LUISS

MSc in Corporate Finance
Department of Business and Management

Course of Asset Pricing

Empirical Stock Market Returns Forecasting: Machine Learning in Modern Portfolio Theory

Prof. Paolo Porchia
SUPERVISOR

Prof. Marco Pirra
CO-SUPERVISOR

Luca Feroce - ID 717901
CANDIDATE

Academic Year 2020/2021

Alla Sig. Luigina, che ammirava il mio percorso dando rilievo ad ogni mio piccolo passo;

Ai miei Nonni, mentori e promotori di diligenza;

A mia madre e mio padre, fonti di saggezza e perspicacia, elementi chiave del mio traguardo;

A mia sorella, compagna di vita ed infinita sorgente di felicità e spensieratezza;

A Serena, amorevole sostenitrice e complice essenziale del percorso di studi fin dall'infanzia;

Agli amici di sempre, Andrea e Federico, fidati alleati di avventure, sapienti consiglieri e stimatori.

TABLE OF CONTENTS

INTRODUCTION

This study aims to clarify the growing role of Machine Learning techniques in the financial industry and in particular in asset pricing. We have a deep investigation on how a representative mean-variance investor should exploits publicly available information to formulate forecasting on excess return and volatility as well as optimal portfolio allocations.

We employ a semi-parametric method known as Boosted Regression Tree (BRT) to forecast European market returns and volatility at the monthly frequency. BRT is a statistical method that generates previsions basing on a large set of conditioning information without imposing strong parametric assumptions such as linearity or monotonicity.

In our research we identify the ability of macroeconomic and financial variables to incorporate conditioning information about expected stock returns and volatility that a mean-variance investor could exploit. By using Python as programming language, we start by forecasting separately both excess return and volatility of the market index and compare the results of the BRT-method with the benchmarks. Our findings highlights the overperformance of BRT forecasts with a more significant market timing.

Subsequently, we test BRT from an asset allocation perspective with the aim to select optimal portfolio weights from excess returns and volatilities forecasts obtained, as well as through a direct BRT selection. Tests of optimal portfolio weights predictability demonstrate that portfolio allocations are time-varying and forecastable with a better performance of the one-step BRT compared to the statistical methods adopted as benchmark.

At the end, we evaluate whether the forecasts are economically valuable by computing the realized utilities level obtained by the investor by comparing BRT-based strategies and passive ones. Again, our results demonstrate the greater performance of the BRT-based model against the passive strategies.

Our paper contributes to the long-standing literature assessing the predictability of stock returns and helps justify the growing role of ML throughout the architecture of the burgeoning FinTech industry. We demonstrate that extending the highly non-linear conditioning information set results in an higher out-of-sample predictive accuracy and

subsequently to a better forecast compared to standard statistical models of the literature adopted as benchmarks.

CHAPTER 1

Machine learning: a real-world perspective

Summary: 1 – Fintech Revolution. –
2 – Stock Market Prediction. – 3 – Research Objective.

1.1 FinTech Revolution

Disruptions have transformed many sectors and in last years have started to affect the financial one as well and becoming the most influential business idea of the early 21st century. The disruptive terminology was defined and first analysed by the American scholar Clayton M. Christensen and his collaborators in 1995. It expresses an innovation that creates a new market and value network and eventually disrupts the existing ones, displacing established market-leading firms, products, and alliances (Ab. Rahman, Airini et al., 2017).

In recent years, the leading disruptive technology in the financial sector is the so-called Financial Technology (FinTech). FinTech firms are companies that use technology for banking, payments, financial data analytics, capital markets and personal financial management (Huang, 2015). Global FinTech investment are continues to rise. According to the Pulse of Fintech H2020[1], overall global FinTech funding across M&A, PE and VC was $105 billion across 2,861 deals in 2020: the third highest level of investment in fintech ever. FinTech companies are entering into a disruptive revolution due to their new alternatives that enhance the efficiency and quality of services.

[1] Bi-annual report on global fintech investment trends published by KPMG.

1.1.1 Machine Learning progress

Machine learning (ML) tools are gaining popularity in the FinTech sector, as can be seen in figure 1.1.

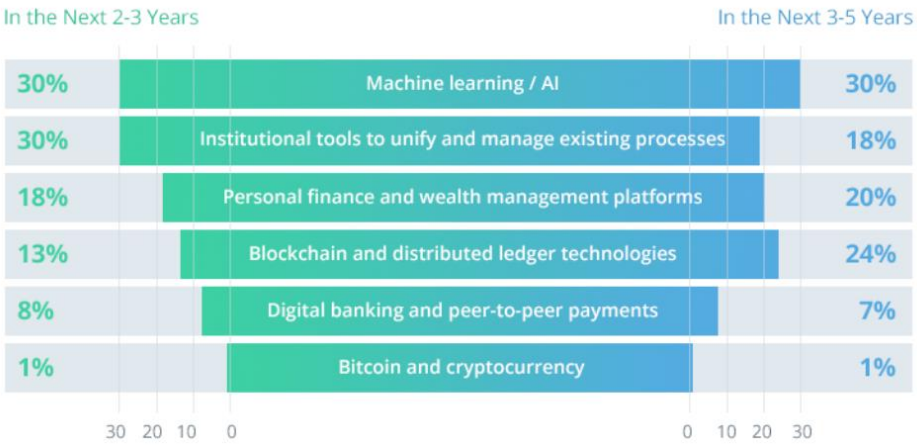


Fig. 1.1: Trends in the FinTech industry
Source: Mediant Investor Communication, 2019

There are many machine learning applications in finance, including for banking and credit offerings, payments and remittances, asset management, personal finance, regulatory and compliance services.

Machine learning algorithms in FinTech are definitely better fortune tellers than any human. Historical data is only the grounds on which predictions are made, because such algorithms are also able to monitor data sources available in real time, such as news or trade results, to reveal patterns indicating stock market dynamics and, for sure, making forecasting. The task left to traders is to determine which ML algorithms to include in their strategies, make a trading forecast, and choose a behavioural pattern.

Following Alpadyn (2004) , we define Machine Learning as programming the computers to optimize a performance criterion using training data or past experience. In detail, we use the term of Machine Learning in order to describe a collection of high-dimensional models for statistical prediction, combined with both the so called "regularization" methods[2] for model selection and mitigation to overfit, and efficient algorithms[3] for searching among an huge number of potential model specifications. Therefore, ML is suited for empirical asset pricing according to three main reasons: first, it is specialized

[2] Refinements in implementation that emphasize stable out-of-sample performance to guard against overfit.
[3] Clever ML tools designed to approximate optimal specification with manageable computational lost.

for prediction tasks, so ideally for the risk premium measurement; second, it is able to reduce the degrees of freedom and to condensate redundant variation among predictors for the risk premium; third, it includes a wide net in its specification search, as well as it is designed to approximate complex non-linear associations of the high-dimensional predictors and it is able to avoid overfit biases and false discovery through parameter penalization and conservative model selection criteria (Gu et al., 2018).

1.2 Stock Market Prediction

Information plays a central role in modern finance. Investors faces an ever-increasing number of new facts, data and statistics every minute of the day. Assessing the predictability of stock returns requires formulating risk premium[4] forecasts on the basis of large sets of conditioning information, but conventional statistical methods fail in such circumstances. In some cases, Machine learning techniques double the performance of leading regression-based strategies from the literature and trace their predictive gains to allowing non-linear predictor interaction missed by other methods (Gu et al., 2018).

Risk premium prediction is the fundamental goal from an asset pricing point of view. For a long time, it was believed that stock returns were not forecastable according to the Random Walk hypothesis[5] (Malkiel and Fama 1970; Malkiel and Burton 2003) and the Efficient Market hypothesis[6] (Jensen 1978) which assess that it is impossible to make economic gains if the market is efficient compared to a current information. Therefore, if it is impossible to outperform the market owing to the randomness in stock prices, economic profits cannot rise.

Stock market trends are almost dynamic, non-parametric[7], chaotic and noisy in nature making investments intrinsically risky. These fluctuations are prominent in the short-run and tend to develop linear trends in the long-run. Market risk and forecasting errors are

[4] Conditional expected stock returns in excess to the risk-free rate. Academic finance traditionally refers to this quantity as the "risk premium" because of its close connection with equilibrium compensation for bearing equity investment risk. We use the terms "expected return" and "risk premium" interchangeably.

[5] Random walk theory suggests that changes in stock prices have the same distribution and are independent of each other. Random walk theory infers that the past movement or trend of a stock price or market cannot be used to predict its future movement.

[6] The efficient-market hypothesis is an hypothesis in financial economics that states that asset prices reflect all available information.

[7] A model is "non-parametric" if all the parameters of the regression are in infinite-dimensional spaces.

strongly correlated each other and they need to be minimized. If expected returns were perfectly observed, investors will be able to increase their financial performance by minimizing risks. Unfortunately, risk premiums are notoriously difficult to measure due to the huge amount of unforecastable news that affects the daily market. Because of its complex and dynamic nature, predicting accurately trends in stock market prices has always been a challenge of interest for researchers and practitioners.

Last decades have seen the use of the increasingly powerful computational abilities to rise. Given different market situations, quantitative investment strategies have the advantage of not being impacted by the human emotions which Keynes in his general theory sees as "animal spirits [...] spontaneous urge to action rather than inaction, and not as the outcome of a weighted average of quantitative benefits multiplied by quantitative probabilities". For this reason, human mind started to exploit ML techniques through for example the so called robo-advisors by seeking low-cost and automated investment advice. These allow investors to set up customized portfolios and give access to other wealth management services such as retirement income planning, portfolio tax efficiency and, in particular, cash flow forecasting, being superior to human financial advisors, as the latter have been shown to display behavioural biases and cognitive limitations (Linnainmaa, et al., 2018). As a result, robo-advisors are quickly attracting attention from policymakers and investors who need less effort to manage their investment portfolio (Rossi and Utkus, 2021).

1.2.1 Literature review

Our work contributes to the finance and economics literatures on stock return prediction which has already made a large number of attempts to add evidence of stock returns and volatility by developing itself during the years.

Starting from the Capital Asset Pricing Model[8] (CAPM), nowadays we are arrived at more complex and efficient methods for stock return prediction through ML techniques. Literature comes from in two basic strands: cross-sectional (i.e. observations that come from different individuals or groups at a single point in time) and time-series (i.e. set of

[8] It assumes that asset's beta with respect to the market portfolio is a sufficient statistic for the cross section of expected returns. See Sharpe (1964), Lintner (1965), and Mossin (1966).

observations collected at usually discrete and equally spaced time intervals) regressions of future stock returns[9].

In the first strand, a relevant contribution is given by Fama and French (1992) with a three-factor model composed by market return and only 2 factors: size risk and a value risk able to explain cross section of stock returns. Following the publication of Fama and French (1992), many researchers tried to identify new characteristics[10] and factor not explained by the three-factor model. They tried to ask the question "Which characteristics really matter by providing independent information about returns?". In order to find these, researchers first have to identify a subset of candidate predictors and then, they have to estimate the quality of these predictors. Authors in this literature usually consider their proposed return predictor in isolation without conditioning on previously discovered return predictors[11], in particular by employing Fama and MacBeth (1973) regressions[12] to combine the information in multiple characteristics. See for instance Harvey et al. (2015), Lewellen (2015), McLean and Pontiff (2016) and Green et al. (2017) who considering a comprehensive set of 94 firm characteristic conclude that relatively few characteristics affect cross-sectional value-weighted expected return. In addition, Han et al. (2019) re-examine the information content of the 94 firm characteristics by applying a robust forecast combination approach to the cross section of returns.

Gagliardini, Ossola, and Scaillet (2016) develop a weighted two-pass cross-sectional regression method to estimate risk premium from an unbalanced panel of individual stocks. Giglio and Xiu (2016) instead use a three-pass regression method that combines principal component analysis[13] (PCA) and a two-stage regression framework to estimate consistent[14] factor risk premium also when not all factors in the model are specified.

[9] There are also papers that combine elements from both cross-sectional and timeseries regressions; see Back et al. (2015), Baker et al. (2017), and Feng et al. (2017).
[10] Size, Market Value, Asset Growth, Momentum, ROA, ROE, Leverage, Beta, Risks etc.
[11] See DeMiguel, Martin-Utrera, Nogales, and Uppal (2016) who extend the parametric portfolio approach of Brandt et al. (2009) to study which characteristics provide valuable information for portfolio optimization.
[12] The Fama–MacBeth regression is a method used to estimate parameters for asset pricing models such as the CAPM. The method estimates the betas and risk premium for any risk factors that are expected to determine asset prices.
[13] Data simplification technique used in factor model to reduce the number of variables describing a set of data to a smaller number of latent variables, limiting the loss of information as much as possible. PCA has been usen to find asset pricing factors among others by Connor and Korajczyk (1988), Connor and Korajczyk (1993), Kozak et al. (2017), Kelly et al. (2017) and Fan et al. (2016).
[14] An estimator is consistent if, as the information increases, i.e. the sample size, its probability distribution is concentrated in correspondence with the value of the parameter to be estimated.

Kelly, Pruitt, and Su (2017) generalize standard PCA to allow for time-varying loadings[15] and extract common factors from the universe of individual stocks. Light et al. (2017) use partial least squares (PLS) to aggregate information on firm characteristics. Huang et al. (2018) demonstrate that in pricing portfolios a reduced-rank approach (RRA) outperforms the Fama-French five-factor model. In addition, among others, Bryan et al. (2018) proposed the Instrumented Principal Component Analysis (IPCA), that allows for latent factors and time-varying loadings[16].

Conversely, in the second strand (i.e. time-series regressions) we can find Welch and Goyal (2008), Koijen and Nieuwerburgh (2011), Rapach, Strauss, and Zhou (2010)[17] and Rapach and Zhou (2013). In particular, Welch and Goyal (2008) find that numerous economic variables with in-sample predictive ability[18] for the equity premium fail to deliver consistent out-of-sample forecasting gains relative to the historical average. Before them, in-sample test has already generated several doubts regarding their accuracy because regressors considered are very persistent[19] (i.e. the value of the variable at a certain date is closely related to the previous value) and data snooping bias (i.e. when exhaustively searching for combinations of variables, the probability that a result arose by chance grow with the number of combinations tested) is a concern if researchers who test for many model specifications report only the significant ones, being less severe compared to out-of-sample ones[20]. Furthermore, in-sample analysis could be interest only from an econometric point of view due to its ex-post assessment, instead out-of-sample tests better help investors to exploit conditioning information for return predictability in real time by choosing ex-ante models[21].

Due to these advantages, a growing body of literature has developed out-of sample tests. See for instance Pesaran and Timmermann (1995, 2000), Bossaerts and Hillion (1999),

[15] Factor loadings are correlation coefficients between observed variables and latent common factors.
[16] PCA has problems identifying factors with a small variance that are important for asset pricing. See also Lettau and Pelger (2018).
[17] They show that combination forecast acts as a shrinkage device, so mitigating overfitting and improving time-series forecast of risk premium.
[18] Roze (1984), Fama and French (1988), Campbell and Shiller (1988a,b), Kothari and Shanken (1997) and Ponti and Schall (1998) find that valuation ratios predict stock returns, particularly so at long horizons; Fama and Schwert (1977), Keim and Stambaugh (1986), Campbell (1987), Fama and French (1989), Hodrick (1992) show that short and long-term treasury and corporate bonds explain variations in stock returns; Lamont (1998), Baker and Wurgler (2000) show that variables related to aggregate corporate payout and financing activity are useful predictors as well.
[19] See Nelson and Kim (1993), Stambaugh (1999), Campbell and Yogo (2006) and Lewellen et al. (2010).
[20] See Lo and MacKinlay (1990), Bossaerts and Hillion (1999) and Sullivan et al. (1999).
[21] For exceptions, see Dangl and Halling (2008) and Johannes et al. (2009).

Marquering and Verbeek (2005), Campbell and Thompson (2008), Goyal and Welch (2003). Nevertheless, results by out-of-sample studies are also related to the model specification and the conditioning variables employed, such as the data frequency[22]. Many literatures are focused for simplicity on simple linear regression forecasting without considering that most asset pricing theories do not imply linear relationships between the equity premium and the predictor variables[23]. Approaching the forecasting through standard non-parametric methods is generally not a viable option because these methods encounter "curse-of-dimensionality" problems when the size of the conditioning information set increases, so overfitting concerns[24]. In addition, another consideration has to be made for transaction costs associated with trading individual characteristics in order to optimize investor's portfolio. Papers have found that combining characteristics helps to reduce transaction costs[25]. Frazzini et al. (2015) explains that value and momentum[26] trades tend to offset each other, resulting in lower turnover which has real transaction costs benefits. DeMiguel et al (2019) show how transaction costs change the number of jointly significant characteristics for an investor's optimal portfolio, so changing the dimension of the cross section of stock returns.

In general, traditional statistical methods as regressions and portfolio sorts are ill-suited to handle the large numbers of predictor variables the literature has accumulated over these years. ML tools help to overcome these issues, helping researchers to analyse Big Data. For this reason, the theoretical framework for ML algorithms started to be studied. The most popular machine learning technique is the penalized regressions called "least absolute shrinkage and selection operator" (Tibshirani 1996, LASSO)[27]. The main characteristic of this method is to avoids the overfitting problem by using a penalty function that reduces regression coefficients to zero. This technique appears for the first time in finance with Repach et al. (2013) to predict monthly cross-sectional returns using

[22] Stock returns are found to be more predictable at quarterly, annual or longer horizons, while returns at the monthly frequency are generally considered the most challenging to predict.

[23] It follows that misspecification implied by linear regressions is economically large as well as training dataset overfitting. In fact, that simpler predictive regressions perform better out-of-sample due to overfitting (DeMiguel, Garlappi, and Uppal, 2009).

[24] The common practice is to use linear models and reduce the dimensionality of the forecasting problem by way of model selection and/or data reduction techniques, but these methods exclude large portions of the conditioning information set and therefore potentially reduce the accuracy of the forecasts.

[25] See, among others, Korajczyk and Sadka (2004), Barroso and Santa-Clara (2015), Novy-Marx and Velikov (2016) and Chen and Velikov (2017)

[26] Rate of acceleration of a security's price. It means the speed at which the price is changing.

[27] See Hastie, Tibshirani, and Friedman (2001) for a general introduction.

lagged returns of all countries and was then practiced by many other several researchers. See Goto and Xu (2015), Huang and Shi (2016), Chinco et al.(2018) and Messmer and Audrino (2017). More recently, Gu et al. (2019) employ LASSO to analyze the time-series predictability of monthly individual stock returns, while Chinco et al. (2019) use the LASSO to predict individual stock returns one minute ahead using the entire cross-section of lagged returns as candidate predictors. Freyberger et al. (2019) apply a non-parametric version of the LASSO to study which characteristics provide incremental information for the cross-section of expected returns, instead Kozak et al. (2019) use the LASSO in a Bayesian context to model the stochastic discount factor based on a large number of firm characteristics.

In addition to these famous ML tool, several papers apply neural networks to forecast derivatives prices[28]. More recently, also Deep Learning solutions have been developed. See Chen et al (2019), Ka Ho Tsang et al. (2019) and Feng et al. (2019) who impose a no-arbitrage constraint by using a set of pre-specified linear asset pricing factors and estimate the risk loadings with a deep neural network[29]. Sirignano, Sadhwani, and Giesecke (2016) estimate a deep neural network for mortgage prepayment, delinquency, and foreclosure or Heaton, Polson, and Witte (2016) for financial prediction problems such as those presented in designing and pricing securities, constructing portfolios, and risk management. At last, we can find Khandani, Kim, and Lo (2010) and Butaru et al. (2016) who use regression trees methodology in order to predict consumer credit card delinquencies and defaults, as well as Rossi and Timmermann (2010) who adopt a novel semi-parametric statistic method known as Boosted Regression Trees (BRT) to study the relation between risk and return.

1.3 Research Objective

Our paper contributes to the long-standing literature assessing the predictability of stock returns and helps justify the growing role of ML throughout the architecture of the burgeoning FinTech industry. Non-linear methods like Tree-based and Neural Network (NN) are the best performing ML techniques for predicting stock returns (Gu et al., 2018).

[28] See among others Hutchinson, Lo, and Poggio (1994), Yao et al. (2000).
[29] Their analysis considers various sets of sorted portfolios but is not applied to individual stock returns.

After comparing their efficiency in our specific context, we decided to focus our research on the first method. Indeed, NN's use in finance industries is very limited, rather used in IT environments for chatbots content creations, voice assistants, etc. Tree-based modelling is an excellent alternative to linear regression analysis for three main reasons: first, it is used both for numerical and categorical data; second, it can handle data that are not normally distributed; third, it is easy to represent visually, making a complex predictive model much easier to interpret.

In this paper we adopt the BRT method employed by Rossi (2018) who forecast stock returns and volatility at a monthly frequency basing on the S&P500 portfolio . BRT is a statistical method that generates forecasts basing on a large set of conditioning information without imposing strong parametric assumptions such as linearity or monotonicity. The result gives to an increase in the stability of the forecasts and therefore protects it against overfitting. Indeed, it does not overfit because it performs a type of model combination that features elements such as shrinkage[30] and subsampling[31]. These features are really important for us in order to condition our forecasts on all the conditioning variables that literature has been considered so far. Moreover, BRT helps us to assess the relative importance of the various predictors at forecasting risk premium and volatility, giving important insights on the limitations of linear regression.

We extend Rossi (2018) analysis trying to replicate his results employing a different market index: EURO STOXX 50[32]. We have a deep investigation on how a representative mean-variance investor should exploits publicly available information to formulate forecasting on excess return and volatility. In fact, much of the literature we presented before doesn't focus on analysing the economic value associated for a representative investor. Therefore, our research answers three questions. The first is whether macroeconomic and financial variables contain information about expected stock returns and volatility that a mean-variance investor[33] could benefit from. Indeed, in order to be exploited the out-of-sample forecast require not only that parameters are estimated

[30] Shrinkage is the reduction in the effects of sampling variation.
[31] Alternative method for approximating the sampling distribution of an estimator.
[32] EURO STOXX 50 is the equity index of the major companies in the eurozone. It is made up of 50 stocks from the 11 eurozone countries: Austria, Belgium, Finland, France, Germany, Ireland, Italy, Luxembourg, the Netherlands, Portugal and Spain.
[33] Modern portfolio theory (MPT) is a mathematical framework for the construction of assets' portfolio such that the expected return is maximized for a certain level of the investor's risk.

recursively, but also that conditioning information employed are selected in real time. The conditioning information we use are the twelve predictor variables proposed so far by Welch and Goyal (2008). We start our analysis by forecasting separately both excess return and volatility of the market index. Our estimates demonstrate that BRT forecasts outperform the Multivariate Linear Model and the GARCH(1,1) with a more significant market timing in both returns and volatility, respectively.

The second question we answer is whether the conditioning information contained in macroeconomic and financial time-series can be exploited to directly select the optimal portfolio weights. We adopt Rossi (2018) approach as a semi-parametric method[34] that avoids the so-called "curse of dimensionality". Tests of optimal portfolio weights predictability demonstrate that portfolio allocations are time-varying and forecastable with a better performance of the one-step BRT compared to the statistical methods adopted as benchmark.

The third question is whether the results forecasts are economically valuable in terms of portfolio allocations profitability. We assess this by computing Mean, Standard Deviation, Sharpe Ratio, VaR and Maximum drawdown of the monthly portfolio returns obtained from our predictor variables also when short selling and borrowing constraints are taken into account.

Furthermore, we compute the realized utilities level obtained by the investor by comparing BRT-based strategies and passive ones. Our findings show the greater performance of the BRT-based model against the passive strategies of portfolios 60/40 and 100% market both accounting and not accounting for estimation uncertainty.

Definitely, our work contributes to the household financial literature by exploiting BRT also in robo-advisory, including portfolio composition and risk-adjusted return[35]. Investors may also obtain financial education benefits, or emotional rewards such as improved financial well-being and peace of mind[36]. Indeed, financial advising can help mitigate under-diversification in order to realize better outcomes, as well as reducing traditional advising cost that are too costly for retail investors compared to FinTech ones.

[34] It is the semi-parametric counterpart of Ait-Sahalia and Brandt (2001) methodology.
[35] See Kim, Maurer, and Mitchell (2016).
[36] See Rossi and Utkus (2019).

Lastly, our study is also relevant to the growing literature on technology adoption, crucial determinants of economic growth[37].

The rest of the paper is organized as follows. Chapter 2 introduces our empirical framework and describes how stock returns and volatility are predicted. Chapter 3 implements BRT to directly select optimal portfolio allocations. Chapter 4 presents and discuss our result for the out-of-sample accuracy of the model, conducts formal tests of market timing in both returns and volatility and evaluates the empirical trading strategies performance. At the end, a brief conclusion.

[37] See Romer (1990) and Aghion and Howitt (1992)

CHAPTER 2

Boosted Regression Trees Forecasting Framework

Summary: 1 – Modern Portfolio Theory. – 2 – Decision Tree Learning. – 3 – Conditioning Information. – 4 –Predicting Optimal Portfolio Allocations.

2.1 Modern Portfolio Theory

In order to introduce our empirical framework on how a representative mean-variance investor should exploits publicly available information to formulate forecasting on excess return and volatility, it is important to have a brief overview of what the Modern portfolio theory is based on. This subject has been extensively tackled by practitioners and academic researchers since of the introduction by Markowitz in 1952 due to its main limitation like the high sensitivity to historical data and the impossibility to include investors views.

The Modern Portfolio Theory (MPT) refers to an investment theory that allows investors to assemble an asset portfolio that maximizes expected return for a given level of risk by assuming that investors are risk-adverse (i.e. for an higher level expected return, rational investors prefer the less risky portfolio, s.t. they want to be compensated with higher expected return if an higher level of risk occurs[38]). The main ideas being that risk and return should be considered together with the aim to diversify the portfolio. This is explained by the fact that when adding negative correlated assets to a portfolio, the losses incurred by one may be offset by the gains of the others.

Given a set of assets, constructing a portfolio consists at choosing the weights to be assigned at each asset component according to performances goals criteria. It results from an optimization problem where many unknow market parameters like expected return,

[38] This concept is called Mean-Variance Criterion.

volatility and correlation, have to be accurately estimated. Every time that these parameters evolve in time, the portfolio has to be rebalanced by re-deriving the assets' weights according to the new incoming data (conditioning information). We can see the mean as an approximation of returns and the standard deviation (i.e. square root of variance) as an approximation of risk. From the mean-variance trade-off, a rational investor should either maximize his return for a given level of risk or minimize his risk for a given level of returns.

The exact trade-off between risk and expected return will not be the same for all investors because they evaluate it differently based on individual risk aversion characteristics. In 1947, John von Neumann and Oskar Morgenstern proved that any individual has an utility function which can be increased (i.e. higher return or lower risk) or decreased (i.e. lower return or higher risk). In the basic mean-variance optimization, we assume an investment on a single time period. The utility function takes the form:

$$U_t = E_t(r_{t+1}) - \frac{1}{2} A \, {\sigma_t}^2(r_{t+1}) \qquad (1)$$

where $E_t(r_{t+1})$ is the expectation of the return at time t+1 given the information available as of time t, ${\sigma_t}^2(r_{t+1})$ is the variance at time t+1 given the information available as of time t and, finally, A is the so called "risk-aversion coefficient" which measure the marginal reward required for additional risk[39].

According to the combination of risk and return pairs acceptable to an investor for a given level of utility function we obtain the indifference curves[40] where all the points on each curve have the same utility (i.e. investor is indifferent) as is shown in fig. 2.1.

[39] Risk averse: A > 0; Risk neutral: A = 0; Risk seeking: A < 0.

[40] Graphically the Risk is measured through Standard Deviation which is the square root of Variance. It is used to determine how widely spread out the assets' value movements are over time. Obviously, assets with a wider range of movements carry higher risk.

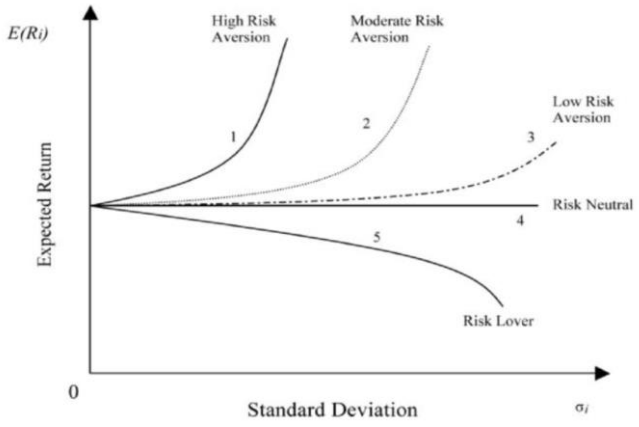


Fig. 2.1: Indifference curves
Source: Chartered Financial Analyst (CFA) institute, 2017

In constructing portfolios, investors often combine risky assets with risk-free assets to reduce risks. A complete portfolio is defined as a combination of a risky asset and a risk-free one[41]. The Expected Return of this portfolio will be a weighted average of its individual assets' expected returns and it is calculated as:

$$E_t(r_{p,t+1}) = \omega_1 E_t(r_{i,t+1}) + (1 - \omega_1) r_{f,t+1} \qquad (2)$$

where $E_t(r_{i,t+1})$ is the expected return of the risky asset whose weight in the portfolio is ω_1 , and $r_{f,t+1}$ is the expected return of the risk-free asset whose portfolio's weight is the remaining part of it, so $(1 - \omega_1)$[42].

Conversely, the Variance of this portfolio will be:

$$\sigma^2{}_t(r_{p,t+1}) = \omega_1^2 \sigma_t^2(r_{i,t+1})\, \omega_2^2 \sigma_t^2(r_{i,t+1}) + 2\omega_1 \omega_2 \rho_{1,2} \sigma_t(r_{i,t+1}) \sigma_t(r_{i,t+1}) \qquad (3)$$

Where$\rho_{1,2}\sigma_t(r_{i,t+1})\sigma_t(r_{i,t+1})$ is the covariance[43] of the two asset whose correlation coefficient is $\rho_{1,2}$. Being the second asset of our portfolio a risk-free one, obviously its

[41] In the mean-variance optimization framework is assumed normally distributed returns for the risky assets. All assets carry some degree of risk, therefore, assets that generally have low default risks and fixed returns are considered risk-free such as government Treasury Bonds.
[42] $\omega_1 + \omega_2 = 1 \Rightarrow \omega_2 = (1 - \omega_1)$.
[43] When the covariance of the two assets is positive, the Variance of the Portfolio will be higher, and vice versa if the covariance is negative. Since variance represents risk, diversification techniques are used in order to minimize portfolio risk by investing in risky assets with negative covariance with the aim to eliminate the so called idiosyncratic (specific) risk of the individual asset by leaving only the systemic risk (i.e. market portfolio risk).

variance, as well as its covariance with the risky asset, are zero, such that the Expected Risk represented by the Standard Deviation of the portfolio will be simply:

$$\sigma_t(r_{p,t+1}) = |\omega_1|\sigma_t(r_{i,t+1}) \tag{4}$$

Where $\sigma_t(r_{i,t+1})$ is the Standard Deviation of the risky asset whose weight is ω_1 as already seen[44]. In fact, $\sigma_t(r_{f,t+1})$ will be zero because of its free-risk characteristic.
By reformulating (4), we obtain:

$$\omega_1 = \frac{\sigma_t(r_{p,t+1})}{\sigma_t(r_{i,t+1})} \tag{5}$$

In addition, by reformulating (2), we obtain:

$$E_t(r_{p,t+1}) = r_{f,t+1} + [E_t(r_{i,t+1}) - r_{f,t+1})]\omega_1 \tag{6}$$

Therefore, by replacing (5) in (6), the Expected Return of the Portfolio will be:

$$E_t(r_{p,t+1}) = r_{f,t+1} + \frac{E_t(r_{i,t+1}) - r_{f,t+1}}{\sigma_t(r_{i,t+1})}\,\sigma_t(r_{p,t+1}) \tag{7}$$

where $E_t(r_{i,t+1}) - r_{f,t+1}$ represents the excess return of the risky asset. The graphical line obtained with (6) is the so-called Capital Allocation Line[45] whose slope is called Sharpe Ratio, or reward-to-risk ratio, which measures increase in expected return per unit of additional standard deviation.

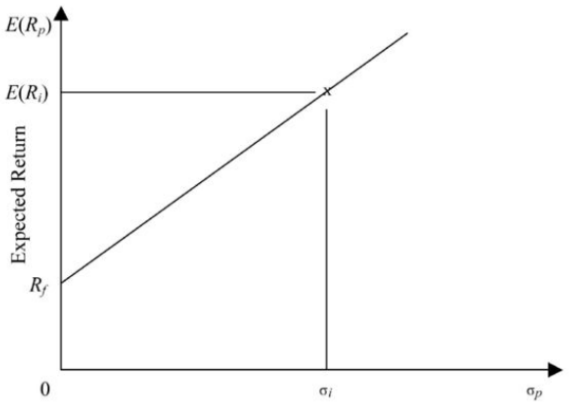


Fig. 2.2: Capital Allocation Line

[44] The absolute value is applied because the variance has to be always a positive value.
[45] It is called Capital Market Line if the risky asset is the market portfolio.

Source: Chartered Financial Analyst (CFA) institute, 2017

Graphically, the optimal portfolio consists of a risk-free asset and a risky asset portfolio is the point where the CAL is tangent to the Indifference Curve of the investor[46].

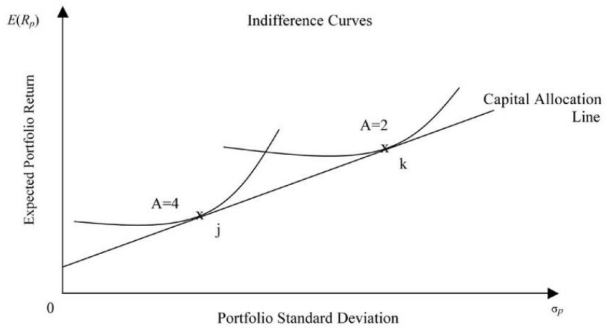


Fig. 2.3: Optimal Portfolio with a risk-free asset and a risky asset
Source: Chartered Financial Analyst (CFA) institute, 2017

Conversely, by applying the Utility Theory to this portfolio, the investor will choose his asset allocation by maximize its utility function, leading to the optimal portfolio weights:

$$\omega^*_{t+1|t} = \frac{E_t(r_{i,t+1}) - r_{f,t+1}}{A\,\sigma_t^2(r_{i,t+1})} \tag{8}$$

As we said at the beginning of this chapter, the Expected Return and the Variance we are considering represent conditional expectations on t+1 on the basis of the investor's conditioning information at time t. Most of the literatures consider these conditional expectations to be linear function of observable macroeconomic and financial time-series, but this may be misleading in terms of forecasting accuracy and portfolio allocation profitability. For this reason, in this paper we consider these conditional expectations to be non-linear functions of observable macroeconomic and financial time-series.

2.2 Decision Tree Learning

Decision tree learning or induction of decision trees is a statistics predictive model used in data mining and machine learning which employ a decision tree as a decision support

[46] For the combination of risky assets, the Efficient Frontier is considered instead of the Indifference Curve. For more information see Markowitz, Todd and Sharpe (2000).

tool that uses a tree-like model of decisions to go from initial observations to one or more target variables. Among all the categories of decision trees, regression trees represent the ones for estimating the relationship of a real number target variable (i.e. dependent variable) basing to the changes in different predictor input variables (i.e. independent variables). Decision trees are among the most popular ML algorithms given their intelligibility and simplicity (Piryonesi and El-Diraby, 2020) and their capability to incorporate multiway predictor interactions (Gu et al., 2018).

A tree "grows" in a sequence of steps where a new "branch" sorts the data leftovers from the preceding step into bins based on one of the predictor variables. This sequential branching divides the space of predictors into rectangular partitions where the simple average of all the outcome variable's value in each partition represents the forecast observation in that partition. Next, one or both of these binary partitions is split into two additional ones, and so on until a stopping criterion is reached. A basic example of this system can be seen in the figure 2.4.

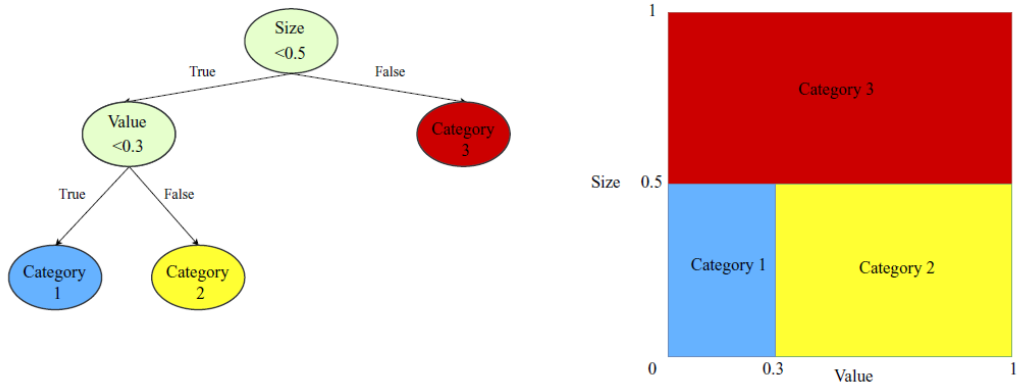


Fig. 2.4: Regression tree example
Left figure presents the diagrams of a regression tree where the terminal nodes of the tree are coloured in blue, yellow, and red. Based on their values of these two characteristics, the sample of individual stocks is divided into three categories (partitions) in the right figure.
Source: Gu et al. (2018)

More formally, suppose we have P potential predictor independent variables x_t with $x_t = x_{t,1}, x_{t,2}, \dots, x_{t,P}$ and a single dependent variable y_{t+1} over T observations for t = 1,…,T. Fitting a regression tree requires deciding about which predictor variables to use to split the sample space and which split points to use (e.g. in figure 2.4 has been chosen size and value as predictors and 0,5 and 0,3 as split points for respectively the size and the value).

The regression trees we use employ recursive binary partitions[47] so the fit of such regression tree can be written as[48]:

$$\mathcal{T}\left(x;\{S_j,c_j\}_{j=1}^{J}\right)=f(x)=\sum_{j=1}^{J}c_j\,I\{x\in S_j\} \tag{9}$$

where S_j with j =1,….J is one of the terminal nodes (i.e. categories or partitions) we split the space of the predictors into, c_j is the constant associated at each partition to model the dependent variable and it is defined to be the sample average of the outcomes within the partition and, finally, $I\{x\in S_j\}$ is an indicator variable. For instance, the regression tree represented in the figure 2.4 has the following prediction equation:

$$\mathcal{T}\left(x;\{S_j,c_j\}_{j=1}^{3}\right)=\ c_1\,I\{size_t<0{,}5\}I\{value_t<0{,}3\}+\ c_2\,I\{size_t<0{,}5\}I\{value_{t,p}\geq 0{,}3\}+\ c_3\,I\{size_t>0{,}5\}$$

As we already said, by using BRT is necessary to define the optimal splitting point, but this is a complex issue, in particular when the number of potential predictors variables is quite large. Hence, a sequential greedy algorithm[49] is employed. Using the full set of data, the algorithm employs a splitting variable p and a split point s to construct half-planes[50]:

$$S_1(p,s)=\{X|X_p\leq s\}\qquad\text{and}\qquad S_2(p,s)=\{X|X_p>s\} \tag{10}$$

that minimize the sum of squared residuals:

$$\min_{p,s}\left[\min_{c_1}\sum_{x_t\in S_1(p,s)}^{J}(y_{t+1}-c_1)^2+\min_{c_2}\sum_{x_t\in S_2(p,s)}^{J}(y_{t+1}-c_2)^2\right] \tag{11}$$

Finally, for a given choice of p and s, the constant values c_j assumes the value of:

$$\widehat{c_1}=\frac{1}{\sum_{t=1}^{T}I\{x_t\in S_j(p,s)\}}\sum_{t=1}^{T}y_{t+1}\,I\{x_t\in S_j(p,s)\} \tag{12}$$

[47] We focus on recursive binary trees for simplicity. For more complex structures see Breiman et al. (1984).
[48] Our methodology's description draws on Hastie et al. (2009) and Rossi and Timmermann (2010) who provide a more in-debt coverage of the approach.
[49] A greedy algorithm is a simple, intuitive algorithm used in optimization problems. The algorithm makes the optimal choice at each step as it attempts to find the overall optimal way to solve the entire problem.
[50] A half-plane is a planar region consisting of all points on one side of an infinite straight line, and no points on the other side.

The best splitting pair (p, s) in the first iteration can be determined by searching through each of the P predictor variables. After having obtained the best partition from the first step, the data is then partitioned into two additional predictors and the splitting process is repeated for each of the subsequent partitions. The choice of splitting variable definitely performs variable selection because if a predictor variable is never used to split the sample space, this means that it will not influence the fit of the model.

Regression trees are generally employed in high-dimensional datasets where the relation between predictor and predicted variables is highly non-linear. This is relevant when modelling stock returns as target variable r_{t+1} , because a very incremental number of predictor variables x_t has been proposed so far in the literature and the theoretical frameworks rarely imply a linear or monotonic relation between predictor and predicted variable. Indeed, advantages of the tree model are the invariance to monotonic transformation of predictors, the possible accommodation of both categorical and numerical data in the same model to better represents non-linearities, and its capability to capture j – 1 interactions. On the other hand, the approach is sequential (i.e. successive splits are performed on ever more few observations) and this increase the risk of overfitting, so without any guarantee on a globally optimal solution. For this reason, trees must be heavily regularized as primary defence against overfitting to improve models' out-of-sample predictive performance. Different regularization methods for model selection and mitigation of overfit, have been developed by literature. The regularization procedures rely on the choice of hyperparameters ("tuning parameters") including, for instance, the penalization parameters in lasso and elastic net, the number of iterated trees in boosting, the number of random trees in a forest, and the depth of the trees[51]. Among all the regularization procedures, we employ the method known as boosting.

2.2.1 Boosting

Boosting consists in the idea that many oversimplified learner regression trees, should form a single "stronger learner" with greater stability, in order to lead to more accurate forecasts than those available from individual model, so reducing the overfitting risk and, subsequently, the accuracy of the forecasts[52]. Boosted regression trees combine the

[51] For more detailed insights see Gu et al. (2018).
[52] For similar results in the context of linear regression see Rapach et al. (2010).

strengths of two algorithms: regression trees (models that relate a response to their predictors by recursive binary splits) and boosting (an adaptive method for combining many simple models to give improved predictive performance). Boosting algorithms iteratively re-weight data used in the initial fit model by adding new trees s.t. the weight of observations modelled poorly by the existing tree is increase, thus they have an higher probability of being selected in the new tree. This means that after the first tree is fitted, the model will take into account the residual of that tree to fit the next tree, and so on. The process continues until some stopping criterion is reached. The number of trees used in the summation is also known as the number of boosting iterations. Formally, a boosted regression tree (BRT) is simply the sum of regression trees for B boosting interactions:

$$f_b(x) = \sum_{b=1}^{B} \mathcal{T}_b\left(x; \{S_{b,j}, c_{b,j}\}_{j=1}^{J}\right) \tag{13}$$

Where $\mathcal{T}_b\left(x; \{S_{b,j}, c_{b,j}\}_{j=1}^{J}\right)$ is the regression tree used in the b[th] boosting interaction. Given the model fitted up to the (b – 1)[th] boosting interaction (i.e. $f_{b-1}(x)$), the subsequent boosting interaction $f_b(x)$ seeks to find parameters $\{S_{b,j}, c_{b,j}\}_{j=1}^{J}$ for the next tree to solve the minimization problem:

$$\{S_{b,j}, c_{b,j}\}_{j=1}^{J} = \min_{\{S_{b,j}, c_{b,j}\}_{j=1}^{J}} \sum_{t=0}^{T-1} \left[y_{t+1} - \left(f_{b-1}(x_t) + \mathcal{T}_b\left(x_t; \{S_{b,j}, c_{b,j}\}_{j=1}^{J}\right)\right)\right]^2 \tag{14}$$

At the end, for a given set of $S_{b,j}$, the optimal constant $c_{b,j}$ in each partition is derived iteratively from the solution of the minimization problem:

$$\hat{c}_{j,b} = \min_{c_{b,j}} \sum_{x_t \in S_{j,b}}^{J} \left[y_{t+1} - f_{b-1}(x_t) - c_{j,b}\right]^2 \tag{15}$$

where $y_{t+1} - f_{b-1}(x_t) = e_{t+1,b-1}$ that is the empirical error after b – 1 boosting interactions. Indeed, the problem to solve in (15) is given by the minimization of the average of the squared residuals $\sum_{t=1 \in S_{j,b}}^{T} e_{t+1,b-1}^2$ of the regression tree. Therefore, the obtained $\hat{c}_{j,b}$ represents the mean of the residuals in the j[th] partition.

Boosting is originally described in Schapire (1990) and Freund (1995) for classification problems to improve the performance of a set of weak learners. Friedman et al. (2000) and Friedman (2001) extend boosting to contexts beyond classification, leading to the gradient boosted regression tree (GBRT) procedure, that is the one we adopt with J=2 nodes. Following Rossi (2018) we adopt three refinements on the basic methodology. The first is *shrinkage*, which is a simple regularization technique to limit overfitting risk that slows the rate at which the empirical risk is minimized on the training sample. In order to do that, we use a shrinkage parameter $0 < \lambda < 1$ which determines how much each boosting iteration contributes to the overall fit:

$$f_b(x) = f_{b-1}(x) + \lambda \sum_{j=1}^{J} c_{j,b}\, I\{x \in S_{j,b}\} \tag{16}$$

Where $\sum_{j=1}^{J} c_{j,b}\, I\{x \in S_{j,b}\}$ is the regression tree $\mathcal{T}_b\left(x; \{S_{b,j}, c_{b,j}\}_{j=1}^{J}\right)$ at the b[th] boosting iteration whose forecast is added to the total with a shrinkage weight of λ, while $f_{b-1}(x)$ is the boosted regression tree fitted up to the last previous boosting interaction. This is iterated until there are a total of B trees in the ensemble. In order to set the value of λ, Friedman (2001) identifies as the best empirical strategy to set λ very small, equal to 0,001 and correspondingly increase the number of boosting iterations to refine the fit[53]. The second refinement is *subsampling* and is inspired by a more general procedure known as bootstrap aggregation, or bagging (Breiman, 1996). The baseline bagging technique consists into three steps: first, it extracts bootstrap samples of the data[54], secondly, it fits a separate regression tree to each and, finally, it averages their forecasts, therefore reducing the variance of the final predictions. In our specific context, at each boosting iteration we sample without replacement one half of the training sample and fit the next tree on the sub-sample obtained.

Finally, according to a large literature which suggests that squared-error loss[55] gives too much weight on observations with large absolute residuals, our analysis employ the *mean absolute errors*, i.e. $\frac{1}{T}\sum_{t=1}^{T}|y_{t+1} - f(x_t)|$, as function to minimize in order to obtain the

[53] See Rossi (2018) in comparing 3 boosting iterations against 5.000 ones.

[54] Bootstrapping is a statistics technique which randomly extract samples from the data in order to approximate the sample distribution of the data by replacing them. It is a resampling method.

[55] The squared-error loss measures the average of the squares of the errors (i.e. the average squared difference between the actual values y_{t+1} and the estimated values $f(x_t)$.

forecast which results to be the conditional median of y_{t+1} rather than the conditional mean given by the squared error loss minimization. By minimizing absolute errors, our regression model is likely to be more robust to outliers, thus reducing the overfitting risk, in particular for fat-tailed distributions such as those for stock returns and volatility.

2.3 Conditioning Information

Information plays a central role for investors who faces every day a lot of information which will affect their investment portfolio. For this reason, an ever-increasing method are developing to study the relation between risk and return.

The conditioning information we are use are the predictor variables already analysed in Welch and Goyal (2008) and by others subsequently. Their predictor variables were available during 1927-2005 and then updated by Welch website until 2020, but we cannot use them because they are referred to the S&P500 Index and the relative American market. For this reason, we replicate these 12 predictor variables[56] by deriving a new dataset referred to our EUROSTOXX50 Index and its relative market, over the sample period from May 2001 up to May 2021. All predictor variables are suitably lagged[57], indeed they are known at time t in order to forecast returns in t+1. The predictor variables consist to three large categories: "risk and return", "fundamental to market value" and "interest rate term structure and default risk"[58]. The first includes lagged returns (exc), long-term bond returns (ltr) and volatility (vol). The second contains the log dividend-price ratio (dp) and the log earnings-price ratio (ep). Finally, the third comprises the three-month T-bill rate (Rfree), the T-bill rate minus a three-month moving average (rrel), the yield on long term government bonds (lty), the term spread measured by the difference between the yield on long-term government bonds and the three-month T-bill rate (tms) and the yield spread between BAA and AAA rated corporate bonds (defspr). Following Rossi (2018), we also include inflation (infl) and the log dividend-earnings ratio (de) for a total of twelve predictor variables.

[56] Following Rossi (2018) a few variables were excluded from the analysis, including net equity expansion, the book-to-market ratio and the consumption wealth ratio.

[57] The lag time is the time between the two correlated time series (e.g. in a time series data at t=0,1,…,n, then taking the autocorrelation of data sets (0,1),(1,2)…(n−1,n) apart would have a lag time of 1. Conversely, if the autocorrelation of data sets to (0,2),(1,3),(n−2,n) that would have lag time 2 etc.)

[58] Additional details on data sources and the construction of these variables are provided by Welch and Goyal (2008).

Stock returns are tracked by EU STOXX 50. They represent the Annual Total Return, including dividends when holding such index. The EU STOXX 50 index is a basket of 50 large EU stocks, weighted by market cap, and is the most widely followed index representing the UE stock market. In order to obtain the excess returns, we subtract the risk-free rate by taking into account the 3-month German Bund return as benchmark for the European Market.

2.3.1 Conditional Stock Returns estimates

Conditional stock returns expectations are for simplicity generally obtained according to the following linear model:

$$E_t(r_{i,t+1}) = \hat{\mu}_{t+1|t} = \beta'_{\mu} x_t \tag{17}$$

where x_t represents a set of publicity available predictor variables and β_{μ} is a vector of parameter estimates obtained via ordinary least square (OLS) method.

Actually, the linear specification imposed brings to some misspecifications. In particular, asset pricing models suggest a wide array of economic variables for both returns and volatility which should be incorporated, but linear specification tends to over-fitting if the number of parameters to be estimated is large compared to the number of observations. Subsequently, researchers tend to exclude a large portion of the conditioning information available. In addition, theoretical frameworks rarely identify linear relations between the variables.

However, in our context, these misspecifications are not a source of concern because it does not mean a lower predictive accuracy, which is definitely what we are focused on for optimal portfolio allocation.

2.3.2 Conditional Volatility estimates

Conditional volatility estimates have been developed in different shapes in the recent past years. Following Rossi (2018), our research bases the estimation for conditional volatility on two main models. The first extends the linear framework presented by Paye (2010), Ludvigson and Ng (2007) and Marquering and Verbeek (2005). They estimate monthly return volatility according to the following linear specification:

$$\sigma_t(r_{i,t+1}) = \hat{\sigma}_{t+1|t}^{lin} = \beta_\sigma' x_t \tag{18}$$

where x_t represents the same set of publicity available predictor variables we employ for conditional expected returns and β_μ is a vector of parameter estimates obtained via ordinary least square (OLS) method. Even if linear estimation methods give interesting in-sample results, their out-of-sample performance was never satisfactory. For this reason, our first approach estimates conditional expected volatility via BRT:

$$\hat{\sigma}_{t+1|t}^{BRT} = f_\sigma(x_t|\hat{\theta}_\sigma) \tag{19}$$

where $\hat{\theta}_\sigma$ are estimates of the parameters obtained via Boosted Regression Tree.
The second model has the aim to forecast volatility according to the GARCH(1,1) where expected conditional variance is defined according to the following formula:

$$\hat{\sigma}_{t+1|t}^{GARCH} = \sqrt{\omega + \alpha\,(r_t - \mu_t)^2 + \beta\sigma_t} = \sqrt{\omega + \alpha\,\sigma_t\epsilon_t^2 + \beta\sigma_t} \tag{22}$$

where ω_d are the optimal weights computed with a specific function called "Beta weights":

$$\omega_d(k_1, k_2) = \frac{\left(\frac{d}{D}\right)^{k_1-1}\left(1-\frac{d}{D}\right)^{k_2-1}}{\sum_{d=0}^{D}\left(\frac{d}{D}\right)^{k_1-1}\left(1-\frac{d}{D}\right)^{k_2-1}} \tag{21}$$

where D represents the maximum lag length (set to 250 days). This estimation technique is also used to estimate the parameters α and β in (22).

2.4 Predicting Optimal Portfolio Allocations

In the previous paragraph we have observed how to forecast returns and volatility in separately ways with the aim to construct optimal portfolio allocations only in a second step. Our objective now is to directly forecast the optimal portfolio allocations given the same set of conditioning information we already seen. Secondary, we are able to assess whether such approach leads to more or less profitable forecasts.

As we seen in (8), conditional optimal portfolio weights for a mean-variance investor are given by the ratio of conditional expected excess return and variance, scaled by the risk-aversion coefficient that we set to 4 as a common choice in the literature. Therefore, in order to make forecasts, conditional optimal portfolio weights represent our target variable as follow:

$$\hat{\omega}^{real}{}_{t+1|t} = f_{\omega}(x_t|\hat{\theta}_{\omega}) \tag{27}$$

where x_t represents the same set of publicly available predictor variables we use for return and volatility and $\hat{\theta}_{\omega}$ are estimates of the parameters obtained via BRT.

There are few papers[59] as well as no economic theories which analyse the direct relationship between economic activity and the publicly conditioning information set variables. Brandt, Santa-Clara and Valkanov (2004) model directly the portfolio weight in each asset as a function of the asset's characteristics where the coefficients are found by optimizing the investor's average utility of the portfolio's return over the sample period. Rossi and Timmermann (2010) observe that the three most important predictors for equity premium are inflation, log earnings price ratio and the de-trended T-bill rate; while for volatility are the lag volatility, the default spread and the lag return on the market portfolio. Only more recently, Rossi (2018) analyses the relation between the optimal portfolio weights and the predictor variables. In his paper, it is highlighted that the predictors with the greatest Relative Influence (i.e. weight) are the log earnings price ratio (ep), the inflation (infl), the log dividend earnings ratio (de), the yield on long term government bonds (lty), the stock market volatility (vol) and the three-month T-bill rate (Rfree). In particular, the relationship between the optimal weight and the log earnings price ratio (ep) is positive, strongest at low or high levels of this ratio and weaker at medium levels. Regarding inflation (infl), at negative levels the relationship is either flat or rising[60] and, vice versa, at positive levels of inflation[61]. The log dividend earnings ratio (de) is inversely related to the conditional portfolio weights and its sensitivity is greatest for medium values of the ratio and weakest at low and high levels of the ratio. Finally,

[59] See Ait-Sahalia and Brandt (2001) for instance.
[60] Thus, in a state of deflation, rising consumer prices leads to a higher exposure to risky assets by the investor (i.e. more weight).
[61] Thus, higher consumer prices become bad news for stocks.

the relation between optimal portfolio weights and long-term government bonds, volatility and the T-bill rate are all highly nonmonotonic, s.t. the optimal investment in risky asset increase for low values of the predictors and decrease for high values of them. These findings imply strong non-linearities in the functional form relating the predictor and predicted variables. Therefore, comparing to linear models, BRT helps us in correcting their misspecification. In order to assess whether correcting for such misspecification translates into greater predictive accuracy, we assess next the out-of-sample forecasting performance of BRT versus the other common benchmark models we have illustrated in paragraph 2.3. We start our analysis by separately forecasting expected returns and conditional volatility of the EUROSTOXX50 Index and secondly uses these results as input for the optimal mean-variance portfolio allocation. Subsequently, we continue by directly estimating portfolio weights according to a one-step BRT. For comparison, we also present results for investment portfolios constructed using stock return predictions from a multivariate linear regression model on the full set of predictor variables and volatility predictions from the most common GARCH (1,1) model which bases the prediction only on the excess return predictor.

CHAPTER 3

Out-of-Sample Empirical Application's Results

Summary: 1 – Data. – 2 – Two Step BRT Model. – 3 – One Step BRT Model. – 4 – Portfolio Allocation Performance.

3.1 Data

We investigate the performance of the strategies discussed in the previous chapter for forecasting the monthly EUROSTOXX50 excess return, set as dependent variable. The dependent variable is always the equity premium, i.e., the total rate of return on the stock market minus the prevailing short-term interest rate. Throughout the empirical section we set May 2001 as the starting value for our out-of-sample evaluation period and May 2021 as end value. The choice is led by the need to have the largest training sample as possible for the European Index, so allowing us for a 20-year test sample.

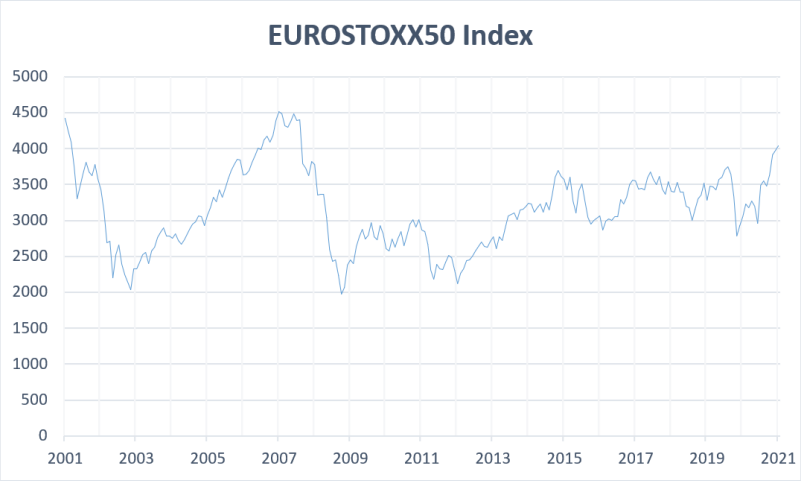


Fig. 3.1: EUROSTOXX50 Index price
Source: Own elaboration, 2021

Our first set of independent variables relate primarily to risk and return:

- Lagged returns (exc): we use EUROSTOXX50 index month-end prices from 2001:05 to 2021:05 from Refinitev Workspace (ex. Thomson-Reuters) add-in in Excel. Stock Returns are the continuously compounded returns on the EUROSTOXX50 index[62]. The risk-free rate taken into consideration to obtain excess return values in such European context is the three-month German Bund monthly Return from the period 2001:05 to 2021:05. Data are obtained from FRED website[63] by expressing them as percentage of the observed listed values and by considering negative values equal to 0.
- Long Term bond returns (ltr): taking into account the European index, our government bond used as benchmark is the German one. In the context of a long-term bond, we considered the 10Y German Bund price from the period 2001:05 to 2021:05 from Refinitev Workspace's month-end values in order to compute relatives returns by Excel.
- Volatility (vol): it is measured by the standard deviation of logarithmic daily returns computed through Excel Pivot table tool from the daily log returns on the EUROSTOXX50 Index prices for the period 2001:05 to 2021:05 from Refinitev Workspace's daily-end values.

Our second set of independent variables relate to fundamental to market value:

- Log dividend-price ratio (dp): it is the difference between the log of dividends and the log of prices. According to logarithms' properties, we compute it as the log of the dividend-price ratio which is directly obtained relative to the EUROSTOXX50 Index by Refinitev Workspace add-in in Excel. For monthly frequency, we can only begin in the Refinitev period which starts from May 2001, so we set this as our start period.
- Log earnings-price ratio (ep): it is the difference between the log of dividends and the log of prices. According to logarithms' properties, we obtain it as the log of the inverse PE monthly ratio which is directly observed relative to the

[62] We have to figure out if everything is defined in simple or log returns. Goyal and Welch explain it in their 2003 Management Science paper "Predicting the Equity Premium with Dividend Ratios") more clearly.
[63] https://fred.stlouisfed.org/series/IR3TIB01DEM156N.

EUROSTOXX50 Index on Refinitev Workspace add-in in Excel from the whole period 2001:05 to 2021:05.

Our third set of independent variables relate to fundamental to interest rate structure and default risk:

- T-bill rate (Rfree): we considered the three-month German Bund Return from the period 2001:05 to 2021:05. Data are obtained from FRED website by expressing them as percentage of the observed listed values. The provided rate is in simple, not log returns, so we compute the right one[64] by Excel.
- Relative T-Bill rate (rrel): given by the difference between the three-month German Bund rate and the moving average of the previous three months' values directly computed manually on Excel from FRED database's values.
- Long Term government bond yeld (lty): we considered the 10Y German Bund price from the period 2001:05 to 2021:05 from Refinitev Workspace's month-end values by expressing them as percentage of the observed listed values.
- Term Spread (tms): we manually obtain this variable on Excel by the difference between the long-term government bond Return and the T-bill rate.
- Default Spread (defspr): given by the Return spread between BAA and AAA rated corporate bonds. Monthly values for each rated corporate bond are obtained from FRED website. After setting them as percentage values, we manually compute the difference on Excel in order to obtain our defspr end-month values.
- Inflation (infl): we obtain monthly values for inflation by the Consumer Price Index (All items for the Euro Area) for the period 2001:05 to 2021:05 FRED database's values by expressing them as percentage of the observed values. Because inflation information is released only in the following month, in our monthly regressions, we inserted one month of waiting before use Inflation.
- Log dividend-earnings ratio (de): we compute these values by dividing our dp and ep predictor predictor's monthly values.

[64] Log return and simple return have the additivity property for, respectively, time-series and cross-section perspectives. In addition, stock return is always assumed to follow a Log Normal Distribution, so that Log return is used for statistical evaluation.

All predictor variables are appropriately lagged, i.e. they are known at time t for forecasting returns at time t+1.

Figure 3.2 portrays the 12 predictor variables for the 2001:05 to 2021:05 sample period. Visually, the predictors variables represent a variety of information sources[65].

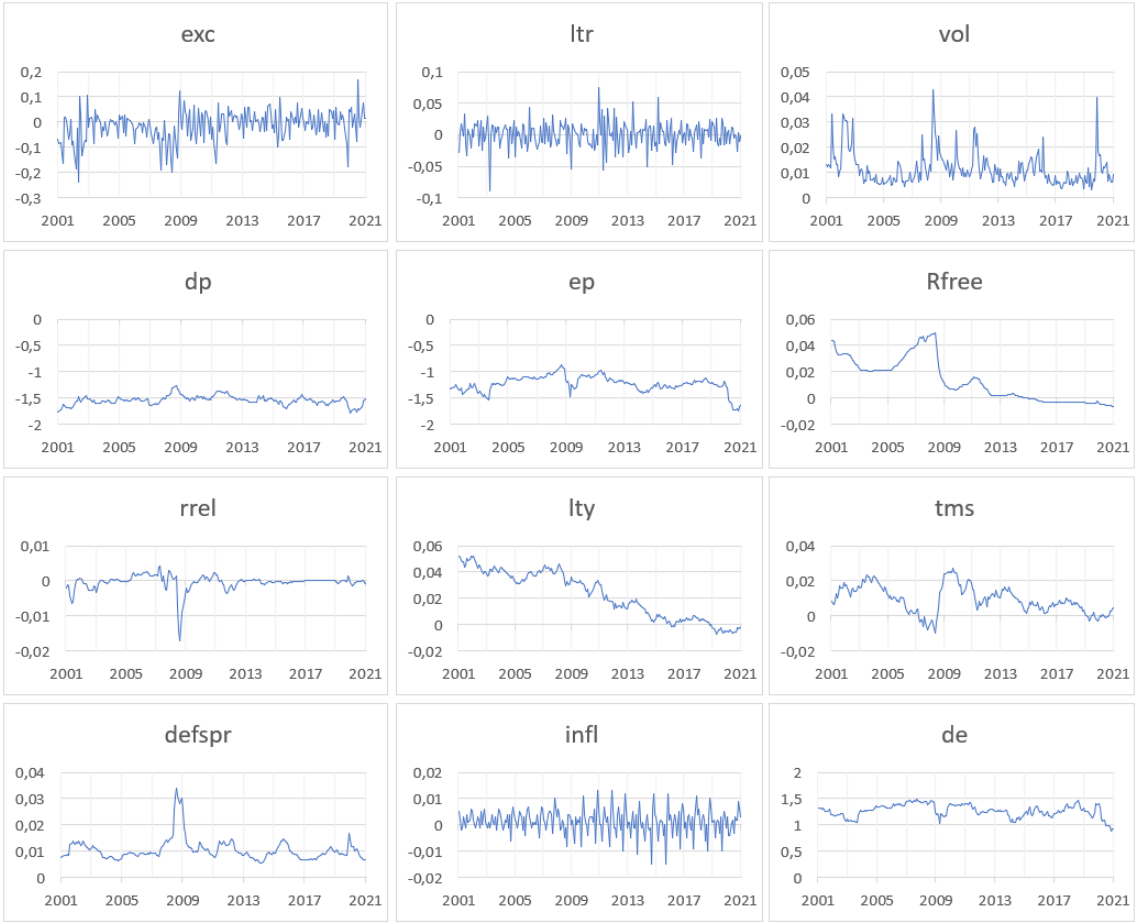


Fig. 3.2: Predictor Variables
Source: Own elaboration, 2021

We start our analysis from a two-step BRT which first forecasts separately expected returns and conditional volatility of the EUROSTOXX50 Index and secondly uses these results as input for the optimal mean-variance portfolio allocation[66]. Subsequently, we continue our analysis by directly estimating portfolio weights according to a one-step BRT[67]. For comparison, we also present results for investment portfolios constructed

[65] See Appendix B for detailed data of predictors.
[66] See section 2.3.
[67] See section 2.4.

using stock return predictions from a multivariate linear regression model on the full set of predictor variables and volatility predictions from the most common GARCH (1,1) model which bases the prediction only on the excess return predictor.

In order to develop our algorithms, we initially exploited scikit-learn[68] libraries and guides to then manually write the code based on our needs, assumptions and available data[69].

The process of writing starts from the download of the programming language Python and the pip installer[70]. Then, from the Command Prompt (cmd.exe) of Windows 10, we install pip and, subsequently, we use it to install other useful libraries[71]. After these preliminary activities, we were able to write the algorithms necessary for our purposes. First, it was necessary to load the data by read the predictors Excel file (properly converted into .csv format) in order to present all our predictor variables at time t and our dependent variable properly lagged at time t+1. Then, data are reprocessed: throughout the empirical section we split our data into a 17-year train sample period (i.e. 2001-2018) and a 3-year test sample (i.e. 2018-2021). The choice is driven by the need to have the largest training sample according to the European Index data. Finally, each regression model is fit such that the output is given, allowing us to test the out-of-sample forecasting accuracy.

In the following section, we present the out-of-sample performance of Boosted Regression Trees and other benchmark forecasting methods. We present the results in terms of Mean Squared Error (MSE)[72] as it is the most common loss function for forecasting problems with continuous dependent variables. In addition, we also present R^2 coefficient as it represents the proportion of the variance for our dependent variable that is explained by the predictors of the regression model. The R-squared, also called the coefficient of determination, is used to explain the degree to which input variables (predictor variables) explain the variation of output variables (predicted variables).

[68] Scikit-learn is a free online software of ML libraries for the Python programming language and it is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.
[69] See Appendix A for detailed information on libraries installation steps.
[70] pip is the package installer for Python used to install packages from the Python Package Index and other indexes.
[71] See Appendix.
[72] The L^2 norm loss function, also known as the least squares error (LSE), is used to minimize the sum of the square of differences between the target value and the estimated value. The MSE is the average squared difference between the estimated values and the actual values of our dependent variable.

Generally speaking, an higher R^2 indicates a better fit for the model. The best possible score is 1.0 and it can be negative (i.e. the model can have a worse fit than the horizontal line). It is given by the following formula:

$$R^2 = 1 - \frac{\sum_{t=1}^{T}(y_t - \hat{y}_t)^2}{\sum_{t=1}^{T}(y_t - \bar{y}_t)^2} \tag{28}$$

where $\sum_{t=1}^{T}(y_t - \hat{y}_t)^2$ is the Residual Sum of Squares given by the difference between the actual values of our dependent variable y_t and the predicted ones $\hat{y}_t$, while $\sum_{t=1}^{T}(y_t - \bar{y}_t)^2$ is the Total Sum of Squares given by the difference between the actual values y_t and its mean value $\bar{y}_t$ over the out-of-sample test period.

Finally, we compute mean directional accuracy (MDA) results as it is known since Leitch and Tanner (1991) that the latter forecast evaluation criterion is closely related to the profitability of financial forecasts. It is given by the mean of the times of correct direction of the time series forecasts. In simple words, MDA provides the probability that the model is able to correctly detect the direction of the time series. It is given by the formula:

$$\frac{1}{N}\sum_{t=1}^{T}\left[1_{sgn}\,(y_t - y_{t-1}) = sgn(\hat{y}_t - y_{t-1})\right] \tag{29}$$

where y_t is the actual dependent variable values over the out-of-sample test period, $\hat{y}_t$ is the predicted dependent variable, 1 is an indicator function that takes the value of 1 when the even happen and 0 when the event does not happen, while sgn() is a sign function that extract the sign of a real number.

3.2 Two Step BRT Model

3.2.1 Return

Fig. 3.3 reports the results for the forecasted monthly excess return in terms of R^2 and MDA. The results are reported for a Boosted Regression tree model with 10,000 boosting

interactions, as well as a competing model proposed in the literature: the Multivariate Linear Model[73].

	R^2	MDA
Boosted Regression Tree	-1,28 %	57,14%
Multivariate Linear Model	-6,31 %	45,71%

Fig. 3.3: Out-of-sample forecasting accuracy for Excess Return forecasts
Source: Own elaboration, 2021

Over the sample 2001-2021 BRT outperforms the benchmark by achieving an out-of-sample R^2 of -1,28 %, which is better relative to the performance of the multivariate linear model which presents an R^2 of -6,31 %.

Our results are consistent with the ones presented by Rossi (2018) in the sense that BRT outperforms the linear model. Indeed, our results demonstrate that even if the linear framework and boosted regression trees conditioning their forecasts on the same set of predictors, only the latter is able to effectively better exploit this information to generate forecasts[74].

As we already said, we also compared the forecast direction to the actual realized direction through the Mean Directional Accuracy coefficient which provides the probability that our forecasting method should detect the correct direction of the time series. For this reason, MDA is a popular metric for forecasting performance in economics and finance. Also with this other ratio, BRT outperforms the multivariate linear model with an MDA of 57,14% compared to the lower 45,71% of the benchmark.

Finally, following Goyal and Welch (2008), we present the so called "CumSum analysis" for stock returns, which is defined in our case as:

$$CumSum(T) = \sum_{t=\mathcal{T}}^{T-1} \left(\hat{\mu}_{t+1|t}^{linear} - r_{t+1}\right)^2 - \left(\hat{\mu}_{t+1|t}^{BRT} - r_{t+1}\right)^2 \qquad (30)$$

[73] See Appendix C for more details about the algorithm steps.
[74] In the context of financial market prediction, an higher R^2, even if it is very low (negative in our case), translate into profitable investment strategies, see Campbell and Thompson (2008).

where $\mathcal{T}$ is the first month we start our calculations from, "linear" stands for multivariate linear model and BRT is the BRT prediction. This measure is useful at tracking the performance of the forecasting framework on each period because an upward sloping "CumSum" curve over a given month entails that the predictive model outperforms the linear, and vice versa. Figure 3.4 shows that BRT outperforms the multi-variate linear model over the sample under consideration.

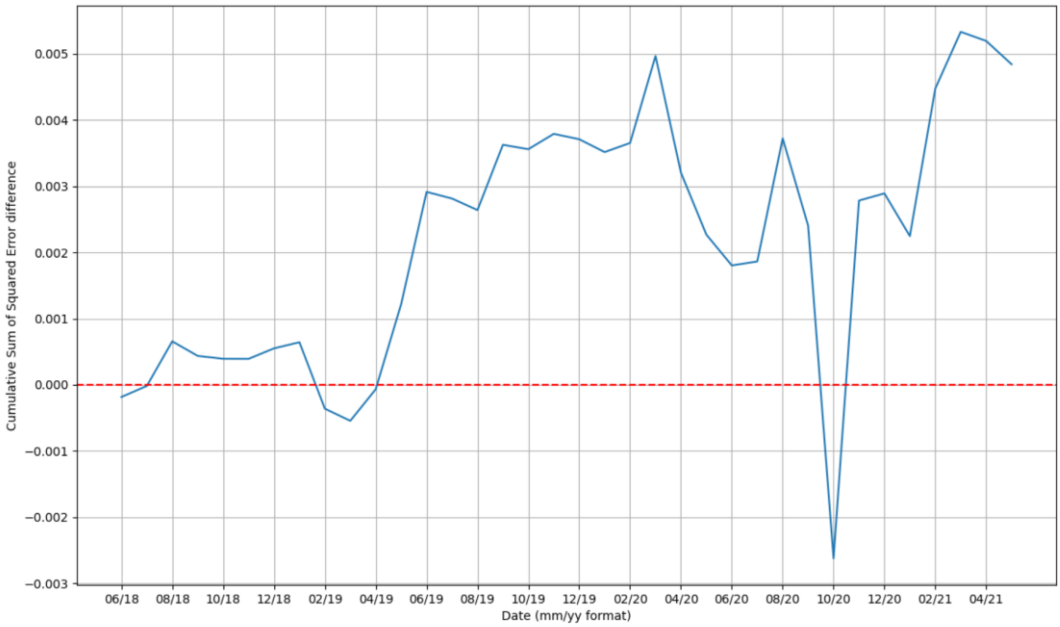


Fig. 3.4: Cumulative Sum of Squared Error Difference of Excess Return forecasts
Source: Own elaboration, 2021

BRT outperforms the multivariate linear method for the whole period from July 2018 to January 2019, from April 2019 to September 2020 and from November 2020 to the end of our out-of-sample period. It underperforms the multivariate linear method only for two short periods: a little lack from January 2019 to April 2019 and a biggest one for the month of October 2020. It is relevant to show that this latter month exactly corresponds to the worst month for stock returns according to the Dow Jones since March 2020 as Coronavirus was spreading. Stock markets in Europe have had their worst month after investors shrugged off record-breaking growth for the eurozone and focused on the likely impact of new coming lockdown restrictions due to the rising of the Coronavirus infection.

In addition to the "CumSum" analysis presented above, we present in figure 3.5 the cumulative out-of-sample mean directional accuracy difference between BRT and the multivariate linear model[75]. The vertical axis is called "Cumulative Sum of MDA difference" and it simply counts the times of differential directional forecasts between the prediction model and our benchmark.

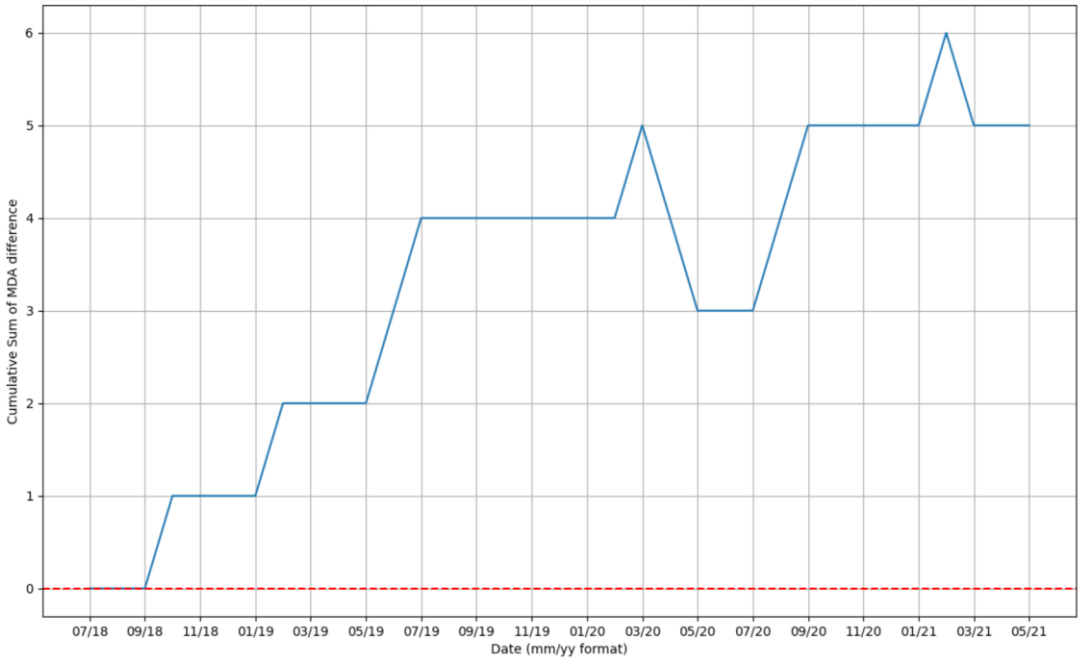


Fig. 3.5: Cumulative Sum of MDA difference of Excess Return forecasts
Source: Own elaboration, 2021

BRT performs very well from the beginning to our out-of-sample test period over March 2020. At this date, the performance is rather poor with a small downhill peak and then immediately resume its ascent after July 2020. By analysing this behaviour for the MDA, we notice that the worst period in terms of predictive directional accuracy corresponds to a negative performance for excess stock returns. In fact, among our predictor variables, the month of March 2020 recorded an excess return of -17%, surely due to the implementation of the international lockdown plans.

Our BRT analysis uses 10,000 boosting iterations to estimate the regression trees. In unreported results we demonstrate that our findings are not sensitive to this choice. Consistently with Rossi (2018), we test our BRT model both with 5,000 and 15,000

[75] For our analysis, comparing to the previous Cumulative Sum of Square Error difference, the order of the BRT model and our benchmark is inverted because an higher MDA is better than a lower one.

boosting iterations and we observe that they both outperform the benchmark in Fig. 3.3. Our conclusion is that boosted regression trees performance is not sensitive to the choice of boosting iterations.

3.2.2 Volatility

Fig. 3.6 reports the results for the forecasted monthly volatility in terms of R^2 and MDA. The results are reported for both our Boosted Regression tree model and the most common competing model proposed in the literature: the GARCH (1,1) model. Setting the regression model, we distinguish the two methods: GARCH (1,1) predicts volatility only from the time series of returns, conversely, in the BRT model algorithm we set the volatility included among our predictors (i.e. vol) as the dependent variable, s.t. the BRT model is able to take into account all the twelve predictors described in paragraph 3.1. in order to obtain the final monthly prediction regarding expected volatility[76].

	R^2	MDA
Boosted Regression Tree	25,33%	40,00%
GARCH (1,1)	-97,96%	48,57%

Fig. 3.6: Out-of-sample forecasting accuracy of Volatility forecasts
Source: Own elaboration, 2021

Over the sample 2001-2021 BRT performs best compared to the benchmark with an out-of-sample R^2 of 25,33%, which is better relative to the performance of the GARCH(1,1) model which achieves a really worst R^2 of -97,96%.
Our results are consistent with the ones presented by Rossi (2018) in the sense that BRT outperforms the most common GARCH(1,1) model.
Also with the volatility, we compared the forecast direction to the actual realized direction through the MDA coefficient to test if our forecasts can correctly detect the direction of the time series. In order to do that, due to the fact that volatility is always positive by nature, following Marquering and Verbeek (2005) we define as positive or negative sign those periods characterized by a volatility lower or greater than the full sample median

[76] See Appendix C for more details about the algorithm steps.

volatility value. Differently from the previous R^2 coefficient, BRT underperforms the GARCH(1,1) model with an MDA of 40% compared to the higher 48,57% of our benchmark.

Finally, we present the results of the modified "CumSum analysis" for the volatility, which is defined as:

$$CumSum(T) = \sum_{t=\mathcal{T}}^{T-1} \left(\hat{\mu}_{t+1|t}^{GARCH\ (1,1)} - r_{t+1}\right)^2 - \left(\hat{\mu}_{t+1|t}^{BRT} - r_{t+1}\right)^2 \qquad (31)$$

where now the GARCH (1,1) model replaces the linear one in the definition and acts as the benchmark. The plot is given in fig. 3.7.

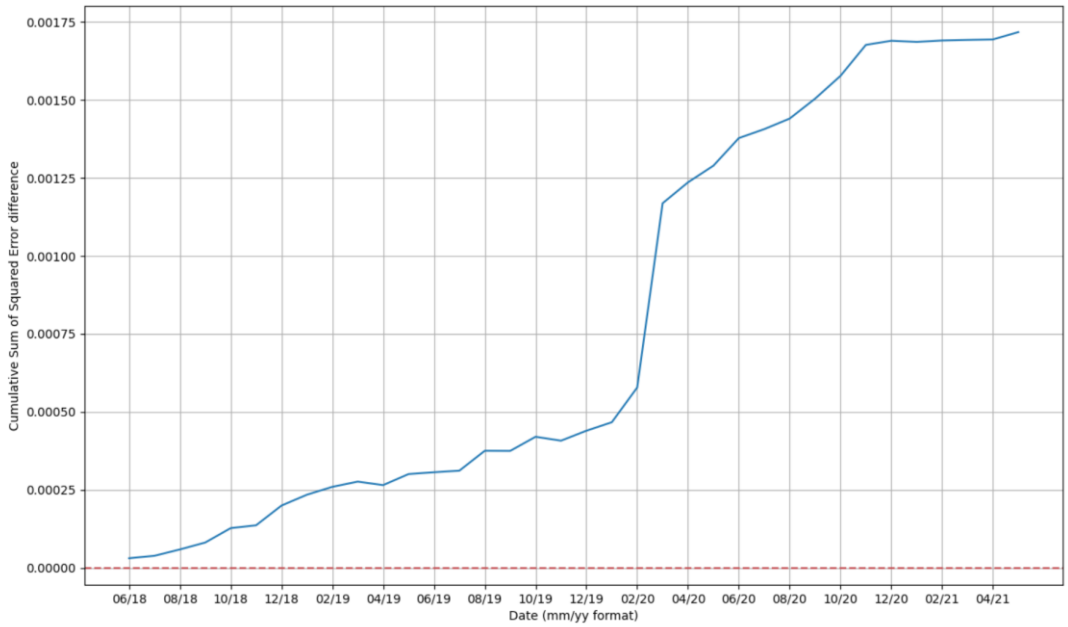


Fig. 3.7: Cumulative Sum of Squared Error Difference of Volatility forecasts
Source: Own elaboration, 2021

A close examination of Figure 3.6 reveals that BRT outperforms the GARCH (1,1) model over the full out-of-sample period taken into account. It is relevant to notice how the predictability of volatility is increased by the Boosted Regression Tree method from March 2020, which corresponds at the month of the first announced lockdown in Europe due to the emerging COVID-19 Pandemic. Indeed, in early March, global markets became extremely volatile and there were large swings. Our result demonstrates how the efficiency of Boosted Regression Tree to predict volatility values is increases for higher

values of volatility. This reveal is consistent with Marquering and Verbeek (2005), which already noticed that volatility values are easier to predict when volatility is high.

Fig. 3.8 repeats the CumSum analysis for the out-of-sample mean directional accuracy difference between BRT model forecasts and GARCH (1,1) that we employ as the benchmark. The vertical axis is called "Cumulative Sum of MDA difference" and it simply counts the times of differential directional forecasts between the prediction model and our benchmark.

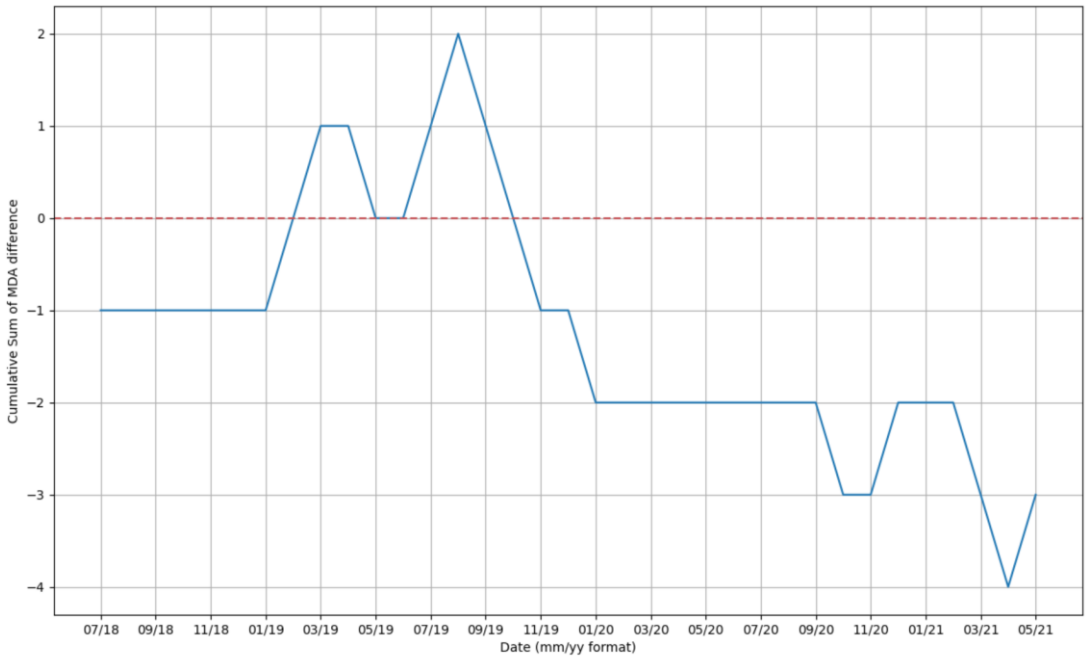


Fig. 3.8: Cumulative Sum of MDA difference of Volatility forecasts
Source: Own elaboration, 2021

The plot highlights that BRT underperforms our benchmark for a longer part of our out-of-sample test period. In particular, BRT beat the benchmark in terms of mean directional accuracy only from March 2019 to November 2019. From this date, the performance is particularly disappointing until the end of the period under consideration. We can explain this behaviour basing on the same previous considerations we present for fig. 3.7: even if the performance of BRT to predict volatility is increased in terms of R^2, for higher values of volatility given by the emerging COVID-19 spread, the mean directional accuracy is conversely more difficult to predict.

In unreported results, by repeating the test made for the excess return forecast, also for the volatility prediction obtained by our Boosted Regression Tree analysis which uses

10,000 boosting iterations, we show that our findings are not sensitive to the choice of boosting interactions. BRT with 5,000 or 15,000 boosting interaction outperforms the benchmark in Fig. 3.5. Again, our conclusion is that boosted regression trees performance is not sensitive to the choice of boosting iterations.

3.2.3 Optimal Portfolio Weights

In the previous paragraph we have illustrated how to separately forecast returns and volatility. Now, we apply our observed prediction algorithms in order to receive optimal portfolio weights as output in a second step (i.e. two-step BRT).

By applying the Utility Theory to this portfolio, the investor will choose his asset allocation by maximize its utility function, leading to the optimal portfolio weights choice given in (8). The only parameter in the latter formula is the risk-aversion coefficient, that we set equal to 4 as a commonly chosen parameter in the literature[77] because it corresponds to a moderately risk-averse investor.

In figure 3.9 we report the results for BRT-based optimal weight predictions in terms of R^2 and MDA. The results are compared with a benchmark strategy which uses stock excess returns predictions from the multivariate linear regression model and volatility predictions from the GARCH(1,1) model[78].

	R^2	MDA
Boosted Regression Tree	6,21%	62,86%
Linear Model & GARCH (1,1)	-7,18%	51,43%

Fig. 3.9: Out-of-sample forecasting accuracy of Optimal Portfolio Weights forecast
Source: Own elaboration, 2021

BRT has a significant overperformance compared to our benchmark with a positive R^2 of 6,21% compared to the negative one of -7,18% of the Linear Model & GARCH (1,1) weights prediction. This overperformance of BRT is also observed regarding the MDA coefficient which is 62,86% with an high-performance differential of 11,43% relative to our benchmark strategy. These results confirm the better performance of the BRT model

[77] See Paragraph 2.1 of Chapter 2.
[78] See Appendix E for more details about the algorithm steps.

forecasts and it is perfectly in compliance with Rossi (2018) achievements for optimal weights predictions.

In order to complete our findings, we presents estimates of the modified "CumSum analysis" for the optimal weights forecast defined as:

$$CumSum(T) = \sum_{t=\mathcal{T}}^{T-1} \left(\hat{\mu}_{t+1|t}^{lin\&GARCH\ (1,1)} - r_{t+1}\right)^2 - \left(\hat{\mu}_{t+1|t}^{BRT} - r_{t+1}\right)^2 \qquad (32)$$

where now the Linear Model & GARCH (1,1) model act as our benchmark. The plot is given in fig. 3.10.

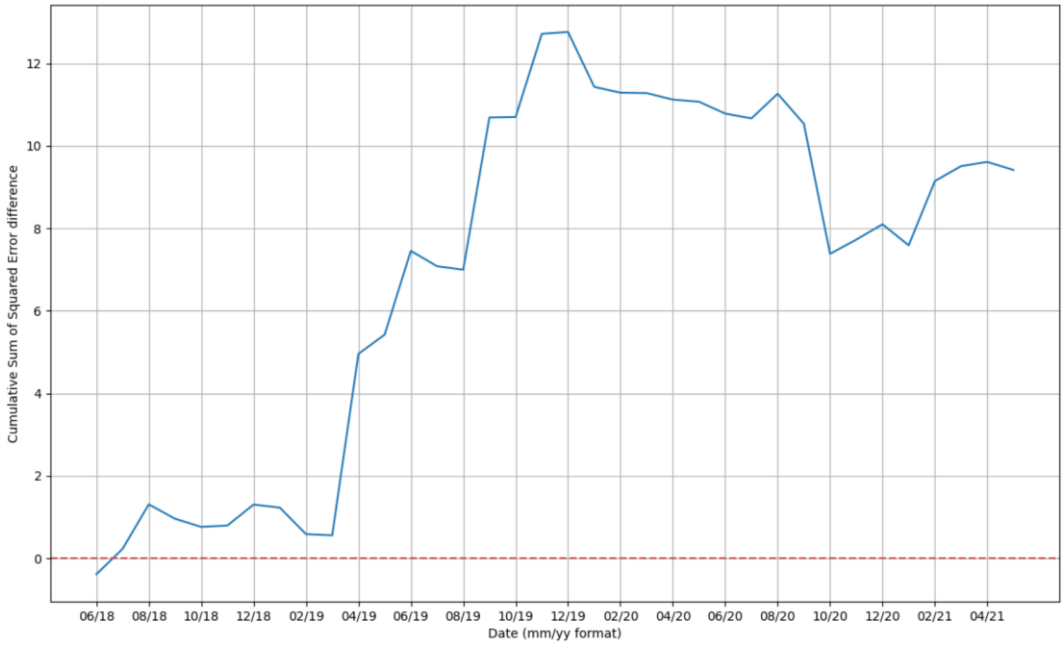


Fig. 3.10: Cumulative Sum of Squared Error Difference of Optimal Weight forecasts
Source: Own elaboration, 2021

The plot gives us the ultimate idea of the outperformance of our BRT model throughout the out-of-sample period. After a continuous rising of performance from early 2019 we observe a short pick fall exactly in the month of October 2020 that corresponds to the worst month for stock returns due to the rising of worries about the COVID-19 spread.

Finally, figure 3.11 plots the cumulative out-of-sample mean directional accuracy difference between BRT and our benchmark strategy of Linear Model & GARCH (1,1).

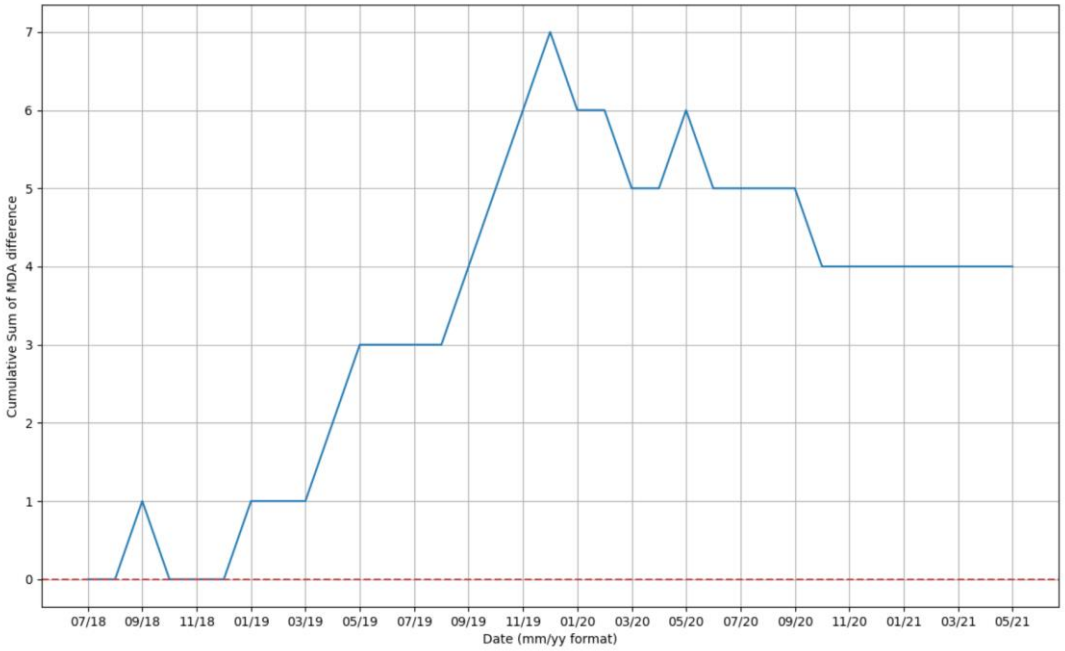


Fig. 3.11: Cumulative Sum of MDA of Optimal Weight forecasts
Source: Own elaboration, 2021

The plot shows that while a great performance of mean directional accuracy was present until December 2019 by the BRT model, over the rest of the out-of-sample test period the performance is mainly falling, probably due to the still in progress emergency situation which created instability on the international market which our BRT algorithms was unable to predict perfectly in terms of directional accuracy comparing to the benchmark. In conclusion, based on the mean square error and mean directional accuracy as performance evaluation criteria, the results reported in this paragraph highlight the superiority of the BRT model compared to the common benchmarks often used in the literature as regards the prediction of the equity premium and associated volatility. Actually, our preliminary results suggest that BRT outperforms the benchmark in terms of R^2 but the evidence for MDA is not convincing in terms of volatility forecasts.

3.3 One Step BRT Model

In the previous paragraph we have focused our attention on two steps: predicting excess returns and volatility in a separately way and then constructing optimal portfolio weights. We now attempt to directly forecast the optimal portfolio allocation for a mean-variance investor by set the risk-aversion coefficient equal to 4 as we already done in the previous

paragraph according to the common practice in the literature for a moderately risk-averse investor. In order to do that, in the initial phase of our algorithm we compute optimal portfolio weights directly from excess return and volatility values of our monthly predictor variables according to formula (8) by creating a new column of the data which set our dependent variable. Subsequently, we split weight results into train sample and test sample. At the end, we set the regression BRT model by fitting it with the predictors described in paragraph 3.1 in order to obtain the final predictions regarding expected monthly optimal portfolio weights[79].

In figure 3.12 we report the results for optimal portfolio weight predictions for the previous two-step BRT model and the new one-step BRT model in terms of R^2 and MDA.

	R^2	MDA
Two-step BRT	6,21%	62,86%
One-step BRT	7,14%	65,71%

Fig. 3.12: Out-of-sample forecasting accuracy of Optimal Portfolio Weights forecast in BRT models
Source: Own elaboration, 2021

One-step BRT has a small overperformance compared to our benchmark with an R^2 of 7,14% compared to the one of 6,21% of the two-step BRT weights prediction. This overperformance is also observed regarding the MDA coefficient which is 65,71% for the one-step BRT model with a relatively small high-performance differential of 2,85% compared to our benchmark model. These results evince the better performance of the BRT model which directly predict optimal portfolio weights in one single step.

In the following paragraph, we construct optimal portfolio allocations based on market excess returns r_{t+1} and volatility σ_{t+1}. We present the results by using our Boosted regression tree models with 10.000 boosting iterations and then we assess the performance by comparing them with the ones obtained in paragraph 3.2.

[79] See Appendix E for more details about the algorithm steps.

3.4 Portfolio Allocation Performance

After predicting the market index over time, our analysis aims to verify predictions' implementation from an asset allocation point of view[80]. In this paragraph we use Market Timing[81] to derive a signal of rebalancing of the stock market index portfolio during time for unconstrained portfolio weights for the mean variance investor described in chapter 2 who uses both the Boosted Regression Tree models and the Linear & GARCH (1,1) one to formulate returns and volatility predictions. We distinguish the two-step BRT where market returns and volatility are forecasted separately and the optimal mean-variance portfolio allocation is computed in a second stage, from the one-step BRT where optimal portfolio weights are forecasted directly from the implemented predictors. For comparison, we also present results for three benchmark strategies. The first one imposes short selling and borrowing constraints to the optimal portfolio weights which have to lie between 0 and 1, such that:

$$\omega^*_{t+1|t} = \begin{cases} 0 & if \quad \omega^*_{t+1|t} < 0, \\ \omega^*_{t+1|t} & if \quad 0 \leq \omega^*_{t+1|t} \leq 1 \\ 1 & if \quad if \quad \omega^*_{t+1|t} > 0 \end{cases} \tag{33}$$

The second one is the classical static portfolio consisting of 60% stocks and 40% bonds. Stock returns are based on the European Market Index EU STOXX 50 compounded monthly returns. On the other hand, Bond returns are based on 3-month German Bund price[82].

Finally, the third benchmark strategy we adopt as benchmark is the passive strategy where the investor purchase our European Market Index and hold it over the long-time horizon without short-term trading[83].

[80] The asset allocation is the percentage composition of the portfolio by financial asset macro classes (stocks, bonds, monetary) or first-level assets with further details for each macro class (e.g. foreign investments, small-cap stocks and large-cap stocks). Asset allocation decisions result from the investor's time horizon, risk tolerance and overall financial knowledge

[81] Market timing consists in anticipating stock market movements in order to buy and sell stocks in a short time period to make a profit. It is appeal when markets are volatile, as the change of making profit quickly is more, even if the risk to lose money is higher as well due to the fact that predicting the stock market consistently is really difficult.

[82] See par. 3.1.

[83] See Appendix F for more details about the algorithm steps.

In order to formulate optimal portfolio weights, we use formula (8) by setting the investor risk coefficient to 4 which corresponds to a moderately risk-averse investor. In constructing portfolios, investors often combine risky assets with risk-free assets to reduce risks[84]. Therefore, our complete portfolio is defined as a combination of the risky asset (i.e. Market Index) and the risk-free one (i.e. 3 month German Bund). The Return of this portfolio will be a weighted average of its individual assets' returns and it is calculated according to formula (34) as follow:

$$r_{p,t+1} = E_t(w_{i,t+1})\, r_{i,t+1} + \left(1 - E_t(w_{i,t+1})\right)\, r_{f,t+1} \qquad (34)$$

where $r_{i,t+1}$ is the return of the risky asset whose expected weight in the portfolio is $E_t(w_{i,t+1})$, and $r_{f,t+1}$ is the return of the risk-free asset whose portfolio's weight is the remaining part of it, so $\left(1 - E_t(w_{i,t+1})\right)$ because portfolio weights sum to 1. Positive weight in the portfolio means that the investor long buy the stock. Conversely, a negative portfolio weight is considered as a negative investment in the asset, s.t. the investor short sell the stock.

The performance of these strategies is evaluated over the three-year test sample period from 2018 to 2021 and it is express in terms of Mean, Standard Deviation, Sharpe Ratio[85], VaR[86] and Maximum drawdown[87].

Fig. 3.13 reports our performance valuation indicators by considering unconstrained weights. Our results confirms that the best model is the BRT that forecasts the optimal portfolio allocation in two steps (i.e. Two-step BRT) with a Sharpe Ratio of 19,30%. The Linear Model & GARCH(1,1) performs significantly worst comparing to other benchmark strategies with a negative Sharpe Ratio of -8,64%. This model beats others

[84] See chapter 2, par. 1

[85] Sharpe Ratio measures the performance of a portfolio compared to a risk-free asset, after adjusting for its risk. Indeed, it is defined as the difference between the portfolio's return and the risk-free return, divided by the portfolio's standard deviation (i.e. its volatility).

[86] Value at risk (VaR) is a statistic for financial risk which provides an estimate of the maximum loss of portfolio over a specific period of time. In our estimations we compute it with a 95% confidence level. This means that with a 95% probability, the maximum expected loss at the end of the considered period will not be higher than the VaR obtained. For sure, nothing tells us what will happen in the remaining 5% of cases.

[87] Maximum Drawdown is the maximum observed loss from a peak to a trough of a portfolio before a new peak occurs. It is an indicator of downside risk over a specified time period, and it is calculating the negative returns, hence it is between 0 and -1.

BRT models only in terms of Standard Deviation with a value of 0,22%, not too far from the 1,83% of the Two-step BRT and the 2,39% of the One-step BRT.

	Mean	St.Dev	Sh.Ratio	VaR	MaxDrawd.
Two-step BRT	-0,06%	1,83%	19,30%	-3,07%	-13,50%
One-step BRT	-0,10%	2,39%	12,82%	-4,03%	-18,24%
Linear Model & GARCH(1,1)	-0,43%	0,22%	-8,64%	-0,78%	-14,02%

Fig. 3.13: Portfolio Allocation Performance with unconstrained weights
Source: Own elaboration, 2021

In figure 3.14 we repeat the same exercise imposing short-selling and borrowing constraints as represented in formula (33). Unexpectedly, these limits increase the profitability of both the two-step BRT-based investment and the Linear Model & GARCH(1.1) strategies: in the first Mean is increased by 0,04% while, in the second, they are increased by 0,04% and 21% respectively. Again, the best investment strategies is the one that adopt the Two-step BRT model in order to make forecasts. It presents a considerably higher Sharpe Ratio of 23,15%. compared to the One-step BRT which has a value drop to 11,59%. The lower Sharpe Ratio is due to both lower Mean and Standard Deviation of this portfolio strategy compared to the one with unconstrained weights. The opposite holds true for the investment strategy that exploits Linear Model & GARCH(1,1) forecasts, where the Sharpe Ratio significantly increase from -8,64% to 12,36%. As also has been observed by Rossi(2018) there are different reasons for this, among which the most important one is that there is a fair degree of estimation error in the estimated optimal portfolio allocations, due to the minimization of the L^2 criterion functions.

	Mean	St.Dev	Sh.Ratio	VaR	MaxDrawd.
Two-step BRT	-0,02%	1,70%	23,15%	-2,81%	-11,03%
One-step BRT	-0,16%	2,11%	11,59%	-3,64%	-16,20%
Linear Model & GARCH(1,1)	-0,39%	0,12%	12,36%	-0,59%	-12,97%

Fig. 3.14: Portfolio Allocation Performance with short selling and borrowing constraints
Source: Own elaboration, 2021

Overall, these results highlights that BRT models outperform the established benchmark in term of constrained and unconstrained portfolio allocation performance. They also demonstrate that, between BRT models, portfolio allocations based on both returns and volatility estimates (i.e. two-step BRT) is more profitable than the one that directly exploit expected weights (i.e. one-step BRT).

At the end, we focus our attention on figure 3.15 which reports results for models that assume a passive strategy. The first is the classical static portfolio consisting of 60% stocks and 40% bonds, while the second one consists in investing 100% of wealth in the risky asset.

	Mean	St.Dev	Sh.Ratio	VaR	MaxDrawd.
Portfolio 60/40	0,12%	3,44%	15,41%	-5,53%	-17,28%
100% Market	0,47%	5,74%	15,36%	-8,97%	-27,25%

Fig. 3.15: Portfolio Allocation Performance with Passive strategies
Source: Own elaboration, 2021

Starting from an internal comparison between them, we denote that the second strategy allows to obtain a more profitable portfolio, but riskier. Indeed, even if they have similar Sharpe Ratios, first strategy has a significantly lower Mean which differ from the benchmark by -0,35%. Conversely, the second strategy has an higher value by considering Standard Deviation and Maximum Drawdown, by differing from the benchmark by 2,3%, 9,97%. In addition, it has a lower VaR of -8,97% compared to the -5,53% of the Portfolio 60/40 investment.

The analysis of these results are also useful in order to better understand how these passive strategies are more or less efficient for a better asset allocation compared to the benchmark active strategies of figure 3.13 and 3.14. We can easily highlight that the Passive strategies are the only ones that present a positive output in terms of Mean, even if the higher Sharpe Ratio is given only by the two-step BRT strategy with 19,30% with unconstrained weights and 23,15% with short selling and borrowing constraints that again has demonstrate its efficiency. Unlike profit, passive strategies are riskier than all the active benchmarks. Indeed, these kind of investments presents almost double Standard

Deviation and also lower VaR and Maximum Drawdown. The riskiest investment is the passive strategy which allocates 100% of the investor's wealth in the market portfolio.
Even though widely used, the performance measures considered in our analysis until now may be not correct measures. For instance, Fleming, Kirby and Ostidiek (2001) and Marquering and Verbeek (2005) argues that the Sharpe Ratio is not efficient as risk-adjusted returns' measure in the presence of time-varying volatility. For this reason, in the next paragraph we focus our analysis on the economic value generated by the trading strategies based on the BRT forecasts.

3.4.1 Mean-Variance Investor's Utility Value Perspective

Following Fleming, Kirby and Ostidiek (2001) and Marquering and Verbeek (2005), we define an utility-based performance measure based on the ex-ante investor's utility function we presented in (1) adapted to the portfolio:

$$U_{p,t} = E_t(r_{p,t+1}) - \frac{1}{2} A \, {\sigma_t}^2(r_{p,t+1}) \tag{35}$$

where $E_t(r_{p,t+1})$ is the expectation of the portfolio return at time t+1 given the information available as of time t, ${\sigma_t}^2(r_{t+1})$ is the variance at time t+1 given the information available as of time t and A is the risk-aversion coefficient.
By substituting (34) in (35), we can re-write the expression in order to construct an ex-post average realized utility function for each investment strategy we observed, by using the following formula:

$$U_p = \frac{1}{T} \sum_{t=0}^{T-1} \left[r_{f,t+1} + w_{i,t+1} \, r_{i,t+1} - \frac{1}{2} A \, w_{i,t+1}^2 {\sigma^2}_{i,t+1} \right] \tag{35}$$

where $r_{f,t+1}$ and $r_{i,t+1}$ are the actual monthly return of the risk-free asset and risky asset respectively between t and t+1, while ${\sigma^2}_{i,t+1}$ is the monthly variance of the risky asset.
The average utility level is express in percentage and refers to the relative satisfaction that an investor derives from the portfolio according to its own risk-aversion perception.

In figure 3.16 we report the output for the average realized utility of the BRT-based[88] portfolio also according to different risk-aversion coefficient ranging from 2 to 10.

		$A = 2$	$A = 4$	$A = 6$	$A = 8$	$A = 10$
Two-step BRT	Unconstrained weights	-0,24%	-0,36%	-0,48%	-0,61%	-0,73%
	Short. and borrow. constraints	-0,24%	-0,35%	-0,46%	-0,58%	-0,69%
One-step BRT	Unconstrained weights	-0,27%	-0,37%	-0,47%	-0,57%	-0,67%
	Short. and borrow. constraints	-0,39%	-0,49%	-0,59%	-0,69%	-0,80%

Fig. 3.16: Average realized utilities for BRT-based strategies
Source: Own elaboration, 2021

In addition to the previous consideration, it is relevant to denote that until now we have ignored estimation uncertainty. In fact, expected returns and volatilities are not know but only estimated from the data. Even though we used the longest possible time-series available, it is not too high in terms of data intensity because the EUROSTOXX50 is not too old as the S&P 500. The noise obtained may be responsible for making the portfolio weights too volatile over time. A number of approaches have been proposed to mitigate the effect of estimation uncertainty[89]. Here, following Rossi (2018) inspired by Marquering and Verbeek (2005), we re-estimate the optimal portfolio weights using a "pseudo-risk aversion coefficient" A which is now considered the double of the true risk-aversion coefficient considered until now. Instead of 4, A is now 8.
In figure 3.17 we report the output for the average realized utility of the portfolio accounting for estimation uncertainty.

		$A = 2$	$A = 4$	$A = 6$	$A = 8$	$A = 10$
Two-step BRT	Unconstrained weights	-0,28%	-0,32%	-0,35%	-0,38%	-0,41%
	Short. and borrow. constraints	-0,29%	-0,32%	-0,35%	-0,38%	-0,40%
One-step BRT	Unconstrained weights	-0,29%	-0,39%	-0,49%	-0,58%	-0,68%
	Short. and borrow. constraints	-0,36%	-0,45%	-0,54%	-0,63%	-0,72%

Fig. 3.17: Average realized utilities for BRT-based strategies accounting for estimation uncertainty
Source: Own elaboration, 2021

[88] Linear Model and GARCH (1,1) performance is not reported for brevity.
[89] See Ter Horst, De Roon, and Werker (2000) and Maenhout (2004).

In addition, for a better comparison, in figure 3.17 we report the output for the average realized utility of the portfolio for an investor who adopt passive strategies by placing 100% of his wealth in the market portfolio or adopt a classical strategy 60/40.

		$A = 2$	$A = 4$	$A = 6$	$A = 8$	$A = 10$
Passive strategies	Portfolio 60/40	-0,48%	-0,83%	-1,18%	-1,53%	-1,88%
	100% Market	-0,91%	-1,89%	-2,86%	-3,84%	-4,81%

Fig. 3.18: Average realized utilities for passive strategies
Source: Own elaboration, 2021

Taking into account the investor's utility function, passive are the worst strategies compared to the BRT-based benchmark. Again, the better performance is given by the two-step BRT, in particular when short selling and borrowing constraints are taking into consideration. For instance, the unconstrained weights investment strategy based on the two-step BRT model that does not correct for estimation uncertainty delivers an average monthly realized utility of -0,24% to an investor with a risk-aversion coefficient of 2. On the other hand, the one-step BRT counterpart presents a value of realized utility equal to -0,27%. Both are much larger than both the -0,48% provided by the portfolio 60/40 strategy and the -0.91% obtained by the investor who allocates 100% of his wealth on the market index. The higher performance of the active strategies increase when the risk-aversion coefficient increase, achieving a significantly gap with a risk-aversion coefficient equal to 10. For instance, with a risk-aversion coefficient equal to 10 the 100% market strategy has a realized utility value of -4,81% compared to the two-step BRT investment correct for estimation uncertainty which gives a value of -0,41%.

From a wider point of view, it is relevant to highlight the greater realized utilities in presence of unconstrained weights. The results hold whether we adopt one or two-step BRT models. All the investment strategies provide higher realized utilities for low coefficient of risk-aversion. Overall, our results indicate that without accounting for estimation uncertainty provides a better realized utility only for a low risk-aversion coefficient. In fact, from $A = 4$ not account for estimation uncertainty result in too volatile portfolio allocations with a subsequently lower utility function's value.

CONCLUSION

Our paper contributes to the long-standing literature assessing the predictability of stock returns and helps justify the growing role of ML throughout the FinTech industry.

We have a deep investigation on how a representative mean-variance investor should exploits publicly available information to formulate forecasting on excess return and volatility as well as optimal portfolio allocations. We adopt the BRT method on the market index EU STOXX 50 that is able to make forecasts basing on a large set of conditioning information without imposing strong parametric assumptions such as linearity or monotonicity. The conditioning information we use are the twelve predictor variables proposed so far by Welch and Goyal (2008).

The first question we answer is whether macroeconomic and financial variables contain information about expected stock returns and volatility that a mean-variance investor could benefit from. We present new evidence of the outperformance of BRT-methods forecasts against the established benchmarks with a more significant market timing in both returns and volatility. BRT-based forecasts method translate into profitable portfolio allocations in terms of both Mean Squared Error and Mean Directional Accuracy.

The second question we answer is whether the conditioning information contained in macroeconomic and financial time-series can be exploited to directly select the optimal portfolio weights. Tests of optimal portfolio weights predictability demonstrate that portfolio allocations are time-varying and forecastable with a better performance of the one-step BRT compared to the statistical methods adopted as benchmark.

The third question is whether the results forecasts are economically valuable in terms of portfolio allocations profitability. We assess this by computing Mean, Standard Deviation, Sharpe Ratio, VaR and Maximum drawdown of the monthly portfolio returns obtained from our predictor variables also when short selling and borrowing constraints are taken into account. Actually, the more significant market timing of BRT-based methods, as well as their better performance compared to classical statistical methods implemented as benchmark, does not translate into profitable performance exploiting the predicted optimal weights of the portfolio. Indeed, passive strategies have demonstrated a better performance in our research compared to the active ones in terms of Mean. The possible reason behind is the scarcity of data available for the European Market Index

that does not allow to have an efficient data intensity, so a longer period, able to make more correct previsions. At this consideration, it is also necessary to add the criticisms happened in our test period (i.e. last years) which includes the word pandemic effect that affected the stock market integrity in terms of higher volatility.

We also calculate the realized utilities level obtained by the investor by comparing BRT-based strategies and passive ones. Our findings show the greater performance of the BRT-based model against the passive strategies of portfolios 60/40 and 100% market. The different results compares to the previous consideration is due to the higher volatility presented in the passive strategies compared to the active investments.

The limits presented by our work in terms of data intensity should be inspirations for future research again in an European context but with a longer period into account. Machine learning in banking and finance is beginning to play a significant role in various processes. However, not many FinTech providers considered machine learning as a key driver of financial services. Easier-to-use machine learning tools, multiple algorithms, and good computing power will only increase its adoption in financial technology, so now is the time to catch up with this performing trend.

REFERENCES

Ait-Sahalia, Y., and Brandt, M. 2001. *Variable Selection for Portfolio Choice.* The Journal of Finance, 56 (4), 1297–1351.

Alpaydin, E. 2020. *Introduction to Machine Learning.* 4th ed. Massachusetts Institute of Technology Press.

Bossaerts, P., and Hillion, P. 1999. *Implementing statistical criteria to select return forecasting models: what do we learn?* Review of Financial Studies, 12 (2), 405–428.

Brandt, M.W., Santa-Clara, P. and Valkanov, R. 2004. *Parametric Portfolio Policies: Exploiting Characteristics in the Cross Section of Equity Returns.* Review of Financial Studies, 22 (9), 3411-3447.

Breen, W., Glosten, L. and Jagannathan, R. 1989. *Economic significance of predictable variations in stock index returns.* Journal of Finance, 44 (5), 1177–89.

Breiman, L., Friedman, J. H. , Olshen, R. A. and Stone, C. J. 1984. *Classication and regression trees. Statistics/Probability Series.* Belmont, California, U.S.A.: Wadsworth Publishing Company.

Brooks, C. 2014. *Introductory Econometrics for Finance.* 3rd ed. New York: Cambridge University Press.

Campbell, J. Y. 1987. *Stock returns and the term structure.* Journal of Financial Economics, 18 (2), 373–399.

Campbell, J. Y. 2006. *Household finance.* The Journal of Finance 61, 1553-604.

Campbell, J., and Shiller, R. 1988. *The dividend-price ratio and expectations of future dividends and discount factors.* Review of Financial Studies, 1(3), 195–228.

Campbell, J., and Shiller, R. 1988. *Stock prices, earnings, and expected dividends.* Journal of Finance, 43 (3), 661–676.

Campbell, J., and Yogo, M. 2006. *Efficient tests of stock return predictability.* Journal of Financial Economics, 81(1), 27–60.

Chinco, A., Clark-Joseph, A. D. and Ye, M. 2019. *Sparse Signals in the Cross-Section of Returns.* The Journal of Finance, 74, 449-492.

Cumby, R., and Modest, D. 1987. *Testing for Market Timing Ability: A Framework for Forecast Evaluation.* Journal of Financial Economics, 19 (1), 169–189.

Dangl, T., and Halling, M. 2008. *Predictive regressions with time-varying coefficients.* Vienna University of Technology. Working Paper.

De'Ath, G. 2007. *Boosted Trees for ecological modelling and prediction.* Ecology, 88 (1), 243-251

DeMiguel, V., Martìn-Utrera, A., Nogales, F. J. and Uppal, R. 2019. *A Transaction-Cost Perspective on the Multitude of Firm Characteristics. The Review of Financial Studies.* 33 (5), 2180–2222.

Deshpande, B. 2011. *4 key advantages of using decision trees for predictive analytics.* Simafore. Working Paper.

Elith J., Leathwick, J.R. and Hastie, T. 2008. *A working guide to boosted regression trees.* Journal of Anlimal Ecology, 77(4), 802-813.

Engle, R., and Rangel, J. 2005. *The Spline GARCH Model for Unconditional Volatility and Its Global Macroeconomic Causes.* Czech National Bank.

Fama, E. F. and French, K. R. 2008. *Dissecting anomalies.* Journal of Finance 63 (4), 1653-1678.

Fama, E. F. and French, K. R. 1992. *The cross-section of expected stock returns.* Journal of Finance 47 (2), 427-465.

Fama, E. F. and French, K. R. 1993. *Common risk factors in the returns on stocks and bonds.* Journal of Financial Economics 33 (1), 3-56.

Fama, E. F. and French, K. R. 1996. *Multifactor explanations of asset pricing anomalies.* Journal of Finance 51 (1), 55-84.

Fama, E. F. and French, K. R. 2015. *A five-factor asset pricing model.* Journal of Financial Economics 116 (1), 1-22.

Fama, E. F. and MacBeth J. D. 1973. *Risk, return, and equilibrium: Empirical tests.* Journal of Political Economy 81 (3), 607-636.

Fama, E., and French, K. 1988. *Dividend yields and expected stock returns.* Journal of Financial Economics, 22(1), 3–25.

Fama, E., and French, K. 1989. *Business conditions and expected returns on stocks and bonds.* Journal of Financial Economics, 25 (1), 23–49.

Fama, E., and Schwert, G. 1977. *Asset Returns and Inflation.* Journal of Financial Economics, 5 (2), 115–146.

Fleming, J., Kirby, C. and Ostdiek, B. 2001. *The Economic Value of Volatility Timing.* Journal of Finance, 56, 329–352.

Freyberger J., Neuhierl, A. and Weber, M. 2018. *Dissecting Characteristics Nonparametrically.* National Bureau of Economic Research Working Paper Series, 23227.

Friedman, J. H. 1999. *Stochastic gradient boosting*. Stanford University. Working Paper.

Friedman, J. H. 2001. *Greedy function approximation: A gradient boosting machine.* Annals of Statistics, 29, 1189–1232.

Géron, A. 2019. *Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow*. 2nd ed. U.S.A.: O'Reilly Media.

Ghysels, E., Santa-Clara, P. and Valkanov, R. 2005. *There is a Risk-Return Tradeoff After All.* Journal of Financial Economics, 76 (3), 509–548.

Green, J., Hand, J. R. and Zhang, X. F. 2017. *The characteristics that provide independent information about average us monthly stock returns.* The Review of Financial Studies 30 (12), 4389-4436.

Gu, S., Kelly, B. and Xiu, D. 2020. *Empirical Asset Pricing via Machine Learning.* The Review of Financial Studies 33, 2223–2273.

Han, Y., He, A. Rapach, D.E. and Zhou, G. 2018. *What Firm Characteristics Drive US Stock Returns?* The Journal of Finance.

Hansen, P., and Lunde, A. 2005. *A forecast comparison of volatility models: Does anything beat a GARCH (1, 1)?* Journal of Applied Econometrics, 20 (7), 873–889.

Hansen. North Holland. Lewellen, J., Nagel, S. and Shanken, J. 2010. *A skeptical appraisal of asset pricing tests.* Journal of Financial Economics, 96(2), 175–194.

Henriksson, R.D. and Merton, R.C. 1981. *On the Market Timing and Investment Performance II: Statistical Procedures for Evaluating Forecasting Skills.* Journal of Business, 54, 513-534.

Hilpisch Y. 2019. *Python for Finance. Mastering Data-Driven Finance*. 2nd ed. Canada: O'Reilly Media.

Hou, K., Karolyi, G. A. and Kho, B.C. 2011. *What factors drive global stock returns?* Review of Financial Studies 24 (8), 2527-2574.

Hull, J. C. 2018. *Options, futures, and other derivatives.* University of Toronto. Tenth edition. New York: Pearson

Jansen S. 2020. *Machine Learning for Algorithmic Trading Predictive models to extract signals from market and alternative data for systematic trading strategies with Python.* 2nd ed. U.K.: Packt Publishing.

Keim, D. and Stambaugh, R. 1986. Predicting returns in the stock and bond markets. Journal of Financial Economics, 17 (2), 357–390.

Kothari, S., and Shanken, J. 1997. *Book-to-market, dividend yield, and expected market returns: A time-series analysis.* Journal of Financial Economics, 44 (2), 169–203.

Leitch, G., and Tanner, J. 1991. *Economic Forecast Evaluation: Profits Versus the Conventional Error Measures.* American Economic Review, 81 (3), 580–590.

Lettau, M., and Ludvigson, S. 2009. *Measuring and Modeling Variation in the Risk-Return Tradeoff.* Handbook of Financial Econometrics.

Lewellen, J. 2015. *The cross section of expected stock returns.* Critical Finance Review 4 (1), 1-44.

Li, Y. and Spigt, R. 2016. *FinTech: Disrupting or constructing?* Erasmus Universiteit Rotterdam. School of Economics. Working Paper.

Li, Y., Spigt, R., Swinkels, L. 2017. *The impact of FinTech start-ups on incumbent retail banks' share prices.* Springer, Heidelberg, 3 (26), 1-16.

Linnainmaa, J. T., Melzer, B. and Previtero, A. 2018. *The Misguided Beliefs of Financial Advisors. Journal of Finance.* Kelley School of Business Woking Paper.

Markowitz, H. 1952 *Portfolio Selection: Efficient Diversification of Investments.* The Journal of Finance, 7 (1),77-91.

Markowitz, H.M., Peter Todd, G. and Sharpe, W. F. 1889. *Mean-variance analysis in portfolio choice and capital markets*. U.S.A.: Wiley.

Marquering W. and Verbeek, M. 2001. *The Economic Value of Predicting Stock Index Returns and Volatility.* The Journal of Finance.

Merton, R. C. 1973. *An intertemporal capital asset pricing model.* Econometrica 41 (5), 867-887.

Paye, B. 2010. *Do Macroeconomic Variables Predict Aggregate Stock Market Volatility?* Working Paper Series.

Pesaran, M., and Timmermann, A. 1995. *Predictability of Stock Returns: Robustness and Economic Significance.* Journal of Finance, 50 (4), 1201–1228.

Pontiff, J., and Schall, L. 1998. *Book-to-market ratios as predictors of market returns.* Journal of Financial Economics, 49(2), 141–160.

Rahman, A. A. 2017. *Emerging Technologies with Disruptive Effects: A Review*. Perints eJournal 7 (2).

Rapach, D. E., Strauss, J. K. and Zhou, G. 2010. *Out-of-Sample Equity Premium Prediction: combination Forecasts and Links to the Real Economy.* Review of Financial Studies 23 (2), 821-862.

Rapach, E. D. and Zhou, G. 2020. *Time-series and Cross-sectional Stock Return Forecasting: New Machine Learning Methods* in *Machine Learning for Asset Management: New Developments and Financial Applications* (edited by) Emmanuel Jurczenko. 1st ed. ISTE Ltd and John Wiley & Sons, Inc.

Rossi, A. G. 2018. *Predicting Stock Market Returns with Machine Learning.* University of Maryland. Working Paper.

Rossi, A., and Timmermann, A. 2010. *What is the Shape of the Risk-Return Relation?* UCSD Discussion Paper.

Rossi, G. and Utkus S. 2021 *Who Benefits from Robo-advising? Evidence from Machine Learning.* The Journal of Finance.

Sharpe, W. F. 1964. *Capital asset prices: A theory of market equilibrium under conditions of risk.* The Journal of Finance 19 (3), 425-442.

Sullivan, R., Timmermann, A. and White, H. 1999. *Data-Snooping, Technical Trading Rule Performance, and the Bootstrap.* The Journal of Finance, 54 (5), 1647–1691.

Tadlaoui, G. 2018. *Intelligent Portfolio Construction: Machine-Learning enabled Mean-Variance Optimization.* Department of Mathematics and Finance, Imperial College London.

Ter Horst, J., De Roon, F. and Werker, B. 2000. *Incorporating Estimation Risk in Portfolio Choice.* CentER Working Papers Series.

Welch, I., and Goyal, A. 2008. *A Comprehensive Look at The Empirical Performance of Equity Premium Prediction.* Review of Financial Studies, 21 (4), 1455–1508.

Zou, H. 2006. *The adaptive Lasso and its oracle properties.* Journal of the American Statistical Association 101 (476), 1418-1429.

https://home.kpmg/xx/en/home/media/press-releases/2021/02/vc-investment-in-fintech-more-than-doubles-in-second-half-of-2020.html accessed 12 March 2021

https://fred.stlouisfed.org/series/EA19CPALTT01GPM accessed 08 June 2021

https://fred.stlouisfed.org/series/AAA accessed 08 June 2021

https://fred.stlouisfed.org/series/BAA accessed 08 June 2021

https://tradingeconomics.com/european-union/inflation-rate accessed 08 June 2021

https://scikit-learn.org/stable/supervised_learning.html#supervised-learning accessed 10 June 2021.

https://towardsdatascience.com/predicting-future-stock-market-trends-with-python-machine-learning-2bf3f1633b3c accessed 10 June 2021.

https://christophj.github.io/replicating/r/replicating-goyal-welch-2008/ accessed 15 June 2021.

https://scikit-learn.org/stable/modules/ensemble.html#gradient-tree-boosting accessed 20 June 2021.

https://machinelearningmastery.com/backtest-machine-learning-models-time-series-forecasting/ accessed 2 July 2021.

https://scikit-learn.org/stable/auto_examples/linear_model/plot_ols.html accessed 2 July 2021.

https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Linear Regression .html#sklearn.linear_model.LinearRegression accessed 2 July 2021.

https://scikit-learn.org/stable/modules/classes.html#module-sklearn.metrics accessed 2 July 2021.

https://github.com/yashveersinghsohi/Statistical_Modeling_for_Time_Series_Forecasting accessed 4 July 2021.

https://pub.towardsai.net/statistical-forecasting-of-time-series-data-part-4-forecasting-volatility-using-garch-1e9ff832f7e5 accessed 4 July 2021.

https://machinelearningmastery.com/develop-arch-and-garch-models-for-time-series-forecasting-in-python/ accessed 4 July 2021.

https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingRegressor.html#sklearn.ensemble.GradientBoostingRegressor accessed 14 July 2021

https://scikit-learn.org/stable/auto_examples/ensemble/plot_gradient_boosting_regression.html accessed 14 July 2021

https://stackoverflow.com/questions/42033720/python-sklearn-multiple-linear-regression-display-r-squared accessed 15 July 2021.

https://gist.github.com/bshishov/5dc237f59f019b26145648e2124ca1c9 accessed 17 July 2021

https://scikit-learn.org/stable/modules/generated/sklearn.metrics.mean_squared_error accessed 17 July 2021.

https://www.theguardian.com/business/2020/oct/30/european-markets-suffer-worst-weekly-falls-since-june accessed 23 July 2021.

https://www.bloomberg.com/news/articles/2020-10-27/asian-stock-futures-signal-drop-dollar-slips-markets-wrap accessed 23 July 2021.

APPENDIX

A. Libraries Installation

Input: `py get-pip.py`

Output:

```
C:\Users\Luca->py get-pip.py
Collecting pip
  Using cached pip-21.1.2-py3-none-any.whl (1.5 MB)
Installing collected packages: pip
  Attempting uninstall: pip
    Found existing installation: pip 21.1.2
    Uninstalling pip-21.1.2:
      Successfully uninstalled pip-21.1.2
Successfully installed pip-21.1.2
```

Input: `pip install numpy`

Output:

```
C:\Users\Luca->pip install numpy
Collecting numpy
  Downloading numpy-1.20.3-cp39-cp39-win_amd64.whl (13.7 MB)
     |████████████████████████████████| 13.7 MB 227 kB/s
Installing collected packages: numpy
Successfully installed numpy-1.20.3
```

Input: `pip install scikit-learn`

Output:

```
C:\Users\Luca->pip install scikit-learn
Collecting scikit-learn
  Downloading scikit_learn-0.24.2-cp39-cp39-win_amd64.whl (6.9 MB)
     |████████████████████████████████| 6.9 MB 656 kB/s
Collecting joblib>=0.11
  Downloading joblib-1.0.1-py3-none-any.whl (303 kB)
     |████████████████████████████████| 303 kB 1.3 MB/s
Collecting scipy>=0.19.1
  Downloading scipy-1.6.3-cp39-cp39-win_amd64.whl (32.7 MB)
     |████████████████████████████████| 32.7 MB 229 kB/s
Collecting threadpoolctl>=2.0.0
  Downloading threadpoolctl-2.1.0-py3-none-any.whl (12 kB)
Requirement already satisfied: numpy>=1.13.3 in c:\users\luca-\appdata\local\programs\python\py
thon39\lib\site-packages (from scikit-learn) (1.20.3)
Installing collected packages: threadpoolctl, scipy, joblib, scikit-learn
Successfully installed joblib-1.0.1 scikit-learn-0.24.2 scipy-1.6.3 threadpoolctl-2.1.0
```

Input: `pip install matplotlib`

Output:

```
C:\Users\Luca-\Desktop\Algorithm Data>pip install matplotlib
Collecting matplotlib
  Using cached matplotlib-3.4.2-cp39-cp39-win_amd64.whl (7.1 MB)
Requirement already satisfied: pyparsing>=2.2.1 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (2.4.7)
Requirement already satisfied: numpy>=1.16 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (1.20.3)
Requirement already satisfied: cycler>=0.10 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (0.10.0)
Requirement already satisfied: kiwisolver>=1.0.1 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (1.3.1)
Requirement already satisfied: pillow>=6.2.0 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (8.2.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from matplotlib) (2.8.1)
Requirement already satisfied: six in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from cycler>=0.10->matplotlib) (1.16.0)
Installing collected packages: matplotlib
Successfully installed matplotlib-3.4.2
```

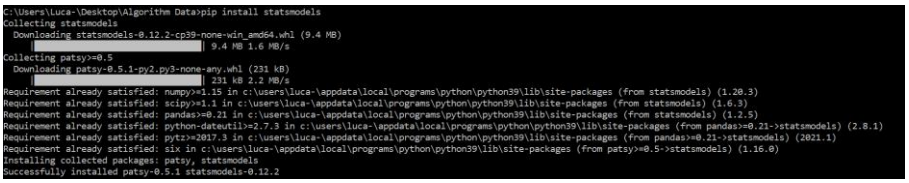
Input: `pip install pandas`

Output:

```
C:\Users\Luca-\Desktop\Algorithm Data>pip install pandas
Collecting pandas
  Using cached pandas-1.2.5-cp39-cp39-win_amd64.whl (9.3 MB)
Requirement already satisfied: numpy>=1.16.5 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from pandas) (1.20.3)
Requirement already satisfied: python-dateutil>=2.7.3 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from pandas) (2.8.1)
Requirement already satisfied: pytz>=2017.3 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from pandas) (2021.1)
Requirement already satisfied: six>=1.5 in c:\users\luca-\appdata\local\programs\python\python39\lib\site-packages (from python-dateutil>=2.7.3->pandas) (1.16.0)
Installing collected packages: pandas
Successfully installed pandas-1.2.5
```

Input: `pip install statsmodels`

Output:



Input: `pip install seaborn`

Output:

B. Selected Predictor variables

date	exc	ltr	vol	dp	ep	Rfree	rrel	lty	tms	defspr	infl	de
200105	-0,06564	-0,02805	0,01313	-1,75945	-1,32552	0,04357	-0,00212	0,05165	0,00808	0,00780	0,00500	1,32737
200106	-0,08577	0,00444	0,01222	-1,73993	-1,31281	0,04370	-0,00118	0,05104	0,00734	0,00790	0,00200	1,32535
200107	-0,07922	0,01689	0,01325	-1,72354	-1,31387	0,04261	-0,00159	0,04881	0,00620	0,00840	-0,00200	1,31181
200108	-0,12779	0,00701	0,01203	-1,67572	-1,28780	0,03906	-0,00424	0,04785	0,00879	0,00830	-0,00100	1,30122
200109	-0,16260	-0,00138	0,03306	-1,62160	-1,23553	0,03537	-0,00642	0,04800	0,01263	0,00860	0,00300	1,31248
200110	0,02043	0,03238	0,02049	-1,64397	-1,28126	0,03330	-0,00572	0,04373	0,01043	0,00880	0,00100	1,28309
200111	0,01745	-0,01422	0,01523	-1,66555	-1,33445	0,03290	-0,00300	0,04557	0,01267	0,00840	-0,00100	1,24811
200112	0,00678	-0,03305	0,01643	-1,69037	-1,35257	0,03284	-0,00101	0,04997	0,01713	0,01280	0,00400	1,24975
200201	-0,06937	0,00787	0,01314	-1,67572	-1,33866	0,03302	0,00001	0,04887	0,01585	0,01320	0,00100	1,25179
200202	-0,04583	-0,00408	0,01334	-1,68613	-1,30492	0,03335	0,00042	0,04946	0,01611	0,01380	0,00200	1,29213
200203	0,00951	-0,02186	0,00858	-1,69897	-1,42144	0,03350	0,00043	0,05253	0,01903	0,01300	0,00600	1,19525
200204	-0,09113	0,00862	0,01068	-1,67985	-1,41414	0,03408	0,00079	0,05134	0,01726	0,01270	0,00500	1,18790
200205	-0,07647	-0,00354	0,01142	-1,66154	-1,39568	0,03405	0,00041	0,05184	0,01779	0,01340	0,00200	1,19049
200206	-0,12275	0,01708	0,02162	-1,62160	-1,37694	0,03353	-0,00035	0,04949	0,01596	0,01320	0,00000	1,17768
200207	-0,18711	0,01513	0,03339	-1,55440	-1,31597	0,03297	-0,00092	0,04755	0,01458	0,01370	-0,00100	1,18118
200208	-0,02385	0,01432	0,03075	-1,55752	-1,31994	0,03256	-0,00095	0,04555	0,01299	0,01210	0,00100	1,17999
200209	-0,23833	0,02200	0,03066	-1,46852	-1,22376	0,03209	-0,00093	0,04268	0,01059	0,01250	0,00300	1,20001
200210	0,10264	-0,01824	0,02999	-1,52578	-1,28171	0,03076	-0,00178	0,04539	0,01463	0,01410	0,00200	1,19042
200211	0,02430	0,00106	0,01968	-1,57025	-1,29973	0,02899	-0,00282	0,04483	0,01584	0,01310	-0,00100	1,20814
200212	-0,13528	0,02180	0,01854	-1,52433	-1,25406	0,02792	-0,00269	0,04189	0,01397	0,01240	0,00500	1,21551
200301	-0,08619	-0,02513	0,01875	-1,49894	-1,22917	0,02652	-0,00270	0,04075	0,01423	0,01180	0,00000	1,21947
200302	-0,07395	0,01252	0,02190	-1,47886	-1,29336	0,02499	-0,00282	0,03897	0,01398	0,01110	0,00400	1,14342
200303	-0,07475	-0,01113	0,03128	-1,45593	-1,33905	0,02502	-0,00146	0,04037	0,01535	0,01060	0,00600	1,08728
200304	0,10826	-0,00280	0,01754	-1,51286	-1,40993	0,02372	-0,00179	0,04068	0,01696	0,01110	0,00100	1,07300
200305	-0,01879	0,02996	0,01475	-1,50307	-1,36361	0,02129	-0,00328	0,03693	0,01564	0,01160	-0,00100	1,10227
200306	0,01659	-0,01014	0,01332	-1,51713	-1,40960	0,02108	-0,00227	0,03797	0,01689	0,01220	0,00100	1,07629
200307	0,01943	-0,08951	0,01305	-1,54821	-1,42716	0,02118	-0,00085	0,04205	0,02087	0,01130	-0,00100	1,08482
200308	-0,00670	0,00083	0,00904	-1,57025	-1,47349	0,02125	0,00006	0,04167	0,02042	0,01130	0,00100	1,06567
200309	-0,08618	0,01491	0,01093	-1,53760	-1,43870	0,02121	0,00004	0,04010	0,01889	0,01070	0,00400	1,06874
200310	0,05076	0,01419	0,01136	-1,58670	-1,47625	0,02136	0,00015	0,04328	0,02192	0,01030	0,00100	1,07482
200311	0,00006	-0,01022	0,00966	-1,59346	-1,48643	0,02124	-0,00004	0,04468	0,02344	0,01010	0,00000	1,07200
200312	0,02763	0,01283	0,00590	-1,59346	-1,51135	0,02068	-0,00059	0,04301	0,02233	0,00980	0,00300	1,05433
200401	0,00753	0,00391	0,00697	-1,60555	-1,52504	0,02049	-0,00060	0,04236	0,02187	0,00900	-0,00200	1,05279
200402	-0,00123	0,01281	0,00670	-1,59176	-1,34163	0,02008	-0,00072	0,04038	0,02030	0,00770	0,00200	1,18644
200403	-0,05750	0,01119	0,01259	-1,56225	-1,22634	0,02028	-0,00014	0,03921	0,01893	0,00780	0,00600	1,27391
200404	-0,02065	-0,02141	0,00963	-1,55440	-1,23477	0,02064	0,00036	0,04213	0,02149	0,00730	0,00500	1,25885
200405	-0,03458	-0,00851	0,01064	-1,54975	-1,22866	0,02091	0,00057	0,04376	0,02285	0,00710	0,00300	1,26134
200406	0,00117	-0,00292	0,00731	-1,56864	-1,23754	0,02094	0,00033	0,04313	0,02219	0,00770	0,00000	1,26754
200407	-0,05384	0,00704	0,00726	-1,55284	-1,22298	0,02092	0,00009	0,04214	0,02122	0,00800	-0,00200	1,26972
200408	-0,03924	0,01541	0,00796	-1,52724	-1,21669	0,02096	0,00004	0,04018	0,01922	0,00810	0,00200	1,25524
200409	-0,00067	0,00835	0,00624	-1,56225	-1,22737	0,02125	0,00030	0,03990	0,01865	0,00810	0,00100	1,27284
200410	0,00938	0,00809	0,00923	-1,57512	-1,24055	0,02147	0,00043	0,03854	0,01707	0,00740	0,00400	1,26969
200411	0,00124	-0,03595	0,00570	-1,58838	-1,24502	0,02150	0,00027	0,03788	0,01638	0,00680	-0,00100	1,27579

200412	0,00438	0,01130	0,00564	-1,60206	-1,25744	0,02123	-0,00018	0,03675	0,01552	0,00680	0,00400	1,27407
200501	-0,00984	0,01137	0,00523	-1,59688	-1,24944	0,02116	-0,00024	0,03543	0,01427	0,00660	-0,00600	1,27807
200502	0,00326	-0,01396	0,00528	-1,54668	-1,19061	0,02115	-0,00015	0,03736	0,01621	0,00620	0,00400	1,29906
200503	-0,02199	0,00338	0,00561	-1,51999	-1,15381	0,02115	-0,00003	0,03621	0,01506	0,00660	0,00700	1,31736
200504	-0,06302	0,02231	0,00780	-1,47886	-1,09342	0,02103	-0,00012	0,03398	0,01295	0,00720	0,00400	1,35251
200505	0,02793	-0,03581	0,00543	-1,50031	-1,12123	0,02089	-0,00022	0,03271	0,01182	0,00860	0,00200	1,33809
200506	0,01254	0,01171	0,00586	-1,51145	-1,13001	0,02097	-0,00005	0,03134	0,01037	0,00900	0,00100	1,33755
200507	0,02346	-0,00209	0,00640	-1,53760	-1,14457	0,02110	0,00014	0,03239	0,01129	0,00890	-0,00100	1,34338
200508	-0,04020	0,00537	0,00732	-1,53760	-1,12385	0,02117	0,00018	0,03103	0,00986	0,00870	0,00200	1,36815
200509	0,02751	-0,00010	0,00739	-1,55284	-1,14737	0,02173	0,00065	0,03152	0,00979	0,00900	0,00500	1,35340
200510	-0,05545	-0,02116	0,00893	-1,53910	-1,12613	0,02333	0,00200	0,03395	0,01062	0,00950	0,00300	1,36672
200511	0,01309	0,01738	0,00477	-1,53611	-1,14019	0,02443	0,00235	0,03449	0,01006	0,00970	-0,00200	1,34723
200512	0,01273	0,01296	0,00547	-1,56225	-1,15806	0,02481	0,00164	0,03306	0,00825	0,00950	0,00300	1,34902
200601	0,00527	-0,01346	0,00786	-1,54821	-1,16791	0,02567	0,00148	0,03477	0,00910	0,00950	-0,00400	1,32563
200602	-0,00460	-0,00140	0,00761	-1,53462	-1,15564	0,02686	0,00189	0,03487	0,00801	0,00920	0,00300	1,32794
200603	-0,00678	-0,02598	0,00606	-1,51286	-1,14051	0,02755	0,00177	0,03785	0,01030	0,00880	0,00600	1,32648
200604	-0,03208	-0,01423	0,00665	-1,51145	-1,14019	0,02848	0,00178	0,03947	0,01099	0,00840	0,00700	1,32561
200605	-0,08366	0,04350	0,01446	-1,48945	-1,09726	0,02942	0,00179	0,03987	0,01045	0,00800	0,00300	1,35743
200606	-0,02733	-0,01000	0,01279	-1,53760	-1,09795	0,03055	0,00207	0,04058	0,01003	0,00890	0,00100	1,40043
200607	-0,02005	0,01059	0,01038	-1,49894	-1,09342	0,03176	0,00227	0,03921	0,00745	0,00910	-0,00100	1,37087
200608	-0,00166	0,00950	0,00608	-1,51999	-1,09447	0,03281	0,00223	0,03755	0,00474	0,00910	0,00100	1,38879
200609	-0,01088	0,00970	0,00672	-1,53018	-1,10278	0,03442	0,00272	0,03706	0,00264	0,00920	0,00000	1,38757
200610	-0,00867	-0,00489	0,00431	-1,55596	-1,11461	0,03534	0,00234	0,03741	0,00207	0,00910	0,00100	1,39596
200611	-0,04058	-0,01560	0,00674	-1,54975	-1,11261	0,03618	0,00199	0,03691	0,00073	0,00870	0,00000	1,39290
200612	-0,00409	-0,02204	0,00659	-1,55440	-1,11261	0,03683	0,00152	0,03959	0,00276	0,00900	0,00400	1,39708
200701	-0,02335	-0,01095	0,00666	-1,53313	-1,13033	0,03747	0,00135	0,04095	0,00348	0,00940	-0,00500	1,35635
200702	-0,06029	0,01410	0,00772	-1,51145	-1,10517	0,03817	0,00134	0,03964	0,00147	0,00890	0,00300	1,36762
200703	-0,01627	-0,01101	0,01016	-1,51145	-1,11628	0,03898	0,00149	0,04064	0,00166	0,00970	0,00700	1,35401
200704	0,00940	-0,01242	0,00665	-1,61261	-1,13418	0,03991	0,00170	0,04146	0,00155	0,00920	0,00600	1,42183
200705	-0,01362	0,02638	0,00625	-1,64207	-1,13577	0,04064	0,00162	0,04410	0,00346	0,00920	0,00200	1,44577
200706	-0,04638	-0,01334	0,00839	-1,64016	-1,13322	0,04130	0,00145	0,04555	0,00425	0,00910	0,00100	1,44735
200707	-0,08398	0,01818	0,01076	-1,61618	-1,10551	0,04443	0,00382	0,04321	-0,00122	0,00920	-0,00200	1,46194
200708	-0,05124	0,00882	0,01466	-1,61798	-1,10619	0,04633	0,00420	0,04249	-0,00384	0,00860	0,00100	1,46266
200709	-0,02572	-0,01134	0,01060	-1,60033	-1,11628	0,04581	0,00179	0,04344	-0,00237	0,00850	0,00400	1,43363
200710	-0,02097	0,01433	0,00606	-1,61083	-1,11727	0,04534	-0,00018	0,04290	-0,00244	0,00820	0,00500	1,44176
200711	-0,06870	-0,01847	0,01035	-1,61798	-1,09726	0,04735	0,00152	0,04159	-0,00576	0,00960	0,00500	1,47457
200712	-0,04276	-0,01331	0,00709	-1,59688	-1,10106	0,04384	-0,00233	0,04324	-0,00060	0,01160	0,00400	1,45031
200801	-0,19113	0,03158	0,02475	-1,53165	-1,03342	0,04270	-0,00281	0,03933	-0,00337	0,01210	-0,00300	1,48211
200802	-0,06311	0,00466	0,01480	-1,52143	-1,03703	0,04494	0,00031	0,03874	-0,00620	0,01290	0,00300	1,46711
200803	-0,07296	-0,00231	0,01529	-1,48678	-1,02202	0,04673	0,00290	0,03901	-0,00772	0,01380	0,01000	1,45475
200804	0,00543	-0,01698	0,01209	-1,50864	-1,05038	0,04743	0,00264	0,04116	-0,00627	0,01420	0,00300	1,43628
200805	-0,06063	-0,00710	0,00693	-1,50307	-1,06145	0,04822	0,00186	0,04454	-0,00368	0,01360	0,00600	1,41605
200806	-0,16777	-0,01382	0,01020	-1,45100	-1,01410	0,04842	0,00096	0,04628	-0,00214	0,01390	0,00400	1,43082
200807	-0,04399	0,02203	0,01329	-1,45223	-1,00432	0,04846	0,00043	0,04352	-0,00494	0,01490	-0,00200	1,44598
200808	-0,04962	0,01411	0,01070	-1,43890	-1,00130	0,04897	0,00061	0,04174	-0,00723	0,01510	-0,00100	1,43703
200809	-0,15222	0,01294	0,02459	-1,40230	-0,96895	0,04987	0,00125	0,04030	-0,00957	0,01660	0,00200	1,44724

200810	-0,20044	0,00811	0,04303	-1,30190	-0,94694	0,04151	-0,00759	0,03878	-0,00273	0,02600	0,00000	1,37484
200811	-0,09671	0,01335	0,03354	-1,31336	-0,93247	0,03240	-0,01439	0,03251	0,00011	0,03090	-0,00500	1,40847
200812	-0,01717	0,02589	0,02539	-1,29158	-0,90363	0,02427	-0,01699	0,02947	0,00520	0,03380	-0,00100	1,42932
200901	-0,10923	-0,02907	0,02238	-1,25259	-0,85612	0,01924	-0,01348	0,03277	0,01353	0,03090	-0,00800	1,46309
200902	-0,14016	0,01502	0,01510	-1,32239	-0,91062	0,01622	-0,00908	0,03091	0,01469	0,02810	0,00400	1,45218
200903	0,03278	0,00900	0,02456	-1,35556	-0,94498	0,01412	-0,00579	0,02997	0,01585	0,02920	0,00400	1,43449
200904	0,12431	-0,01428	0,01794	-1,41567	-1,14983	0,01274	-0,00379	0,03183	0,01909	0,03000	0,00300	1,23119
200905	0,01925	-0,05355	0,01539	-1,43771	-1,19673	0,01220	-0,00216	0,03612	0,02392	0,02520	0,00000	1,20136
200906	-0,03012	0,01650	0,01492	-1,42713	-1,17406	0,00970	-0,00332	0,03385	0,02415	0,01890	0,00200	1,21555
200907	0,08533	0,00759	0,01368	-1,46218	-1,25551	0,00857	-0,00298	0,03304	0,02447	0,01680	-0,00700	1,16461
200908	0,04295	0,00272	0,01225	-1,49485	-1,46404	0,00769	-0,00247	0,03268	0,02499	0,01320	0,00300	1,02104
200909	0,02717	0,00238	0,01099	-1,51570	-1,25237	0,00735	-0,00131	0,03206	0,02471	0,01180	0,00000	1,21027
200910	-0,05313	-0,00030	0,01479	-1,49349	-1,26694	0,00714	-0,00073	0,03241	0,02527	0,01140	0,00200	1,17882
200911	0,01231	-0,01429	0,01273	-1,51004	-1,29994	0,00709	-0,00030	0,03160	0,02451	0,01130	0,00100	1,16162
200912	0,05145	-0,01954	0,00826	-1,54821	-1,32715	0,00677	-0,00042	0,03401	0,02724	0,01110	0,00300	1,16657
201001	-0,07215	0,01666	0,01074	-1,51286	-1,27807	0,00660	-0,00041	0,03200	0,02540	0,00990	-0,00800	1,18371
201002	-0,02400	0,00753	0,01381	-1,49757	-1,13481	0,00643	-0,00039	0,03107	0,02464	0,00990	0,00300	1,31966
201003	0,06523	-0,00004	0,00728	-1,53462	-1,12287	0,00643	-0,00017	0,03102	0,02459	0,01000	0,01100	1,36669
201004	-0,04662	0,01083	0,01385	-1,53462	-1,08778	0,00684	0,00036	0,02965	0,02281	0,00960	0,00400	1,41078
201005	-0,08342	0,00616	0,02659	-1,46980	-1,05192	0,00725	0,00068	0,02653	0,01928	0,01090	0,00100	1,39725
201006	-0,02271	0,00887	0,01599	-1,45717	-1,05038	0,00845	0,00161	0,02552	0,01707	0,01350	0,00000	1,38728
201007	0,05463	-0,01049	0,01311	-1,49757	-1,08063	0,00892	0,00140	0,02672	0,01780	0,01290	-0,00400	1,38584
201008	-0,05321	-0,01582	0,01149	-1,46471	-1,07737	0,00877	0,00056	0,02112	0,01235	0,01170	0,00200	1,35952
201009	0,03661	-0,01383	0,01159	-1,48017	-1,08529	0,00993	0,00122	0,02285	0,01292	0,01130	0,00300	1,36385
201010	0,02436	-0,02188	0,00878	-1,51286	-1,09656	0,01037	0,00116	0,02524	0,01487	0,01040	0,00300	1,37964
201011	-0,08079	0,00833	0,01197	-1,46852	-1,06333	0,01017	0,00048	0,02667	0,01650	0,01050	0,00100	1,38105
201012	0,04200	-0,02381	0,00848	-1,50585	-1,08565	0,01012	-0,00003	0,02949	0,01937	0,01080	0,00600	1,38705
201101	0,04517	-0,01816	0,00958	-1,51145	-1,10992	0,01081	0,00059	0,03163	0,02082	0,01050	-0,00700	1,36177
201102	0,00825	-0,00095	0,00833	-1,53760	-1,08778	0,01169	0,00132	0,03171	0,02002	0,00930	0,00400	1,41352
201103	-0,04762	-0,01501	0,01096	-1,51856	-1,09096	0,01313	0,00225	0,03358	0,02045	0,00900	0,01300	1,39194
201104	0,01974	0,07448	0,00855	-1,52578	-1,08099	0,01415	0,00228	0,03236	0,01821	0,00860	0,00600	1,41147
201105	-0,06564	0,01821	0,00919	-1,47237	-1,05269	0,01478	0,00179	0,03021	0,01543	0,00820	0,00000	1,39867
201106	-0,02054	-0,00079	0,01140	-1,47756	-1,05038	0,01585	0,00183	0,03028	0,01443	0,00760	0,00000	1,40669
201107	-0,07999	0,04069	0,01282	-1,44249	-1,03782	0,01540	0,00048	0,02545	0,01005	0,00830	-0,00600	1,38992
201108	-0,16365	-0,05662	0,02722	-1,39469	-0,98408	0,01525	-0,00009	0,02222	0,00697	0,00990	0,00200	1,41726
201109	-0,07028	0,02952	0,02808	-1,37059	-0,96284	0,01564	0,00014	0,01888	0,00324	0,01180	0,00700	1,42348
201110	0,07538	-0,01352	0,02053	-1,37059	-1,00561	0,01474	-0,00069	0,02025	0,00551	0,01390	0,00300	1,36295
201111	-0,03740	-0,04401	0,02537	-1,36552	-1,05423	0,01416	-0,00105	0,02281	0,00865	0,01270	0,00100	1,29528
201112	-0,01812	0,04194	0,01612	-1,35754	-1,01953	0,01215	-0,00270	0,01825	0,00610	0,01320	0,00300	1,33153
201201	0,03188	0,00052	0,01093	-1,38195	-1,09202	0,01043	-0,00325	0,01793	0,00750	0,01380	-0,00800	1,26550
201202	0,03019	-0,00245	0,00752	-1,39469	-1,14457	0,00855	-0,00370	0,01811	0,00956	0,01290	0,00500	1,21853
201203	-0,02138	0,00252	0,01144	-1,40340	-1,17493	0,00742	-0,00296	0,01798	0,01056	0,01240	0,01300	1,19445
201204	-0,07829	-0,00921	0,01675	-1,38091	-1,15106	0,00683	-0,00197	0,01663	0,00980	0,01230	0,00500	1,19968
201205	-0,09135	0,04094	0,01091	-1,37161	-1,11594	0,00657	-0,00103	0,01211	0,00554	0,01270	-0,00100	1,22910
201206	0,06158	-0,03570	0,01551	-1,40561	-1,13799	0,00496	-0,00198	0,01579	0,01083	0,01380	-0,00100	1,23517
201207	0,02326	0,02831	0,01572	-1,45967	-1,15987	0,00332	-0,00280	0,01283	0,00951	0,01470	-0,00500	1,25848

201208	0,04580	-0,00531	0,01290	-1,48017	-1,19117	0,00246	-0,00249	0,01340	0,01094	0,01430	0,00400	1,24262
201209	0,00346	-0,02912	0,01187	-1,48017	-1,19033	0,00208	-0,00150	0,01434	0,01226	0,01350	0,00700	1,24350
201210	0,01800	-0,00551	0,00997	-1,48945	-1,20952	0,00192	-0,00070	0,01459	0,01267	0,01110	0,00200	1,23145
201211	0,02635	0,01019	0,00911	-1,49894	-1,17725	0,00185	-0,00030	0,01383	0,01198	0,01010	-0,00200	1,27326
201212	0,02124	0,00575	0,00609	-1,50585	-1,17984	0,00205	0,00010	0,01306	0,01101	0,00980	0,00400	1,27631
201301	0,02288	-0,03416	0,00620	-1,52578	-1,16791	0,00223	0,00029	0,01677	0,01454	0,00930	-0,01000	1,30642
201302	-0,02808	0,02037	0,01322	-1,51286	-1,17114	0,00206	0,00001	0,01458	0,01252	0,00950	0,00400	1,29178
201303	-0,00571	0,01637	0,00943	-1,51713	-1,16137	0,00209	-0,00003	0,01282	0,01073	0,00920	0,01200	1,30633
201304	0,03097	0,00643	0,01205	-1,53313	-1,18949	0,00201	-0,00012	0,01211	0,01010	0,00860	-0,00100	1,28890
201305	0,01893	-0,02773	0,00674	-1,54212	-1,22141	0,00210	0,00005	0,01511	0,01301	0,00840	0,00100	1,26257
201306	-0,06442	-0,01956	0,01133	-1,51286	-1,19728	0,00221	0,00015	0,01730	0,01509	0,00920	0,00100	1,26358
201307	0,05941	0,00492	0,00853	-1,54363	-1,19728	0,00226	0,00015	0,01676	0,01450	0,00980	-0,00500	1,28928
201308	-0,01927	-0,01628	0,00859	-1,53611	-1,21192	0,00223	0,00004	0,01855	0,01632	0,00880	0,00100	1,26750
201309	0,05895	0,05169	0,00611	-1,55440	-1,21906	0,00226	0,00002	0,01780	0,01554	0,00830	0,00500	1,27508
201310	0,05643	0,00887	0,00643	-1,57025	-1,24846	0,00223	-0,00002	0,01679	0,01456	0,00780	-0,00100	1,25774
201311	0,00334	-0,00154	0,00497	-1,57349	-1,24477	0,00273	0,00049	0,01693	0,01420	0,00750	-0,00100	1,26408
201312	0,00430	-0,02174	0,00908	-1,57512	-1,24576	0,00292	0,00051	0,01941	0,01649	0,00760	0,00400	1,26438
201401	-0,03392	0,00342	0,00963	-1,57187	-1,23019	0,00288	0,00025	0,01658	0,01370	0,00700	-0,01100	1,27774
201402	0,04085	0,00258	0,00766	-1,58503	-1,24895	0,00305	0,00021	0,01628	0,01323	0,00650	0,00300	1,26908
201403	0,00063	0,00512	0,01025	-1,58503	-1,33163	0,00329	0,00034	0,01570	0,01241	0,00680	0,00900	1,19029
201404	0,00833	0,00873	0,00714	-1,58503	-1,34596	0,00324	0,00017	0,01470	0,01146	0,00660	0,00100	1,17762
201405	0,01194	-0,01166	0,00676	-1,48413	-1,37273	0,00241	-0,00078	0,01355	0,01114	0,00600	-0,00100	1,08115
201406	-0,00710	0,00938	0,00443	-1,46092	-1,38166	0,00205	-0,00093	0,01250	0,01045	0,00550	0,00100	1,05737
201407	-0,03746	0,00751	0,00835	-1,50724	-1,36549	0,00191	-0,00065	0,01166	0,00975	0,00570	-0,00600	1,10381
201408	0,01719	0,02486	0,00720	-1,49214	-1,40841	0,00097	-0,00115	0,00887	0,00790	0,00610	0,00100	1,05945
201409	0,01584	-0,04999	0,00780	-1,46092	-1,39005	0,00083	-0,00082	0,00946	0,00863	0,00690	0,00400	1,05099
201410	-0,03634	0,00916	0,01444	-1,55129	-1,37912	0,00081	-0,00043	0,00841	0,00760	0,00770	-0,00100	1,12484
201411	0,04244	0,01330	0,00988	-1,54363	-1,39777	0,00081	-0,00006	0,00702	0,00621	0,00870	-0,00200	1,10436
201412	-0,03330	0,01450	0,01363	-1,55284	-1,34341	0,00063	-0,00019	0,00541	0,00478	0,00950	-0,00100	1,15590
201501	0,06264	-0,02380	0,01428	-1,51570	-1,38561	0,00048	-0,00027	0,00312	0,00264	0,00990	-0,01500	1,09389
201502	0,07099	-0,00114	0,00675	-1,50446	-1,32756	0,00027	-0,00037	0,00324	0,00297	0,00900	0,00600	1,13325
201503	0,02692	0,01329	0,00880	-1,49757	-1,29754	0,00005	-0,00041	0,00183	0,00178	0,00900	0,01200	1,15416
201504	-0,02237	-0,01733	0,01065	-1,51286	-1,29425	-0,00010	-0,00037	0,00363	0,00373	0,00960	0,00400	1,16891
201505	-0,01247	-0,01168	0,01103	-1,54061	-1,26928	-0,00014	-0,00021	0,00487	0,00501	0,00910	0,00300	1,21377
201506	-0,04189	-0,02666	0,01345	-1,55909	-1,24650	-0,00019	-0,00012	0,00768	0,00787	0,00940	0,00000	1,25078
201507	0,05023	0,05889	0,01453	-1,53313	-1,27830	-0,00028	-0,00013	0,00650	0,00678	0,01050	-0,00600	1,19936
201508	-0,09645	-0,01418	0,01622	-1,57349	-1,21378	-0,00037	-0,00017	0,00793	0,00830	0,01150	0,00000	1,29635
201509	-0,05306	0,01906	0,01563	-1,60906	-1,20222	-0,00054	-0,00026	0,00587	0,00641	0,01270	0,00200	1,33842
201510	0,09750	0,00571	0,01066	-1,55440	-1,27114	-0,00088	-0,00048	0,00523	0,00611	0,01390	0,00100	1,22283
201511	0,02548	0,00428	0,00907	-1,56384	-1,33284	-0,00126	-0,00067	0,00475	0,00601	0,01400	-0,00400	1,17331
201512	-0,07057	-0,01495	0,01339	-1,59007	-1,30038	-0,00146	-0,00057	0,00635	0,00781	0,01490	0,00000	1,22277
201601	-0,07050	-0,01731	0,01644	-1,65170	-1,26905	-0,00184	-0,00064	0,00335	0,00519	0,01450	-0,01500	1,30153
201602	-0,03317	0,02180	0,01911	-1,69037	-1,24650	-0,00229	-0,00077	0,00110	0,00339	0,01380	0,00200	1,35609
201603	0,01989	-0,00480	0,01049	-1,69680	-1,26811	-0,00250	-0,00063	0,00155	0,00405	0,01310	0,01200	1,33806
201604	0,00772	-0,01220	0,01122	-1,65365	-1,29645	-0,00258	-0,00037	0,00280	0,00538	0,01170	0,00200	1,27552
201605	0,01158	0,01249	0,01034	-1,61979	-1,29951	-0,00268	-0,00023	0,00146	0,00414	0,01030	0,00400	1,24646

201606	-0,06707	0,02544	0,02400	-1,61083	-1,27021	-0,00295	-0,00037	-0,00127	0,00168	0,01030	0,00200	1,26816
201607	0,04305	-0,04685	0,01001	-1,60555	-1,28375	-0,00299	-0,00025	-0,00120	0,00179	0,00940	-0,00500	1,25067
201608	0,01076	-0,00595	0,00777	-1,56067	-1,29645	-0,00302	-0,00015	-0,00061	0,00241	0,00920	0,00100	1,20380
201609	-0,00693	0,00540	0,00867	-1,54061	-1,30600	-0,00309	-0,00011	-0,00117	0,00192	0,00900	0,00400	1,17964
201610	0,01750	-0,02774	0,00593	-1,57512	-1,31513	-0,00313	-0,00010	0,00162	0,00475	0,00870	0,00200	1,19769
201611	-0,00119	-0,01077	0,00922	-1,58336	-1,25792	-0,00316	-0,00008	0,00275	0,00591	0,00850	-0,00400	1,25871
201612	0,07538	0,00715	0,00656	-1,52143	-1,28959	-0,00326	-0,00013	0,00207	0,00533	0,00770	0,00500	1,17978
201701	-0,01835	0,00136	0,00539	-1,50585	-1,27623	-0,00329	-0,00011	0,00437	0,00766	0,00740	-0,00900	1,17992
201702	0,02715	0,02284	0,00511	-1,46980	-1,27231	-0,00330	-0,00006	0,00206	0,00536	0,00690	0,00400	1,15523
201703	0,05318	-0,01213	0,00553	-1,44009	-1,26055	-0,00331	-0,00003	0,00331	0,00662	0,00670	0,00800	1,14243
201704	0,01662	0,00037	0,00896	-1,47237	-1,25624	-0,00330	0,00000	0,00323	0,00653	0,00700	0,00600	1,17205
201705	-0,00140	0,00262	0,00519	-1,50169	-1,24452	-0,00331	0,00000	0,00300	0,00631	0,00700	-0,00100	1,20664
201706	-0,03222	-0,01600	0,00670	-1,50585	-1,22917	-0,00331	0,00000	0,00473	0,00804	0,00690	0,00100	1,22509
201707	0,00217	0,01781	0,00570	-1,53018	-1,22789	-0,00330	0,00001	0,00540	0,00870	0,00690	-0,00500	1,24619
201708	-0,00812	0,01632	0,00642	-1,53313	-1,19811	-0,00330	0,00000	0,00361	0,00691	0,00680	0,00300	1,27963
201709	0,04943	-0,00962	0,00385	-1,51145	-1,22089	-0,00330	0,00000	0,00463	0,00793	0,00670	0,00400	1,23799
201710	0,02176	0,00940	0,00369	-1,51570	-1,22634	-0,00330	0,00000	0,00363	0,00693	0,00720	0,00000	1,23595
201711	-0,02872	-0,00136	0,00487	-1,53910	-1,22037	-0,00328	0,00001	0,00368	0,00696	0,00700	-0,00200	1,26118
201712	-0,01865	-0,00463	0,00621	-1,57512	-1,23172	-0,00329	0,00000	0,00424	0,00753	0,00710	0,00300	1,27879
201801	0,02962	-0,02593	0,00563	-1,55596	-1,24452	-0,00329	0,00000	0,00695	0,01024	0,00710	-0,00900	1,25024
201802	-0,04834	0,00310	0,01048	-1,59007	-1,19424	-0,00328	0,00000	0,00659	0,00987	0,00690	0,00200	1,33145
201803	-0,02278	0,01632	0,00944	-1,64207	-1,16316	-0,00329	0,00000	0,00497	0,00826	0,00770	0,01100	1,41173
201804	0,05076	-0,00662	0,00545	-1,57675	-1,19948	-0,00326	0,00003	0,00562	0,00888	0,00820	0,00400	1,31453
201805	-0,03741	0,02091	0,00540	-1,58004	-1,17782	-0,00323	0,00005	0,00343	0,00666	0,00830	0,00600	1,34149
201806	-0,00325	0,00311	0,00767	-1,58004	-1,17696	-0,00321	0,00005	0,00308	0,00629	0,00870	0,00100	1,34248
201807	0,03754	-0,03708	0,00491	-1,56067	-1,18498	-0,00320	0,00004	0,00443	0,00763	0,00920	-0,00200	1,31705
201808	-0,03834	0,01105	0,00634	-1,57840	-1,18949	-0,00319	0,00002	0,00332	0,00651	0,00890	0,00200	1,32695
201809	0,00186	-0,01374	0,00642	-1,59860	-1,19257	-0,00318	0,00002	0,00474	0,00792	0,00900	0,00400	1,34047
201810	-0,06117	0,00880	0,00868	-1,63451	-1,15594	-0,00317	0,00002	0,00384	0,00701	0,00930	0,00200	1,41401
201811	-0,00765	0,00732	0,00599	-1,62525	-1,14051	-0,00312	0,00006	0,00309	0,00621	0,01000	-0,00600	1,42502
201812	-0,05563	0,00605	0,00987	-1,63451	-1,11594	-0,00308	0,00007	0,00246	0,00554	0,01110	0,00000	1,46469
201901	0,05130	0,00896	0,00803	-1,60206	-1,13925	-0,00309	0,00004	0,00155	0,00464	0,01190	-0,01000	1,40624
201902	0,04300	-0,00303	0,00777	-1,57840	-1,18837	-0,00310	0,00000	0,00186	0,00496	0,01160	0,00300	1,32821
201903	0,01608	0,02500	0,00707	-1,54212	-1,20249	-0,00311	-0,00002	-0,00068	0,00243	0,01070	0,01000	1,28244
201904	0,04746	-0,00849	0,00365	-1,56225	-1,22660	-0,00312	-0,00003	0,00011	0,00323	0,01010	0,00700	1,27364
201905	-0,06896	0,02043	0,00868	-1,57349	-1,19562	-0,00329	-0,00018	-0,00203	0,00126	0,00960	0,00100	1,31604
201906	0,05725	0,01160	0,00621	-1,54975	-1,22167	-0,00366	-0,00048	-0,00328	0,00038	0,01040	0,00200	1,26855
201907	-0,00197	-0,01104	0,00581	-1,56543	-1,23930	-0,00409	-0,00073	-0,00442	-0,00033	0,00990	-0,00500	1,26316
201908	-0,01163	0,02678	0,01174	-1,54975	-1,23603	-0,00418	-0,00051	-0,00703	-0,00285	0,00890	0,00100	1,25381
201909	0,04080	-0,01424	0,00471	-1,53313	-1,25237	-0,00414	-0,00016	-0,00572	-0,00158	0,00880	0,00200	1,22419
201910	0,00975	-0,01704	0,00967	-1,49757	-1,26293	-0,00402	0,00011	-0,00403	-0,00001	0,00910	0,00100	1,18580
201911	0,02714	-0,00489	0,00336	-1,48413	-1,27623	-0,00395	0,00016	-0,00360	0,00035	0,00880	-0,00300	1,16290
201912	0,01116	-0,01642	0,00754	-1,46597	-1,28240	-0,00392	0,00012	-0,00187	0,00205	0,00870	0,00300	1,14315
202001	-0,02823	0,02638	0,00675	-1,51713	-1,28035	-0,00410	-0,00013	-0,00434	-0,00024	0,00830	-0,01000	1,18493
202002	-0,08941	0,01670	0,01394	-1,56067	-1,25551	-0,00417	-0,00018	-0,00608	-0,00191	0,00830	0,00200	1,24305
202003	-0,17789	-0,01583	0,03976	-1,66154	-1,17464	-0,00254	0,00152	-0,00469	-0,00215	0,01270	0,00500	1,41451

202004	0,04936	0,01225	0,01853	-1,69465	-1,23121	-0,00272	0,00088	-0,00589	-0,00317	0,01700	0,00300	1,37640
202005	0,04091	-0,01407	0,01679	-1,79317	-1,28126	-0,00377	-0,00062	-0,00448	-0,00071	0,01450	-0,00100	1,39954
202006	0,05853	0,00325	0,01722	-1,73518	-1,30835	-0,00445	-0,00144	-0,00453	-0,00008	0,01200	0,00300	1,32624
202007	-0,01865	0,00689	0,01077	-1,71220	-1,54195	-0,00481	-0,00116	-0,00531	-0,00050	0,01170	-0,00400	1,11041
202008	0,03047	-0,01409	0,00969	-1,69250	-1,58184	-0,00493	-0,00058	-0,00398	0,00095	0,01020	-0,00400	1,06996
202009	-0,02441	0,01193	0,01249	-1,71220	-1,57101	-0,00510	-0,00038	-0,00521	-0,00011	0,01050	0,00100	1,08987
202010	-0,07657	0,00986	0,01176	-1,76447	-1,61511	-0,00522	-0,00028	-0,00625	-0,00103	0,01090	0,00200	1,09248
202011	0,16604	-0,00561	0,01402	-1,70997	-1,72337	-0,00540	-0,00031	-0,00571	-0,00031	0,01000	-0,00300	0,99222
202012	0,01706	-0,00023	0,00680	-1,71220	-1,73223	-0,00549	-0,00025	-0,00575	-0,00026	0,00900	0,00300	0,98843
202101	-0,02024	-0,00351	0,00940	-1,70997	-1,71941	-0,00543	-0,00006	-0,00520	0,00023	0,00790	0,00200	0,99450
202102	0,04356	-0,02614	0,00743	-1,65561	-1,70757	-0,00541	0,00003	-0,00257	0,00284	0,00720	0,00200	0,96957
202103	0,07488	0,00353	0,00615	-1,54668	-1,74414	-0,00540	0,00004	-0,00292	0,00248	0,00700	0,00900	0,88679
202104	0,01407	-0,00985	0,00621	-1,53018	-1,66539	-0,00603	-0,00062	-0,00200	0,00403	0,00700	0,00600	0,91881
202105	0,01615	-0,00173	0,00922	-1,51713	-1,64532	-0,00643	-0,00082	-0,00183	0,00460	0,00660	0,00300	0,92208

C. Excess Return forecasts

Input: `py exc_BRT.py`

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]



# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data. We will also set the regression model parameters.
#
# n_estimators : the number of boosting stages that will be performed.
# Later, we will plot deviance against boosting iterations.
#
# max_depth : limits the number of nodes in the tree.
# The best value depends on the interaction of the input variables.
#
# min_samples_split : the minimum number of samples required to split an
# internal node.
#
# learning_rate : how much the contribution of each tree will shrink.
#
# loss : loss function to optimize. The least squares function is  used in this
# case however, there are many other options
# There is an additional parameter to tune the L2 regularization term which is called regularization rate (lambda).

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)

print (prediction)

#Out-of-sample Forecasting Accurancy


print('R2 equal to: %.4f' % r2_score(y_test, prediction))

def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.mean((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int))

print('MDA equal to: %.4f' % mda(np.array(y_test), prediction))
```

Output:

```
[ 1.25760453e-02  4.98750781e-03 -1.43551445e-02  1.62290069e-02
  9.89775276e-04 -4.39765962e-03 -6.80558121e-03 -2.91011478e-03
 -9.19645459e-03 -4.31291003e-04  2.44943072e-02 -3.55421140e-03
  1.58634397e-02  1.52039026e-02  2.08482164e-03  3.69436505e-02
  1.86864677e-04  1.87746796e-02  5.18939733e-04  1.59252863e-02
 -8.96495831e-04  2.41748169e-05 -6.76036128e-03 -2.10957103e-03
 -1.21595334e-02  2.89626035e-03  1.83296242e-02  1.53488051e-02
  2.59460768e-02  2.03417842e-03  1.96205222e-02  8.62503163e-03
  1.72123068e-02  8.38439494e-04  2.99368505e-03 -5.54168075e-03]
R2 equal to: -0.0128
MDA equal to: 0.5714
```

Input: py exc_Linear.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble, linear_model
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

print (X_test)

# Fit regression model (reg)
# -------------------------------------
# Now we will initiate the multivariate linear regression model.


reg = linear_model.LinearRegression()

# Train the model using the training sets

reg.fit(X_train, y_train)

print (y_test)
# Make predictions using the testing set

prediction = reg.predict(X_test)
print(prediction)

#Out-of-sample Forecasting Accurancy

print('R2 equal to: %.4f' % r2_score(y_test, prediction))

def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.mean((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int))

print('MDA equal to: %.4f' % mda(np.array(y_test), prediction))
```

Output:

```
[ 0.00467011  0.00384202 -0.00138216  0.0010178   0.00159252 -0.00425209
 -0.00771597 -0.00650063 -0.00184704  0.00708922  0.01519662  0.00638546
 -0.00163967  0.00862373 -0.00774597  0.00913112  0.00414464  0.00941402
  0.00490575  0.01228959 -0.00037468  0.00371609  0.0116769   0.00858873
 -0.00958564  0.00264382 -0.01492789 -0.01041114 -0.00624001 -0.01644619
  0.00660016 -0.01068614 -0.0111532  -0.0046994   0.01279592  0.00541455]
R2 equal to: -0.0631
MDA equal to: 0.4571
```

Input: py exc_CumSum.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble, linear_model
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

#BRT
# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data. We will also set the regression model parameters.
#
# n_estimators : the number of boosting stages that will be performed.
# Later, we will plot deviance against boosting iterations.
#
# max_depth : limits the number of nodes in the tree.
# The best value depends on the interaction of the input variables.
#
# min_samples_split : the minimum number of samples required to split an
# internal node.
#
# learning_rate : how much the contribution of each tree will shrink.
#
# loss : loss function to optimize. The least squares function is  used in this
# case however, there are many other options
# There is an additional parameter to tune the L2 regularization term which is called regularization rate (lambda).

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)

print (prediction)


#LINEAR
# Fit regression model (reg)
# -------------------------------------
# Now we will initiate the multivariate linear regression model.


reg2 = linear_model.LinearRegression(normalize=True)

# Train the model using the training sets

reg2.fit(X_train, y_train)

# Make predictions using the testing set

prediction2 = reg2.predict(X_test)
print(prediction2)


CumSum_series =((prediction2-y_test)**2) - ((prediction-y_test)**2)
print(CumSum_series)
series = pd.Series(CumSum_series)
cumsum =  series.cumsum()
print(cumsum)
print (np.array(cumsum))
```

```
#PLOT
dates = []
ticks = []
count = 0
to_plot = data['date'].iloc[205:]
for i in to_plot:
  i = str(i)
  c = i[4:6] + '/' + i[2:4]
  i = c
  dates.append(i)
  if count%2 == 0:
    ticks.append(i)
  count += 1

Y_plot = []

for i in cumsum:
  Y_plot.append(i)


plt.style.use('default')
plt.plot(dates, Y_plot)
plt.ylabel('Cumulative Sum of Squared Error difference')
plt.xlabel('Date (mm/yy format)')
plt.xticks(ticks)
plt.axhline(y=0.0, color = 'r', linestyle = '--')
plt.grid()
plt.show()
```

Output:

```
[-1.63871088e-04 -8.53390253e-05  6.67099269e-04  4.78278110e-04
  6.03520380e-04  5.84234722e-04  4.41135704e-04  1.13666526e-03
  4.73914146e-04  2.88601904e-04  7.65353118e-04  2.31080656e-03
  4.00840077e-03  3.85603819e-03  3.68217263e-03  4.66023302e-03
  4.57077156e-03  4.81248325e-03  4.75617736e-03  4.40657143e-03
  4.18254812e-03  5.37255289e-03  3.83807709e-03  2.93353191e-03
  2.89630945e-03  2.86619031e-03  4.77037002e-03  3.42598206e-03
 -1.87971601e-03  4.17408038e-03  4.27731453e-03  3.54558455e-03
  5.86333142e-03  6.86642674e-03  6.77078655e-03  6.45205159e-03]
```

Input: py exc_CumSumMDA.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble, linear_model
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

#BRT
# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data. We will also set the regression model parameters.
#
# n_estimators : the number of boosting stages that will be performed.
# Later, we will plot deviance against boosting iterations.
#
# max_depth : limits the number of nodes in the tree.
# The best value depends on the interaction of the input variables.
#
# min_samples_split : the minimum number of samples required to split an
# internal node.
#
# learning_rate : how much the contribution of each tree will shrink.
#
# loss : loss function to optimize. The least squares function is  used in this
# case however, there are many other options
# There is an additional parameter to tune the L2 regularization term which is called regularization rate (lambda).

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)

print (prediction)


#LINEAR
# Fit regression model (reg)
# -------------------------------------
# Now we will initiate the multivariate linear regression model.


reg2 = linear_model.LinearRegression()

# Train the model using the training sets

reg2.fit(X_train, y_train)

# Make predictions using the testing set

prediction2 = reg2.predict(X_test)
print(prediction2)


def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.cumsum((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int))

cumsum = mda(np.array(y_test), prediction) - mda(np.array(y_test), prediction2)
print(cumsum)
```

```
#PLOT
dates = []
ticks = []
count = 0
to_plot = data['date'].iloc[206:]
for i in to_plot:
  i = str(i)
  c = i[4:6] + '/' + i[2:4]
  i = c
  dates.append(i)
  if count%2 == 0:
    ticks.append(i)
  count += 1

Y_plot = []

for i in cumsum:
  Y_plot.append(i)


plt.style.use('default')
plt.plot(dates, Y_plot)
plt.ylabel('Cumulative Sum of MDA difference')
plt.xlabel('Date (mm/yy format)')
plt.xticks(ticks)
plt.axhline(y=0.0, color = 'r', linestyle = '--')
plt.grid()
plt.show()
```

Output:

```
[0 0 0 1 1 1 1 2 2 2 2 3 4 4 4 4 4 4 4 4 5 4 3 3 3 4 5 5 5 5 5 6 5 5 5]
```

D. Volatility forecasts

Input: `py vol_BRT.py`

```
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
vol = data['vol']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], vol.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

print (X_test)

# %%
# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.We will also set the regression model parameters.
#
# n_estimators : the number of boosting stages that will be performed.
# Later, we will plot deviance against boosting iterations.
#
# max_depth : limits the number of nodes in the tree.
# The best value depends on the interaction of the input variables.
#
# min_samples_split : the minimum number of samples required to split an
# internal node.
#
# learning_rate : how much the contribution of each tree will shrink.
#
# loss : loss function to optimize. The least squares function is  used in this
# case however, there are many other options

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)

print(prediction)

#Out-of-sample Forecasting Accurancy

import statistics
median_vol = statistics.median(vol)
print(median_vol)
y_test2 = y_test - median_vol
print(y_test2)
prediction2= prediction - median_vol
print(prediction2)

print('R2 equal to: %.4f' % r2_score(y_test, prediction))
def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.mean((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int))

print('MDA equal to: %.4f' % mda(np.array(y_test2), prediction2))
```

Output:

```
[-0.00399994 -0.00291321 -0.00272694 -0.00357878 -0.0036897  -0.00130705
 -0.00249267 -0.00027543 -0.00046992 -0.00094037 -0.00246539 -0.00287472
 -0.00160634 -0.00301368 -0.0027662  -0.00230304 -0.00349723 -0.00253359
 -0.0035696  -0.00239561 -0.00277776  0.0024503   0.00840091  0.00234889
  0.00156248  0.00130671 -0.00077306 -0.00227049 -0.00183183 -0.00011312
 -0.00160643 -0.00320625 -0.00333494 -0.00254164 -0.00369792 -0.00344971]
R2 equal to: 0.2532
MDA equal to: 0.4000
```

Input: py vol_GARCH.py

```
#importing Required Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.stattools import adfuller, kpss
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from arch import arch_model
import warnings

sns.set()
warnings.filterwarnings("ignore")

# Load the data

data_backup = pd.read_csv("predictors.csv")

data = data_backup[["vol", "exc"]]

#Train and Test Split

train_df = data.iloc[:205]
test_df = data.iloc[205:]



# Building Dataframe for fitting GARCH model
garch_df = pd.DataFrame(data["exc"].shift(1).loc[data.index])
#garch_df return the exc shifted of one row down (exc t-1, subsequently in next step we will take from row 1

garch_df.at[train_df.index, "exc"] = train_df["exc"]

# Instantating the model with the full dataset, parameters and specifying the model to be a GARCH model
# model = arch_model(train, mean='Zero', vol='GARCH', p=1, q=1)
model = arch_model(garch_df["exc"][1:], p = 1, q = 1, vol = "GARCH")

# Fitting the model on all the data just before the date specified in "last_obs" argument
model_results = model.fit(last_obs = test_df.index[0], update_freq = 5)


# Printing the Summary table of the fitted model
model_results.summary()

# FORECASTING
# Building Predictions Data
predictions_df = test_df.copy()
# Predictions
predictions_df["Predictions"] = model_results.forecast().residual_variance.loc[test_df.index]

#Tests
y_test=predictions_df["vol"]
prediction=predictions_df["Predictions"]

print(y_test)
print(np.array(prediction))


import statistics
median_vol = statistics.median(data_backup["vol"])
print(median_vol)
y_test2 = y_test - median_vol
print(y_test2)
prediction2= prediction - median_vol
print(np.array(prediction2))


print('R2 equal to: %.4f' % r2_score(y_test, prediction))
def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.mean((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int))

print('MDA equal to: %.4f' % mda(np.array(y_test2), np.array(prediction2)))
```

Output:

```
[-0.00807854 -0.00842227 -0.00809328 -0.00816888 -0.00846237 -0.0078712
 -0.00829187 -0.00791735 -0.00737028 -0.00726497 -0.00774622 -0.00737963
 -0.00691725 -0.00651659 -0.00740294 -0.00798207 -0.00772294 -0.00811503
 -0.00812195 -0.00836071 -0.00848864 -0.00682315  0.00046127 -0.00198209
 -0.00381711 -0.00445374 -0.00603307 -0.00670102 -0.00744937 -0.00668624
  0.00080916 -0.00248116 -0.00473921 -0.0055378  -0.00490212 -0.00623415]
R2 equal to: -0.9796
MDA equal to: 0.4857
```

Input: py vol_CumSum.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
vol = data['vol']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], vol.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

print (X_test)


# %%
# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.We will also set the regression model parameters.
#
# n_estimators : the number of boosting stages that will be performed.
# Later, we will plot deviance against boosting iterations.
#
# max_depth : limits the number of nodes in the tree.
# The best value depends on the interaction of the input variables.
#
# min_samples_split : the minimum number of samples required to split an
# internal node.
#
# learning_rate : how much the contribution of each tree will shrink.
#
# loss : loss function to optimize. The least squares function is  used in this
# case however, there are many other options

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)

print(prediction)



#GARCH
#importing Required Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.stattools import adfuller, kpss
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from arch import arch_model
import warnings

sns.set()
warnings.filterwarnings("ignore")

# Load the data

data_backup = pd.read_csv("predictors.csv")

data2 = data_backup[["vol", "exc"]]

#Train and Test Split

train_df = data2.iloc[:205]
test_df = data2.iloc[205:]
```

```
# Building Dataframe for fitting GARCH model
garch_df = pd.DataFrame(data2["exc"].shift(1).loc[data.index])
#garch_df return the exc shifted of one row down (exc t-1, subsequently in next step we will take from row 1

garch_df.at[train_df.index, "exc"] = train_df["exc"]

# Instantating the model with the full dataset, parameters and specifying the model to be a GARCH model
# model = arch_model(train, mean='Zero', vol='GARCH', p=1, q=1)
model = arch_model(garch_df["exc"][1:], p = 1, q = 1, vol = "GARCH")

# Fitting the model on all the data just before the date specified in "last_obs" argument
model_results = model.fit(last_obs = test_df.index[0], update_freq = 5)


# Printing the Summary table of the fitted model
model_results.summary()

# FORECASTING
# Building Predictions Data
predictions_df = test_df.copy()
# Predictions
predictions_df["Predictions"] = model_results.forecast().residual_variance.loc[test_df.index]


#Tests
prediction2=predictions_df["Predictions"]

print(prediction2)
print(np.array(prediction2))

CumSum_series =((prediction2-y_test)**2) - ((prediction-y_test)**2)
print(CumSum_series)
series = pd.Series(CumSum_series)
cumsum =  series.cumsum()
print(np.array(cumsum))
#PLOT
dates = []
ticks = []
count = 0
to_plot = data['date'].iloc[205:]
for i in to_plot:
  i = str(i)
  c = i[4:6] + '/' + i[2:4]
  i = c
  dates.append(i)
  if count%2 == 0:
    ticks.append(i)
  count += 1

Y_plot = []

for i in cumsum:
  Y_plot.append(i)

plt.style.use('default')
plt.plot(dates, Y_plot)
plt.ylabel('Cumulative Sum of Squared Error difference')
plt.xlabel('Date (mm/yy format)')
plt.xticks(ticks)
plt.axhline(y=0.0, color = 'r', linestyle = '--')
plt.grid()
plt.show()
```

Output:

```
[3.12648089e-05 3.87837114e-05 5.89693931e-05 8.11832397e-05
 1.27658091e-04 1.36891562e-04 1.99209705e-04 2.33594215e-04
 2.58671831e-04 2.74910139e-04 2.63855124e-04 2.99399176e-04
 3.05466490e-04 3.10560651e-04 3.74314332e-04 3.73499411e-04
 4.18609261e-04 4.05252357e-04 4.37309313e-04 4.64291540e-04
 5.75529098e-04 1.16566927e-03 1.23281445e-03 1.28950265e-03
 1.37994943e-03 1.40844656e-03 1.44229477e-03 1.50669940e-03
 1.57968063e-03 1.67854667e-03 1.69162983e-03 1.68780315e-03
 1.69246932e-03 1.69464416e-03 1.69618507e-03 1.71947430e-03]
```

Input: py vol_CumSumMDA.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
vol = data['vol']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], vol.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

print (X_test)


# %%
# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.We will also set the regression model parameters.
#
# n_estimators : the number of boosting stages that will be performed.
# Later, we will plot deviance against boosting iterations.
#
# max_depth : limits the number of nodes in the tree.
# The best value depends on the interaction of the input variables.
#
# min_samples_split : the minimum number of samples required to split an
# internal node.
#
# learning_rate : how much the contribution of each tree will shrink.
#
# loss : loss function to optimize. The least squares function is  used in this
# case however, there are many other options

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)

print(prediction)



#GARCH
#importing Required Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.stattools import adfuller, kpss
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from arch import arch_model
import warnings

sns.set()
warnings.filterwarnings("ignore")

# Load the data

data_backup = pd.read_csv("predictors.csv")

data2 = data_backup[["vol", "exc"]]

#Train and Test Split

train_df = data2.iloc[:205]
test_df = data2.iloc[205:]
```

```
# Building Dataframe for fitting GARCH model
garch_df = pd.DataFrame(data2["exc"].shift(1).loc[data.index])
#garch_df return the exc shifted of one row down (exc t-1, subsequently in next step we will take from row 1

garch_df.at[train_df.index, "exc"] = train_df["exc"]

# Instantating the model with the full dataset, parameters and specifying the model to be a GARCH model
# model = arch_model(train, mean='Zero', vol='GARCH', p=1, q=1)
model = arch_model(garch_df["exc"][1:], p = 1, q = 1, vol = "GARCH")

# Fitting the model on all the data just before the date specified in "last_obs" argument
model_results = model.fit(last_obs = test_df.index[0], update_freq = 5)


# Printing the Summary table of the fitted model
model_results.summary()

# FORECASTING
# Building Predictions Data
predictions_df = test_df.copy()
# Predictions
predictions_df["Predictions"] = model_results.forecast().residual_variance.loc[test_df.index]


#Tests
prediction2=np.array(predictions_df["Predictions"])

print(prediction2)



#CumSum
def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.cumsum((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(i

cumsum = mda(np.array(y_test), prediction) - mda(np.array(y_test), prediction2)
print(cumsum)


#PLOT
dates = []
ticks = []
count = 0
to_plot = data['date'].iloc[206:]
for i in to_plot:
  i = str(i)
  c = i[4:6] + '/' + i[2:4]
  i = c
  dates.append(i)
  if count%2 == 0:
    ticks.append(i)
  count += 1

Y_plot = []

for i in cumsum:
  Y_plot.append(i)


plt.style.use('default')
plt.plot(dates, Y_plot)
plt.ylabel('Cumulative Sum of MDA difference')
plt.xlabel('Date (mm/yy format)')
plt.xticks(ticks)
plt.axhline(y=0.0, color = 'r', linestyle = '--')
plt.grid()
plt.show()
```

Output:

```
[-1 -1 -1 -1 -1 -1 -1  0  1  1  0  0  1  2  1  0 -1 -1 -2 -2 -2 -2 -2 -2
 -2 -2 -2 -3 -3 -2 -2 -2 -3 -4 -3]
```

E. Optimal Portfolio Weight forecasts

Input: py TWOSTEP_weight_BRT.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]
print (X_test)

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)
print(prediction)

#data2


data2 = pd.read_csv("predictors.csv")
vol = data['vol']
predictors2 = data.shift(1)

print(data2)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X2,y2 = predictors2.iloc[1:,1:], vol.iloc[1:]

print (X2)
print (y2)


X_train2 = X2.iloc[:204]
X_test2 = X2.iloc[204:]

y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

print (X_test2)
```

```
# Fit regression model
# ------------------------------------
#
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg2 = ensemble.GradientBoostingRegressor(**params)
reg2.fit(X_train2, y_train2)

print (y_test2)

prediction2 = reg2.predict(X_test2)

#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)


#Out-of-sample Forecasting Accurancy


print('R2 equal to: %.4f' % r2_score(w_real, w_predict))

def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.mean((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int))

print('MDA equal to: %.4f' % mda(np.array(w_real), w_predict))
```

Output:

```
[ 0.54419109  0.18215553 -0.47862012  0.64434332  0.01631073 -0.07667559
 -0.25004163 -0.0814226  -0.24686665 -0.03147235  0.83173907 -0.13086117
  0.44705948  0.43649024  0.04000801  1.19820371 -0.01116832  0.64495468
  0.03710989  0.53472315 -0.06753659  0.01611634 -0.09137568 -0.05831176
 -0.26459689  0.03578023  0.43754288  0.43590355  0.73113331 -0.00926241
  0.53682067  0.27795616  0.59092118 -0.01118838  0.08455766 -0.22396574]
R2 equal to: 0.0621
MDA equal to: 0.6286
```

Input: py weight_LinearGARCH.py

```
#LINEAR

import numpy as np
import pandas as pd
from sklearn import datasets, ensemble, linear_model
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

print (X_test)

# Fit regression model (reg)
# -------------------------------------
# Now we will initiate the multivariate linear regression model.


reg = linear_model.LinearRegression()

# Train the model using the training sets

reg.fit(X_train, y_train)

print (y_test)
# Make predictions using the testing set

prediction = reg.predict(X_test)
print(prediction)


#GARCH

#importing Required Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.stattools import adfuller, kpss
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from arch import arch_model
import warnings

sns.set()
warnings.filterwarnings("ignore")

# Load the data

data_backup = pd.read_csv("predictors.csv")

data2 = data_backup[["vol", "exc"]]

#Train and Test Split

train_df = data2.iloc[:205]
test_df = data2.iloc[205:]



# Building Dataframe for fitting GARCH model
garch_df = pd.DataFrame(data2["exc"].shift(1).loc[data2.index])
garch_df.at[train_df.index, "exc"] = 10*train_df["exc"]

# Instantating the model with the full dataset, parameters and specifying the model to be a GARCH model
model = arch_model(garch_df["exc"][1:], p = 1, q = 1, vol = "GARCH")

# Fitting the model on all the data just before the date specified in "last_obs" argument
model_results = model.fit(last_obs = test_df.index[0], update_freq = 5)

# Printing the Summary table of the fitted model
model_results.summary()
```

```
# FORECASTING
# Building Predictions Data
predictions_df = test_df.copy()
# Predictions
predictions_df["Predictions"] = model_results.forecast().residual_variance.loc[test_df.index]


#Tests
y_test2=predictions_df["vol"].iloc[:36]
vol_prediction= predictions_df["Predictions"]
prediction2 = np.array(vol_prediction.iloc[:36])

print(y_test2)
print(prediction2)

#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)


#Out-of-sample Forecasting Accurancy

print('R2 equal to: %.4f' % r2_score(w_real, w_predict))

def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.mean((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int))

print('MDA equal to: %.4f' % mda(np.array(w_real), w_predict))
```

Output:

```
[ 0.00759179  0.00744685 -0.00300983  0.00247988  0.00415459 -0.01181475
 -0.02211117 -0.01925167 -0.00532953  0.02024336  0.04384645  0.01817363
 -0.00483719  0.02470527 -0.0226005   0.02707605  0.01206332  0.02759936
  0.0143059   0.03604806 -0.00112118  0.0113243   0.03470392  0.02487548
 -0.0274792   0.00742949 -0.04327896 -0.03007733 -0.01846339 -0.04979547
  0.01704133 -0.02880722 -0.03145469 -0.0132168   0.0350367   0.01520988]
R2 equal to: -0.0718
MDA equal to: 0.5143
```

Input: py ONESTEP_weight_BRT.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")
predictors = data.shift(1)
X = predictors.iloc[1:,1:]


X_train = X.iloc[:204]
X_test = X.iloc[204:]



#EXC
exc = data['exc']
y1 = exc.iloc[1:]
y_train1 = y1.iloc[:204]
y_test1 = y1.iloc[204:]


#VOL
vol = data['vol']
y2 = vol.iloc[1:]
y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

#WEIGHT
w_real_train = np.divide(y_train1, 4*y_train2)
w_real_test = np.divide(y_test1, 4*y_test2)


y_train = w_real_train
y_test = w_real_test

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

prediction = reg.predict(X_test)
print(prediction)

#Out-of-sample Forecasting Accurancy


print('R2 equal to: %.4f' % r2_score(y_test, prediction))

def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.mean((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int)

print('MDA equal to: %.4f' % mda(np.array(y_test), prediction))
```

Output:

```
C:\Users\Luca-\Desktop\Algorithm Data>py ONESTEP_weight_BRT.py
[ 0.1206457   0.05479233 -0.39479817  0.34665722 -0.12894714 -0.02984201
 -0.20804468  0.02016893 -0.20514239  0.00613254  0.49712317 -0.26422695
  0.4361013   0.41574808  0.04342673  0.80682747 -0.07236697  0.40932635
 -0.073692    0.31592157  0.07162726  0.4668207   0.02605962  0.26460306
 -0.04635001  0.24444181  0.4716295   0.36775684  0.64732261  0.41584241
  0.42658681  0.1955776   0.4285386   0.05863657  0.05896566 -0.15324646]
R2 equal to: 0.0714
MDA equal to: 0.6571
```

Input: py weight_CumSum.py

```
#pt 1 LinearGARCH

#LINEAR

import numpy as np
import pandas as pd
from sklearn import datasets, ensemble, linear_model
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# ------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# ------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

print (X_test)

# Fit regression model (reg)
# ------------------------------------
# Now we will initiate the multivariate linear regression model.


reg = linear_model.LinearRegression()

# Train the model using the training sets

reg.fit(X_train, y_train)

print (y_test)

# Make predictions using the testing set

prediction = reg.predict(X_test)
print(prediction)


#GARCH

#importing Required Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.stattools import adfuller, kpss
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from arch import arch_model
import warnings

sns.set()
warnings.filterwarnings("ignore")

# Load the data

data_backup = pd.read_csv("predictors.csv")

data2 = data_backup[["vol", "exc"]]

#Train and Test Split

train_df = data2.iloc[:205]
test_df = data2.iloc[205:]



# Building Dataframe for fitting GARCH model
garch_df = pd.DataFrame(data2["exc"].shift(1).loc[data2.index])
garch_df.at[train_df.index, "exc"] = 10*train_df["exc"]

# Instantating the model with the full dataset, parameters and specifying the model to be a GARCH model
model = arch_model(garch_df["exc"][1:], p = 1, q = 1, vol = "GARCH")

# Fitting the model on all the data just before the date specified in "last_obs" argument
model_results = model.fit(last_obs = test_df.index[0], update_freq = 5)

# Printing the Summary table of the fitted model
model_results.summary()

# FORECASTING
# Building Predictions Data
predictions_df = test_df.copy()
# Predictions
predictions_df["Predictions"] = model_results.forecast().residual_variance.loc[test_df.index]
```

```
#Tests
y_test2=predictions_df["vol"].iloc[:36]
vol_prediction= predictions_df["Predictions"]
prediction2 = np.array(vol_prediction.iloc[:36])

print(y_test2)
print(prediction2)

#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)



#Pt. 2

#exc_BRT
# %%
# Load the data
# ---------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data3 = pd.read_csv("predictors.csv")
exc3 = data3['exc']
predictors3 = data3.shift(1)

print(data3)

# Data preprocessing
# ---------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X3,y3 = predictors3.iloc[1:,1:], exc3.iloc[1:]

print (X3)
print (y3)


X_train3 = X3.iloc[:204]
X_test3 = X3.iloc[204:]

y_train3 = y3.iloc[:204]
y_test3 = y3.iloc[204:]
print (X_test3)

# Fit regression model
# ---------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train3, y_train3)

print (y_test3)

prediction3 = reg.predict(X_test3)
print(prediction3)



#volBRT


data4 = pd.read_csv("predictors.csv")
vol4 = data4['vol']
predictors4 = data4.shift(1)

print(data4)

# Data preprocessing
# ---------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X4,y4 = predictors4.iloc[1:,1:], vol4.iloc[1:]

print (X4)
print (y4)


X_train4 = X4.iloc[:204]
X_test4 = X4.iloc[204:]

y_train4 = y4.iloc[:204]
y_test4 = y4.iloc[204:]

print (X_test4)
```

```
# Fit regression model
# ------------------------------------
#
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg4 = ensemble.GradientBoostingRegressor(**params)
reg4.fit(X_train4, y_train4)

print (y_test4)

prediction4 = reg4.predict(X_test4)


#weight BRT

w_real2 = np.divide(y_test3, 4*y_test4)
print(w_real2)

w_predict2 = np.divide(prediction3, 4*prediction4)
print(w_predict2)


#CUMSUM
CumSum_series =((w_predict-w_real)**2) - ((w_predict2-w_real2)**2)
print(CumSum_series)
series = pd.Series(CumSum_series)
cumsum =  series.cumsum()
print(cumsum)
print (np.array(cumsum))

#PLOT
dates = []
ticks = []
count = 0
to_plot = data['date'].iloc[205:]
for i in to_plot:
  i = str(i)
  c = i[4:6] + '/' + i[2:4]
  i = c
  dates.append(i)
  if count%2 == 0:
    ticks.append(i)
  count += 1

Y_plot = []

for i in cumsum:
  Y_plot.append(i)

plt.style.use('default')
plt.plot(dates, Y_plot)
plt.ylabel('Cumulative Sum of Squared Error difference')
plt.xlabel('Date (mm/yy format)')
plt.xticks(ticks)
plt.axhline(y=0.0, color = 'r', linestyle = '--')
plt.grid()
plt.show()
```

Output:

```
[-0.41486284  0.19286504  1.43721369  1.07921419  0.88204963  0.91742752
  1.43591791  1.37125439  0.65275493  0.60265339  4.99907225  5.61324358
  7.69222405  7.38050887  7.33693051 10.95192272 10.90417736 12.90227248
 12.93336403 11.56622208 11.40602102 11.31096909 11.18773219 11.13676246
 10.74125874 10.68666049 11.26988979 10.50377446  7.26099622  7.98923524
  8.36132015  7.9028747   9.55037848  9.96543372 10.05215777  9.8431141 ]
```

Input: py weight_CumSumMDA.py

```
#pt 1 LinearGARCH

#LINEAR

import numpy as np
import pandas as pd
from sklearn import datasets, ensemble, linear_model
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

print (X_test)

# Fit regression model (reg)
# -------------------------------------
# Now we will initiate the multivariate linear regression model.


reg = linear_model.LinearRegression()

# Train the model using the training sets

reg.fit(X_train, y_train)

print (y_test)

# Make predictions using the testing set

prediction = reg.predict(X_test)
print(prediction)


#GARCH

#importing Required Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.stattools import adfuller, kpss
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from arch import arch_model
import warnings

sns.set()
warnings.filterwarnings("ignore")

# Load the data

data_backup = pd.read_csv("predictors.csv")

data2 = data_backup[["vol", "exc"]]

#Train and Test Split

train_df = data2.iloc[:205]
test_df = data2.iloc[205:]



# Building Dataframe for fitting GARCH model
garch_df = pd.DataFrame(data2["exc"].shift(1).loc[data2.index])
garch_df.at[train_df.index, "exc"] = 10*train_df["exc"]

# Instantating the model with the full dataset, parameters and specifying the model to be a GARCH model
model = arch_model(garch_df["exc"][1:], p = 1, q = 1, vol = "GARCH")

# Fitting the model on all the data just before the date specified in "last_obs" argument
model_results = model.fit(last_obs = test_df.index[0], update_freq = 5)

# Printing the Summary table of the fitted model
model_results.summary()

# FORECASTING
# Building Predictions Data
predictions_df = test_df.copy()
# Predictions
predictions_df["Predictions"] = model_results.forecast().residual_variance.loc[test_df.index]
```

```
#Tests
y_test2=predictions_df["vol"].iloc[:36]
vol_prediction= predictions_df["Predictions"]
prediction2 = np.array(vol_prediction.iloc[:36])

print(y_test2)
print(prediction2)

#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)



#Pt. 2

#exc_BRT
# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data3 = pd.read_csv("predictors.csv")
exc3 = data3['exc']
predictors3 = data3.shift(1)

print(data3)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X3,y3 = predictors3.iloc[1:,1:], exc3.iloc[1:]

print (X3)
print (y3)


X_train3 = X3.iloc[:204]
X_test3 = X3.iloc[204:]

y_train3 = y3.iloc[:204]
y_test3 = y3.iloc[204:]
print (X_test3)

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train3, y_train3)

print (y_test3)

prediction3 = reg.predict(X_test3)
print(prediction3)



#volBRT


data4 = pd.read_csv("predictors.csv")
vol4 = data4['vol']
predictors4 = data4.shift(1)

print(data4)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X4,y4 = predictors4.iloc[1:,1:], vol4.iloc[1:]

print (X4)
print (y4)


X_train4 = X4.iloc[:204]
X_test4 = X4.iloc[204:]

y_train4 = y4.iloc[:204]
y_test4 = y4.iloc[204:]

print (X_test4)
```

```
# Fit regression model
# ------------------------------------
#
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg4 = ensemble.GradientBoostingRegressor(**params)
reg4.fit(X_train4, y_train4)

print (y_test4)

prediction4 = reg4.predict(X_test4)


#weight BRT

w_real2 = np.divide(y_test3, 4*y_test4)
print(w_real2)

w_predict2 = np.divide(prediction3, 4*prediction4)
print(w_predict2)


#CumSum
def mda(actual: np.ndarray, predicted: np.ndarray):
    """ Mean Directional Accuracy """
    return np.cumsum((np.sign(actual[1:] - actual[:-1]) == np.sign(predicted[1:] - predicted[:-1])).astype(int))

cumsum = mda(np.array(y_test), prediction) - mda(np.array(y_test), prediction2)
print(cumsum)


#PLOT
dates = []
ticks = []
count = 0
to_plot = data['date'].iloc[206:]
for i in to_plot:
  i = str(i)
  c = i[4:6] + '/' + i[2:4]
  i = c
  dates.append(i)
  if count%2 == 0:
    ticks.append(i)
  count += 1

Y_plot = []

for i in cumsum:
  Y_plot.append(i)


plt.style.use('default')
plt.plot(dates, Y_plot)
plt.ylabel('Cumulative Sum of MDA difference')
plt.xlabel('Date (mm/yy format)')
plt.xticks(ticks)
plt.axhline(y=0.0, color = 'r', linestyle = '--')
plt.grid()
plt.show()
```

Output:

```
[0 0 1 0 0 0 1 1 1 2 3 3 3 3 4 5 6 7 6 6 5 5 6 5 5 5 5 4 4 4 4 4 4 4 4]
```

F. Portfolio Performance

Input: py ONESTEP_BRT_UnconstrainedWeights.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")
predictors = data.shift(1)
X = predictors.iloc[1:,1:]


X_train = X.iloc[:204]
X_test = X.iloc[204:]



#EXC
exc = data['exc']
y1 = exc.iloc[1:]
y_train1 = y1.iloc[:204]
y_test1 = y1.iloc[204:]


#VOL
vol = data['vol']
y2 = vol.iloc[1:]
y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

#WEIGHT
w_real_train = np.divide(y_train1, 4*y_train2)
w_real_test = np.divide(y_test1, 4*y_test2)


y_train = w_real_train
y_test = w_real_test
# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

prediction = reg.predict(X_test)
print(prediction)

#Portfolio Performance

exp_Stock_weight = prediction
Rfree_weight = 1 - prediction

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)

#Portfolio metrics

#Mean
mean = np.mean(P_Return)
print('Mean is equal to: %.4f' % mean)

#Standard Deviation
std = np.std(P_Return)
print('Standard Deviation is equal to: %.4f' % std)
```

```
#SharpeRatio
sharpe_ratio = (np.mean(P_Return) - np.mean(Rfree_return)) / np.std(P_Return)
print('Sharpe Ratio is equal to: %.4f' % sharpe_ratio)

#Maximum Drawdown
#https://www.kaggle.com/prakharrathi25/maximum-drawdown-strategy
        #1)Compute the Wealth Index: Value of a portfolio when it compounds over time over returns.
            # Compute the wealth index by starting with 1000 dollars
            # The starting value won't matter with drawdowns
wealth_index = 1000*(1+P_Return).cumprod()

        #2)Compute the previous peak
previous_peaks = wealth_index.cummax()


        #3)Compute drawdown
drawdown = (wealth_index - previous_peaks)/previous_peaks

        #4)Get maximum drawdown
maximum_drawdown = drawdown.min()
print('Maximum Drawdown is equal to: %.4f' % maximum_drawdown)

#Value at Risk
from scipy.stats import norm
    #If I wanto to get the graph
        #import seaborn
        #import matplotlib.mlab as mlab
        #P_Return.hist(bins=40, normed=True, histtype='stepfilled', alpha=0.5)
        #x = np.linspace(mean - 3*std, mean + 3*std, 100)
        #plt.plot(x,norm.pdf(x, mean, std),"r")
        #plt.show()

VaR_95 = norm.ppf(1-0.95, mean, std)
print('Value at Risk with a 95 percent confidence level is equal to: %.4f' % VaR_95)
```

Output:

```
Mean is equal to: -0.0010
Standard Deviation is equal to: 0.0239
Return is equal to: -0.0370
Sharpe Ratio is equal to: 0.1282
Maximum Drawdown is equal to: -0.1824
Value at Risk with a 95 percent confidence level is equal to: -0.0403
```

Input: py ONESTEP_BRT_ShortsellingAndBorrowingConstraints.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")
predictors = data.shift(1)
X = predictors.iloc[1:,1:]


X_train = X.iloc[:204]
X_test = X.iloc[204:]



#EXC
exc = data['exc']
y1 = exc.iloc[1:]
y_train1 = y1.iloc[:204]
y_test1 = y1.iloc[204:]


#VOL
vol = data['vol']
y2 = vol.iloc[1:]
y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

#WEIGHT
w_real_train = np.divide(y_train1, 4*y_train2)
w_real_test = np.divide(y_test1, 4*y_test2)


y_train = w_real_train
y_test = w_real_test

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}
reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

prediction = reg.predict(X_test)
print(prediction)

#Portfolio Performance

exp_Stock_weight = []
for i in prediction:
    if i < 0:
        exp_Stock_weight.append(0)
    elif i > 1:
        exp_Stock_weight.append(1)
    else:
        exp_Stock_weight.append(i)

E_stock_weight=np.array(exp_Stock_weight)
print (E_stock_weight)

Rfree_weight = 1 - E_stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)
P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)

#Portfolio metrics

#Mean
mean = np.mean(P_Return)
print('Mean is equal to: %.4f' % mean)

#Standard Deviation
std = np.std(P_Return)
print('Standard Deviation is equal to: %.4f' % std)
```

```
#SharpeRatio
sharpe_ratio = (np.mean(P_Return) - np.mean(Rfree_return)) / np.std(P_Return)
print('Sharpe Ratio is equal to: %.4f' % sharpe_ratio)

#Maximum Drawdown
#https://www.kaggle.com/prakharrathi25/maximum-drawdown-strategy
        #1)Compute the Wealth Index: Value of a portfolio when it compounds over time over returns.
            # Compute the wealth index by starting with 1000 dollars
            # The starting value won't matter with drawdowns
wealth_index = 1000*(1+P_Return).cumprod()

        #2)Compute the previous peak
previous_peaks = wealth_index.cummax()


        #3)Compute drawdown
drawdown = (wealth_index - previous_peaks)/previous_peaks

        #4)Get maximum drawdown
maximum_drawdown = drawdown.min()
print('Maximum Drawdown is equal to: %.4f' % maximum_drawdown)

#Value at Risk
from scipy.stats import norm
    #If I wanto to get the graph
        #import seaborn
        #import matplotlib.mlab as mlab
        #P_Return.hist(bins=40, normed=True, histtype='stepfilled', alpha=0.5)
        #x = np.linspace(mean - 3*std, mean + 3*std, 100)
        #plt.plot(x,norm.pdf(x, mean, std),"r")
        #plt.show()

VaR_95 = norm.ppf(1-0.95, mean, std)
print('Value at Risk with a 95 percent confidence level is equal to: %.4f' % VaR_95)
```

Output:

```
Mean is equal to: -0.0016
Standard Deviation is equal to: 0.0211
Return is equal to: -0.0591
Sharpe Ratio is equal to: 0.1159
Maximum Drawdown is equal to: -0.1620
Value at Risk with a 95 percent confidence level is equal to: -0.0364
```

Input: py TWOSTEP_BRT_UnconstrainedWeights.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]
print (X_test)

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)
prediction = reg.predict(X_test)
print(prediction)



#data2


data2 = pd.read_csv("predictors.csv")
vol = data['vol']
predictors2 = data.shift(1)

print(data2)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X2,y2 = predictors2.iloc[1:,1:], vol.iloc[1:]

print (X2)
print (y2)


X_train2 = X2.iloc[:204]
X_test2 = X2.iloc[204:]

y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

print (X_test2)

# Fit regression model
# -------------------------------------
#
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg2 = ensemble.GradientBoostingRegressor(**params)
reg2.fit(X_train2, y_train2)

print (y_test2)
```

```
prediction2 = reg2.predict(X_test2)


#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)

#Portfolio Performance

exp_Stock_weight = w_predict
Rfree_weight = 1 - w_predict

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)

#Portfolio metrics

#Mean
mean = np.mean(P_Return)
print('Mean is equal to: %.4f' % mean)

#Standard Deviation
std = np.std(P_Return)
print('Standard Deviation is equal to: %.4f' % std)

#Return
print('Return is equal to: %.4f' % np.sum(P_Return))

#SharpeRatio
sharpe_ratio = (np.mean(P_Return) - np.mean(Rfree_return)) / np.std(P_Return)
print('Sharpe Ratio is equal to: %.4f' % sharpe_ratio)



#Maximum Drawdown
#https://www.kaggle.com/prakharrathi25/maximum-drawdown-strategy
        #1)Compute the Wealth Index: Value of a portfolio when it compounds over time over returns.
            # Compute the wealth index by starting with 1000 dollars
            # The starting value won't matter with drawdowns
wealth_index = 1000*(1+P_Return).cumprod()

        #2)Compute the previous peak
previous_peaks = wealth_index.cummax()


        #3)Compute drawdown
drawdown = (wealth_index - previous_peaks)/previous_peaks

        #4)Get maximum drawdown
maximum_drawdown = drawdown.min()
print('Maximum Drawdown is equal to: %.4f' % maximum_drawdown)

#Value at Risk
from scipy.stats import norm
    #If I wanto to get the graph
        #import seaborn
        #import matplotlib.mlab as mlab
        #P_Return.hist(bins=40, normed=True, histtype='stepfilled', alpha=0.5)
        #x = np.linspace(mean - 3*std, mean + 3*std, 100)
        #plt.plot(x,norm.pdf(x, mean, std),"r")
        #plt.show()

VaR_95 = norm.ppf(1-0.95, mean, std)
print('Value at Risk with a 95 percent confidence level is equal to: %.4f' % VaR_95)
```

Output:

```
Mean is equal to: -0.0006
Standard Deviation is equal to: 0.0183
Return is equal to: -0.0200
Sharpe Ratio is equal to: 0.1930
Maximum Drawdown is equal to: -0.1350
Value at Risk with a 95 percent confidence level is equal to: -0.0307
```

Input: py TWOSTEP_BRT_ShortsellingAndBorrowingConstraints.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]
print (X_test)

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)
prediction = reg.predict(X_test)
print(prediction)



#data2


data2 = pd.read_csv("predictors.csv")
vol = data['vol']
predictors2 = data.shift(1)

print(data2)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X2,y2 = predictors2.iloc[1:,1:], vol.iloc[1:]

print (X2)
print (y2)


X_train2 = X2.iloc[:204]
X_test2 = X2.iloc[204:]

y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

print (X_test2)

# Fit regression model
# -------------------------------------
#
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg2 = ensemble.GradientBoostingRegressor(**params)
reg2.fit(X_train2, y_train2)

print (y_test2)
```

```
prediction2 = reg2.predict(X_test2)


#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)

#Portfolio Performance

exp_Stock_weight = []
for i in w_predict:
    if i < 0:
        exp_Stock_weight.append(0)
    elif i > 1:
        exp_Stock_weight.append(1)
    else:
        exp_Stock_weight.append(i)

E_stock_weight=np.array(exp_Stock_weight)
print (E_stock_weight)

Rfree_weight = 1 - E_stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)

#Portfolio metrics

#Mean
mean = np.mean(P_Return)
print('Mean is equal to: %.4f' % mean)

#Standard Deviation
std = np.std(P_Return)
print('Standard Deviation is equal to: %.4f' % std)


#SharpeRatio
sharpe_ratio = (np.mean(P_Return) - np.mean(Rfree_return)) / np.std(P_Return)
print('Sharpe Ratio is equal to: %.4f' % sharpe_ratio)

#Maximum Drawdown
#https://www.kaggle.com/prakharrathi25/maximum-drawdown-strategy
        #1)Compute the Wealth Index: Value of a portfolio when it compounds over time over returns.
            # Compute the wealth index by starting with 1000 dollars
            # The starting value won't matter with drawdowns
wealth_index = 1000*(1+P_Return).cumprod()

        #2)Compute the previous peak
previous_peaks = wealth_index.cummax()


        #3)Compute drawdown
drawdown = (wealth_index - previous_peaks)/previous_peaks

        #4)Get maximum drawdown
maximum_drawdown = drawdown.min()
print('Maximum Drawdown is equal to: %.4f' % maximum_drawdown)

#Value at Risk
from scipy.stats import norm
    #If I wanto to get the graph
        #import seaborn
        #import matplotlib.mlab as mlab
        #P_Return.hist(bins=40, normed=True, histtype='stepfilled', alpha=0.5)
        #x = np.linspace(mean - 3*std, mean + 3*std, 100)
        #plt.plot(x,norm.pdf(x, mean, std),"r")
        #plt.show()

VaR_95 = norm.ppf(1-0.95, mean, std)
print('Value at Risk with a 95 percent confidence level is equal to: %.4f' % VaR_95)
```

Output:

```
Mean is equal to: -0.0002
Standard Deviation is equal to: 0.0170
Return is equal to: -0.0056
Sharpe Ratio is equal to: 0.2315
Maximum Drawdown is equal to: -0.1103
Value at Risk with a 95 percent confidence level is equal to: -0.0281
```

Input: py LinearGARCH_UnconstrainedWeights.py

```
#LINEAR

import numpy as np
import pandas as pd
from sklearn import datasets, ensemble, linear_model
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

print (X_test)

# Fit regression model (reg)
# -------------------------------------
# Now we will initiate the multivariate linear regression model.


reg = linear_model.LinearRegression()

# Train the model using the training sets

reg.fit(X_train, y_train)

print (y_test)
# Make predictions using the testing set

prediction = reg.predict(X_test)
print(prediction)


#GARCH

#importing Required Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.stattools import adfuller, kpss
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from arch import arch_model
import warnings

sns.set()
warnings.filterwarnings("ignore")

# Load the data

data_backup = pd.read_csv("predictors.csv")

data2 = data_backup[["vol", "exc"]]

#Train and Test Split

train_df = data2.iloc[:205]
test_df = data2.iloc[205:]



# Building Dataframe for fitting GARCH model
garch_df = pd.DataFrame(data2["exc"].shift(1).loc[data2.index])
garch_df.at[train_df.index, "exc"] = 10*train_df["exc"]

# Instantating the model with the full dataset, parameters and specifying the model to be a GARCH model
model = arch_model(garch_df["exc"][1:], p = 1, q = 1, vol = "GARCH")

# Fitting the model on all the data just before the date specified in "last_obs" argument
model_results = model.fit(last_obs = test_df.index[0], update_freq = 5)

# Printing the Summary table of the fitted model
model_results.summary()
```

```
# FORECASTING
# Building Predictions Data
predictions_df = test_df.copy()
# Predictions
predictions_df["Predictions"] = model_results.forecast().residual_variance.loc[test_df.index]


#Tests
y_test2=predictions_df["vol"].iloc[:36]
vol_prediction= predictions_df["Predictions"]
prediction2 = np.array(vol_prediction.iloc[:36])

print(y_test2)
print(prediction2)

#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)


#Portfolio Performance

exp_Stock_weight = w_predict
Rfree_weight = 1 - w_predict

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)
#Portfolio metrics

#Mean
mean = np.mean(P_Return)
print('Mean is equal to: %.4f' % mean)

#Standard Deviation
std = np.std(P_Return)
print('Standard Deviation is equal to: %.4f' % std)

#Return
print('Return is equal to: %.4f' % np.sum(P_Return))

#SharpeRatio
sharpe_ratio = (np.mean(P_Return) - np.mean(Rfree_return)) / np.std(P_Return)
print('Sharpe Ratio is equal to: %.4f' % sharpe_ratio)

#Maximum Drawdown
#https://www.kaggle.com/prakharrathi25/maximum-drawdown-strategy
        #1)Compute the Wealth Index: Value of a portfolio when it compounds over time over returns.
            # Compute the wealth index by starting with 1000 dollars
            # The starting value won't matter with drawdowns
wealth_index = 1000*(1+P_Return).cumprod()

        #2)Compute the previous peak
previous_peaks = wealth_index.cummax()


        #3)Compute drawdown
drawdown = (wealth_index - previous_peaks)/previous_peaks

        #4)Get maximum drawdown
maximum_drawdown = drawdown.min()
print('Maximum Drawdown is equal to: %.4f' % maximum_drawdown)

#Value at Risk
from scipy.stats import norm
    #If I wanto to get the graph
        #import seaborn
        #import matplotlib.mlab as mlab
        #P_Return.hist(bins=40, normed=True, histtype='stepfilled', alpha=0.5)
        #x = np.linspace(mean - 3*std, mean + 3*std, 100)
        #plt.plot(x,norm.pdf(x, mean, std),"r")
        #plt.show()

VaR_95 = norm.ppf(1-0.95, mean, std)
print('Value at Risk with a 95 percent confidence level is equal to: %.4f' % VaR_95)
```

Output:

```
Mean is equal to: -0.0043
Standard Deviation is equal to: 0.0022
Return is equal to: -0.1539
Sharpe Ratio is equal to: -0.0864
Maximum Drawdown is equal to: -0.1402
Value at Risk with a 95 percent confidence level is equal to: -0.0078
```

Input: py LinearGARCH_ShortsellingAndBorrowingConstraints.py

```
#LINEAR

import numpy as np
import pandas as pd
from sklearn import datasets, ensemble, linear_model
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split


# %%
# Load the data
# ------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# ------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]

print (X_test)

# Fit regression model (reg)
# ------------------------------------
# Now we will initiate the multivariate linear regression model.


reg = linear_model.LinearRegression()

# Train the model using the training sets

reg.fit(X_train, y_train)

print (y_test)

# Make predictions using the testing set

prediction = reg.predict(X_test)
print(prediction)


#GARCH

#importing Required Packages
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import statsmodels.graphics.tsaplots as sgt
from statsmodels.tsa.stattools import adfuller, kpss
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from arch import arch_model
import warnings

sns.set()
warnings.filterwarnings("ignore")

# Load the data

data_backup = pd.read_csv("predictors.csv")

data2 = data_backup[["vol", "exc"]]

#Train and Test Split

train_df = data2.iloc[:205]
test_df = data2.iloc[205:]



# Building Dataframe for fitting GARCH model
garch_df = pd.DataFrame(data2["exc"].shift(1).loc[data2.index])
garch_df.at[train_df.index, "exc"] = 10*train_df["exc"]

# Instantating the model with the full dataset, parameters and specifying the model to be a GARCH model
model = arch_model(garch_df["exc"][1:], p = 1, q = 1, vol = "GARCH")

# Fitting the model on all the data just before the date specified in "last_obs" argument
model_results = model.fit(last_obs = test_df.index[0], update_freq = 5)

# Printing the Summary table of the fitted model
model_results.summary()
```

```
# FORECASTING
# Building Predictions Data
predictions_df = test_df.copy()
# Predictions
predictions_df["Predictions"] = model_results.forecast().residual_variance.loc[test_df.index]


#Tests
y_test2=predictions_df["vol"].iloc[:36]
vol_prediction= predictions_df["Predictions"]
prediction2 = np.array(vol_prediction.iloc[:36])

print(y_test2)
print(prediction2)

#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)


#Portfolio Performance

exp_Stock_weight = []
for i in w_predict:
    if i < 0:
        exp_Stock_weight.append(0)
    elif i > 1:
        exp_Stock_weight.append(1)
    else:
        exp_Stock_weight.append(i)

E_stock_weight=np.array(exp_Stock_weight)
print (E_stock_weight)

Rfree_weight = 1 - E_stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)
print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)

#Portfolio metrics

#Mean
mean = np.mean(P_Return)
print('Mean is equal to: %.4f' % mean)

#Standard Deviation
std = np.std(P_Return)
print('Standard Deviation is equal to: %.4f' % std)

#Return
print('Return is equal to: %.4f' % np.sum(P_Return))

#SharpeRatio
sharpe_ratio = (np.mean(P_Return) - np.mean(Rfree_return)) / np.std(P_Return)
print('Sharpe Ratio is equal to: %.4f' % sharpe_ratio)

#Maximum Drawdown
#https://www.kaggle.com/prakharrathi25/maximum-drawdown-strategy
        #1)Compute the Wealth Index: Value of a portfolio when it compounds over time over returns.
            # Compute the wealth index by starting with 1000 dollars
            # The starting value won't matter with drawdowns
wealth_index = 1000*(1+P_Return).cumprod()

        #2)Compute the previous peak
previous_peaks = wealth_index.cummax()


        #3)Compute drawdown
drawdown = (wealth_index - previous_peaks)/previous_peaks

        #4)Get maximum drawdown
maximum_drawdown = drawdown.min()
print('Maximum Drawdown is equal to: %.4f' % maximum_drawdown)

#Value at Risk
from scipy.stats import norm
    #If I wanto to get the graph
        #import seaborn
        #import matplotlib.mlab as mlab
        #P_Return.hist(bins=40, normed=True, histtype='stepfilled', alpha=0.5)
        #x = np.linspace(mean - 3*std, mean + 3*std, 100)
        #plt.plot(x,norm.pdf(x, mean, std),"r")
        #plt.show()

VaR_95 = norm.ppf(1-0.95, mean, std)
print('Value at Risk with a 95 percent confidence level is equal to: %.4f' % VaR_95)
```

Output:

```
Mean is equal to: -0.0039
Standard Deviation is equal to: 0.0012
Return is equal to: -0.1418
Sharpe Ratio is equal to: 0.1236
Maximum Drawdown is equal to: -0.1297
Value at Risk with a 95 percent confidence level is equal to: -0.0059
```

Input: py Portfolio6040.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")


#Portfolio Performance

exp_Stock_weight = 0.6
Rfree_weight = 1 - exp_Stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)

#Portfolio metrics

#Mean
mean = np.mean(P_Return)
print('Mean is equal to: %.4f' % mean)

#Standard Deviation
std = np.std(P_Return)
print('Standard Deviation is equal to: %.4f' % std)


#SharpeRatio
sharpe_ratio = (np.mean(P_Return) - np.mean(Rfree_return)) / np.std(P_Return)
print('Sharpe Ratio is equal to: %.4f' % sharpe_ratio)

#Maximum Drawdown
#https://www.kaggle.com/prakharrathi25/maximum-drawdown-strategy
        #1)Compute the Wealth Index: Value of a portfolio when it compounds over time over returns.
            # Compute the wealth index by starting with 1000 dollars
            # The starting value won't matter with drawdowns
wealth_index = 1000*(1+P_Return).cumprod()

        #2)Compute the previous peak
previous_peaks = wealth_index.cummax()


        #3)Compute drawdown
drawdown = (wealth_index - previous_peaks)/previous_peaks

        #4)Get maximum drawdown
maximum_drawdown = drawdown.min()
print('Maximum Drawdown is equal to: %.4f' % maximum_drawdown)

#Value at Risk
from scipy.stats import norm
    #If I wanto to get the graph
        #import seaborn
        #import matplotlib.mlab as mlab
        #P_Return.hist(bins=40, normed=True, histtype='stepfilled', alpha=0.5)
        #x = np.linspace(mean - 3*std, mean + 3*std, 100)
        #plt.plot(x,norm.pdf(x, mean, std),"r")
        #plt.show()

VaR_95 = norm.ppf(1-0.95, mean, std)
print('Value at Risk with a 95 percent confidence level is equal to: %.4f' % VaR_95)
```

Output:

```
Mean is equal to: 0.0012
Standard Deviation is equal to: 0.0344
Return is equal to: 0.0433
Sharpe Ratio is equal to: 0.1541
Maximum Drawdown is equal to: -0.1728
Value at Risk with a 95 percent confidence level is equal to: -0.0553
```

Input: py 100MarketPortfolio.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")


#Portfolio Performance

exp_Stock_weight = 1
Rfree_weight = 1 - exp_Stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)

#Portfolio metrics

#Mean
mean = np.mean(P_Return)
print('Mean is equal to: %.4f' % mean)

#Standard Deviation
std = np.std(P_Return)
print('Standard Deviation is equal to: %.4f' % std)

#SharpeRatio
sharpe_ratio = (np.mean(P_Return) - np.mean(Rfree_return)) / np.std(P_Return)
print('Sharpe Ratio is equal to: %.4f' % sharpe_ratio)

#Maximum Drawdown
#https://www.kaggle.com/prakharrathi25/maximum-drawdown-strategy
        #1)Compute the Wealth Index: Value of a portfolio when it compounds over time over returns.
            # Compute the wealth index by starting with 1000 dollars
            # The starting value won't matter with drawdowns
wealth_index = 1000*(1+P_Return).cumprod()

        #2)Compute the previous peak
previous_peaks = wealth_index.cummax()


        #3)Compute drawdown
drawdown = (wealth_index - previous_peaks)/previous_peaks

        #4)Get maximum drawdown
maximum_drawdown = drawdown.min()
print('Maximum Drawdown is equal to: %.4f' % maximum_drawdown)

#Value at Risk
from scipy.stats import norm
    #If I wanto to get the graph
        #import seaborn
        #import matplotlib.mlab as mlab
        #P_Return.hist(bins=40, normed=True, histtype='stepfilled', alpha=0.5)
        #x = np.linspace(mean - 3*std, mean + 3*std, 100)
        #plt.plot(x,norm.pdf(x, mean, std),"r")
        #plt.show()

VaR_95 = norm.ppf(1-0.95, mean, std)
print('Value at Risk with a 95 percent confidence level is equal to: %.4f' % VaR_95)
```

Output:

```
Mean is equal to: 0.0047
Standard Deviation is equal to: 0.0574
Return is equal to: 0.1704
Sharpe Ratio is equal to: 0.1536
Maximum Drawdown is equal to: -0.2725
Value at Risk with a 95 percent confidence level is equal to: -0.0897
```

Input: py TWOSTEP_BRT_UnconstrainedWeights_UTILITY.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]
print (X_test)

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)
print(prediction)
#data2


data2 = pd.read_csv("predictors.csv")
vol = data['vol']
predictors2 = data.shift(1)

print(data2)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X2,y2 = predictors2.iloc[1:,1:], vol.iloc[1:]

print (X2)
print (y2)


X_train2 = X2.iloc[:204]
X_test2 = X2.iloc[204:]

y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

print (X_test2)

# Fit regression model
# -------------------------------------
#
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg2 = ensemble.GradientBoostingRegressor(**params)
reg2.fit(X_train2, y_train2)

print (y_test2)

prediction2 = reg2.predict(X_test2)
```

```
#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)

#Portfolio Performance

exp_Stock_weight = w_predict
Rfree_weight = 1 - w_predict

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)


#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A1 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A2 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A3 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A4 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A5 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0024
Utility value with a risk-aversion of 4 is equal to: -0.0036
Utility value with a risk-aversion of 6 is equal to: -0.0048
Utility value with a risk-aversion of 8 is equal to: -0.0061
Utility value with a risk-aversion of 10 is equal to: -0.0073
```

Input: py TWOSTEP_BRT_ShortsellingAndBorrowingConstraints_UTILITY.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]
print (X_test)

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)
prediction = reg.predict(X_test)
print(prediction)



#data2


data2 = pd.read_csv("predictors.csv")
vol = data['vol']
predictors2 = data.shift(1)

print(data2)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X2,y2 = predictors2.iloc[1:,1:], vol.iloc[1:]

print (X2)
print (y2)


X_train2 = X2.iloc[:204]
X_test2 = X2.iloc[204:]

y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

print (X_test2)

# Fit regression model
# -------------------------------------
#
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg2 = ensemble.GradientBoostingRegressor(**params)
reg2.fit(X_train2, y_train2)

print (y_test2)
```

```
prediction2 = reg2.predict(X_test2)


#weight

w_real = np.divide(y_test, 4*y_test2)
print(w_real)

w_predict = np.divide(prediction, 4*prediction2)
print(w_predict)

#Portfolio Performance

exp_Stock_weight = []
for i in w_predict:
    if i < 0:
        exp_Stock_weight.append(0)
    elif i > 1:
        exp_Stock_weight.append(1)
    else:
        exp_Stock_weight.append(i)

E_stock_weight=np.array(exp_Stock_weight)
print (E_stock_weight)

Rfree_weight = 1 - E_stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)


#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A1 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A2 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A3 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A4 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A5 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0024
Utility value with a risk-aversion of 4 is equal to: -0.0035
Utility value with a risk-aversion of 6 is equal to: -0.0046
Utility value with a risk-aversion of 8 is equal to: -0.0058
Utility value with a risk-aversion of 10 is equal to: -0.0069
```

Input: py ONESTEP_BRT_UnconstrainedWeights_UTILITY.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")
predictors = data.shift(1)
X = predictors.iloc[1:,1:]


X_train = X.iloc[:204]
X_test = X.iloc[204:]



#EXC
exc = data['exc']
y1 = exc.iloc[1:]
y_train1 = y1.iloc[:204]
y_test1 = y1.iloc[204:]


#VOL
vol = data['vol']
y2 = vol.iloc[1:]
y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

#WEIGHT
w_real_train = np.divide(y_train1, 4*y_train2)
w_real_test = np.divide(y_test1, 4*y_test2)


y_train = w_real_train
y_test = w_real_test

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}
reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

prediction = reg.predict(X_test)
print(prediction)

#Portfolio Performance

exp_Stock_weight = prediction
Rfree_weight = 1 - prediction

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)


#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A1 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A2 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A3 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A4 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A5 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0027
Utility value with a risk-aversion of 4 is equal to: -0.0037
Utility value with a risk-aversion of 6 is equal to: -0.0047
Utility value with a risk-aversion of 8 is equal to: -0.0057
Utility value with a risk-aversion of 10 is equal to: -0.0067
```

Input: py ONESTEP_BRT_ShortsellingAndBorrowingConstraints_UTILITY.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")
predictors = data.shift(1)
X = predictors.iloc[1:,1:]


X_train = X.iloc[:204]
X_test = X.iloc[204:]



#EXC
exc = data['exc']
y1 = exc.iloc[1:]
y_train1 = y1.iloc[:204]
y_test1 = y1.iloc[204:]


#VOL
vol = data['vol']
y2 = vol.iloc[1:]
y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

#WEIGHT
w_real_train = np.divide(y_train1, 4*y_train2)
w_real_test = np.divide(y_test1, 4*y_test2)


y_train = w_real_train
y_test = w_real_test

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}
reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

prediction = reg.predict(X_test)
print(prediction)

#Portfolio Performance

exp_Stock_weight = []
for i in prediction:
    if i < 0:
        exp_Stock_weight.append(0)
    elif i > 1:
        exp_Stock_weight.append(1)
    else:
        exp_Stock_weight.append(i)

E_stock_weight=np.array(exp_Stock_weight)
print (E_stock_weight)

Rfree_weight = 1 - E_stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)
P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)
```

```
#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A1 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A2 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A3 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A4 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A5 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0039
Utility value with a risk-aversion of 4 is equal to: -0.0049
Utility value with a risk-aversion of 6 is equal to: -0.0059
Utility value with a risk-aversion of 8 is equal to: -0.0069
Utility value with a risk-aversion of 10 is equal to: -0.0080
```

Input: py TWOSTEP_BRT_UnconstrainedWeights_UTILITY_Uncertainty.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]
print (X_test)

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)
print(prediction)
#data2


data2 = pd.read_csv("predictors.csv")
vol = data['vol']
predictors2 = data.shift(1)

print(data2)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X2,y2 = predictors2.iloc[1:,1:], vol.iloc[1:]

print (X2)
print (y2)


X_train2 = X2.iloc[:204]
X_test2 = X2.iloc[204:]

y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

print (X_test2)

# Fit regression model
# -------------------------------------
#
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg2 = ensemble.GradientBoostingRegressor(**params)
reg2.fit(X_train2, y_train2)

print (y_test2)

prediction2 = reg2.predict(X_test2)
```

```
#weight

w_real = np.divide(y_test, 8*y_test2)
print(w_real)

w_predict = np.divide(prediction, 8*prediction2)
print(w_predict)

#Portfolio Performance

exp_Stock_weight = w_predict
Rfree_weight = 1 - w_predict

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)


#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A1 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A2 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A3 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A4 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A5 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0028
Utility value with a risk-aversion of 4 is equal to: -0.0032
Utility value with a risk-aversion of 6 is equal to: -0.0035
Utility value with a risk-aversion of 8 is equal to: -0.0038
Utility value with a risk-aversion of 10 is equal to: -0.0041
```

Input: py TWOSTEP_BRT_ShortsellingAndBorrowingConstraints_UTILITY_Uncertainty.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.

data = pd.read_csv("predictors.csv")
exc = data['exc']
predictors = data.shift(1)

print(data)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X,y = predictors.iloc[1:,1:], exc.iloc[1:]

print (X)
print (y)


X_train = X.iloc[:204]
X_test = X.iloc[204:]

y_train = y.iloc[:204]
y_test = y.iloc[204:]
print (X_test)

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

print (y_test)

prediction = reg.predict(X_test)
print(prediction)

#data2


data2 = pd.read_csv("predictors.csv")
vol = data['vol']
predictors2 = data.shift(1)

print(data2)

# Data preprocessing
# -------------------------------------
#
# Next, we will split our dataset to use 85% for training and leave the rest
# for testing.

X2,y2 = predictors2.iloc[1:,1:], vol.iloc[1:]

print (X2)
print (y2)


X_train2 = X2.iloc[:204]
X_test2 = X2.iloc[204:]

y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

print (X_test2)

# Fit regression model
# -------------------------------------
#
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.

params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg2 = ensemble.GradientBoostingRegressor(**params)
reg2.fit(X_train2, y_train2)

print (y_test2)

prediction2 = reg2.predict(X_test2)
```

```
#weight

w_real = np.divide(y_test, 8*y_test2)
print(w_real)

w_predict = np.divide(prediction, 8*prediction2)
print(w_predict)

#Portfolio Performance

exp_Stock_weight = []
for i in w_predict:
    if i < 0:
        exp_Stock_weight.append(0)
    elif i > 1:
        exp_Stock_weight.append(1)
    else:
        exp_Stock_weight.append(i)

E_stock_weight=np.array(exp_Stock_weight)
print (E_stock_weight)

Rfree_weight = 1 - E_stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)


#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A1 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A2 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A3 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A4 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A5 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0029
Utility value with a risk-aversion of 4 is equal to: -0.0032
Utility value with a risk-aversion of 6 is equal to: -0.0035
Utility value with a risk-aversion of 8 is equal to: -0.0038
Utility value with a risk-aversion of 10 is equal to: -0.0040
```

Input: py ONESTEP_BRT_UnconstrainedWeights_UTILITY_Uncertainty.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")
predictors = data.shift(1)
X = predictors.iloc[1:,1:]


X_train = X.iloc[:204]
X_test = X.iloc[204:]



#EXC
exc = data['exc']
y1 = exc.iloc[1:]
y_train1 = y1.iloc[:204]
y_test1 = y1.iloc[204:]


#VOL
vol = data['vol']
y2 = vol.iloc[1:]
y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

#WEIGHT
w_real_train = np.divide(y_train1, 4*y_train2)
w_real_test = np.divide(y_test1, 4*y_test2)


y_train = w_real_train
y_test = w_real_test

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}

reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

prediction = reg.predict(X_test)
print(prediction)

#Portfolio Performance

exp_Stock_weight = prediction
Rfree_weight = 1 - prediction

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)
```

```
#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A1 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A2 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A3 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A4 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A5 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0029
Utility value with a risk-aversion of 4 is equal to: -0.0039
Utility value with a risk-aversion of 6 is equal to: -0.0049
Utility value with a risk-aversion of 8 is equal to: -0.0058
Utility value with a risk-aversion of 10 is equal to: -0.0068
```

Input: py ONESTEP_BRT_ShortsellingAndBorrowingConstraints_UTILITY_Uncertainty.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")
predictors = data.shift(1)
X = predictors.iloc[1:,1:]


X_train = X.iloc[:204]
X_test = X.iloc[204:]



#EXC
exc = data['exc']
y1 = exc.iloc[1:]
y_train1 = y1.iloc[:204]
y_test1 = y1.iloc[204:]


#VOL
vol = data['vol']
y2 = vol.iloc[1:]
y_train2 = y2.iloc[:204]
y_test2 = y2.iloc[204:]

#WEIGHT
w_real_train = np.divide(y_train1, 4*y_train2)
w_real_test = np.divide(y_test1, 4*y_test2)


y_train = w_real_train
y_test = w_real_test

# Fit regression model
# -------------------------------------
# Now we will initiate the gradient boosting regressors and fit it with our
# training data.
params = {'n_estimators': 10000,
          'max_depth': 2,
          'learning_rate': 0.001,
          'loss': 'lad',
          'subsample': 0.5,}
reg = ensemble.GradientBoostingRegressor(**params)
reg.fit(X_train, y_train)

prediction = reg.predict(X_test)
print(prediction)

#Portfolio Performance

exp_Stock_weight = []
for i in prediction:
    if i < 0:
        exp_Stock_weight.append(0)
    elif i > 1:
        exp_Stock_weight.append(1)
    else:
        exp_Stock_weight.append(i)

E_stock_weight=np.array(exp_Stock_weight)
print (E_stock_weight)

Rfree_weight = 1 - E_stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)
P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)
```

```
#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A1 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A2 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A3 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A4 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + E_stock_weight * Stock_return - 1/2 * A5 * E_stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0036
Utility value with a risk-aversion of 4 is equal to: -0.0045
Utility value with a risk-aversion of 6 is equal to: -0.0054
Utility value with a risk-aversion of 8 is equal to: -0.0063
Utility value with a risk-aversion of 10 is equal to: -0.0072
```

Input: py Portfolio6040_UTILITY.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")


#Portfolio Performance

exp_Stock_weight = 0.6
Rfree_weight = 1 - exp_Stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)


#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A1 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A2 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A3 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A4 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A5 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0048
Utility value with a risk-aversion of 4 is equal to: -0.0083
Utility value with a risk-aversion of 6 is equal to: -0.0118
Utility value with a risk-aversion of 8 is equal to: -0.0153
Utility value with a risk-aversion of 10 is equal to: -0.0188
```

Input: py 100MarketPortfolio_UTILITY.py

```
import matplotlib.pyplot as plt
import numpy as np
import pandas as pd
from sklearn import datasets, ensemble
from sklearn.inspection import permutation_importance
from sklearn.metrics import mean_squared_error, r2_score, accuracy_score
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import scale, MinMaxScaler

# %%
# Load the data
# -------------------------------------
# First we need to load the data.
# Prediction is at T+1. Data are modelled to predict exc based on t-1 predictors.


data = pd.read_csv("predictors.csv")


#Portfolio Performance

exp_Stock_weight = 1
Rfree_weight = 1 - exp_Stock_weight

#Negative Risk-free rate is considered 0 when used for excess return
Rfree_return = data['Rfree'].iloc[205:]

r_free_excReturn = []
for i in Rfree_return:
    if i < 0:
        r_free_excReturn.append(0)
    else:
        r_free_excReturn.append(i)

Stock_return = data['exc'].iloc[205:]+ r_free_excReturn
print(Stock_return)

print(Rfree_return)

P_Return = exp_Stock_weight*Stock_return + Rfree_weight*Rfree_return

print(P_Return)


#UTILITY FUNCTION

Stock_vol = data['vol'].iloc[205:]

A1 = 2
U_p1 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A1 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 2 is equal to: %.4f' % U_p1)

A2 = 4
U_p2 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A2 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 4 is equal to: %.4f' % U_p2)

A3 = 6
U_p3 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A3 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 6 is equal to: %.4f' % U_p3)

A4 = 8
U_p4 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A4 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 8 is equal to: %.4f' % U_p4)

A5 = 10
U_p5 = np.mean(Rfree_return + exp_Stock_weight * Stock_return - 1/2 * A5 * exp_Stock_weight**2 * Stock_vol)
print('Utility value with a risk-aversion of 10 is equal to: %.4f' % U_p5)
```

Output:

```
Utility value with a risk-aversion of 2 is equal to: -0.0091
Utility value with a risk-aversion of 4 is equal to: -0.0189
Utility value with a risk-aversion of 6 is equal to: -0.0286
Utility value with a risk-aversion of 8 is equal to: -0.0384
Utility value with a risk-aversion of 10 is equal to: -0.0481
```