"2003"]

     for i in alpha5f_fundsp1.columns:
             ff_ = ff.copy()["1998":"2003"]
             ff_[i] = data_f[i]["1998":"2003"]
             ff_["Excess"] = ff_[i] - ff_["RF"]
             mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
     ↪CMA",window=12,data=ff_)
             rres = mod.fit()
             alpha5f_fundsp1[i] = rres.params["Intercept"]
             beta5f_fundsp1[i] = rres.params["mkt_excess"]

     alpha5f_fundsp2 = pd.DataFrame(columns=data_f["2004":"2007"].dropna(axis=1).
     ↪columns,index=data_f.index)["2004":"2007"]
     beta5f_fundsp2 = pd.DataFrame(columns=data_f["2004":"2007"].dropna(axis=1).
     ↪columns,index=data_f.index)["2004":"2007"]
```

```
for i in alpha5f_fundsp2.columns:
        ff_ = ff.copy()["2004":"2007"]
        ff_[i] = data_f[i]["2004":"2007"]
        ff_["Excess"] = ff_[i] - ff_["RF"]
        mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
 ↪CMA",window=12,data=ff_)
        rres = mod.fit()
        alpha5f_fundsp2[i] = rres.params["Intercept"]
        beta5f_fundsp2[i] = rres.params["mkt_excess"]

alpha5f_fundsp3 = pd.DataFrame(columns=data_f["2008":"2012"].dropna(axis=1).
 ↪columns,index=data_f.index)["2008":"2012"]
beta5f_fundsp3 = pd.DataFrame(columns=data_f["2008":"2012"].dropna(axis=1).
 ↪columns,index=data_f.index)["2008":"2012"]


for i in alpha5f_fundsp3.columns:
        ff_ = ff.copy()["2008":"2012"]
        ff_[i] = data_f[i]["2008":"2012"]
        ff_["Excess"] = ff_[i] - ff_["RF"]
        mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
 ↪CMA",window=12,data=ff_)
        rres = mod.fit()
        alpha5f_fundsp3[i] = rres.params["Intercept"]
        beta5f_fundsp3[i] = rres.params["mkt_excess"]

alpha5f_fundsp4 = pd.DataFrame(columns=data_f["2013":"2017"].dropna(axis=1).
 ↪columns,index=data_f.index)["2013":"2017"]
beta5f_fundsp4 = pd.DataFrame(columns=data_f["2013":"2017"].dropna(axis=1).
 ↪columns,index=data_f.index)["2013":"2017"]

for i in alpha5f_fundsp4.columns:
        ff_ = ff.copy()["2013":"2017"]
        ff_[i] = data_f[i]["2013":"2017"]
        ff_["Excess"] = ff_[i] - ff_["RF"]
        mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
 ↪CMA",window=12,data=ff_)
        rres = mod.fit()
        alpha5f_fundsp4[i] = rres.params["Intercept"]
        beta5f_fundsp4[i] = rres.params["mkt_excess"]


alpha5f_fundsp5 = pd.DataFrame(columns=data_f["2018":"2021"].dropna(axis=1).
 ↪columns,index=data_f.index)["2018":"2021"]
beta5f_fundsp5 = pd.DataFrame(columns=data_f["2018":"2021"].dropna(axis=1).
 ↪columns,index=data_f.index)["2018":"2021"]
```

```
for i in alpha5f_fundsp5.columns:
        ff_ = ff.copy()["2018":"2021"]
        ff_[i] = data_f[i]["2018":"2021"]
        ff_["Excess"] = ff_[i] - ff_["RF"]
        mod = RollingOLS.from_formula("Excess ~ mkt_excess + SMB + HML + RMW +␣
 ↪CMA",window=12,data=ff_)
        rres = mod.fit()
        alpha5f_fundsp5[i] = rres.params["Intercept"]
        beta5f_fundsp5[i] = rres.params["mkt_excess"]

alpha5f_fundsp1.to_csv("Computations/alpha5f_fundsp1.csv")
alpha5f_fundsp2.to_csv("Computations/alpha5f_fundsp2.csv")
alpha5f_fundsp3.to_csv("Computations/alpha5f_fundsp3.csv")
alpha5f_fundsp4.to_csv("Computations/alpha5f_fundsp4.csv")
alpha5f_fundsp5.to_csv("Computations/alpha5f_fundsp5.csv")

beta5f_fundsp1.to_csv("Computations/beta5f_fundsp1.csv")
beta5f_fundsp2.to_csv("Computations/beta5f_fundsp2.csv")
beta5f_fundsp3.to_csv("Computations/beta5f_fundsp3.csv")
beta5f_fundsp4.to_csv("Computations/beta5f_fundsp4.csv")
beta5f_fundsp5.to_csv("Computations/beta5f_fundsp5.csv")
```

1.3 To save time, we can load the data

[14]:
```
alpha5f_fundsp1 = pd.read_csv("Computations/alpha5f_fundsp1.
 ↪csv",index_col=0,parse_dates=True)
alpha5f_fundsp2 = pd.read_csv("Computations/alpha5f_fundsp2.
 ↪csv",index_col=0,parse_dates=True)
alpha5f_fundsp3 = pd.read_csv("Computations/alpha5f_fundsp3.
 ↪csv",index_col=0,parse_dates=True)
alpha5f_fundsp4 = pd.read_csv("Computations/alpha5f_fundsp4.
 ↪csv",index_col=0,parse_dates=True)
alpha5f_fundsp5 = pd.read_csv("Computations/alpha5f_fundsp5.
 ↪csv",index_col=0,parse_dates=True)

beta5f_fundsp1 = pd.read_csv("Computations/beta5f_fundsp1.
 ↪csv",index_col=0,parse_dates=True)
beta5f_fundsp2 = pd.read_csv("Computations/beta5f_fundsp2.
 ↪csv",index_col=0,parse_dates=True)
beta5f_fundsp3 = pd.read_csv("Computations/beta5f_fundsp3.
 ↪csv",index_col=0,parse_dates=True)
beta5f_fundsp4 = pd.read_csv("Computations/beta5f_fundsp4.
 ↪csv",index_col=0,parse_dates=True)
beta5f_fundsp5 = pd.read_csv("Computations/beta5f_fundsp5.
 ↪csv",index_col=0,parse_dates=True)
```

```
alpha5f_portfoliosp1 = pd.read_csv("Computations/alpha5f_portfoliosp1.
↪csv",index_col=0,parse_dates=True)
alpha5f_portfoliosp2 = pd.read_csv("Computations/alpha5f_portfoliosp2.
↪csv",index_col=0,parse_dates=True)
alpha5f_portfoliosp3 = pd.read_csv("Computations/alpha5f_portfoliosp3.
↪csv",index_col=0,parse_dates=True)
alpha5f_portfoliosp4 = pd.read_csv("Computations/alpha5f_portfoliosp4.
↪csv",index_col=0,parse_dates=True)
alpha5f_portfoliosp5 = pd.read_csv("Computations/alpha5f_portfoliosp5.
↪csv",index_col=0,parse_dates=True)

beta5f_portfoliosp1 = pd.read_csv("Computations/beta5f_portfoliosp1.
↪csv",index_col=0,parse_dates=True)
beta5f_portfoliosp2 = pd.read_csv("Computations/beta5f_portfoliosp2.
↪csv",index_col=0,parse_dates=True)
beta5f_portfoliosp3 = pd.read_csv("Computations/beta5f_portfoliosp3.
↪csv",index_col=0,parse_dates=True)
beta5f_portfoliosp4 = pd.read_csv("Computations/beta5f_portfoliosp4.
↪csv",index_col=0,parse_dates=True)
beta5f_portfoliosp5 = pd.read_csv("Computations/beta5f_portfoliosp5.
↪csv",index_col=0,parse_dates=True)
```

1.3.1 Now, we group the funds or portfolios with alpha in terms of their beta

```
[ ]: alpha5f_bybetas_fundsp1 = pd.DataFrame(index=alpha5f_fundsp1.
     ↪index,columns=["low","mid","high"])

     for i in alpha5f_fundsp1[alpha5f_fundsp1>0].index:

         lista_alpha = alpha5f_fundsp1.loc[i][alpha5f_fundsp1.loc[i]> 0].index.
     ↪to_list()

         alpha5f_bybetas_fundsp1.loc[i]["low"] = len(beta5f_fundsp1.
     ↪loc[i][lista_alpha][beta5f_fundsp1.loc[i][lista_alpha] < 0.9])

         alpha5f_bybetas_fundsp1.loc[i]["high"] = len(beta5f_fundsp1.
     ↪loc[i][lista_alpha][beta5f_fundsp1.loc[i][lista_alpha] > 1.1])

         alpha5f_bybetas_fundsp1.loc[i]["mid"] = len(beta5f_fundsp1.
     ↪loc[i][lista_alpha][  (beta5f_fundsp1.loc[i][lista_alpha] < 1.1) &␣
     ↪(beta5f_fundsp1.loc[i][lista_alpha] > 0.9)])



     alpha5f_bybetas_fundsp2 = pd.DataFrame(index=alpha5f_fundsp2.
     ↪index,columns=["low","mid","high"])
```

```
for i in alpha5f_fundsp2[alpha5f_fundsp2>0].index:

    lista_alpha = alpha5f_fundsp2.loc[i][alpha5f_fundsp2.loc[i]> 0].index.
 ↪to_list()

    alpha5f_bybetas_fundsp2.loc[i]["low"] = len(beta5f_fundsp2.
 ↪loc[i][lista_alpha][beta5f_fundsp2.loc[i][lista_alpha] < 0.9])

    alpha5f_bybetas_fundsp2.loc[i]["high"] = len(beta5f_fundsp2.
 ↪loc[i][lista_alpha][beta5f_fundsp2.loc[i][lista_alpha] > 1.1])

    alpha5f_bybetas_fundsp2.loc[i]["mid"] = len(beta5f_fundsp2.
 ↪loc[i][lista_alpha][  (beta5f_fundsp2.loc[i][lista_alpha] < 1.1) &␣
 ↪(beta5f_fundsp2.loc[i][lista_alpha] > 0.9)])


alpha5f_bybetas_fundsp3 = pd.DataFrame(index=alpha5f_fundsp3.
 ↪index,columns=["low","mid","high"])

for i in alpha5f_fundsp3[alpha5f_fundsp3>0].index:

    lista_alpha = alpha5f_fundsp3.loc[i][alpha5f_fundsp3.loc[i]> 0].index.
 ↪to_list()

    alpha5f_bybetas_fundsp3.loc[i]["low"] = len(beta5f_fundsp3.
 ↪loc[i][lista_alpha][beta5f_fundsp3.loc[i][lista_alpha] < 0.9])

    alpha5f_bybetas_fundsp3.loc[i]["high"] = len(beta5f_fundsp3.
 ↪loc[i][lista_alpha][beta5f_fundsp3.loc[i][lista_alpha] > 1.1])

    alpha5f_bybetas_fundsp3.loc[i]["mid"] = len(beta5f_fundsp3.
 ↪loc[i][lista_alpha][  (beta5f_fundsp3.loc[i][lista_alpha] < 1.1) &␣
 ↪(beta5f_fundsp3.loc[i][lista_alpha] > 0.9)])

alpha5f_bybetas_fundsp4 = pd.DataFrame(index=alpha5f_fundsp4.
 ↪index,columns=["low","mid","high"])

for i in alpha5f_fundsp4[alpha5f_fundsp4>0].index:

    lista_alpha = alpha5f_fundsp4.loc[i][alpha5f_fundsp4.loc[i]> 0].index.
 ↪to_list()

    alpha5f_bybetas_fundsp4.loc[i]["low"] = len(beta5f_fundsp4.
 ↪loc[i][lista_alpha][beta5f_fundsp4.loc[i][lista_alpha] < 0.9])
```

```
    alpha5f_bybetas_fundsp4.loc[i]["high"] = len(beta5f_fundsp4.
 ↪loc[i][lista_alpha][beta5f_fundsp4.loc[i][lista_alpha] > 1.1])

    alpha5f_bybetas_fundsp4.loc[i]["mid"] = len(beta5f_fundsp4.
 ↪loc[i][lista_alpha][  (beta5f_fundsp4.loc[i][lista_alpha] < 1.1) &␣
 ↪(beta5f_fundsp4.loc[i][lista_alpha] > 0.9)])


alpha5f_bybetas_fundsp5 = pd.DataFrame(index=alpha5f_fundsp5.
 ↪index,columns=["low","mid","high"])

for i in alpha5f_fundsp5[alpha5f_fundsp5>0].index:

    lista_alpha = alpha5f_fundsp5.loc[i][alpha5f_fundsp5.loc[i]> 0].index.
 ↪to_list()

    alpha5f_bybetas_fundsp5.loc[i]["low"] = len(beta5f_fundsp5.
 ↪loc[i][lista_alpha][beta5f_fundsp5.loc[i][lista_alpha] < 0.9])

    alpha5f_bybetas_fundsp5.loc[i]["high"] = len(beta5f_fundsp5.
 ↪loc[i][lista_alpha][beta5f_fundsp5.loc[i][lista_alpha] > 1.1])

    alpha5f_bybetas_fundsp5.loc[i]["mid"] = len(beta5f_fundsp5.
 ↪loc[i][lista_alpha][  (beta5f_fundsp5.loc[i][lista_alpha] < 1.1) &␣
 ↪(beta5f_fundsp5.loc[i][lista_alpha] > 0.9)])


alpha5f_bybetas_fundsp1.to_csv("Computations/alpha5f_bybetas_fundsp1.csv")
alpha5f_bybetas_fundsp2.to_csv("Computations/alpha5f_bybetas_fundsp2.csv")
alpha5f_bybetas_fundsp3.to_csv("Computations/alpha5f_bybetas_fundsp3.csv")
alpha5f_bybetas_fundsp4.to_csv("Computations/alpha5f_bybetas_fundsp4.csv")
alpha5f_bybetas_fundsp5.to_csv("Computations/alpha5f_bybetas_fundsp5.csv")
```

```
[ ]: alpha5f_bybetas_portfoliosp1 = pd.DataFrame(index=alpha5f_portfoliosp1.
 ↪index,columns=["low","mid","high"])

for i in alpha5f_portfoliosp1[alpha5f_portfoliosp1>0].index:

    lista_alpha = alpha5f_portfoliosp1.loc[i][:-2][alpha5f_portfoliosp1.loc[i][:
 ↪-2]> 0].index.to_list()

    alpha5f_bybetas_portfoliosp1.loc[i]["low"] = len(beta5f_portfoliosp1.
 ↪loc[i][:-2][lista_alpha][beta5f_portfoliosp1.loc[i][:-2][lista_alpha] < 0.9])

    alpha5f_bybetas_portfoliosp1.loc[i]["high"] = len(beta5f_portfoliosp1.
 ↪loc[i][:-2][lista_alpha][beta5f_portfoliosp1.loc[i][:-2][lista_alpha] > 1.1])
```

```
    alpha5f_bybetas_portfoliosp1.loc[i]["mid"] = len(beta5f_portfoliosp1.
 ↪loc[i][:-2][lista_alpha][  (beta5f_portfoliosp1.loc[i][:-2][lista_alpha] < 1.
 ↪1) &
                                                      (beta5f_portfoliosp1.loc[i][:
 ↪-2][lista_alpha] > 0.9)])


alpha5f_bybetas_portfoliosp2 = pd.DataFrame(index=alpha5f_portfoliosp2.
 ↪index,columns=["low","mid","high"])

for i in alpha5f_portfoliosp2[alpha5f_portfoliosp2>0].index:

    lista_alpha = alpha5f_portfoliosp2.loc[i][:-2][alpha5f_portfoliosp2.loc[i][:
 ↪-2]> 0].index.to_list()

    alpha5f_bybetas_portfoliosp2.loc[i]["low"] = len(beta5f_portfoliosp2.
 ↪loc[i][:-2][lista_alpha][beta5f_portfoliosp2.loc[i][:-2][lista_alpha] < 0.9])

    alpha5f_bybetas_portfoliosp2.loc[i]["high"] = len(beta5f_portfoliosp2.
 ↪loc[i][:-2][lista_alpha][beta5f_portfoliosp2.loc[i][:-2][lista_alpha] > 1.1])

    alpha5f_bybetas_portfoliosp2.loc[i]["mid"] = len(beta5f_portfoliosp2.
 ↪loc[i][:-2][lista_alpha][  (beta5f_portfoliosp2.loc[i][:-2][lista_alpha] < 1.
 ↪1) &
                                                      (beta5f_portfoliosp2.loc[i][:
 ↪-2][lista_alpha] > 0.9)])


alpha5f_bybetas_portfoliosp3 = pd.DataFrame(index=alpha5f_portfoliosp3.
 ↪index,columns=["low","mid","high"])

for i in alpha5f_portfoliosp3[alpha5f_portfoliosp3>0].index:

    lista_alpha = alpha5f_portfoliosp3.loc[i][:-2][alpha5f_portfoliosp3.loc[i][:
 ↪-2]> 0].index.to_list()

    alpha5f_bybetas_portfoliosp3.loc[i]["low"] = len(beta5f_portfoliosp3.
 ↪loc[i][:-2][lista_alpha][beta5f_portfoliosp3.loc[i][:-2][lista_alpha] < 0.9])

    alpha5f_bybetas_portfoliosp3.loc[i]["high"] = len(beta5f_portfoliosp3.
 ↪loc[i][:-2][lista_alpha][beta5f_portfoliosp3.loc[i][:-2][lista_alpha] > 1.1])

    alpha5f_bybetas_portfoliosp3.loc[i]["mid"] = len(beta5f_portfoliosp3.
 ↪loc[i][:-2][lista_alpha][  (beta5f_portfoliosp3.loc[i][:-2][lista_alpha] < 1.
 ↪1) &
```

```
                                                    (beta5f_portfoliosp3.loc[i][:
↪-2][lista_alpha] > 0.9)])


alpha5f_bybetas_portfoliosp4 = pd.DataFrame(index=alpha5f_portfoliosp4.
↪index,columns=["low","mid","high"])

for i in alpha5f_portfoliosp4[alpha5f_portfoliosp4>0].index:

    lista_alpha = alpha5f_portfoliosp4.loc[i][:-2][alpha5f_portfoliosp4.loc[i][:
↪-2]> 0].index.to_list()

    alpha5f_bybetas_portfoliosp4.loc[i]["low"] = len(beta5f_portfoliosp4.
↪loc[i][:-2][lista_alpha][beta5f_portfoliosp4.loc[i][:-2][lista_alpha] < 0.9])

    alpha5f_bybetas_portfoliosp4.loc[i]["high"] = len(beta5f_portfoliosp4.
↪loc[i][:-2][lista_alpha][beta5f_portfoliosp4.loc[i][:-2][lista_alpha] > 1.1])

    alpha5f_bybetas_portfoliosp4.loc[i]["mid"] = len(beta5f_portfoliosp4.
↪loc[i][:-2][lista_alpha][  (beta5f_portfoliosp4.loc[i][:-2][lista_alpha] < 1.
↪1) &
                                                    (beta5f_portfoliosp4.loc[i][:
↪-2][lista_alpha] > 0.9)])


alpha5f_bybetas_portfoliosp5 = pd.DataFrame(index=alpha5f_portfoliosp5.
↪index,columns=["low","mid","high"])

for i in alpha5f_portfoliosp5[alpha5f_portfoliosp5>0].index:

    lista_alpha = alpha5f_portfoliosp5.loc[i][:-2][alpha5f_portfoliosp5.loc[i][:
↪-2]> 0].index.to_list()

    alpha5f_bybetas_portfoliosp5.loc[i]["low"] = len(beta5f_portfoliosp5.
↪loc[i][:-2][lista_alpha][beta5f_portfoliosp5.loc[i][:-2][lista_alpha] < 0.9])

    alpha5f_bybetas_portfoliosp5.loc[i]["high"] = len(beta5f_portfoliosp5.
↪loc[i][:-2][lista_alpha][beta5f_portfoliosp5.loc[i][:-2][lista_alpha] > 1.1])

    alpha5f_bybetas_portfoliosp5.loc[i]["mid"] = len(beta5f_portfoliosp5.
↪loc[i][:-2][lista_alpha][  (beta5f_portfoliosp5.loc[i][:-2][lista_alpha] < 1.
↪1) &
                                                    (beta5f_portfoliosp5.loc[i][:
↪-2][lista_alpha] > 0.9)])
```

```
alpha5f_bybetas_portfoliosp1.to_csv("Computations/alpha5f_bybetas_portfoliosp1.
 ↪csv")
alpha5f_bybetas_portfoliosp2.to_csv("Computations/alpha5f_bybetas_portfoliosp2.
 ↪csv")
alpha5f_bybetas_portfoliosp3.to_csv("Computations/alpha5f_bybetas_portfoliosp3.
 ↪csv")
alpha5f_bybetas_portfoliosp4.to_csv("Computations/alpha5f_bybetas_portfoliosp4.
 ↪csv")
alpha5f_bybetas_portfoliosp5.to_csv("Computations/alpha5f_bybetas_portfoliosp5.
 ↪csv")
```

1.3.2 Or load the data

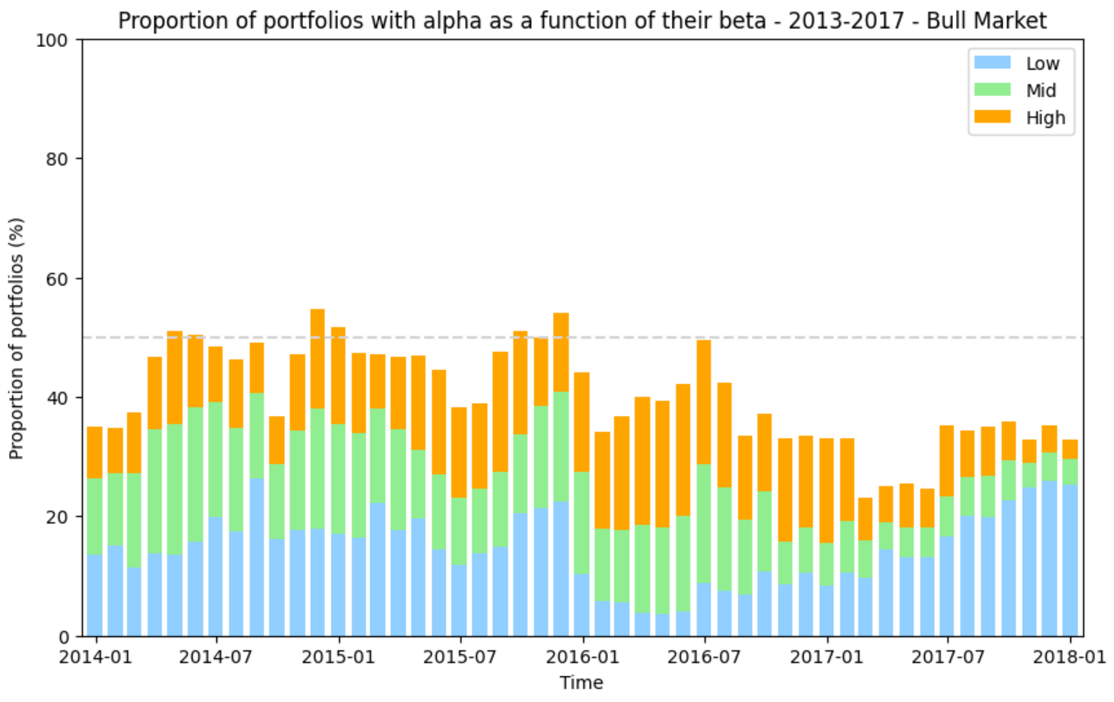
```
[15]: alpha5f_bybetas_portfoliosp1 = pd.read_csv("Computations/
       ↪alpha5f_bybetas_portfoliosp1.csv",index_col=0,parse_dates=True)
      alpha5f_bybetas_portfoliosp2 = pd.read_csv("Computations/
       ↪alpha5f_bybetas_portfoliosp2.csv",index_col=0,parse_dates=True)
      alpha5f_bybetas_portfoliosp3 = pd.read_csv("Computations/
       ↪alpha5f_bybetas_portfoliosp3.csv",index_col=0,parse_dates=True)
      alpha5f_bybetas_portfoliosp4 = pd.read_csv("Computations/
       ↪alpha5f_bybetas_portfoliosp4.csv",index_col=0,parse_dates=True)
      alpha5f_bybetas_portfoliosp5 = pd.read_csv("Computations/
       ↪alpha5f_bybetas_portfoliosp5.csv",index_col=0,parse_dates=True)

      alpha5f_bybetas_fundsp1 = pd.read_csv("Computations/alpha5f_bybetas_fundsp1.
       ↪csv",index_col=0,parse_dates=True)
      alpha5f_bybetas_fundsp2 = pd.read_csv("Computations/alpha5f_bybetas_fundsp2.
       ↪csv",index_col=0,parse_dates=True)
      alpha5f_bybetas_fundsp3 = pd.read_csv("Computations/alpha5f_bybetas_fundsp3.
       ↪csv",index_col=0,parse_dates=True)
      alpha5f_bybetas_fundsp4 = pd.read_csv("Computations/alpha5f_bybetas_fundsp4.
       ↪csv",index_col=0,parse_dates=True)
      alpha5f_bybetas_fundsp5 = pd.read_csv("Computations/alpha5f_bybetas_fundsp5.
       ↪csv",index_col=0,parse_dates=True)
```

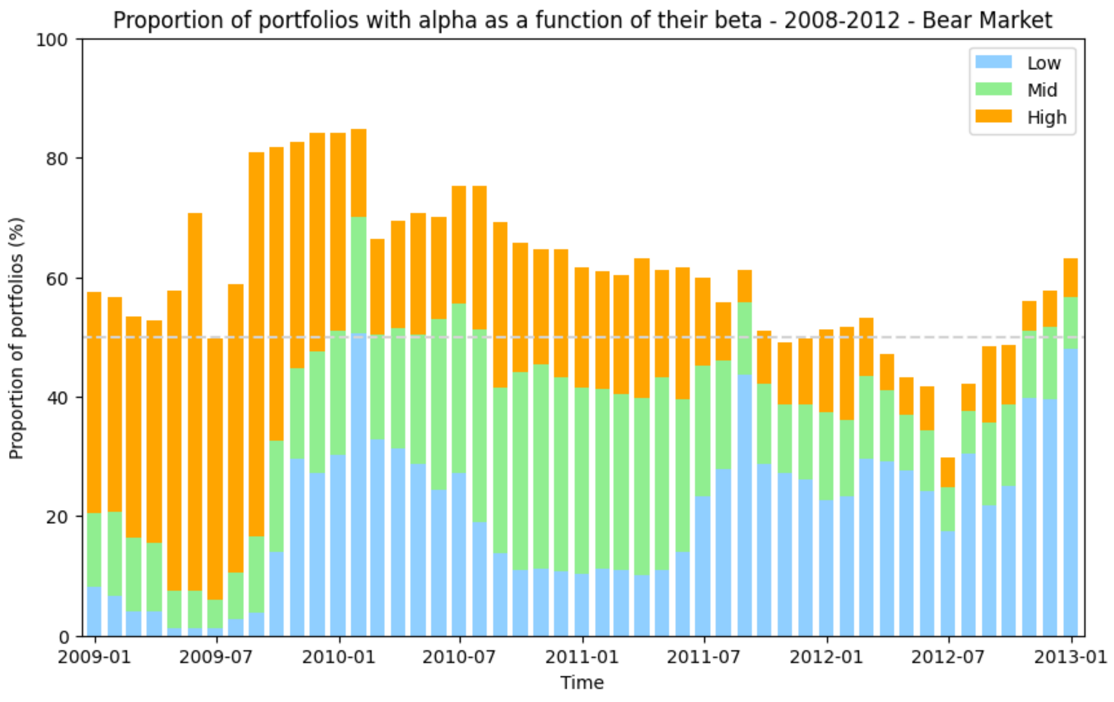
1.3.3 Now, we can visualize the proportion of portfolios and funds with alpha

```
[70]: fig,ax1 = plt.subplots(1,figsize=(10,6))

      ax1.bar(alpha5f_bybetas_portfoliosp4.
       ↪index,height=alpha5f_bybetas_portfoliosp4["low"]/
       ↪100,width=22,color="#90cfff",label="Low");
```

```
ax1.bar(alpha5f_bybetas_portfoliosp4.
 ↪index,height=alpha5f_bybetas_portfoliosp4["mid"]/
 ↪100,width=22,color="lightgreen",bottom=alpha5f_bybetas_portfoliosp4["low"]/
 ↪100,label="Mid");
ax1.bar(alpha5f_bybetas_portfoliosp4.
 ↪index,height=alpha5f_bybetas_portfoliosp4["high"]/
 ↪100,width=22,color="orange",
        bottom=alpha5f_bybetas_portfoliosp4["low"]/100 +␣
 ↪alpha5f_bybetas_portfoliosp4["mid"]/100,label="High");
plt.title("Proportion of portfolios with alpha as a function of their beta -␣
 ↪2013-2017 - Bull Market");
plt.legend()
ax1.set_ylabel("Proportion of portfolios (%)")
ax1.set_xlabel("Time")
ax1.set_xlim(16050,17550)
ax1.set_ylim(0,100)
ax1.axhline(y=50,color="lightgrey",linestyle="--")
ax1.get_xlim()

plt.savefig("Graphs/Proportion_portfolios_alpha_by_beta_carteras_p4.png")


fig,ax1 = plt.subplots(1,figsize=(10,6))

ax1.bar(alpha5f_bybetas_portfoliosp3.
 ↪index,height=alpha5f_bybetas_portfoliosp3["low"]/
 ↪100,width=22,color="#90cfff",label="Low");
ax1.bar(alpha5f_bybetas_portfoliosp3.
 ↪index,height=alpha5f_bybetas_portfoliosp3["mid"]/
 ↪100,width=22,color="lightgreen",bottom=alpha5f_bybetas_portfoliosp3["low"]/
 ↪100,label="Mid");
ax1.bar(alpha5f_bybetas_portfoliosp3.
 ↪index,height=alpha5f_bybetas_portfoliosp3["high"]/
 ↪100,width=22,color="orange",
        bottom=alpha5f_bybetas_portfoliosp3["low"]/100 +␣
 ↪alpha5f_bybetas_portfoliosp3["mid"]/100,label="High");
plt.title("Proportion of portfolios with alpha as a function of their beta -␣
 ↪2008-2012 - Bear Market");
plt.legend()
ax1.set_ylabel("Proportion of portfolios (%)")
ax1.set_xlabel("Time")
ax1.set_xlim(14225,15725)
ax1.set_ylim(0,100)
ax1.axhline(y=50,color="lightgrey",linestyle="--")
ax1.get_xlim()
```

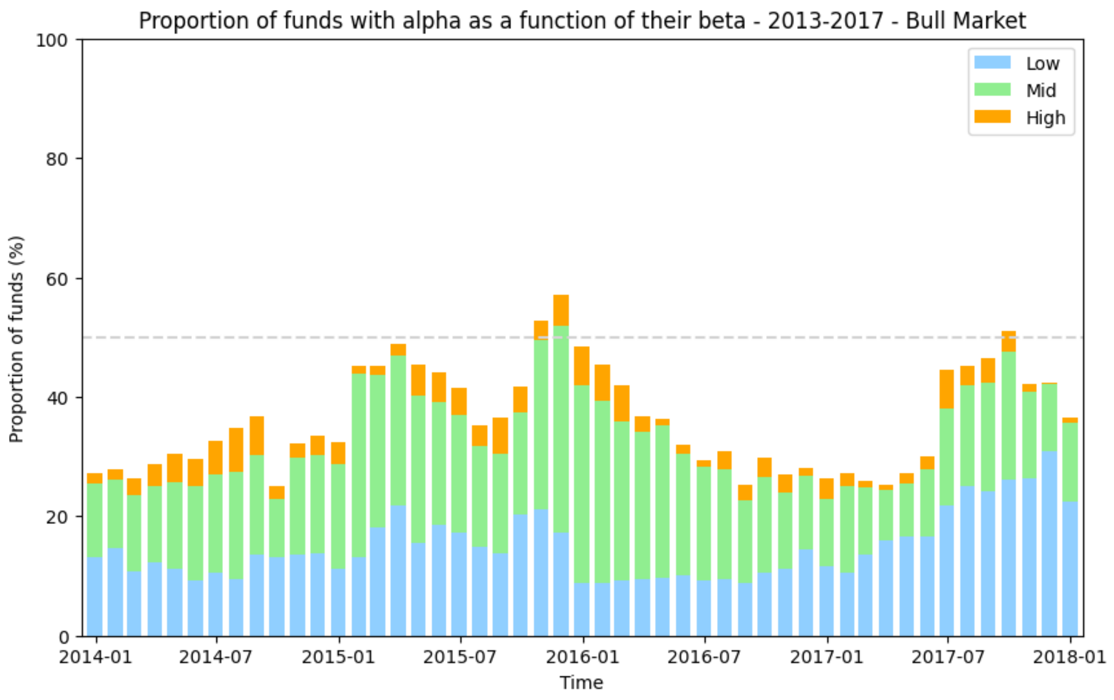
```python
plt.savefig("Graphs/Proportion_portfolios_alpha_by_beta_carteras_p3.png")
```

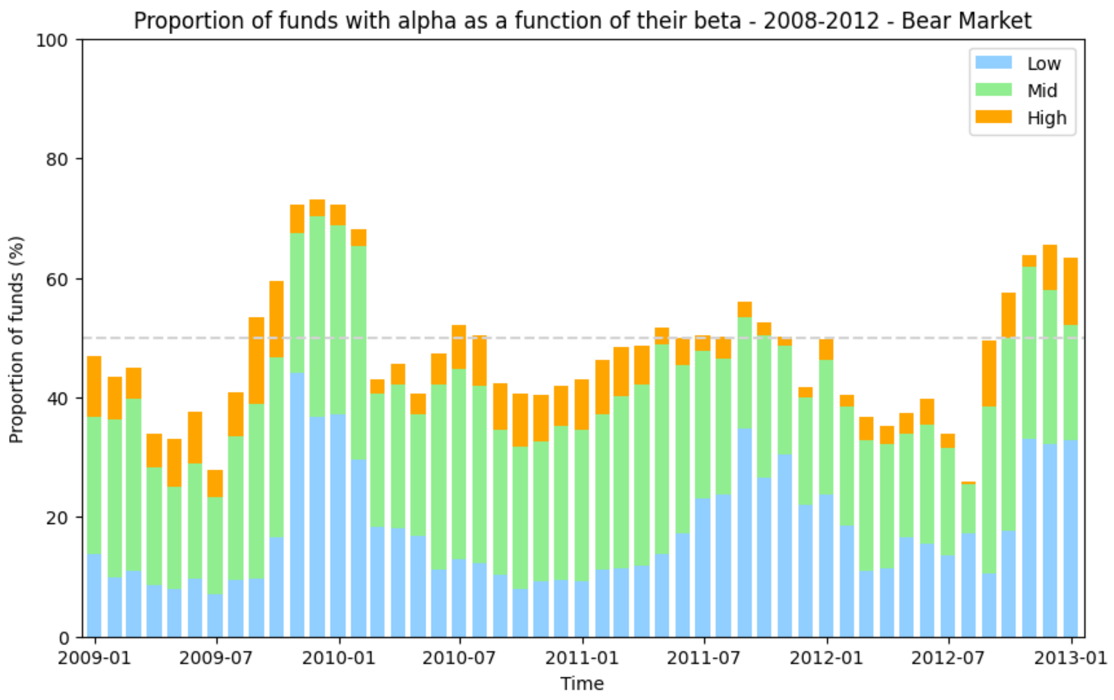




[40]:
```
fig,ax1 = plt.subplots(1,figsize=(10,6))

ax1.bar(alpha5f_bybetas_fundsp4.index,height=alpha5f_bybetas_fundsp4["low"]/
↪497*100,width=22,color="#90cfff",label="Low");
ax1.bar(alpha5f_bybetas_fundsp4.index,height=alpha5f_bybetas_fundsp4["mid"]/
↪497*100,width=22,color="lightgreen",bottom=alpha5f_bybetas_fundsp4["low"]/
↪497*100,label="Mid");
ax1.bar(alpha5f_bybetas_fundsp4.index,height=alpha5f_bybetas_fundsp4["high"]/
↪497*100,width=22,color="orange",
        bottom=alpha5f_bybetas_fundsp4["low"]/497*100 +␣
↪alpha5f_bybetas_fundsp4["mid"]/497*100,label="High");

plt.title("Proportion of funds with alpha as a function of their beta -␣
↪2013-2017 - Bull Market");
plt.legend()
ax1.set_ylabel("Proportion of funds (%)")
ax1.set_xlabel("Time")
ax1.set_xlim(16050,17550)
ax1.set_ylim(0,100)
ax1.axhline(y=50,color="lightgrey",linestyle="--")
ax1.get_xlim()

plt.savefig("Graphs/Proportion_funds_alpha_by_beta_fondos_p4.png")


fig,ax1 = plt.subplots(1,figsize=(10,6))

ax1.bar(alpha5f_bybetas_fundsp3.index,height=alpha5f_bybetas_fundsp3["low"]/
↪430*100,width=22,color="#90cfff",label="Low");
ax1.bar(alpha5f_bybetas_fundsp3.index,height=alpha5f_bybetas_fundsp3["mid"]/
↪430*100,width=22,color="lightgreen",bottom=alpha5f_bybetas_fundsp3["low"]/
↪430*100,label="Mid");
ax1.bar(alpha5f_bybetas_fundsp3.index,height=alpha5f_bybetas_fundsp3["high"]/
↪430*100,width=22,color="orange",
        bottom=alpha5f_bybetas_fundsp3["low"]/430*100 +␣
↪alpha5f_bybetas_fundsp3["mid"]/430*100,label="High");

plt.title("Proportion of funds with alpha as a function of their beta -␣
↪2008-2012 - Bear Market");
plt.legend()
ax1.set_ylabel("Proportion of funds (%)")
ax1.set_xlabel("Time")
ax1.set_xlim(14225,15725)
ax1.set_ylim(0,100)
ax1.axhline(y=50,color="lightgrey",linestyle="--")
ax1.get_xlim()
```

```python
plt.savefig("Graphs/Proportion_funds_alpha_by_beta_fondos_p3.png")
```





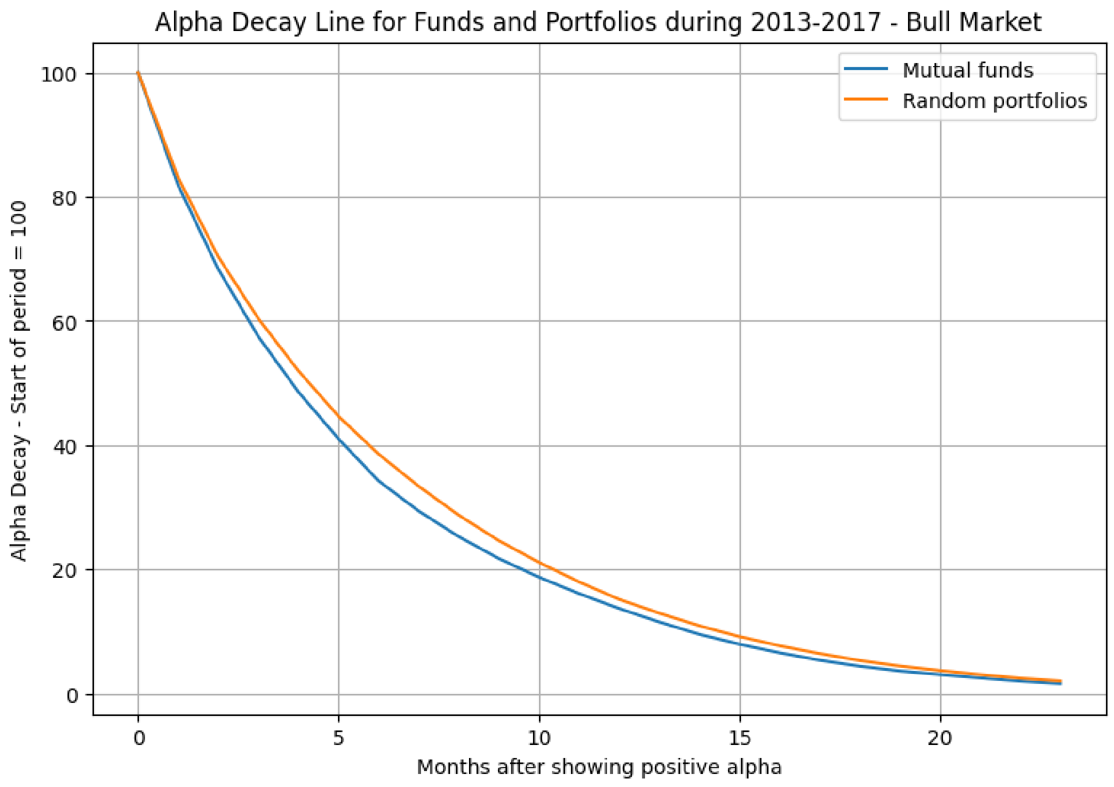
1.4 As an additional test, we introduce the "Alpha Decay Line" (ADL), where we can compute the proportion of portfolios/funds that conserve alpha following the month they achived it.

```
[16]: def filter_positive_values_first_row(dataframe,column):
          return (dataframe.iloc[column][dataframe.iloc[column] >0]).index.to_list()


      def filter_list_components(lista1,lista2):
          return list(set(lista1).intersection(lista2))


      def alpha_decay(dataframe,months):

          selection = dataframe

          number_months = months
          number_periods = len(selection.dropna())-number_months

          dataframe_number = pd.
       ↪DataFrame(columns=range(1,number_periods),index=range(0,number_months))


          for period in range(1,number_periods):

              selection_ = selection.dropna().iloc[period:,]

              temp_list_0 = filter_positive_values_first_row(selection_.dropna(),0)

              temp_list_1 = filter_list_components(temp_list_0,selection_.dropna().
       ↪iloc[1][selection_.dropna().iloc[1] >0].index.to_list())

              temp_list_filtered = temp_list_1

              lista_numbers = [len(temp_list_0)]

              for month in range(1,(number_months)):

                  temp_list_filtered = filter_list_components(temp_list_filtered,␣
       ↪selection_.dropna().iloc[month][selection_.dropna().iloc[month] >0].index.
       ↪to_list())

                  lista_numbers.append(len(temp_list_filtered))

              dataframe_number[period] = lista_numbers

          return dataframe_number
```

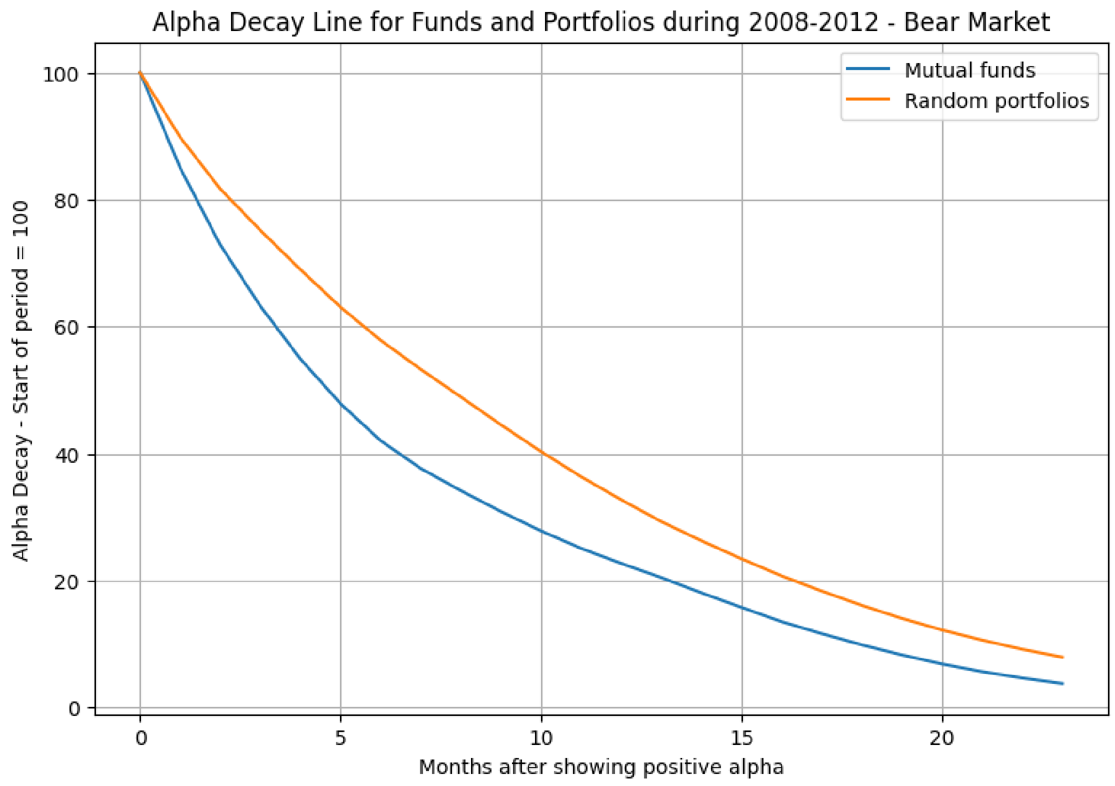
[17]:
```
alpha_decayf3 = alpha_decay(alpha5f_fundsp3,24)
alpha_decayf4 = alpha_decay(alpha5f_fundsp4,24)

alpha_decayp3 = alpha_decay(alpha5f_portfoliosp3,24)
alpha_decayp4 = alpha_decay(alpha5f_portfoliosp4,24)
```

1.4.1 We plot the ADL

[25]:
```
fig,ax1 = plt.subplots(1,figsize=(9,6))

ax1.plot((alpha_decayf4.mean(axis=1)/alpha_decayf4.mean(axis=1).
 ↪iloc[0])*100,label="Mutual funds");
ax1.plot((alpha_decayp4.mean(axis=1)/alpha_decayp4.mean(axis=1).
 ↪iloc[0])*100,label="Random portfolios");
ax1.set_title("Alpha Decay Line for Funds and Portfolios during 2013-2017 -␣
 ↪Bull Market");
ax1.set_xlabel("Months after showing positive alpha");
ax1.set_ylabel("Alpha Decay - Start of period = 100");
plt.grid(True);
ax1.legend();

plt.savefig("Graphs/alpha_decay_bull.png",dpi=300);
```

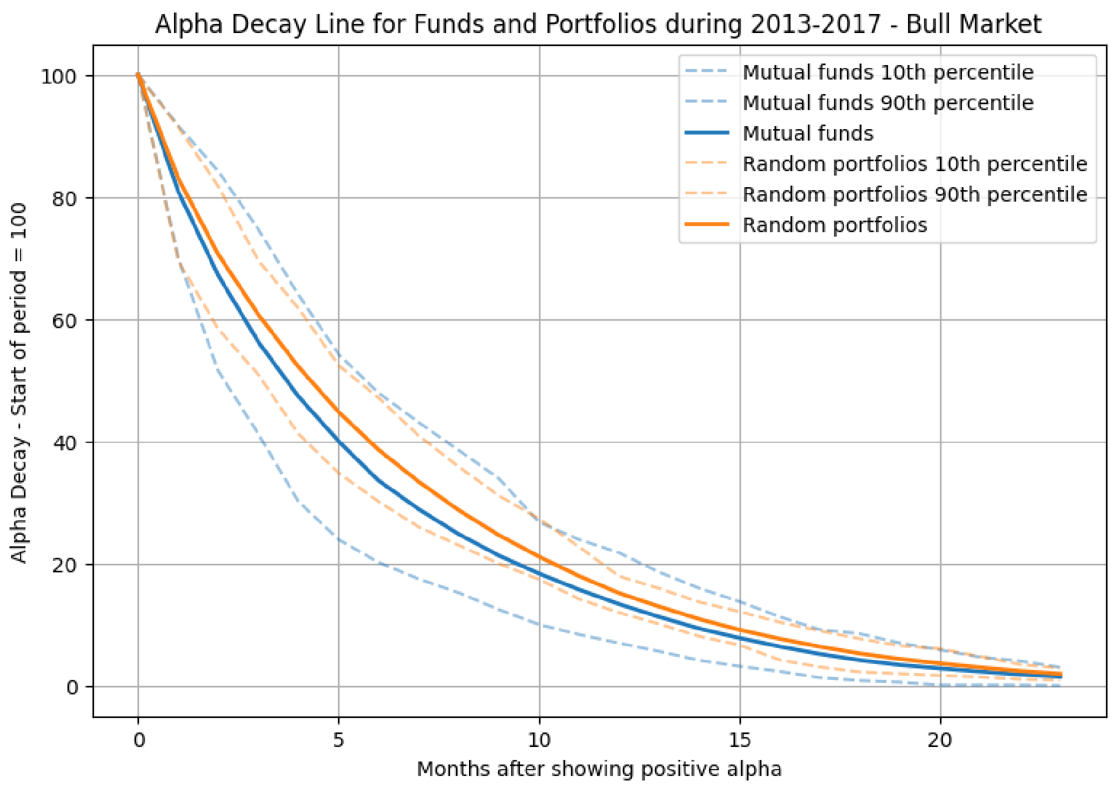


```
[26]: fig,ax1 = plt.subplots(1,figsize=(9,6))

      ax1.plot((alpha_decayf3.mean(axis=1)/alpha_decayf3.mean(axis=1).
       ↪iloc[0])*100,label="Mutual funds");
      ax1.plot((alpha_decayp3.mean(axis=1)/alpha_decayp3.mean(axis=1).
       ↪iloc[0])*100,label="Random portfolios");
      ax1.set_title("Alpha Decay Line for Funds and Portfolios during 2008-2012 -␣
       ↪Bear Market");
      ax1.set_xlabel("Months after showing positive alpha");
      ax1.set_ylabel("Alpha Decay - Start of period = 100");
      plt.grid(True);
      ax1.legend();

      plt.savefig("Graphs/alpha_decay_bear.png",dpi=300);
```



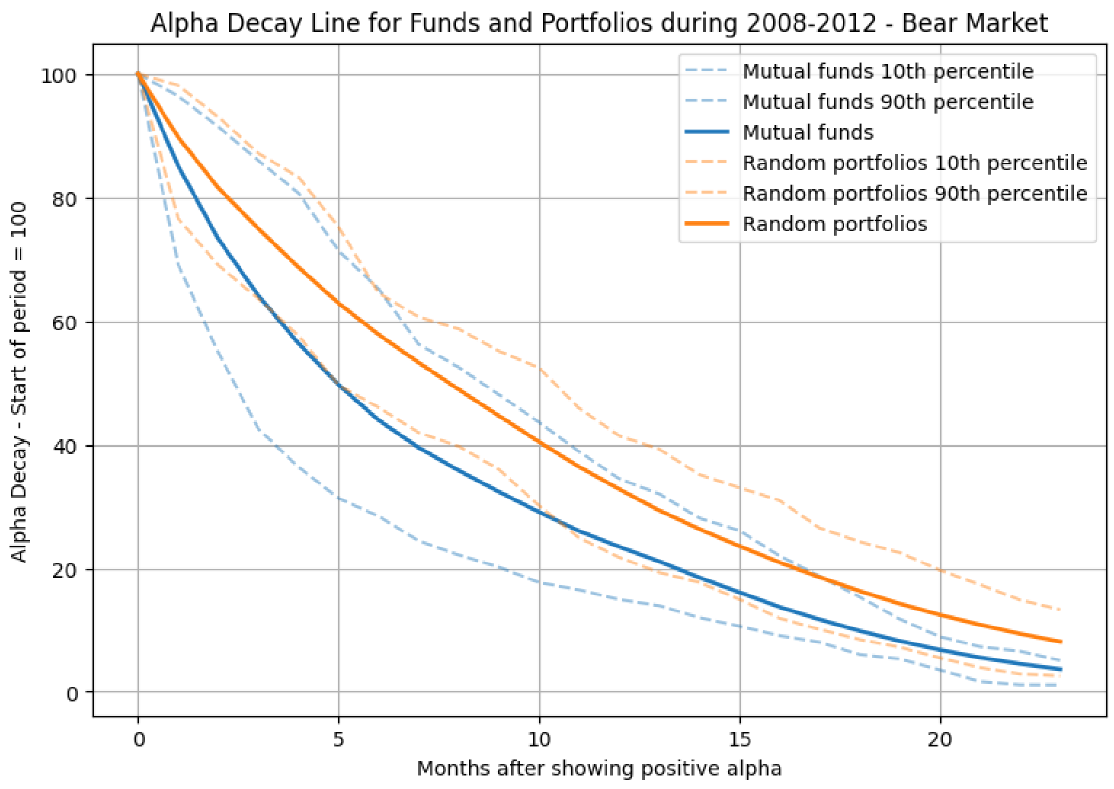
1.4.2 To visualize the dispersion, we plot as well the 10th and 90th percentile

```
[29]: fig,ax1 = plt.subplots(1,figsize=(9,6))

ax1.plot((alpha_decayf4/alpha_decayf4.iloc[0]).quantile(0.
 ↪1,axis=1)*100,label="Mutual funds 10th percentile",color="#1f77ba",alpha=0.
 ↪45,linestyle="dashed");
ax1.plot((alpha_decayf4/alpha_decayf4.iloc[0]).quantile(0.
 ↪9,axis=1)*100,label="Mutual funds 90th percentile",color="#1f77ba",alpha=0.
 ↪45,linestyle="dashed");
ax1.plot((alpha_decayf4/alpha_decayf4.iloc[0]).mean(axis=1)*100,label="Mutual␣
 ↪funds",color="#1f77ba",alpha=1,linewidth=2);

ax1.plot((alpha_decayp4/alpha_decayp4.iloc[0]).quantile(0.
 ↪1,axis=1)*100,label="Random portfolios 10th␣
 ↪percentile",color="#ff7f0e",alpha=0.45,linestyle="dashed");
ax1.plot((alpha_decayp4/alpha_decayp4.iloc[0]).quantile(0.
 ↪9,axis=1)*100,label="Random portfolios 90th␣
 ↪percentile",color="#ff7f0e",alpha=0.45,linestyle="dashed");
ax1.plot((alpha_decayp4/alpha_decayp4.iloc[0]).mean(axis=1)*100,label="Random␣
 ↪portfolios",color="#ff7f0e",alpha=1,linewidth=2);

ax1.set_title("Alpha Decay Line for Funds and Portfolios during 2013-2017 -␣
 ↪Bull Market");
ax1.set_xlabel("Months after showing positive alpha");
ax1.set_ylabel("Alpha Decay - Start of period = 100");
plt.grid(True);
ax1.legend();

plt.savefig("Graphs/alpha_decay_bull_v2.png",dpi=300);
```



[30]:
```
fig,ax1 = plt.subplots(1,figsize=(9,6))

ax1.plot((alpha_decayf3/alpha_decayf3.iloc[0]).quantile(0.
 ↪1,axis=1)*100,label="Mutual funds 10th percentile",color="#1f77ba",alpha=0.
 ↪45,linestyle="dashed");
ax1.plot((alpha_decayf3/alpha_decayf3.iloc[0]).quantile(0.
 ↪9,axis=1)*100,label="Mutual funds 90th percentile",color="#1f77ba",alpha=0.
 ↪45,linestyle="dashed");
ax1.plot((alpha_decayf3/alpha_decayf3.iloc[0]).mean(axis=1)*100,label="Mutual␣
 ↪funds",color="#1f77ba",alpha=1,linewidth=2);

ax1.plot((alpha_decayp3/alpha_decayp3.iloc[0]).quantile(0.
 ↪1,axis=1)*100,label="Random portfolios 10th␣
 ↪percentile",color="#ff7f0e",alpha=0.45,linestyle="dashed");
ax1.plot((alpha_decayp3/alpha_decayp3.iloc[0]).quantile(0.
 ↪9,axis=1)*100,label="Random portfolios 90th␣
 ↪percentile",color="#ff7f0e",alpha=0.45,linestyle="dashed");
ax1.plot((alpha_decayp3/alpha_decayp3.iloc[0]).mean(axis=1)*100,label="Random␣
 ↪portfolios",color="#ff7f0e",alpha=1,linewidth=2);
```

```
ax1.set_title("Alpha Decay Line for Funds and Portfolios during 2008-2012 -␣
↪Bear Market");
ax1.set_xlabel("Months after showing positive alpha");
ax1.set_ylabel("Alpha Decay - Start of period = 100");
plt.grid(True);
ax1.legend();

plt.savefig("Graphs/alpha_decay_bear_v2.png",dpi=300);
```



1.5 As the last round of tests, we perform classification with k-means and Support Vector Machine algorithms

1.5.1 First, we label the data that will be used for SVM and other tests performed in Rapid Miner

[49]:
```
label_funds = pd.DataFrame(index=["Label"],columns=factorsfp3.dropna(axis=1).
↪columns,data="Mutual fund")
factorsfp3.dropna(axis=1).append(label_funds)

label_portfolios = pd.DataFrame(index=["Label"],columns=factorsp3[:-2].
↪columns,data="Random Portfolio")
```

```
factorsp3.append(label_portfolios)

test_labelled_p3 = factorsfp3.dropna(axis=1).append(label_funds).join(factorsp3.
 ↪append(label_portfolios))

test_labelled_p3.T.to_csv("Computations/ClassificationP3.csv")


label_funds = pd.DataFrame(index=["Label"],columns=factorsfp4.dropna(axis=1).
 ↪columns,data="Mutual fund")
factorsfp4.dropna(axis=1).append(label_funds)

label_cartera = pd.DataFrame(index=["Label"],columns=factorsp4[:-2].
 ↪columns,data="Random Portfolio")
factorsp4.append(label_cartera)

test_labelled_p4 = factorsfp4.dropna(axis=1).append(label_funds).join(factorsp4.
 ↪append(label_portfolios))

test_labelled_p4.T.to_csv("Computations/ClassificationP4.csv")
```

1.5.2 We build a SVM model and compute its accuracy and confusion matrix, for both periods

[51]:
```
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix

target = test_labelled_p3.T["Label"]
features = test_labelled_p3.T.drop(["R2","Label","P-value Alpha"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(features,target, test_size␣
 ↪= 0.5, random_state = 668410)
from sklearn.svm import SVC

# Building a Support Vector Machine on train data
svc_model = SVC(C= .1, kernel='linear', gamma= 1)
svc_model.fit(X_train, y_train)

prediction = svc_model .predict(X_test)
# check the accuracy on the training set
print(svc_model.score(X_train, y_train))
print(svc_model.score(X_test, y_test))

print("Confusion Matrix:\n",confusion_matrix(y_test,prediction))
```

```
0.9581975071907958
0.959731543624161
```

```
Confusion Matrix:
 [[   2  210]
 [   0 5003]]
```

[52]:
```
from sklearn.model_selection import train_test_split

from sklearn.metrics import accuracy_score, confusion_matrix

target = test_labelled_p4.T["Label"]
features = test_labelled_p4.T.drop(["R2","Label","P-value Alpha"], axis=1)
X_train, X_test, y_train, y_test = train_test_split(features,target, test_size␣
↪= 0.5, random_state = 668410)
from sklearn.svm import SVC

# Building a Support Vector Machine on train data
svc_model = SVC(C= .1, kernel='linear', gamma= 1)
svc_model.fit(X_train, y_train)

prediction = svc_model.predict(X_test)
# check the accuracy on the training set
print(svc_model.score(X_train, y_train))
print(svc_model.score(X_test, y_test))

print("Confusion Matrix:\n",confusion_matrix(y_test,prediction))
```

```
0.9523628048780488
0.9529434177938655
Confusion Matrix:
 [[   0  247]
 [   0 5002]]
```

1.5.3 Now, we do the same for k-means

[59]:
```
############## k means

from sklearn.cluster import KMeans
from sklearn.metrics import accuracy_score, confusion_matrix
kmeans = KMeans(n_clusters=2)
kmeans.fit(test_labelled_p3.T.drop(["R2","Label","P-value Alpha"], axis=1))


y_kmeans = kmeans.predict(test_labelled_p3.T.drop(["R2","Label","P-value␣
↪Alpha"], axis=1))

#confusion_matrix()

print(accuracy_score(test_labelled_p3.T["Label"].factorize()[0], y_kmeans))
```

```
confusion_matrix(test_labelled_p3.T["Label"].factorize()[0], y_kmeans)
```

```
0.5179290508149569
```

```
[59]: array([[ 407,   23],
             [5005, 4995]], dtype=int64)
```

```
[58]: kmeans = KMeans(n_clusters=2)
kmeans.fit(test_labelled_p4.T.drop(["R2","Label","P-value Alpha"], axis=1))



y_kmeans = kmeans.predict(test_labelled_p4.T.drop(["R2","Label","P-value␣
 ↪Alpha"], axis=1))

#confusion_matrix()

print(accuracy_score(test_labelled_p4.T["Label"].factorize()[0], y_kmeans))
confusion_matrix(test_labelled_p4.T["Label"].factorize()[0], y_kmeans)
```

```
0.527102981804325
```

```
[58]: array([[ 460,   37],
             [4927, 5073]], dtype=int64)
```