VILNIUS UNIVERSITY

FACULTY OF MATHEMATICS AND INFORMATICS

DATA SCIENCE STUDY PROGRAMME

Master's thesis

Financial Risk Management in the Baltic and Nordic Stock Markets Using Value-at-Risk and Expected Shortfall

Rizikuojamoji vertė ir tikėtinas vertės trūkumas finansinės rizikos valdymui Baltijos ir Šiaurės šalių akcijų rinkose

Inesa Burkevičiūtė

Supervisor : Assist. Prof. Dr Andrius Buteikis

Reviewer : Assoc. Prof. Dr Dmitrij Celov

Vilnius
2025

Summary

A comprehensive analysis from the financial risk perspective is performed for a portfolio consisting of three Baltic and five Nordic indices for the period spanning from 2016-12-20 to 2024-09-30 by estimating Value-at-Risk (VaR) and Expected Shortfall (ES). A combination of various backtesting methods, both traditional and comparative, is employed in order to assess the adequacy and compare the estimated VaR and ES models at several confidence levels and using four different calibration windows corresponding to one, two, three, and four years. It is found that the hierarchical Clayton copula with uniform margins produced by fitting the normal inverse Gaussian distribution to standardised residuals from GARCH-filtered logarithmic returns outperforms the benchmark estimates given by the historical simulation, multivariate normal inverse Gaussian distribution, and the hierarchical Frank copula approaches. The age-weighted historical simulation produces comparable results for a longer calibration window. Moreover, simple forecast combination strategies are explored, and an incremental VaR analysis is done for each of the eight indices in the portfolio by using the hierarchical Clayton copula approach.

Keywords: Clayton copula, Comparative backtesting, Expected Shortfall, Forecast combination, GARCH-Copula, Hierarchical Archimedean copula, OMX, Value-at-Risk

Santrauka

Darbe pateikiama išsami finansinės rizikos analizė portfeliui, sudarytam iš trijų Baltijos ir penkių Šiaurės šalių akcijų indeksų, laikotarpiui nuo 2016-12-20 iki 2024-09-30 įvertinant rizikuojamosios vertės (VaR) ir tikėtino vertės trūkumo (ES) modelius. Siekiant įvertinti modelių adekvatumą ir palyginti VaR ir ES modelių prognozavimo gebą keliems pasikliovimo lygmenims ir naudojant keturis skirtingus modelių kalibravimo periodus, atitinkančius vienerius, dvejus, trejus ir ketverius metus, naudojamas įvairių tradicinių ir palyginamųjų grįžtamojo patikrinimo metodų derinys. Nustatyta, kad hierarchinė Clayton'o kopula su tolygiai pasiskirsčiusiais marginaliaisiais skirstiniais, gautais įvertinant normalųjį atvirkštinį Gauso skirstinį standartizuotoms liekanoms iš GARCH modelio logaritminėms grąžoms, pranoksta istorinės simuliacijos, daugiamačio normaliojo atvirkštinio Gauso skirstinio ir hierarchinės Frank'o kopulos metodų įverčius. Svertinė pagal stebinių amžių istorinė simuliacija pasižymi panašiais rezultatais ilgesniam kalibravimo periodui. Taip pat darbe nagrinėjamos paprasti prognozių agregavimo būdai ir, naudojant hierarchinį Clayton'o kopulos metodą, kiekvienam iš aštuonių portfelį sudarančių akcijų indeksų atliekama prieauginė rizikuojamosios vertės analizė (angl. *incremental VaR*).

Raktiniai žodžiai: Clayton'o kopula, GARCH-kopula, hierarchinė Archimedo kopula, OMX, palyginimasis grįžtamasis patikrinimas, rizikuojamoji vertė, tikėtinas vertės trūkumas

List of figures

List of tables

Contents

List of abbreviations

AIC	Akaike information criterion
AWHS	age-weighted historical simulation
BCBS	Basel Committee on Banking Supervision
EBA	European Banking Authority
ES	Expected Shortfall
HAC	hierarchical Archimedean copula
HCC	hierarchical Clayton copula
HFC	hierarchical Frank copula
HS	(basic) historical simulation
MNIG	multivariate normal inverse Gaussian (distribution)
P&L	profit and loss
VaR	Value-at-Risk

Introduction

The Basel Committee, which today sets the international regulatory standards for financial institutions with the goal to ensure effective risk management and financial stability, was established in 1974 as a result of grave disturbances observed in international financial markets [1]. Hence, the initial accord was defined by the Committee that was meant to even out the different national capital requirements and strengthen the stability of the international banking system. In that accord, called the Basel Capital Accord, one of the earliest amendments introduced the Value-at-Risk model in the regulatory context for market risk capital measurement requirements by allowing banks to use internal models based on this risk measure, and it has been used for the minimum market risk capital requirements calculation and more general market risk management since then. However, an updated set of proposals for the market risk framework, the Fundamental Review of the Trading Book (FRTB), has been proposed which includes a shift from using Value-at-Risk models to Expected Shortfall in order to be more mindful of the potential tail risks rather than just focus on one quantile of the loss distribution. Nevertheless, both of these risk measures are closely related, and it is meaningful and often required to consider them in conjunction when, for instance, validating the estimated ES models or comparing the forecast accuracy of competing estimation procedures.

The purpose of this work is to explore how well methods for estimating VaR and ES of different complexity and for several confidence levels (95%, 97.5%, and 99%) are able to assess the risk measures for a portfolio consisting of Baltic and Nordic stock indices. Five different models (historical simulation, age-weighted historical simulation, multivariate normal inverse Gaussian distribution, and the hierarchical Clayton and Frank copulas) are estimated and analysed by applying both traditional and comparative backtesting exploiting the joint elicitability of VaR and ES. Additionally, forecast combinations are calculated from the best-performing models and assessed for their goodness-of-fit and predictive power, relative to other methods. Incremental VaR analysis is also considered in order to determine the influence that each of the individual assets in the portfolio has on the total portfolio VaR.

The rest of this thesis is structured as follows: Section 1 provides a review of the existing literature for different estimation methods of VaR and ES; Section 2 outlines the methodology employed in this research, describing all estimation, backtesting methods, and the analysed data; Section 3 presents the results of the portfolio analysis, followed by a concluding Section for the results and conclusions, which summarises the key findings of the thesis and suggests potential avenues for future research.

1 Literature review

The following subsections are intended to present the general overview of the Value-at-Risk and Expected Shortfall risk measures and the reasoning behind the methods chosen in this thesis.

1.1 Desirable properties of risk measures

Artzner et al. [2] define a coherent risk measure as one that satisfies a set of four axioms of translation invariance, subadditivity, positive homogeneity, and monotonicity. Subadditivity implies that the merging of two portfolios should result in a risk measure value no larger than the sum of the risk measure values for individual portfolios. Hence, VaR is often criticised for violating the subadditivity property for non-elliptical distributions. However, it is not necessarily a reasonable argument for rejecting it as a useful risk measure – for instance, Danielsson et al. find that in most practical applications, VaR does not violate the subadditivity axiom [3]. On the other hand, even though ES is a coherent risk measure, it lacks another feature called elicitability:

1.1.1 Definition. A statistics $\psi(Y)$ of a random variable Y is said to be *elicitable* if it minimises the expected value of a scoring function S:

$$\psi = \operatorname*{argmin}_x \mathbb{E}[S(x, Y)].$$

For predictions x_t and the actual observed values y_t of the random variable Y, the forecasting model is considered to be best if the associated mean score

$$\bar{S} = \frac{1}{T} \sum_{t=1}^{T} S(x_t, y_t)$$

produces values as low as possible (like, for instance, the mean minimises the mean square error) [4, p. 1–2]. Though no such scoring function exists for ES, it has been found that ES is elicitable jointly with VaR. Elicitability is needed to perform model selection, hence, comparative backtesting of ES forecasts produced by different models can be done by using a scoring function for the pair of VaR and ES risk measures.

As an alternative to both VaR and ES, the use of expectiles for measuring financial risk has also been explored in the literature [5]. However, even though expectiles are both coherent and elicitable, they violate another axiom of comonotonic additivity: if assets X_1 and X_2 are increasing functions of a common underlying asset, for a risk measure ρ, the identity $\rho(X_1 + X_2) = \rho(X_1) + \rho(X_2)$ should hold. This essentially means that by moving in the same direction, such assets do not hedge each other [6, p. 2]. Yet the only risk measure that is characterised by coherence, elicitability, and comonotonic additivity is the expected loss [6, p. 14].

1.2 Ways to estimate VaR and ES

According to the European Banking Authority (EBA) report for the results from the 2023 market risk benchmarking exercise [7, p. 49], the historical simulation (HS) approach is still the most popular method for modelling market risk with 70% of the surveyed European banks opting for it, as opposed to using Monte Carlo simulation, parametric, or combination/other methods. Hence, the HS procedure is also considered in this work as one of the benchmark models for VaR and ES together with a simple extension based on weighting the observations by their age in order to account for and emphasise the recency of the information contained in the data.

For capturing the relationships between the individual assets in a portfolio, models based on estimating the multivariate distributions can be used in VaR and ES estimation. Byun and Song [8] consider the normal and the multivariate normal inverse Gaussian distributions together with the elliptical, vine, and hierarchical copulas for VaR estimation on a portfolio of four prominent technology companies in the American market: Facebook, Amazon, Netflix, and Google (FANG). A simulation study for the copula modelling was also performed. Both the simulation results in terms of violation rates and backtesting the models estimated on the real portfolio using the Kupiec test indicate the superiority of the hierarchical Clayton copula model with normal inverse Gaussian margins due to its ability to reflect the left distribution tail dependence, whereas the performance of vine copulas is potentially hindered by overfitting resulting from a large number of parameters that need to be estimated. The multivariate normal inverse Gaussian distribution appears to produce forecasts that are not outperformed by other copula models, for instance, C-vine, D-vine, or hierarchical Frank copulas. As Byun and Song propose, in this work, a larger dimension portfolio is analysed employing the multivariate normal inverse Gaussian distribution, hierarchical Clayton copula and hierarchical Frank copula. The hierarchical Clayton copula is chosen due to its promising predictive power, and hierarchical Frank copula together with the multivariate normal inverse Gaussian distribution are chosen for comparison purposes. However, a further extension in this thesis is considered by fitting GARCH models to the margins to filter for volatility in data and only then fitting the NIG distribution to the standardised GARCH residuals for each of the assets in the portfolio. Generally, approaches based on this procedure are also called copula-GARCH in the literature [9]. Another extension lies in estimating ES in addition to VaR in this work by using the hierarchical Archimedean copulas.

1.3 Research on the Baltic and Nordic Markets

The literature for the analysis from the financial risk management perspective for the Baltic stock markets appears to be limited. Papers that are centred around the equity indices of the Baltic countries mainly focus on the VaR measure. For example, Radivojevic et al. [10] assess the applicability of the filtered historical simulation (FHS) model proposed by Hull and White for the Baltic stock indices during 2013–2016. The FHS procedure is based on using a volatility-weighting scheme for the observed returns. The authors use the Kupiec's unconditional coverage and Christoffersen's conditional coverage tests to backtest VaR. They find that no single specification of the GARCH model for the volatility forecasts in FHS results in an estimating procedure which uniformly produces the

best forecasts for each of the indices (VaR is estimated for each Baltic index separately instead of considering their portfolio). Jurgilas [11] performs the VaR analysis for a portfolio of four Nordic and three Baltic indices for the years 2000–2011 by considering 17 different models for estimating VaR, generally representing the historical simulation, unconditional models based on some distribution, volatility-based, and Extreme Value Theory-focused models. The estimated models are also back-tested using the same Kupiec and Christoffersen tests. It is observed that the unconditional VaR models (including HS and distribution-based approaches) fail to account for volatility clustering and produce very static estimates of risk. A similar conclusion is made to the previously mentioned paper for the Baltic equity market – estimating VaR models for each considered index results in no obvious superior choice for the estimation procedure across the different assets.

Hence, in this thesis, a portfolio of eight indices representing the Baltic and Nordic stock markets is constructed and analysed using VaR and ES risk measures for a period around the years 2017–2024. Moreover, a comprehensive backtesting methodology is applied: instead of only assessing the proportion of violations for VaR via, for example, the Kupiec or Basel Traffic Light tests, the mixed Kupiec test is also applied in order to determine whether the estimated models produce forecasts resulting in violations of VaR that are independent from each other. In addition, the ES values are validated by applying three types of goodness-of-fit tests, and both the produced VaR and ES forecasts are compared by using a statistic of the Diebold-Mariano test type. Two other types of methods that do not seem to be that much explored in the literature, especially for a portfolio of Baltic and Nordic equity indices, are the incremental VaR analysis, meant for assessing the impact of individual assets on the overall portfolio VaR, and combining strategies of VaR and ES forecasts. For instance, Le [12] finds that for the Vietnamese stock market, combining estimates produced by seven risk models by applying 10 different aggregation strategies have adequate predictive power, although they are frequently inferior to some individual risk model. Furthermore, it is observed that more complex combining methods based on optimised weighting functions are not superior to simple combination strategies. Thus, three forecast combination methods are also considered in this work by applying them to the best-performing models based on the traditional and comparative backtesting results.

2 Methodology

The following subsections describe all VaR and ES estimation and backtesting methods employed in this work.

2.1 Risk measures: VaR and ES

The idea behind Value-at-Risk (VaR) is to provide an estimate for the largest portfolio loss that would not be exceeded with a given high probability over some considered time interval (also called horizon, holding period). It can be defined in the following way:

2.1.1 Definition. For a given confidence level $q \in (0, 1)$ and time t, the Value-at-Risk of a portfolio with loss X is given by the smallest number x such that the probability of the loss X_t exceeding x (in absolute value) is no greater than $1 - q$:

$$\mathsf{VaR}_q^t = \mathsf{sup}\{x \in \mathbb{R} : \mathbb{P}(X_t \leq x) \leq 1 - q\}.$$

Usually, q is chosen to be 0.95 or higher. Since all estimation methods used in this work are based on simulating potential return scenarios, VaR is given by the empirical $1 - q$ level quantile of the scenario sample, which can be estimated in several ways. This is considered in the European Central Bank (ECB) guide to internal models, which is meant to ensure consistent adherence of supervised institutions to regulatory standards (based on the current applicable European Union and national law). In the section describing the methodology for VaR and stressed VaR, it is underlined that when using a simulation approach to estimate these models, the quantile estimation method should be "asymptotically unbiased, distribution-free, and assume that the probability of experiencing a P&L lower (or higher) than the lowest (or highest) simulated value is strictly greater than zero" [13, p. 183–184]. Among the referenced estimators, a simplified method is provided, which corresponds to Definition 6 from an article by R. J. Hyndman and Y. Fan reviewing different sample quantile estimators implemented in statistical computer packages [14]: it satisfies five out of six desirable properties for a sample quantile for which the authors check in each of the analysed definitions. However, in contrast to the suggestion in the ECB document, the authors of the latter paper recommend using Definition 8 (given in the same paper [14]), as it results in approximately median-unbiased estimates of the quantile while satisfying the same properties as Definition 6 [14, p. 364]. Hence, the median-unbiased quantile estimator is used in this work for all estimation procedures (except the age-weighted historical simulation).

2.1.2 Definition. A VaR exception (also called breach, exceedance, failure, or violation in the literature) happens when the portfolio return is lower than the forecasted VaR. The associated exception indicator (hit series) can be defined as:

$$I_t(q) \coloneqq \mathbb{1}_{\{X_t < \mathsf{VaR}_q^t\}} = \begin{cases} 0 & \text{if } X_t \geq \mathsf{VaR}_q^t, \\ 1 & \text{if } X_t < \mathsf{VaR}_q^t. \end{cases}$$

The Expected Shortfall (ES) measure represents the average return distribution tail risk by considering returns which exceed the VaR level:

2.1.3 Definition. For a given confidence level $q \in (0,1)$ and time t, the Expected Shortfall of a portfolio with loss X is given by:

$$\mathsf{ES}_q^t = \mathbb{E}[X_t | X_t \leq \mathsf{VaR}_q^t] = \frac{1}{1-q} \int_0^{1-q} \mathsf{VaR}_{1-u}^t \mathsf{d}u.$$

It should be noted that in this work, VaR and ES are negative values, i.e. the portfolio loss is not converted to be positive, hence, all the notation and used methods are defined having that assumption in mind. The one-day horizon is considered when estimating both VaR and ES.

2.2 Estimation methods

In this thesis, five different estimation methods coming from three more general groups are employed to produce VaR and ES forecasts at 95%, 97.5%, and 99% confidence levels using four different calibration windows $w \in \{250, 500, 750, 1000\}$ (days). In general, all methods rely on the price changes for each individual index: the total portfolio value scenarios are generated by combining the simulated price scenarios for each index in the portfolio, and only then the total portfolio log returns are calculated and used as the simulated sample for VaR and ES estimation. Specifically when the historical simulation method is described in the literature, frequently the additional information that considering individual assets in the portfolio might provide is omitted by only using total portfolio returns in risk measure estimation.

All models are estimated using a rolling-window procedure: for instance, for the calibration window w of 250 days and the first backtesting window point $t = w + 1 = 251$, the first 250 (log return) observations are used to produce estimates of VaR and ES for $t = 251$; then the beginning and end of the calibration period, $t = 1$ and $t = 250$, is shifted by one date to $t = 2$ and $t = 251$, and the risk measures are forecasted for $t = 252$. This procedure is done until $t = T + w = 1811$, i.e., when the last forecasting date is reached. Naturally, since the dataset has the same number of observations while the length of w varies, the backtesting window T depends on w and is equal to $1811 - w$.

2.2.1 Historical simulation (HS)

The historical simulation approach for estimating risk measures makes no parametric assumptions about the underlying return distribution; it is only assumed that the portfolio changes observed in the past will be appropriate for forecasting the potential changes in the future, i.e., the underlying return distribution stays the same. If v_i is defined as the value of the asset in the portfolio on day i and the forecasting is performed for day $n + 1$, scenario i in the historical simulation provides the potential value of the asset for day $n + 1$ as [15, p. 517]

$$pv_i = v_n \frac{v_i}{v_{i-1}}.$$

Then, the portfolio value scenarios are given by the weighted sum of the individual asset scenarios, in this work calculated as

$$PV_i = 10 \cdot \sum_{a=1}^{A} pv_i^a,$$

assuming 10 units of each asset $a = 1, ..., A$ in the portfolio. VaR and ES are determined by taking the quantile or the conditional mean, respectively, from the log-return scenario sample:

$$r_i^{sim} = \ln\left(\frac{PV_i}{PV_n}\right), \quad i = 1, ..., w; \quad w \in \{250, 500, 750, 1000\}.$$

Here PV_n denotes the actual portfolio value observed on day n.

A simple extension of the HS procedure can be achieved by applying an age-related weighting of the observations in order to consider the recency of the information given by the observed returns. For the age-weighted historical simulation (AWHS) procedure, instead of giving equal weights to the generated log-return scenarios, they are given weights (as per [15, p. 521])

$$\frac{\lambda^{w-i}(1-\lambda)}{1-\lambda^w}, \quad \sum_{i=1}^{w} \frac{\lambda^{w-i}(1-\lambda)}{1-\lambda^w} = 1.$$

Here $\lambda \in (0, 1)$ is the rate at which the weights decline (this work uses λ = 0.98), w denotes the number of scenarios (which coincides with the length of the calibration window for historical simulation approaches), and $i = 1, ..., w$. Higher i values are associated with more recent scenarios and larger weights. After ranking the generated portfolio return scenarios, VaR_q^t is the value for which the cumulative weight equals to $1 - q$ (if such an observation does not exist, the value is linearly interpolated between scenarios that have cumulative weights around $1 - q$). ES is calculated as the weighted average of the scenarios equal to and exceeding VaR using the assigned cumulative weights.

2.2.2 Multivariate normal inverse Gaussian distribution (MNIG)

A d-dimensional random variable that follows the multivariate normal inverse Gaussian distribution is a variance-mean mixture of a d-dimensional Gaussian random variable $\boldsymbol{Y}$ with a univariate inverse Gaussian distributed mixing variable Z. There are several specifications of the parameters available, but using the $(\alpha, \boldsymbol{\mu}, \boldsymbol{\Gamma}, \delta, \boldsymbol{\beta})$ parametrisation, a MNIG distributed random variable can be derived from

$$\boldsymbol{X} = \boldsymbol{\mu} + Z\boldsymbol{\Gamma}\boldsymbol{\beta} + \sqrt{Z}\boldsymbol{\Gamma}^{1/2}\boldsymbol{Y},$$

where $\boldsymbol{\beta} \in \mathbb{R}^d$, $\boldsymbol{\mu} \in \mathbb{R}^d$, and $\boldsymbol{\Gamma} \in \mathbb{R}^{d \times d}$; $\boldsymbol{Y} \sim \mathcal{N}_d(\boldsymbol{0}, \boldsymbol{I})$; Z is an inverse Gaussian distributed random variable, $Z \sim \mathcal{IG}(\delta^2, \alpha^2 - \boldsymbol{\beta}^T\boldsymbol{\Gamma}\boldsymbol{\beta})$, with $\alpha > 0$ and $\delta > 0$ [16].

To estimate VaR and ES, for each backtesting period date t, a log return sample of calibration window w observations is used to fit the MNIG distribution. Then, 10000 samples of dimension $d = 8$ are generated from the fitted distribution. The generation of return scenarios is mostly equivalent to the way it is done for the historical simulation, however, the number of generated scenarios r_i for the return distribution is $i = 1, ..., 10000$, and the potential values for the individual assets are given

by

$$pv_i = v_n \cdot \exp\{r_i\},$$

since the generated r_i scenario is the logarithmic return.

Fitting of the MNIG distribution is performed by using a type of expectation-maximisation (EM) algorithm called the multi-cycle, expectation, conditional maximisation (MCECM) algorithm given by Algorithm 3.14 by Mcneil et al. in [17, p. 83]. This EM estimation method is implemented in the **QRM** package in R via the `fit.mNH` function [18]. Sampling from the estimated multivariate NIG distribution is performed using the **ghyp** package in R [19]. Since it is suitable for heavy-tailed and skewed data, the univariate version of this distribution is also used for fitting the distribution parameters in order to convert the standardised residuals from fitted GARCH models into uniform margins for copula modelling. This is done using the `nigFit` function from the **GeneralizedHyperbolic** package in R, which implements the maximum likelihood estimation (MLE) method [20].

2.2.3 Hierarchical Archimedean copulas (HAC)

Simply put, copulas are a specific type of multivariate cumulative distribution functions:

2.2.1 Definition. A copula is a multivariate distribution function with standard uniform univariate margins U_i, $U_i \sim \mathcal{U}(0,1)$.

The main families of copula functions are the elliptical (Gaussian and t) and Archimedean copulas. A multivariate Archimedean copula can be defined as follows:

2.2.2 Definition. An Archimedean copula is a function $C : [0,1]^k \to [0,1]$ given by

$$C(u_1, ..., u_k) = \phi\left\{\phi^{-1}(u_1) + \cdots + \phi^{-1}(u_k)\right\}.$$

$\phi : [0,\infty) \to [0,1]$ is a continuous strictly decreasing convex function, called the *generator* of C with $\phi(0) = 1, \phi(\infty) = 0$, and ϕ^{-1} is its pseudo-inverse [21, p. 2]. ϕ commonly depends on a single parameter θ.

Two types of Archimedean copulas are considered in this work, namely Clayton and Frank copulas. The Clayton copula is characterised by its ability to model the lower tail dependency between the univariate margins. In contrast, the Frank copula has radial symmetry without tail dependence [22, p. 151]. A visual example of samples produced by bivariate Clayton and Frank copulas is provided in Figure 1. These copulas appear to produce bivariate relationships comparable to what is observed in the data used in this thesis (see Figures 19 and 20 in the Appendix).
Table 1 provides the generator functions for the Clayton and Frank copulas, alongside their parameter ranges.

The simple Archimedean copula is rather restrictive in its nature, since it only depends on one parameter θ and produces a symmetric dependency of the margins with respect to the permutation of variables (such a distribution is called exchangeable) [23, p. 3]. It may be unrealistic to assume that all variables of some analysed multivariate distribution are related in the same way, especially

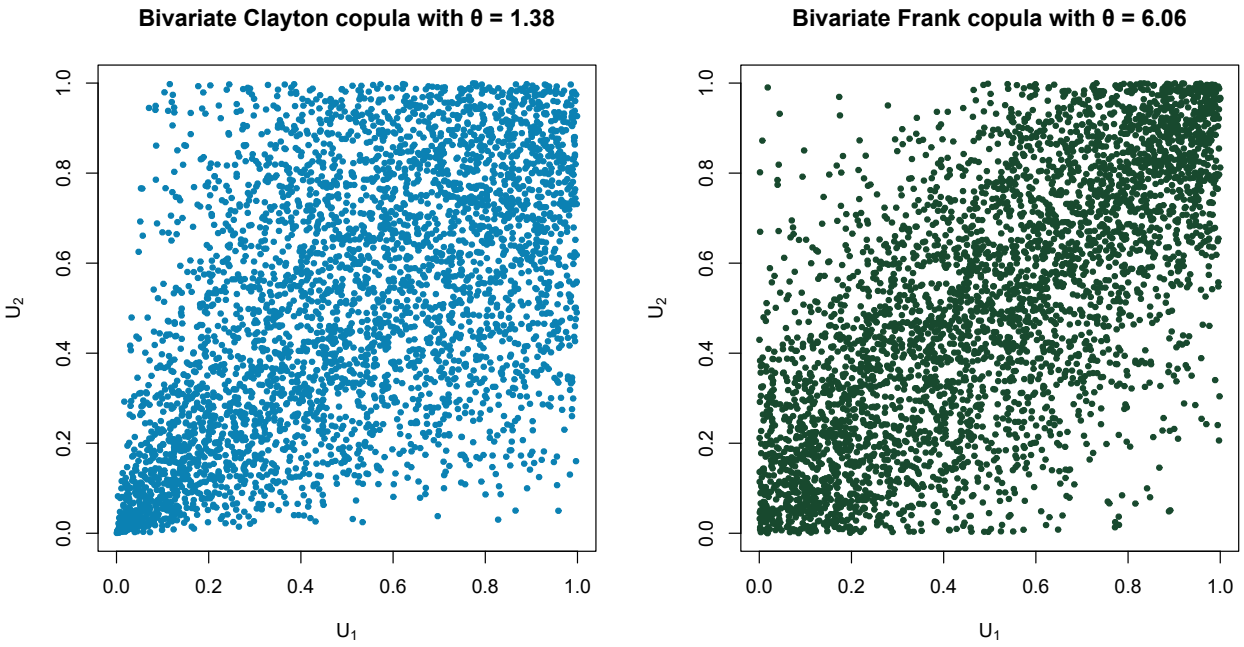


Figure 1. *Examples for scatter plots of simulated observations from the Clayton and Frank copulas*

Table 1. *Generator functions ϕ for the Clayton and Frank copulas.*

Generator	Range of θ	$\phi(u;\theta)$
Clayton	$(0,\infty)$	$(u+1)^{-1/\theta}$
Frank	$(0,\infty)$	$-\ln\left[\frac{1-\{1-\exp\{-\theta\}\}\exp\{-u\}}{\theta}\right]$

in the context of financial asset dynamics. Hence, an extension of the Archimedean copula provided by the hierarchical Archimedean copulas (HAC) is considered here. As is described by Okhrin et al. in [24], the dependence structure of HACs is defined recursively via bivariate copulas. For a fully nested copula,

$$C(u_1,...,u_k) = \phi_{k-1}(\phi_{k-1}^{-1} \circ \{\phi_{k-2}[\dots(\phi_2^{-1} \circ \phi_1[\phi_1^{-1}(u_1) + \phi_1^{-1}(u_2)] + \phi_2^{-1}(u_3)) + \\ + \dots + \phi_{k-2}^{-1}(u_{k-1})]\} + \phi_{k-1}^{-1}(u_k)).$$

There are $k-1$ parameters produced by the bivariate copulas in the HAC model that have to satisfy the condition of $\theta_1 > \dots > \theta_{k-1}$ [8, p. 65]. An example of how the tree structure of a partially nested Clayton copula can look like is provided in Figure 2 (for the Frank copula, see Figure 18 in the Appendix). It can be seen that the strongest relationship was identified between OMXS30GI and OMXH25GI indices, and the estimated structure can be denoted as

$$s = ((((((\text{OMXS30GI}, \text{OMXH25GI}), \text{OBX}), \text{OMXC25GI}), (\text{OMXVGI}, \text{OMXTGI})), \text{OMXRGI}), \text{OMXIGI}) .$$

The procedure that is used to estimate VaR and ES is similar to that as given by Lu et al. in [9]:

1. For all individual indices in the portfolio, univariate AR(1)-GARCH(1, 1) models are estimated using w observations. An AR(1)-GARCH(1,1) specification is used for all indices. In case the es-

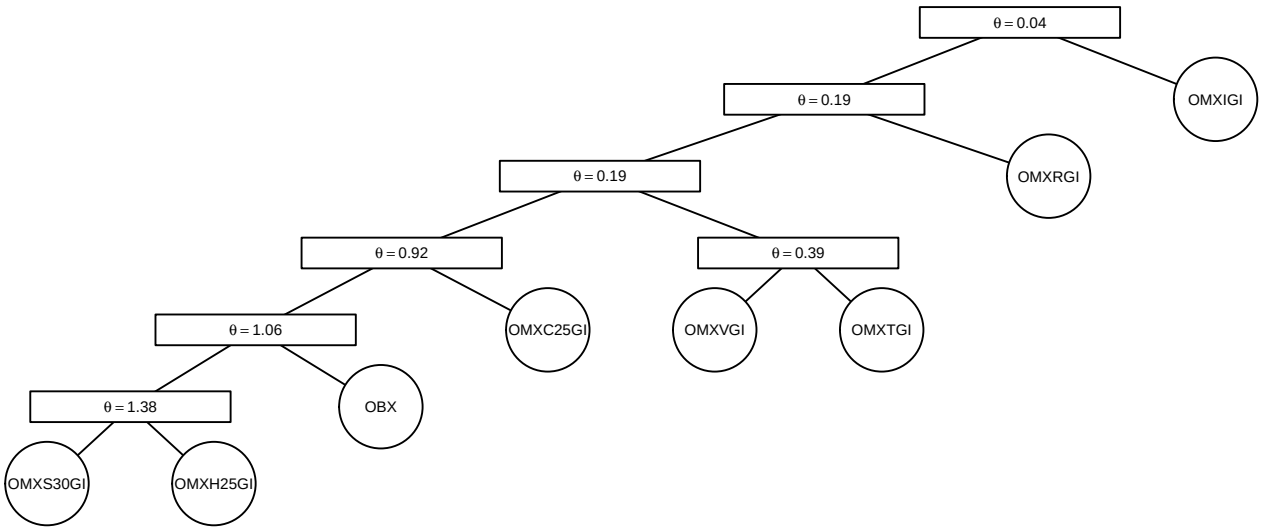


Figure 2. *Example of a partially nested Archimedean copula tree produced by fitting the hierarchical Clayton copula to the uniform margins from standardised AR(1)-GARCH(1, 1) residuals from the first 500 observations in the dataset*

timation procedure fails to converge, another specification for the conditional mean is chosen based on AIC (however, that was necessary for only a few iterations for the shorter calibration windows for the OMXC25GI index).

2. The one-step return mean and standard deviation forecasts for time $t+1$ are retrieved from the estimated models (for asset a, $\hat{\mu}^a_{t+1}$ and $\hat{\sigma}^a_{t+1}$, respectively).

3. For the standardised GARCH residuals z^a_t, univariate normal inverse Gaussian distribution parameters are estimated and used to transform into uniform margins to be used for copula estimation.

4. Using the uniform margins, either the hierarchical Clayton copula or the hierarchical Frank copula is fitted.

5. $M = 10000$ random variables of dimension $d = 8$ are simulated from the estimated copula.

6. The simulated standardised residuals are obtained by using the inverse functions of the estimated marginal NIG distributions for each asset.

7. The simulated log returns are retrieved by multiplying the simulated standardised residuals by the volatility forecast and adding the mean forecast to that multiplication.

After getting the simulated sample of log returns, the subsequent steps are equivalent to those performed for the MNIG-type VaR and ES estimation. The AR(1)-GARCH(1, 1) model is given by

$$\begin{cases} r_t &= \mu + \phi r_{t-1} + \epsilon_t, \\ \epsilon_t &= \sigma_t z_t,\ z_t \sim \text{i.i.d. } (0,1), \\ \sigma_t^2 &= \omega + \alpha_1 \epsilon_{t-1}^2 + \beta_1 \sigma_{t-1}^2. \end{cases}$$

Here σ_t^2 indicates the conditional variance, ϵ_t is the residual, and z_t is the standardised residual with zero mean and unit variance.

Estimation of HAC for the univariate margins derived from the standardised GARCH residuals is done using the **HAC** package in R [23], which, by default, uses a maximum likelihood setup based on realised pseudo-variables for the multistage estimation of HAC parameters [23, p. 5].

2.3 Backtesting

2.3.1 Traditional backtesting

In this work, the term *traditional backtesting* is adopted from Nolde and Ziegel [25, p. 1834] to mean statistical tests for VaR and ES that check the following null hypothesis:

$$H_0 : \text{The risk measurement procedure is correct.}$$

The analysed model is considered to be adequate if the null hypothesis is not rejected. Such tests can be used for assessing the goodness of fit of estimation procedures but are not suitable for their comparison.

Basel Traffic Light test

The Basel Traffic Light test for assessing the VaR model performance was proposed by the Basel Committee on Banking Supervision (BCBS) in 1996 [26]. Its goal is to compare the percentage of losses that are covered by the VaR model with the expected confidence level (the BCBS document specifies the 99% level, however, the test can be extended to any chosen q). This approach is based on the sum of exceptions that occurred during the backtesting window of T days (using similar notation as in [27]):

$$X_{VaR}^T(1-q) \coloneqq \sum_{t=1}^{T} I_t(q).$$

Under the null hypothesis, $X_{VaR}^T(1-q) = X_{VaR}^T(p) \sim \text{Bin}(T, p)$ with $p \coloneqq 1-q$:

$$H_0 : \mathbb{E}\left[X_{VaR}^T(p)\right] = Tp.$$

Three colour zones for decision making are defined using the cumulative distribution function of $X_{VaR}^T(p)$:

- The green zone is defined as the number of exceptions N for which $\mathbb{P}\left[X_{VaR}^T(p) \leq N\right] < 0.95$, which results in the VaR model being considered as adequate
- The yellow/amber zone arises from N for which $0.95 \leq \mathbb{P}\left[X_{VaR}^T(p) \leq N\right] < 0.9999$, which suggests potential problems with the VaR model
- The red zone is assigned when for the number of exceptions N, $\mathbb{P}\left[X_{VaR}^T(p) \leq N\right] \geq 0.9999$, which indicates that improvement of the VaR model is required.

The null hypothesis of an appropriate VaR model is rejected only when there is underestimation of risk observed, since overestimation is not an issue from the regulatory perspective. In the current regulatory requirements, this approach is used to assess the ES model at the 97.5% level by backtesting $\text{VaR}_{97.5}$ and VaR_{99} over the last 250 trading days (corresponding to one year) [28].

Kupiec Proportion of Failures (PoF) test

The Proportion of Failures test was proposed by Paul H. Kupiec in 1995 and is considered to be the earliest published statistical backtest for VaR [29, p. 4], [30]. It is an unconditional coverage test which is again meant to check whether the proportion of exceptions to the total number of observations corresponds to the expected coverage rate p (e.g. 5% or 1%). The following null and alternative hypotheses are used:

$$\begin{cases} H_0 : p = \dfrac{N}{T}, \\ H_1 : p \neq \dfrac{N}{T}, \end{cases}$$

where N is the number of exceptions and T is the number of observations in the backtesting window. Rather than directly calculating the probabilities from the binomial distribution (like for $X^T_{VaR}(p)$ in the Basel Traffic Light approach), a log-likelihood ratio test statistic is formulated as

$$LR_{PoF} = -2 \ln \left((1-p)^{(T-N)} p^N \right) + 2 \ln \left(\left(1 - \frac{N}{T} \right)^{(T-N)} \left(\frac{N}{T} \right)^N \right).$$

Here p, N, and T are defined as before. It relies on the central limit theorem – for a large T, the binomial distribution can be approximated by the normal distribution. Hence, LR_{PoF} is asymptotically χ^2_1-distributed under the null hypothesis H_0. It is rejected at the significance level $\alpha = 0.05$ if the p-value $= 1 - F_{\chi^2}(LR_{PoF}, 1) \leq 0.05$. In contrast to the Basel approach, this test can indicate both underestimation and overestimation of the portfolio's underlying level of risk.

Mixed Kupiec test

The mixed Kupiec test, formulated by Marcus Haas, suggests combining the ideas of another test that was proposed by Kupiec, the Time until First Failure (TUFF), and Christoffersen's Independence test [31, p. 12]. The TUFF test measures the time until the first VaR violation in the backtesting window, and the Independence test is a likelihood ratio test for identifying unusually frequent consecutive violations. Their statistics are also assumed to be χ^2_1-distributed under H_0. The extension proposed by the mixed Kupiec test is based on measuring the time between two exceptions via

$$LR_i = -2 \ln \left(\frac{p(1-p)^{\nu_i - 1}}{\hat{p}(1-\hat{p})^{\nu_i - 1}} \right).$$

Here $\hat{p} = N/T$ and ν_i is the time between exceptions i and $i-1$. Hence, N statistics are produced (corresponding to each exception). The null hypothesis is defined as

$$H_0 : \text{VaR exceptions are independent from each other,}$$

thus, the test statistics are also independent and can be summed, resulting in the following mixed Kupiec test statistic:

$$LR_{ind} = \sum_{i=2}^{N} \left(-2 \ln \left(\frac{p(1-p)^{\nu_i - 1}}{\hat{p}(1-\hat{p})^{\nu_i - 1}} \right) \right) - 2 \ln \left(\frac{p(1-p)^{\nu - 1}}{\hat{p}(1-\hat{p})^{\nu - 1}} \right).$$

Here ν stands for the time until the first exception. Since the sum of independent chi-square random variables is also χ^2-distributed, LR_{ind} is asymptotically χ^2_N-distributed. It is joined with LR_{PoF} to get

$$LR_{mix} = LR_{ind} + LR_{PoF} \sim \chi^2_{N+1}.$$

Similarly to the PoF test, the null hypothesis is rejected at the significance level $\alpha = 0.05$ if the p-value $= 1 - F_{\chi^2}(LR_{mix}, N+1) \leq 0.05$.

Multinomial test (MT): Implicit ES backtesting

In order to indirectly backtest the Expected Shortfall produced by an estimated model, Kratz et al. [32] rely on the idea of simultaneously backtesting several VaR estimates given by the same model at levels $q_1, ..., q_N$ when formulating their test (here N indicates the number of VaR estimates used to approximate ES rather than the number of exceptions). The used levels can be defined by

$$q_j = q + \frac{j-1}{N}(1-q), \quad j = 1, ..., N, \quad N \in \mathbb{N},$$

for the starting level q at which ES is to be backtested. $q_0 = 0$ and $q_{N+1} = 1$ are also defined. Simultaneously testing VaR forecasts at N levels results in a multinomial distribution: by setting $X_t = \sum_{j=1}^{N} I_t(q_j)$, a sequence $\{X_t\}_{t=1,...,T}$ is derived that represents the counts of breached VaR levels for each time t in the backtesting window. The cell counts representing the frequency with which a j number of levels were exceeded over time are given by

$$O_j = \sum_{t=1}^{T} \mathbb{1}_{\{X_t = j\}}, \quad j = 0, \cdots, N.$$

For the random vector $(O_0, ..., O_N)$ to satisfy the unconditional coverage and independence hypotheses, $(O_0, ..., O_N) \sim \text{MN}(T, (q_1 - q_0, ..., q_{N+1} - q_N))$, where $\text{MN}(T, (q_1 - q_0, ..., q_{N+1} - q_N))$ denotes the multinomial distribution with T trials and probabilities $q_1 - q_0, ..., q_{N+1} - q_N$ for each of $N+1$ possible events. For instance, if $q = 0.95$ and $N = 4$, $(O_0, O_1, O_2, O_3, O_4) \sim \text{MN}(T, (0.95, 0.0125, 0.0125, 0.0125, 0.0125))$. For the general case, where $(O_0, ..., O_N) \sim \text{MN}(T, (\theta_1 - \theta_0, ..., \theta_{N+1} - \theta_N))$ with $0 = \theta_0 < \theta_1 < \cdots < \theta_N < \theta_{N+1} = 1$, Kratz et al. [32,

p. 395] formulate the following hypotheses for the multinomial test:

$$\begin{cases} H_0 : \theta_j = q_j & \text{for all } j \in \{1, ..., N\}, \\ H_1 : \theta_j \neq q_j & \text{for at least one } j \in \{1, ..., N\}. \end{cases}$$

Several specifications of the test statistic for evaluating these hypotheses are available, including Pearson chi-squared, Nass, and the likelihood ratio tests. In this work, the Nass test with a level number $N = 4$ is used, since the authors of the multinomial test found that as long as the backtesting window exceeds 250 observations, the Nass test tends to be stable in terms of size and power when N changes from 4 to 8. Moreover, it is preferable when the cell counts are small, which is expected when backtesting VaR exception counts [32, p. 395]. The Nass test is based on the Pearson chi-squared test but uses an improved approximation to the distribution of the statistic S_N:

$$S_N = \sum_{j=0}^{N} \frac{(O_j - T(q_{j+1} - q_j))^2}{T(q_{j+1} - q_j)} \underset{H_0}{\overset{d}{\sim}} \chi^2_N,$$

by using an adjustment factor c:

$$cS_N \underset{H_0}{\overset{d}{\sim}} \chi^2_\nu \quad \text{with} \quad c = \frac{2 \cdot \mathbb{E}(S_N)}{\mathsf{Var}(S_N)}, \quad \nu = c \cdot \mathbb{E}(S_N),$$

where

$$\mathbb{E}(S_N) = N \quad \text{and} \quad \mathsf{Var}(S_N) = 2N - \frac{N^2 + 4N + 1}{T} + \frac{1}{T} \sum_{j=0}^{N} \frac{1}{q_{j+1} - q_j}.$$

Then, H_0 is rejected when $cS_N > \chi^2_\nu(1 - 0.05) = \chi^2_\nu(0.95)$ at the significance level equal to 0.05 and $\chi^2_\nu(0.95)$ denoting the 0.95-quantile of the χ^2_ν distribution. Similarly to the Basel's approach, three colour zones can also be defined based on the cumulative distribution function of the χ^2_ν distribution. Kratz et al. [32, p. 403] propose using the same boundaries as in the Basel Traffic Light test in order to identify the zones that the test results fall into.

Exceedance residual (ER) test

McNeil and Frey [33] define a bootstrap test for backtesting the ES model based on the differences between losses and estimates for ES when a VaR exception is observed, i.e.,

$$er_t = (X_t - \widehat{ES}^t_q) I_t(q).$$

These differences are called exceedance residuals. It is also possible to standardise these residuals by some volatility forecast, but as Bayer and Dimitriadis point out in the supplementary appendix to their paper on regression-based backtesting of ES [34], when an ES backtest relies not only on ES forecasts but also on some other inputs, rejection of the null hypothesis of a correct model might be caused by some other incorrect component that was used for backtesting rather than ES itself. In light of this, they propose using fewer input parameters for goodness-of-fit assessment for ES by

considering the raw (non-standardised) exceedance residuals. Under the null hypothesis of a correct ES estimation procedure, it is assumed that $\mathbb{E}[er_t] = 0$. The test might be formulated by using either the one-sided (H_0^{1s}, H_1^{1s}) or two-sided hypotheses (H_0^{2s}, H_1^{2s}):

$$\begin{cases} H_0^{1s}: & \overline{er} \geq 0, \\ H_1^{1s}: & \overline{er} < 0, \end{cases}$$

and

$$\begin{cases} H_0^{2s}: & \overline{er} = 0, \\ H_1^{2s}: & \overline{er} \neq 0 \end{cases}$$

with

$$\overline{er} = \frac{1}{\sum_{t=1}^{T} I_t(q)} \sum_{t=1}^{T} er_t.$$

$\overline{er} > 0$ implies that, on average, the ES estimates are larger in absolute value than the losses, meaning the risk is not underestimated. Since no assumption about the underlying distribution of er_t is made, a bootstrapping procedure is used to derive the p-values for the test, which was described by Efron and Tibshirani [35, p. 224–227] (also, specifically for er_t – by Spring [36, p. 610]). For all N exceptions observed over the backtesting window T, adjusted exceedance residuals, er_t^{adj}, are calculated by subtracting $\overline{er}$ derived from the observed losses and risk measure estimates from er_t: $er_t^{\text{adj}} = (er_t - \overline{er}) \cdot \mathbb{1}_{\{er_t \neq 0\}}$. Then, for M bootstrap trials, draw N observations with replacement from the estimated sample of er_t^{adj}, and for each $i \in \{1, ..., M\}$, calculate $\overline{er}_i = \dfrac{\overline{er_i^{\text{adj}}}}{\sigma_i / \sqrt{N}}$, where $\overline{er_i^{\text{adj}}}$ and σ_i denote the mean and standard deviation of the bootstrap sample i, respectively. In this thesis, M = 10 000 is used. For the one-sided test formulation, the p-value is calculated by

$$p\text{-value} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}_{\{\overline{er}_i < \overline{er}\}},$$

and for the two-tailed test,

$$p\text{-value} = \frac{1}{M} \sum_{i=1}^{M} \mathbb{1}_{\{|\overline{er}_i| > |\overline{er}|\}}.$$

The null hypothesis of a correct ES model is rejected when the p-value is smaller than the significance level (0.05). Both H^{1s} and H^{2s} specifications are considered in this work, because although from the regulatory perspective only underestimation of risk is problematic, it is meaningful for a financial firm implementing and assessing their models of risk measures to also identify estimation procedures which produce forecasts that are too conservative, as they might lead to unnecessarily high capital requirements.

2.3.2 Comparative backtesting

In comparative (as opposed to traditional) backtesting, the following null hypotheses are considered:

$$\begin{cases} H_0^- : & \text{The internal model predicts at least as well as the standard model,} \\ H_0^+ : & \text{The internal model predicts at most as well as the standard model.} \end{cases}$$

The first null hypothesis, H_0^-, is equivalent to the null hypothesis of a correct estimation procedure in the case of traditional backtesting, whereas H_0^+ is formulated in such a way that the comparative backtest is passed when H_0^+ is rejected. This allows one to explicitly control the type I error of an inferior internal model being accepted over an established standard estimation procedure [25, p. 1848]. H_0^- and H_0^+ are tested by using the following test statistic:

$$\Delta_T \overline{S} \coloneqq \frac{1}{T} \sum_{t=1}^{T} \left(S(\hat{R}_t, X_t) - S(\hat{R}_t^*, X_t) \right),$$

which, for a large enough backtesting window T, has $\mathbb{E}[\Delta_T \overline{S}] \leq 0$ under H_0^- and $\mathbb{E}[\Delta_T \overline{S}] \geq 0$ under H_0^+. $\hat{R}_t^*$ represents the risk measure estimate produced by the standard model (the model whose forecasts some new, alternative (internal) model's forecasts are compared to), $\hat{R}_t$ stands for the risk measure value given by the internal procedure, and $S(R_t, X_t)$ denotes the scoring function. Testing the defined null hypotheses based on the $\Delta_T \overline{S}$ statistic results in the Diebold-Mariano test meant for comparing the accuracy of forecasts produced by two different models. Its statistic is defined as

$$T_4 = \frac{\Delta_T \overline{S}}{\hat{\sigma}_T / \sqrt{T}} \underset{H_0}{\overset{d}{\sim}} \mathcal{N}(0, 1).$$

Here $\hat{\sigma}_T^2$ is a heteroskedasticity- and autocorrelation-consistent (HAC) estimate of the asymptotic variance $\sigma_T^2 = \mathrm{Var}(\sqrt{T} \Delta_T \overline{S})$. In order to estimate σ_T^2, a HAC estimator defined by Diks et al. [37, p. 221–222] is used in this thesis:

$$\hat{\sigma}_T^2 = \hat{\gamma}_0 + 2 \sum_{k=1}^{K-1} a_k \hat{\gamma}_k.$$

Here $\hat{\gamma}_k$ stands for the k-lag sample covariance of the $\{\Delta_T \overline{S}\}_{t=1}^T$ sequence, and a_k are the Bartlett weights $a_k = 1 - k/K$ with $K = \lfloor T^{1/4} \rfloor$. As was the case with the Basel approach and the multinomial test, a three-zone approach can also be followed in case of comparative backtesting (see Figure 3).

The choice of the scoring function $S(R_t, X_t)$ (where R_t is either VaR_q^t or the pair $(\mathsf{VaR}_q^t, \mathsf{ES}_q^t)$ in this work) is not unique, however, Nolde and Ziegel [25, p. 1871] suggest using the so called *0-homogeneous* scoring function for backtesting ES:

$$S((\mathsf{VaR}_q^t, \mathsf{ES}_q^t), X_t) = \mathbb{1}_{\{X_t < \mathsf{VaR}_q^t\}} \frac{X_t - \mathsf{VaR}_q^t}{\mathsf{ES}_q^t} + (1 - q) \left(\frac{\mathsf{VaR}_q^t}{\mathsf{ES}_q^t} - 1 + \ln\left(-\mathsf{ES}_q^t\right) \right),$$

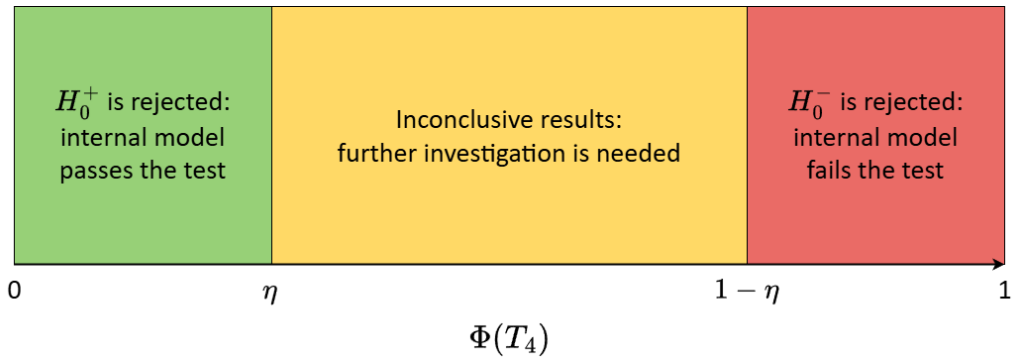


Figure 3. *The three-zone approach for comparative VaR and ES backtesting with significance level η suggested by Fissler et al. [38]*

and, analogously for VaR, the 0-homogeneous scoring function is given by:

$$S(\mathsf{VaR}_q^t, X_t) = (1 - q - \mathbb{1}_{\{X_t < \mathsf{VaR}_q^t\}}) \ln\left(-\mathsf{VaR}_q^t\right) + \mathbb{1}_{\{X_t < \mathsf{VaR}_q^t\}} \ln(-X_t).$$

Comparative backtests require elicitability of the considered models. As has previously been discussed, VaR is an elicitable risk measure, whereas ES is only jointly elicitable with VaR. Hence, for ES, the scoring functions used in comparative backtesting are based on both ES and VaR estimates. However, the weakness of this approach for backtesting ES lies in the fact that having to consider VaR forecasts in the scoring function might lead to an incorrect ranking of the ES estimates, i.e., an incorrect VaR model may prevent one from identifying an adequate model for ES [39, p. 9].

2.4 Incremental VaR

In case there is a need to determine how an existing asset in the portfolio contributes to its risk or how adding a new position would impact the current portfolio, it can be done using several methods. The marginal contribution of the individual existing or new portfolio components on the total portfolio VaR can be assessed by component VaR, marginal VaR, and incremental VaR. As defined in [40], the component VaR shows how the portfolio VaR would be influenced by removing that component from the portfolio. The marginal VaR indicates the change in the portfolio's risk caused by an increase or decrease in the amount of money invested in a certain asset. The incremental VaR reflects the change in the portfolio VaR when a new instrument is added to it. Since the precise definitions of these marginal measures vary in the literature, in this work, the idea of incremental VaR (IVaR) is used to analyse the portfolio at hand in the following way:

$$\mathsf{IVaR}_q^t(i) = |\mathsf{VaR}_q^t| - |\mathsf{VaR}_{q,excl.i}^t|,$$

where $\mathsf{IVaR}_q^t(i)$ denotes the incremental VaR for the individual asset i, and $\mathsf{VaR}_{q,excl.i}^t$ indicates VaR_q^t for the portfolio without the asset i. It is assumed that the full portfolio for which VaR is estimated in the work is the portfolio with the new asset i in relation to each index. Taking absolute values of the VaR_q^t for different portfolios results in $\mathsf{IVaR}_q^t(i) > 0$ in cases when the addition of the asset i to the portfolio increases risk, and $\mathsf{IVaR}_q^t(i) < 0$ indicating a reduction of the portfolio VaR after adding

a certain position i. The analysis is done for each index by using the best model, as determined by traditional and comparative backtesting results.

2.5 Data description

The analysed dataset consists of eight equity indices, which, in terms of publicly traded equities, generally cover the Baltic and Nordic financial markets. These indices are constructed and calculated by Nasdaq, Inc. subsidiaries in the Baltic and Nordic regions, with the exception of the OBX Index, which is composed of shares traded on the Oslo Stock Exchange. In each case, the Gross Index (GI) type (also called total return index) is chosen, which means that the index tracks the gross return of the shares it includes (any dividends, for instance, are assumed to be reinvested) rather than only the price movements. Such indices are generally considered to provide a more accurate reflection of the market's performance [41]. In addition, GI are chosen for all time series for consistency, since none of the Baltic countries have a version of a price index available.

Specifically, the Baltic indices are:

- OMX Riga All-Share Gross Index (OMXRGI) – the index comprises all stocks listed on Nasdaq Riga [42]. There are 8 constituents, largely reflecting the financials industry[1] [44].
- OMX Tallinn All-Share Gross Index (OMXTGI) – the index includes the shares traded on Nasdaq Tallinn [45]. As of October, it is composed of 19 components, mostly representing the financials, utilities, consumer discretionary, energy, and industrials sectors [46].
- OMX Vilnius All-Share Gross Index (OMXVGI) – the index consists of shares which are listed on Nasdaq Vilnius Stock Exchange [47]. As of October, it has 20 components, mainly from the utilities, telecommunications, financials, and consumer staples sectors [48].

It is noted in the descriptions of the Baltic indices that they are all meant to represent the general state of the respective country's economy. Hence, no filtering for liquidity is done when choosing the underlying stocks. This could be considered as a drawback of using these indices as benchmarks for considered markets' performances, since the prices of illiquid shares might be outdated and not reflect the current market demand, thus impacting the values of the indices they are included in.

The Nordic market is chosen to be represented by the following indices:

- OMX Copenhagen 25 GI Index (OMXC25GI) – it contains 25 most traded stocks on Nasdaq Copenhagen, primarily concentrated in the health care, industrials, and financials sectors (as of 2024-09-30) [49].
- OMX Helsinki 25 GI Index (OMXH25GI) – the index comprises 25 most actively traded shares on Nasdaq Helsinki [50]. As of the end of third quarter of 2024, the underlying stocks are predominantly from the industrials, financial services, basic materials, and technology sectors [51].

[1] The Industry Classification Benchmark (ICB) methodology is used to categorise the listed companies into 11 industry sectors by both Nasdaq Nordic and the Oslo Stock Exchange, the latter of which is controlled by Euronext [43].

- OMX Iceland All-Share Index (OMXIGI) – it consists of all 28 shares listed on Nasdaq Iceland [52]. The constituents mostly represent the financials, health care, consumer staples, industrials, and consumer discretionary sectors [53].

- OMX Stockholm 30 GI Index (OMXS30GI) – this index consists of 30 most actively traded securities on Nasdaq Stockholm, which, as of the end of September 2024, largely represent the industrials and financial services sectors [54].

- OBX Total Return Index (OBX) – it is a gross return index consisting of 25 most traded shares on the Oslo Stock Exchange in Norway. The components are largely from the energy, industrials, financials, consumer staples, and basic materials sectors (as of the end of September 2024) [55].

In contrast to the Baltic states's indices, most of the considered Nordic indices are characterised by better liquidity, since the underlying shares are chosen based on their ranking by turnover rather than just the fact of being listed on a certain securities exchange (except for the OMX Iceland All-Share Index) [50], [54].

Data for OMX indices is collected from the Nasdaq Nordic website [56], and the historical OBX values are retrieved from Yahoo Finance by using the *yfinance* library in Python. It is assumed that there are 10 units of each index in the analysed portfolio, and it is valued in euros. Hence, all Nordic indices (except for OMXH25GI, which is already measured in euros) are adjusted by using the respective exchange rates, which are also extracted using the *yfinance* library. After calculating daily logarithmic (log) returns for each index and removing the dates for which at least one of the indices did not have a value available, the log-return dataset contains 1811 observations for each time series, spanning from 2016-12-20 to 2024-09-30. The dataset with index values instead of returns is also used in the estimation of models – it contains the same observation dates as the log-return dataset (and the first date used for calculating the returns, i.e. 2016-12-19).

3 Analytical part

3.1 Initial data analysis

Table 2 presents an overview of the main descriptive statistics of the log returns (expressed in percentage for convenience) for each index in the analysed portfolio. It can be seen that over the period of around eight years that the data spans, all indices are characterised by very close to zero but slightly positive average and median values, and the minimum return being more extreme than the maximum. All assets display positive excess kurtosis, implying heavier tails than that of the normal distribution, and negative skewness, indicating a longer left tail of the distribution, which means that one could expect to observe frequent small positive returns and some large losses. For the Jarque-Bera test, the null hypothesis of normality is rejected for all indices, of which the excess kurtosis and skewness values were indicative. The Augmented Dickey-Fuller (ADF) test, whose null hypothesis states that there is a unit root present in the time series, allows one to conclude that there is no unit root in the series for any index, since all p-values are smaller than the significance level of 0.05. The results of the Engle's Lagrange multiplier test for autoregressive conditional heteroskedasticity (ARCH) effects done for 12 lags (denoted as ARCH(12)) indicate that conditional heteroskedasticity is present in the data, as the test's null hypothesis is formulated for missing ARCH effects (at least one of the 12 coefficients in the linear regression of the squared return lags is statistically significantly different from zero). Finally, the statistics of the Ljung-Box test with 10 lags (denoted as Q(10)) indicate autocorrelation of the log returns for all but one index. Hence, the results imply that the multivariate normal inverse Gaussian distribution might provide a good fit for the data due to its skewness and heavy tails, and the lack of a unit root with the presence of ARCH effects suggest that it is reasonable to use GARCH models for the indices.

In order to determine the complete model specification to use when estimating the conditional volatility for each log-return series before fitting the copulas, three conditional mean specifications are compared based on the resulting Akaike information criterion (AIC) values from estimating the univariate μ-GARCH(1, 1) models using two 500-day calibration windows. Tables 3 and 4 show that it is not a straightforward choice, since the best fit varies not only between indices but also over time. Nevertheless, the AR(1)-GARCH(1, 1) provides either the best or the second-best fit based on the AIC value in most cases, hence, it is used as the main GARCH model specification in copula modelling.

Table 5 shows that according to the Kolmogorov–Smirnov one-sample test, there is no ground to reject the null hypothesis of the standardised residuals' distributions being equal to the fitted normal inverse Gaussian (NIG) distributions for each index. The NIG distribution provides an adequate fit for the standardised residuals and, therefore, is used to transform the data into uniform margins before estimating the copula models.

Figure 4 (and Figures 19 and 20 in the Appendix) reveals that the standardised residuals which have been transformed into uniform margins after retrieving them from fitted univariate GARCH models to the log-return series display changing relationships over time: it can be seen that the indices are characterised, in general, by stronger correlation during the more volatile calibration window,

Table 2. *Descriptive statistics of logarithmic returns for each index in the portfolio. Statistics' values in bold indicate p-values < 0.05.*

	OMXVGI	OMXRGI	OMXTGI	OMXS30GI	OMXIGI	OMXH25GI	OMXC25GI	OBX
Minimum	-9.834	-22.565	-10.603	-10.978	-7.293	-10.666	-8.266	-12.473
Mean	0.034	0.009	0.020	0.031	0.031	0.026	0.044	0.039
Median	0.035	0.015	0.026	0.085	0.026	0.064	0.087	0.058
Maximum	4.699	12.086	5.197	7.619	6.068	6.665	4.153	6.020
Standard deviation	0.578	1.283	0.803	1.148	1.131	1.104	1.068	1.234
Excess kurtosis	63.595	77.918	34.369	7.455	3.389	8.394	3.311	10.971
Skewness	-3.385	-3.785	-2.501	-0.554	-0.046	-0.690	-0.478	-1.159
Jarque-Bera test	**308637.513**	**462453.145**	**91022.603**	**4286.333**	**867.409**	**5460.906**	**896.160**	**9487.393**
ADF test	**-9.021**	**-18.543**	**-9.986**	**-40.121**	**-15.065**	**-15.620**	**-25.162**	**-13.005**
ARCH(12)	**350.869**	**76.173**	**541.453**	**410.972**	**152.466**	**484.491**	**227.092**	**537.833**
Q(10)	**74.387**	**85.930**	**61.630**	14.681	**23.287**	**21.365**	**19.051**	**29.794**

Table 3. *AIC values for μ-GARCH(1, 1) models with different specifications of the conditional mean model using a 500-day calibration window (2016-12-20–2019-02-06). Number in bold indicates the smallest AIC value for each index.*

	OMXVGI	OMXRGI	OMXTGI	OMXS30GI	OMXIGI	OMXH25GI	OMXC25GI	OBX
ARMA(0,0)	-8.1691	-6.7268	-7.8238	**-6.7345**	-6.3600	**-6.8146**	-6.8641	-6.5778
ARMA(1,0)	**-8.1724**	-6.7707	**-7.8426**	-6.7317	**-6.3561**	-6.8110	-6.8710	-6.5739
ARMA(1,1)	-8.1682	**-6.7712**	-7.8389	-6.7301	-6.3522	-6.8099	**-6.8721**	**-6.5837**

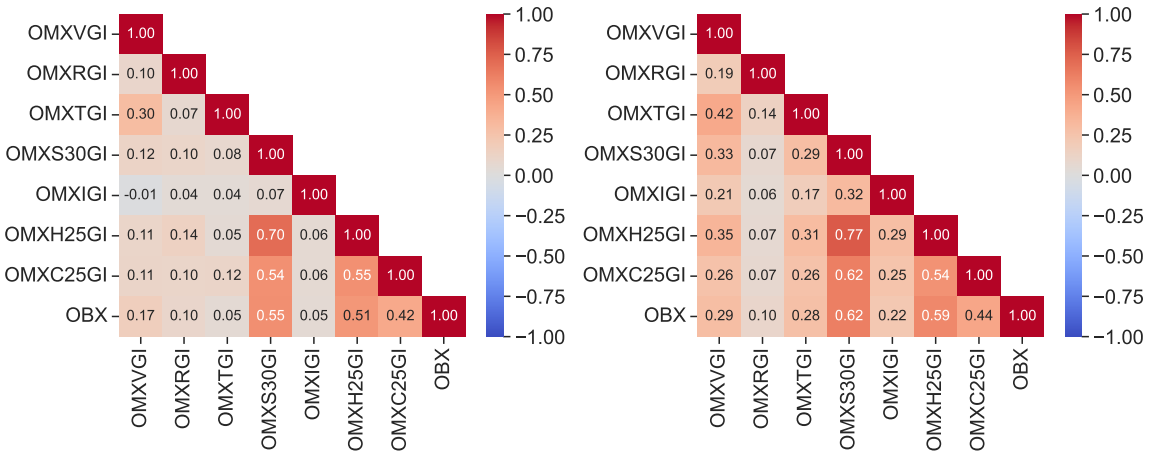
overlapping with the COVID-19 pandemic. Hence, during VaR and ES model estimation, the copulas are reestimated on each date in the rolling-window procedure.

Table 4. *AIC values for μ-GARCH(1, 1) models with different specifications of the conditional mean model using a 500-day calibration window (2019-12-17–2022-02-15). Number in bold indicates the smallest AIC value for each index.*

	OMXVGI	OMXRGI	OMXTGI	OMXS30GI	OMXIGI	OMXH25GI	OMXC25GI	OBX
ARMA(0,0)	-7.5604	-6.8152	-6.7940	-6.0146	-6.2159	-6.1623	-5.9973	-5.8665
ARMA(1,0)	-7.5604	-6.8863	-6.8414	-6.0146	**-6.2212**	-6.1610	**-5.9996**	**-5.8753**
ARMA(1,1)	**-7.5617**	**-6.9037**	**-6.8431**	**-6.0210**	-6.2173	**-6.1705**	-5.9970	-5.8718

Table 5. *Kolmogorov–Smirnov test p-values for standardised residuals from the AR(1)-GARCH(1, 1) models with a NIG distribution fit.*

	OMXVGI	OMXRGI	OMXTGI	OMXS30GI	OMXIGI	OMXH25GI	OMXC25GI	OBX
2016-12-20–2019-02-06	0.827	0.980	0.989	0.989	0.984	0.998	0.898	0.995
2019-12-17–2022-02-15	0.997	0.575	0.942	0.968	0.917	0.983	0.995	0.990



(a) *500-day calibration window: 2016-12-20–2019-02-06*

(b) *500-day calibration window: 2019-12-17–2022-02-15*

Figure 4. *Pearson correlation heatmaps of standardised residuals from GARCH models transformed into uniform margins using fitted univariate normal inverse Gaussian distribution parameters*

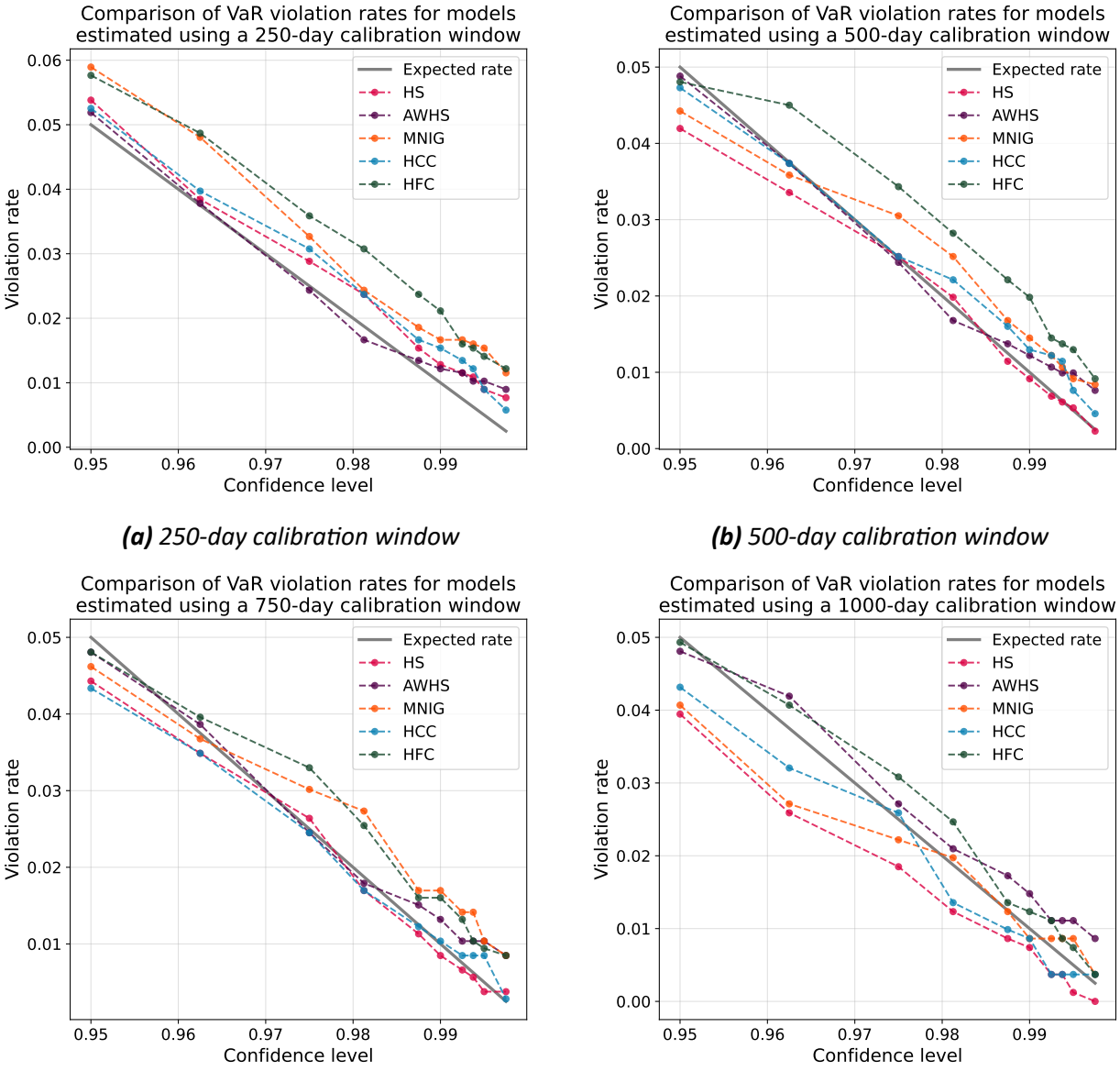
3.2 Estimation results

The VaR levels for which the violation rates are calculated in Figure 5 correspond to the levels required for the multinomial test at 95%, 97.5%, and 99% confidence levels with the number of tested levels $N = 4$ for each q. In general, longer calibration windows w tend to result in, on average, more conservative estimates of VaR for all estimated models, though especially for the basic historical simulation (HS): a longer historical interval is more likely to capture the more extreme observations arising from periods of higher volatility. For the calibration window corresponding to one year (Figure 5a), the age-weighted historical simulation (AWHS) produces VaR estimates with violation rates that are closest to the expected rate for different confidence levels (except for the very highest). The basic historical simulation is comparable in its performance to the age-weighted one, whereas the hierarchical Frank copula (HFC) and multivariate normal inverse Gaussian (MNIG) distribution methods lead to violation rates consistently higher than the expected level; hence, for the latter methods, the one-year calibration window appears to be too short to produce accurate estimates. The HFC model appears to consistently underestimate the risk at confidence levels $q > 0.95$ for different calibration windows, although when w reaches four years, the violation rates for HFC become closer to the expected ones. For $w = 750$ days, HS, AWHS, and HCC approaches seem to match the expected exception rate the most, whereas for $w = 1000$ days HS results in the most conservative estimates, with AWHS, on average, slightly underestimating VaR for different q, and HCC producing the closest failure rates on the conservative side.

Figure 6 shows the estimated 95% confidence level VaR and ES series by the basic historical simulation method up to T = 2024-09-30 for different calibration windows. This approach produces rather static estimates for the considered risk measures which fail to quickly adapt to new information coming from the data. Besides, jumps in risk measure forecasts are observed caused by a more extreme return either entering or leaving the calibration sample and all data points having the same weight regardless of the time they were observed. For all calibration windows the VaR exceptions appear to be clustered together around periods of higher volatility. Similar results are observed for the higher confidence levels, with an increasing q being associated with more pronounced static periods that change suddenly (see Figures 21 and 26).

The estimated age-weighted historical simulation VaR and ES values for the 95% confidence level are shown in Figure 7. In contrast to the basic historical simulation, the forecasts tend to adapt to the changing market environment better, since giving more weight to the recent returns in the sample reduces the impact of old values which might be reflecting information that is not relevant any more. It can be noted that after a period of higher volatility, the AWHS estimates appear to be lagging behind and stay more conservative, which seems to be a distinct feature of the age-weighted historical simulation. This especially stands out for the higher confidence levels (see Figures 22 and 27 in the Appendix). Nevertheless, observation weighing according to their recency does produce estimates which reflect the variability of the sample more accurately.

The risk measure estimates produced by fitting the multivariate normal inverse Gaussian distribution for different calibration windows at $q = 0.95$ are provided in Figure 8. A more complicated estimation method does not seem to produce estimates which are any better than those given by the



(a) *250-day calibration window*

(b) *500-day calibration window*

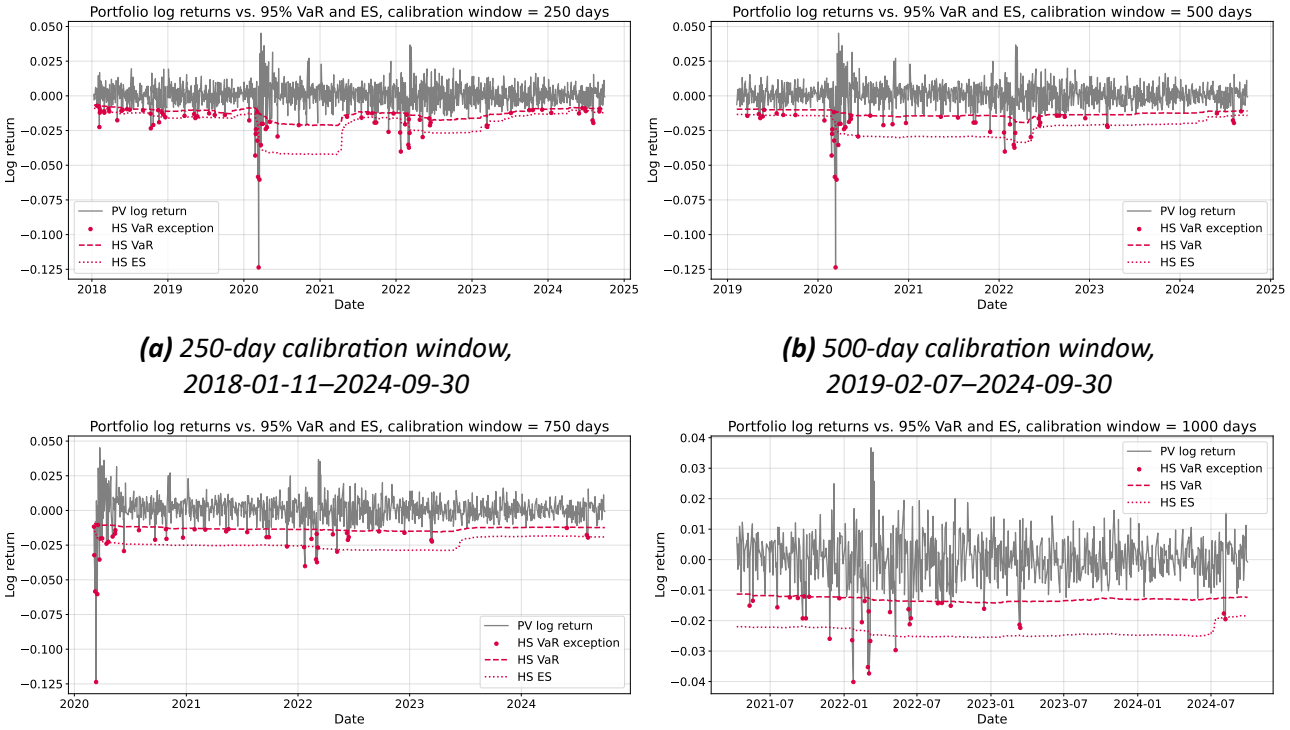
(c) *750-day calibration window*

(d) *1000-day calibration window*

Figure 5. *VaR violation rates vs. VaR levels for all estimated models grouped by calibration window length*

basic historical simulation: although the VaR and ES forecasts are varying, relatively to the observed returns they are still static and comparable to HS estimates for different w. One obvious difference between MNIG and HS lies in the conservativeness of the ES estimates – the HS method for ES is much more influenced by extreme observations in the sample, with this effect being even more pronounced for the larger confidence levels q (see Figures 21 and 26 in the Appendix); however, MNIG ES estimates do not display such gaps between the VaR and ES levels at the same confidence level. The same can be said about risk measure estimates for the higher confidence levels, as per Figures 23 and 28 in the Appendix.

As opposed to the methods discussed previously, the VaR and ES estimates at the 95% confidence level produced by the combination of univariate GARCH models, NIG distribution fit, and hierarchical Clayton copula appear to follow the market tendencies much more closely, both when



(a) *250-day calibration window, 2018-01-11–2024-09-30*

(b) *500-day calibration window, 2019-02-07–2024-09-30*

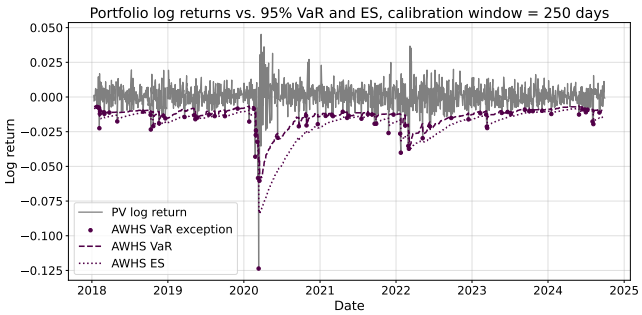
(c) *750-day calibration window, 2020-03-05–2024-09-30*

(d) *1000-day calibration window, 2021-04-09–2024-09-30*

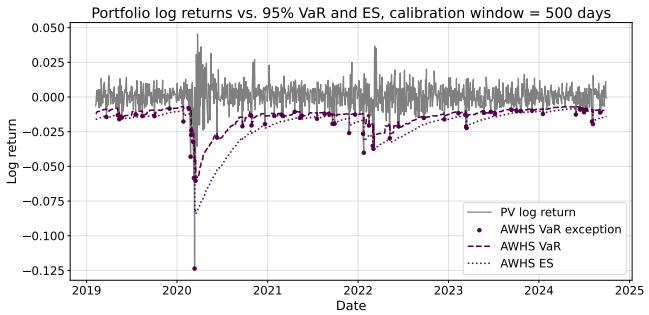
Figure 6. *95% basic historical simulation (HS) VaR and ES forecasts using different calibration windows*

the volatility increases and when a calmer period is observed, as per Figure 9. More conservative estimates seem to be given by models estimated using longer calibration windows, which might be resulting from the fact that it is usually recommended to use a longer calibration sample for the GARCH models. Similar comments are relevant for the higher confidence levels (see Figures 24 and 29 in the Appendix).

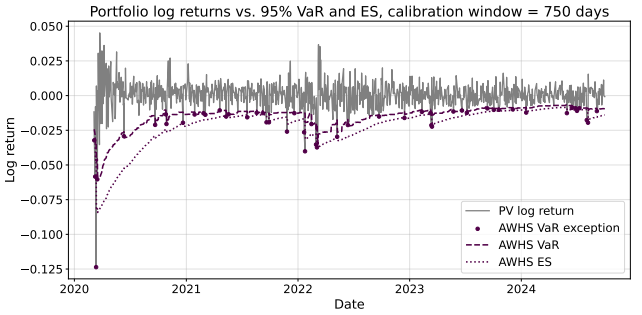
The hierarchical Frank copula model for the VaR and ES estimates at the 95% confidence level follow similar patterns as in the case of the hierarchical Clayton copula, with some periods appearing to produce marginally lower (in absolute value) estimates for the risk values and smaller differences between VaR and ES estimated at the same confidence level, as can be seen in Figure 9. Equivalent remarks could be made about higher confidence levels for the HFC model (see Figures 24 and 29 in the Appendix).



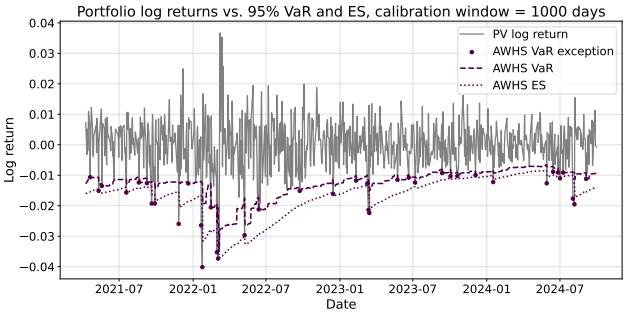
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

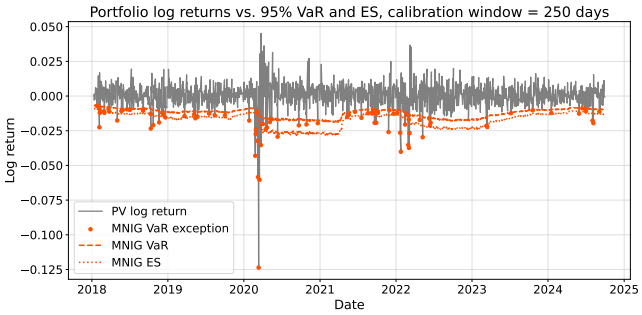


(c) *750-day calibration window, 2020-03-05–2024-09-30*

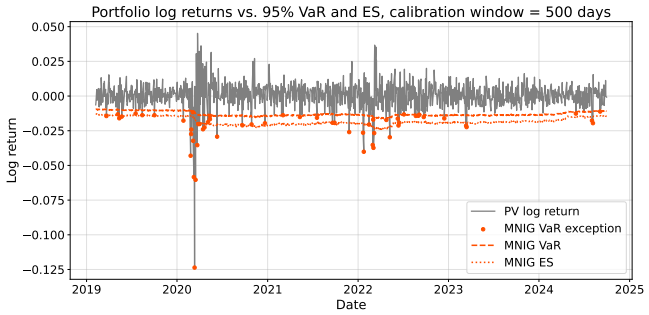


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

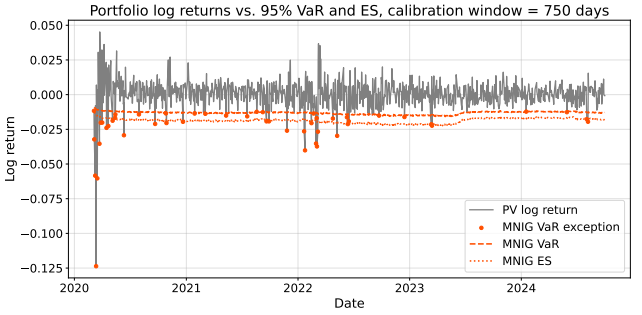
Figure 7. *95% age-weighted historical simulation (AWHS) VaR and ES forecasts using different calibration windows*



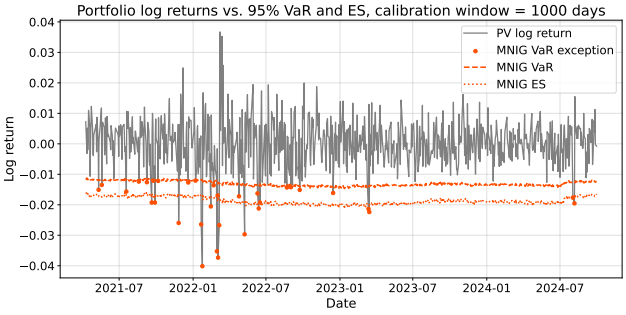
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

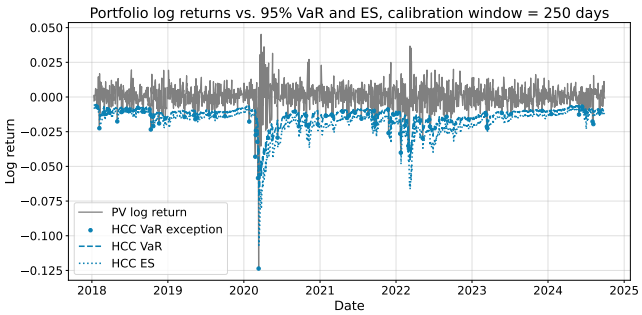


(c) *750-day calibration window, 2020-03-05–2024-09-30*

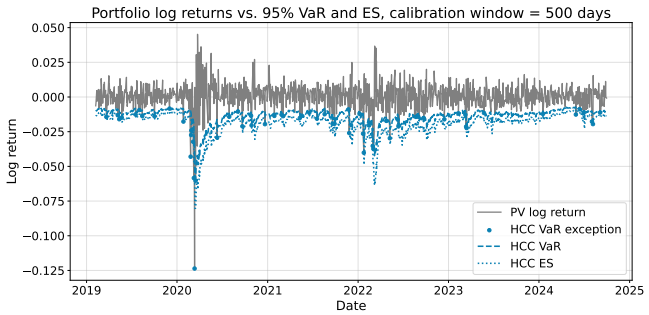


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

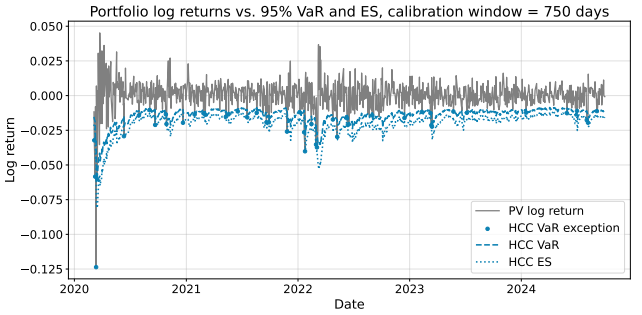
Figure 8. *95% multivariate normal inverse Gaussian distribution (MNIG) VaR and ES forecasts using different calibration windows*



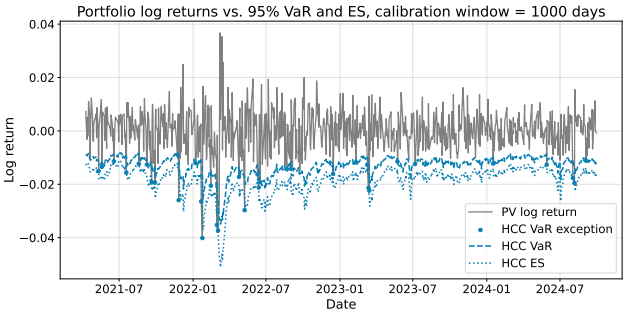
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

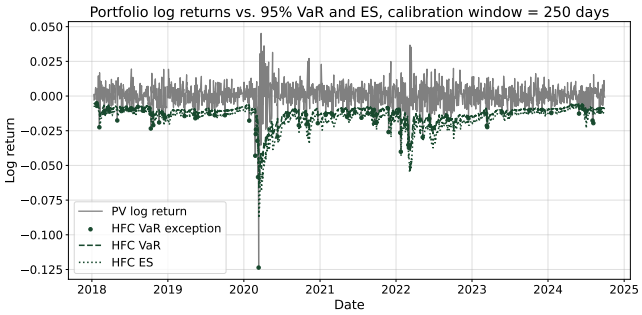


(c) *750-day calibration window, 2020-03-05–2024-09-30*

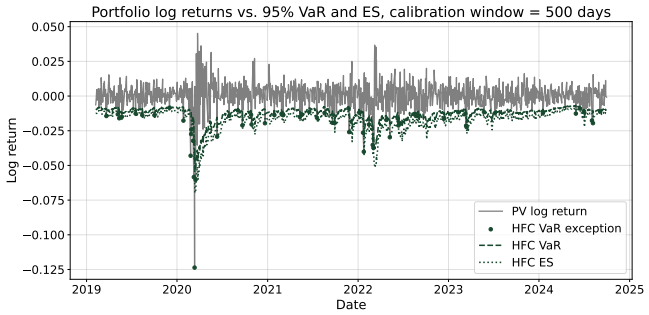


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

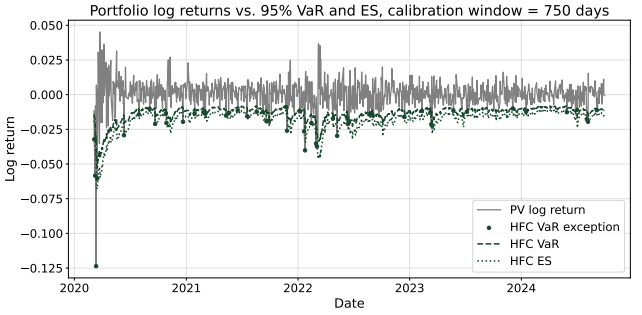
Figure 9. *95% hierarchical Clayton copula (HCC) VaR and ES forecasts using different calibration windows*



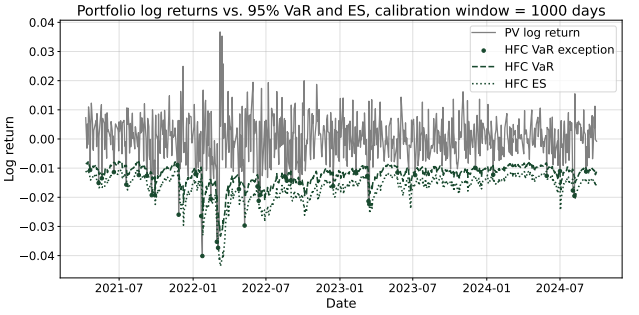
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*



(c) *750-day calibration window, 2020-03-05–2024-09-30*



(d) *1000-day calibration window, 2021-04-09–2024-09-30*

Figure 10. *95% hierarchical Frank copula (HFC) VaR and ES forecasts using different calibration windows*

3.3 Backtesting

The expected numbers of VaR violations for different calibration windows and confidence levels are provided in Table 6.

Table 6. *Expected number of VaR breaches for each calibration window w and confidence level q. T indicates the length of the corresponding backtesting window.*

w (T) \ q	95	97.5	99
250 (1561)	78	39	16
500 (1311)	66	33	13
750 (1061)	53	27	11
1000 (811)	41	20	8

Tables 7, 8, 9, and 10 present the results of traditional backtesting of the estimated VaR models for different w and q. Generally, the Basel Traffic Light test indicates that using a longer calibration window for estimating the VaR models at different confidence levels results in more conservative estimates in the sense that the proportion of VaR violations is similar to or lower than the expected violation rate. HS and AWHS models pass this test for all calibration windows, with HCC also exhibiting good results except for the 99% confidence level with w = 250 days. MNIG and HFC models are in the green zone for all confidence levels only for the highest calibration window. The Kupiec PoF test outcomes tend to overlap with the results of the Basel approach, however, the PoF test would not indicate any issue with the MNIG model calibrated to 500 observations and the HFC model estimated using 750 data points, although the Traffic Light test assigned a yellow zone for at least one q. In contrast to the previous tests, the mixed Kupiec test for which the null hypothesis states the independence of VaR exceptions indicates rejection of H_0 for the majority of the models. Nevertheless, for a calibration window of 1000 days, the AWHS model for $q = 0.95$ with the HCC and HFC models for $q = 0.99$ pass the mixed Kupiec test, as there is no ground to reject the null hypothesis.

Table 7. *VaR traditional backtesting results for models estimated using a 250-day calibration window. Values in bold indicate p-values < 0.05.*

Model	HS			AWHS			MNIG			HCC			HFC		
q (%)	95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99
No. of exceptions	84	45	20	81	38	19	92	51	26	82	48	24	90	56	33
Basel Traffic Light	0.775	0.853	0.890	0.661	0.476	0.840	0.950	0.975	0.995	0.702	0.934	0.983	0.923	0.996	1.000
Kupiec	0.495	0.344	0.285	0.733	0.867	0.404	0.115	0.064	**0.016**	0.649	0.160	**0.048**	0.175	**0.010**	**0.000**
Mixed Kupiec	**0.000**	**0.000**	**0.000**	**0.001**	**0.000**	**0.001**	**0.000**	**0.000**	**0.000**	**0.001**	**0.000**	**0.000**	**0.000**	**0.000**	**0.000**

Table 8. *VaR traditional backtesting results for models estimated using a 500-day calibration window. Values in bold indicate p-values < 0.05.*

Model	HS			AWHS			MNIG			HCC			HFC		
q (%)	95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99
No. of exceptions	55	33	12	64	32	16	58	40	19	62	33	17	63	45	26
Basel Traffic Light	0.099	0.562	0.450	0.455	0.492	0.829	0.187	0.911	0.955	0.356	0.562	0.885	0.404	0.984	0.9995
Kupiec	0.170	0.968	0.755	0.844	0.891	0.438	0.330	0.216	0.126	0.650	0.968	0.302	0.745	**0.040**	**0.002**
Mixed Kupiec	**0.000**	**0.000**	**0.000**	**0.044**	**0.004**	**0.003**	**0.000**	**0.000**	**0.000**	**0.007**	**0.000**	**0.000**	**0.006**	**0.000**	**0.000**

Table 9. *VaR traditional backtesting results for models estimated using a 750-day calibration window. Values in bold indicate p-values < 0.05.*

Model	HS			AWHS			MNIG			HCC			HFC		
q (%)	95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99
No. of exceptions	47	28	9	51	26	14	49	32	18	46	26	11	51	35	17
Basel Traffic Light	0.220	0.661	0.383	0.421	0.511	0.882	0.314	0.878	0.988	0.179	0.511	0.626	0.421	0.956	0.977
Kupiec	0.385	0.774	0.610	0.771	0.918	0.319	0.564	0.297	**0.038**	0.310	0.918	0.905	0.771	0.112	0.070
Mixed Kupiec	**0.000**	**0.000**	**0.000**	**0.021**	**0.001**	**0.000**	**0.000**	**0.000**	**0.000**	**0.002**	**0.000**	**0.000**	**0.005**	**0.000**	**0.000**

As for the traditional ES backtesting results presented in Table 11, all tests are passed by the HCC 750, HCC 1000, HS 500 and HS 750 models. The multinomial test reveals that similarly to estimating VaR, longer calibration periods result in ES estimates which are conservative enough for more models at different confidence levels. Only the AWHS model has at least one non-green MT zone assigned for some confidence level for all calibration windows. The one-sided ER test null hypothesis is rejected for the majority of the models for both 250- and 500-day calibration windows, meaning that they underestimate ES (in absolute value). The two-sided ER test null hypothesis is rejected only for the HS 1000 at all confidence levels, indicating that the ES estimates produced by the model are too conservative, which also corresponds to what was observed in the estimate graphs for different confidence levels (e.g. Figure 21).

For comparative backtesting of risk measures at different confidence levels, the models were selected based on the results of traditional backtesting, i.e., for VaR and each q, models which passed the Basel Traffic Light and Kupiec PoF tests are chosen, whereas for ES and each q, models which pass all three tests (the multinomial test and both specifications of the exceedance residual test) are analysed further.

The traffic light matrix for the VaR with 95% confidence level, shown in Figure 31, reveals that there is not enough evidence for any single estimation method producing more accurate forecasts than other models. On the other hand, the MNIG 1000 model is outperformed by six methods, including all basic historical simulation models (which MNIG appeared to be comparable to visually), MNIG itself with a shorter calibration window of 750 days, and HFC 750. For the 97.5% confidence

Table 10. *VaR traditional backtesting results for models estimated using a 1000-day calibration window. Values in bold indicate p-values < 0.05.*

Model	HS			AWHS			MNIG			HCC			HFC		
q (%)	95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99
No. of exceptions	32	15	6	39	22	12	33	18	7	35	21	7	40	25	10
Basel Traffic Light	0.094	0.140	0.299	0.442	0.701	0.932	0.126	0.356	0.437	0.210	0.621	0.437	0.506	0.878	0.806
Kupiec	0.153	0.214	0.435	0.802	0.702	0.200	0.209	0.602	0.688	0.360	0.871	0.688	0.929	0.305	0.520
Mixed Kupiec	**0.000**	**0.000**	**0.011**	0.194	**0.008**	**0.001**	**0.000**	**0.000**	**0.001**	**0.009**	**0.001**	0.113	**0.046**	**0.001**	0.063

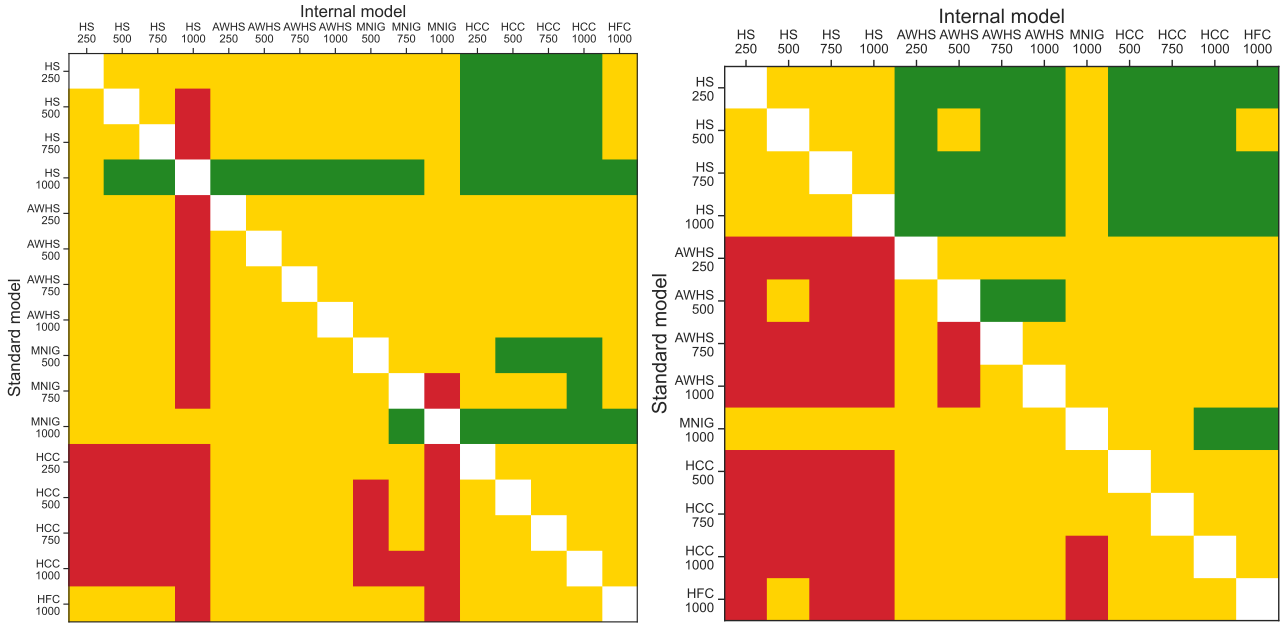
Table 11. *ES traditional backtesting results for different confidence levels. MT denotes the multinomial test, and ER indicates the exceedance residuals test. Values in bold indicate p-values < 0.05.*

Model			HS			AWHS			MNIG			HCC			HFC		
q (%)			95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99	95	97.5	99
250 days	MT		0.515	0.112	0.002	0.925	0.061	0.000	0.172	0.000	0.000	0.398	0.045	0.053	0.002	0.000	0.000
	ER	H_0^{2s}	0.326	0.162	0.242	0.344	0.114	0.146	0.063	0.074	0.098	0.140	0.178	0.246	0.060	0.083	0.149
		H_0^{1s}	0.112	**0.011**	**0.045**	0.118	**0.001**	**0.001**	**0.000**	**0.000**	**0.000**	**0.010**	**0.014**	**0.040**	**0.000**	**0.000**	**0.003**
500 days	MT		0.417	0.849	0.934	0.955	0.162	0.005	0.056	0.260	0.001	0.523	0.082	0.187	0.001	0.014	0.000
	ER	H_0^{2s}	0.639	0.983	0.439	0.409	0.188	0.205	0.088	0.154	0.164	0.247	0.237	0.264	0.089	0.164	0.193
		H_0^{1s}	0.266	0.471	0.150	0.145	**0.011**	**0.005**	**0.001**	**0.005**	**0.003**	**0.048**	**0.026**	**0.029**	**0.000**	**0.006**	**0.012**
750 days	MT		0.571	0.742	0.457	0.657	0.261	0.002	0.240	0.002	0.005	0.763	0.621	0.143	0.134	0.288	0.004
	ER	H_0^{2s}	0.834	0.839	0.695	0.571	0.303	0.320	0.097	0.143	0.181	0.319	0.393	0.370	0.163	0.256	0.288
		H_0^{1s}	0.384	0.549	0.298	0.230	**0.028**	**0.017**	**0.001**	**0.003**	**0.008**	0.071	0.104	0.061	**0.002**	**0.021**	**0.023**
1000 days	MT		0.508	0.728	0.528	0.301	0.415	0.010	0.364	0.454	0.181	0.400	0.171	0.181	0.558	0.397	0.709
	ER	H_0^{2s}	**0.024**	**0.020**	**0.018**	0.286	0.176	0.243	0.268	0.187	0.081	0.952	0.187	0.409	0.147	0.516	0.298
		H_0^{1s}	0.977	0.981	0.984	0.124	0.069	0.099	0.103	0.061	**0.024**	0.506	0.875	0.802	**0.042**	0.232	0.122

level VaR, the results are presented in Figure 11a. In this case, the majority of methods outperform the HS 1000 model, and the HCC models provide more accurate forecasts than all HS models and at least some MNIG models. HCC 1000 stands out as the procedure producing more accurate forecasts than 7 other analysed models. Finally, for the 99% VaR, the HCC models result in better forecasts than all the HS models, however, so do the AWHS models for the most part. The HFC 1000 model also gives more accurate forecasts than most HS procedures and the MNIG 1000 model. It can be concluded that for the higher confidence levels, the AWHS and HCC methods together with the HFC 1000 model produce the best VaR forecasts, with the HCC 1000 procedure collecting the most passes of the comparative backtest.

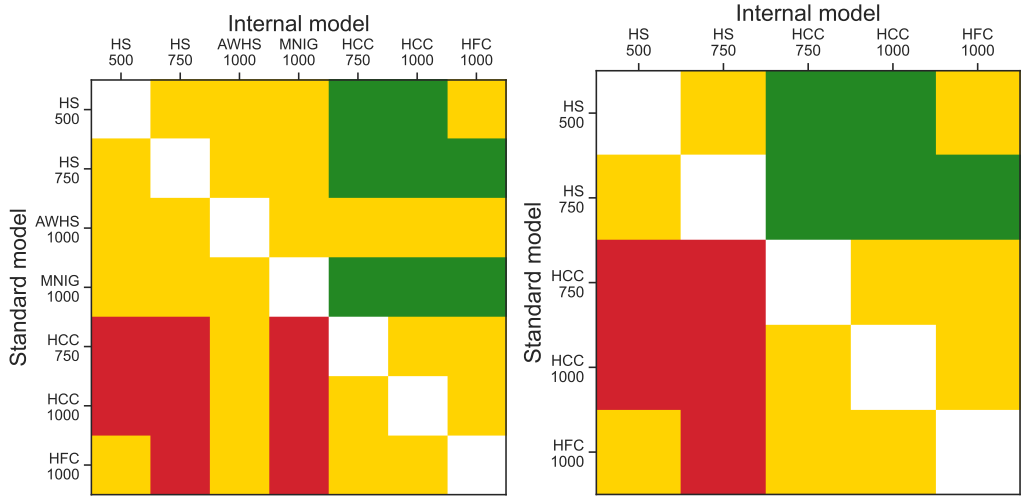
For comparative backtesting of ES, the results appear to be consistent with the VaR analysis outcomes. Figure 32 shows the comparison of models for the 95% ES level. Both MNIG 1000 and HS 750 method forecasts are worse than those produced by AWHS and HCC models, yet the comparison between these models is inconclusive. AWHS with a calibration window of 1000 days outperforms the

AWHS 750 procedure. For the higher confidence levels of ES, the HCC 750 and HCC 1000 again produce more accurate estimates than the HS methods, with HFC 1000 also estimating ES more precisely than HS 750.



(a) *Traffic light matrix for 97.5% VaR* ***(b)*** *Traffic light matrix for 99% VaR*

Figure 11. *Traffic light matrices for 97.5% and 99% VaR comparative backtesting results*



(a) *Traffic light matrix for 97.5% ES* ***(b)*** *Traffic light matrix for 99% ES*

Figure 12. *Traffic light matrices for 97.5% and 99% ES comparative backtesting results*

3.4 Combining forecasts

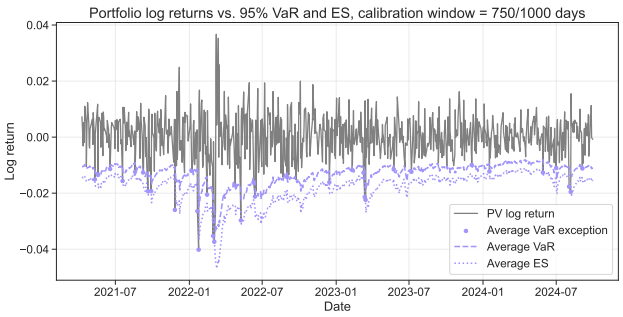
Instead of using forecasts produced by one model, it is also possible to synthesise the predictive information available from several models by combining their forecasts. The AWHS 1000, HCC 750, and HCC 1000 model forecasts are considered, since these methods show consistently good results

across different significance levels q for both VaR and ES. Three combination procedures are explored:

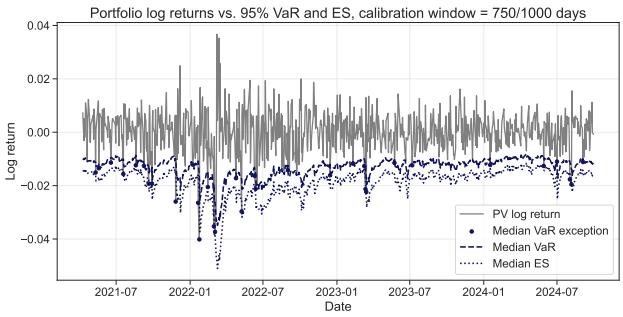
$$\begin{aligned}
\text{Simple average value (Average)} : R^t_{q,Average} &= \frac{R^t_{q,\text{AWHS 1000}} + R^t_{q,\text{HCC 750}} + R^t_{q,\text{HCC 1000}}}{3}, \\
\text{Median value (Median)} : R^t_{q,Median} &= \text{Med}\{R^t_{q,m}\}, \\
\text{Minimum value (Minimum)} : R^t_{q,Minimum} &= \text{Min}\{R^t_{q,m}\}, \\
m &\in \{\text{AWHS 1000, HCC 750, HCC 1000}\}
\end{aligned}$$

with $R^t_{q,m}$ corresponding to either VaR or ES value produced by the m model at confidence level q and time t. The backtesting period corresponds to that of models estimated using w = 1000, since there is not enough data for the AWHS 1000 and HCC 1000 to use the same forecasting dates as for HCC 750.

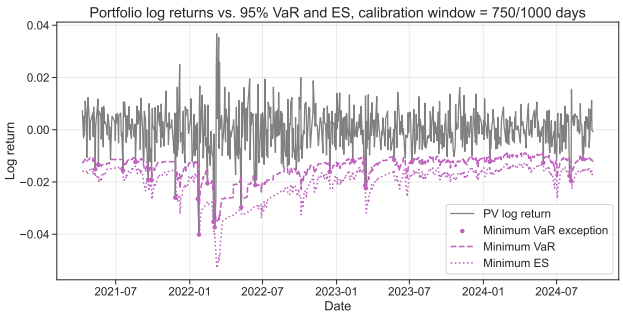
Figure 13 shows the resulting risk measure forecasts for the three combination strategies at q = 0.95. All three methods tend to reflect the variation produced by the HCC models. Averaging results in slightly smoothed risk measure values over time, whereas the median mostly preserves the patterns observed for the HCC 750/1000 models. The general dynamics of forecasts produced by taking the minimum are similar to the averaged and median cases, however, it leads to more conservative values during periods when volatility decreases, which is a direct impact of the AWHS 1000 model estimates – a long calibration window for the AWHS estimation procedure causes lagging in volatility reduction, albeit with a smaller effect than for the basic historical simulation. Similar conclusions can be derived for 97.5% and 99% confidence levels (see Figures 33 and 34).



(a) *95% VaR and ES produced by averaging forecasts*



(b) *95% VaR and ES produced by the median of forecasts*



(c) *95% VaR and ES produced by the minimum of forecasts*

Figure 13. *95% VaR and ES from combining forecasts produced by AWHS (w=1000), HCC (w=750), HCC (w=1000) models*

Table 12 reflects the traditional backtesting results of the combination methods for VaR at different confidence levels q. All methods pass the Basel Traffic Light test, implying that the overall violation rate of the models is conservative enough. However, the Kupiec test indicates that the VaR forecasts produced by taking the minimum of the forecasts from considered models are inaccurate and, based on the observed number of exceptions, cause risk overestimation at q equal to 95% and 99%. The mixed Kupiec test again indicates that the models tend to be characterised by dependent VaR exceedances, although for the 99% confidence level of the median combination, there is no sufficient evidence for rejecting the null hypothesis of independent VaR violations.

Table 12. *VaR traditional backtesting results for combinations of AWHS (w=1000), HCC (w=750), HCC (w=1000) model forecasts for the backtesting window of 811 observations. Values in bold indicate p-values < 0.05.*

Model	Average			Median			Minimum		
q (%)	95	97.5	99	95	97.5	99	95	97.5	99
No. of exceptions	34	18	5	33	20	7	26	14	3
Basel Traffic Light	0.165	0.356	0.180	0.126	0.534	0.437	0.009	0.092	0.039
Kupiec	0.278	0.602	0.237	0.209	0.951	0.688	**0.012**	0.135	**0.038**
Mixed Kupiec	**0.035**	**0.013**	**0.020**	**0.015**	**0.001**	0.102	**0.001**	**0.001**	**0.006**

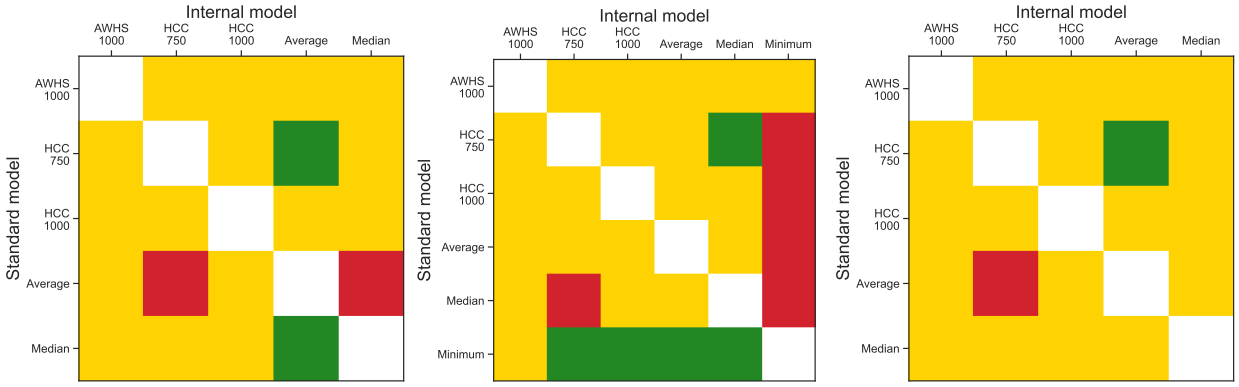
The results for traditional backtesting of ES for the combination strategies are provided in Table 13. Both averaging and median methods produce correct ES estimates at different confidence levels according to both the multinomial and exceedance residual tests. Hence, also considering the results for VaR traditional backtesting, for comparative backtesting the average and median procedures are included for all levels, and the minimum forecast procedure – only the confidence level of 97.5%.

Table 13. *ES traditional backtesting results for combinations of AWHS (w=1000), HCC (w=750), HCC (w=1000) model forecasts for the backtesting window of 811 observations. Values in bold indicate p-values < 0.05.*

Model		Average			Median			Minimum		
q (%)		95	97.5	99	95	97.5	99	95	97.5	99
MT		0.727	0.575	0.600	0.318	0.315	0.342	0.175	0.550	0.443
ER	H_0^{2s}	0.971	0.825	0.397	0.992	0.186	0.496	0.785	0.308	0.331
	H_0^{1s}	0.468	0.574	0.184	0.481	0.875	0.743	0.596	0.822	**0.039**

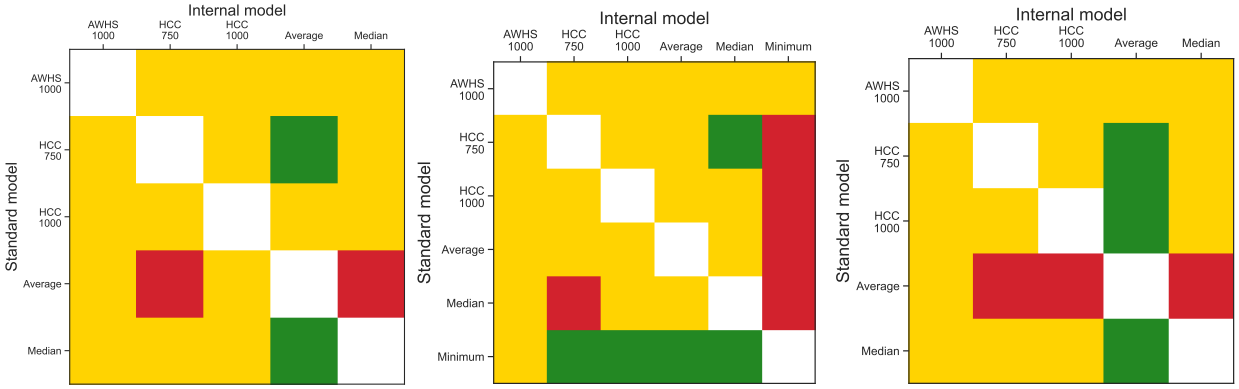
The traffic light matrices for comparative VaR and ES backtesting of the combined forecasts alongside the models they were calculated from are given in Figures 14 and 15. It can be seen that taking the minimum of AWHS 1000, HCC 750, and HCC 1000 model estimates results in forecasts which are less accurate than the ones produced by HCC 750 and HCC 1000 on their own or by the average or median combinations of considered models for $q = 0.975$ in both VaR and ES. On the other hand, both the average and median combinations produce forecasts that are better than from the HCC 750 method at different levels for VaR. In case of ES, the average combination stands out for

$q = 0.99$, as it outperforms the HCC 750, HCC 1000, and median methods. Nevertheless, AWHS 1000 and HCC 1000 models produce forecasts which are not statistically significantly outperformed by the new combination models across different confidence levels for both VaR and ES.



(a) *Traffic light matrix for 95% VaR* ***(b)*** *Traffic light matrix for 97.5% VaR* ***(c)*** *Traffic light matrix for 99% VaR*

Figure 14. *Traffic light matrices for 95%, 97.5%, and 99% VaR comparative backtesting results for combined forecasts*



(a) *Traffic light matrix for 95% ES* ***(b)*** *Traffic light matrix for 97.5% ES* ***(c)*** *Traffic light matrix for 99% ES*

Figure 15. *Traffic light matrices for 95%, 97.5%, and 99% ES comparative backtesting results for combined forecasts*

Although no single combination method is evidently better than all other combination procedures and models used for producing the combined forecasts for both VaR and ES at different q, even taking the simple average or median of forecasts produced by well-performing models gives promising results and indicates a potential avenue for further exploration of risk measure forecasting.

3.5 Incremental VaR analysis

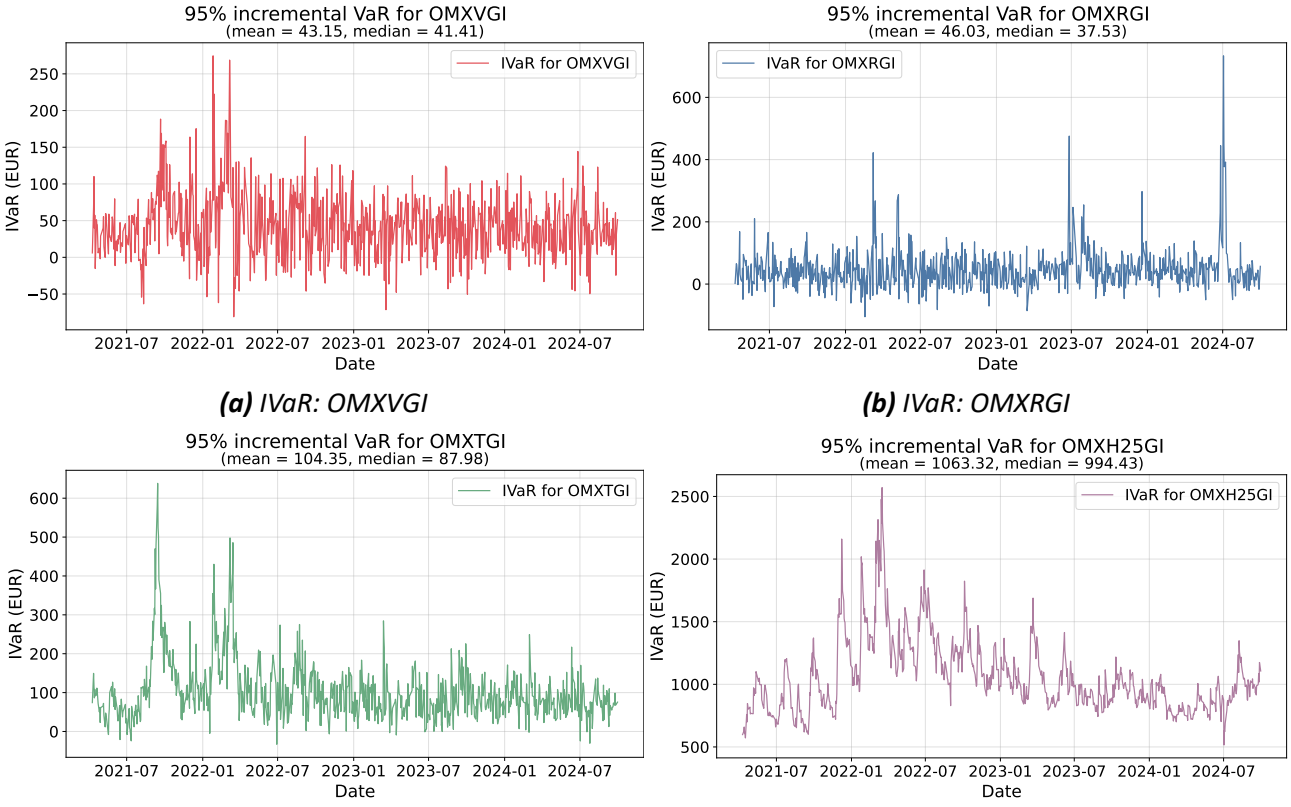
For the incremental VaR (IVaR) analysis, the hierarchical Clayton copula model estimated using a calibration window of 1000 days (HCC (1000)) is used, since it displayed consistently good backtesting results for both VaR and ES across different confidence levels. In order to calculate IVaR, after fitting the univariate margins, the hierarchical Clayton copula was estimated eight times (excluding one of

the indices from the portfolio each time) for every iteration of the rolling-window procedure. Since throughout the thesis the forecasted VaR for time t is expressed in terms of log returns, to get values in euros, the following transformation is applied:

$$\text{VaR}^t_{q,\text{ excl. } i}\,(\text{EUR}) = \text{e}^{\text{VaR}^t_q} \cdot PV_{\text{excl. } i,t-1} - PV_{\text{excl. } i,t-1},$$

where i stands for the excluded index, and $PV_{\text{excl. } i,t-1}$ indicates the value of the relevant portfolio at time $t-1$ (which is the same as the one used during estimation to produce the log-return scenarios). When IVaR is calculated, the difference is taken between absolute values of VaR so that a positive IVaR would indicate a reduction of risk in the portfolio due to exclusion of a given index, whereas a negative IVaR would imply an increase in portfolio VaR without the excluded asset.

Figure 16 depicts the 95% IVaR series for the portfolio indices originally measured in euros, i.e., the Baltic indices and Finland's OMXH25GI. On average, the exclusion of each index leads to a decrease in financial risk of the portfolio, with OMXH25GI (Figure 16d) providing the largest reduction. This is expected, since all indices tend to be positively correlated throughout time, and due to equal weighting of the assets in the portfolio in terms of units, OMXH25GI has the largest weight in absolute value. Similar conclusions apply for 97.5% and 99% IVaR values (see Figures 35 and 37 in the Appendix).



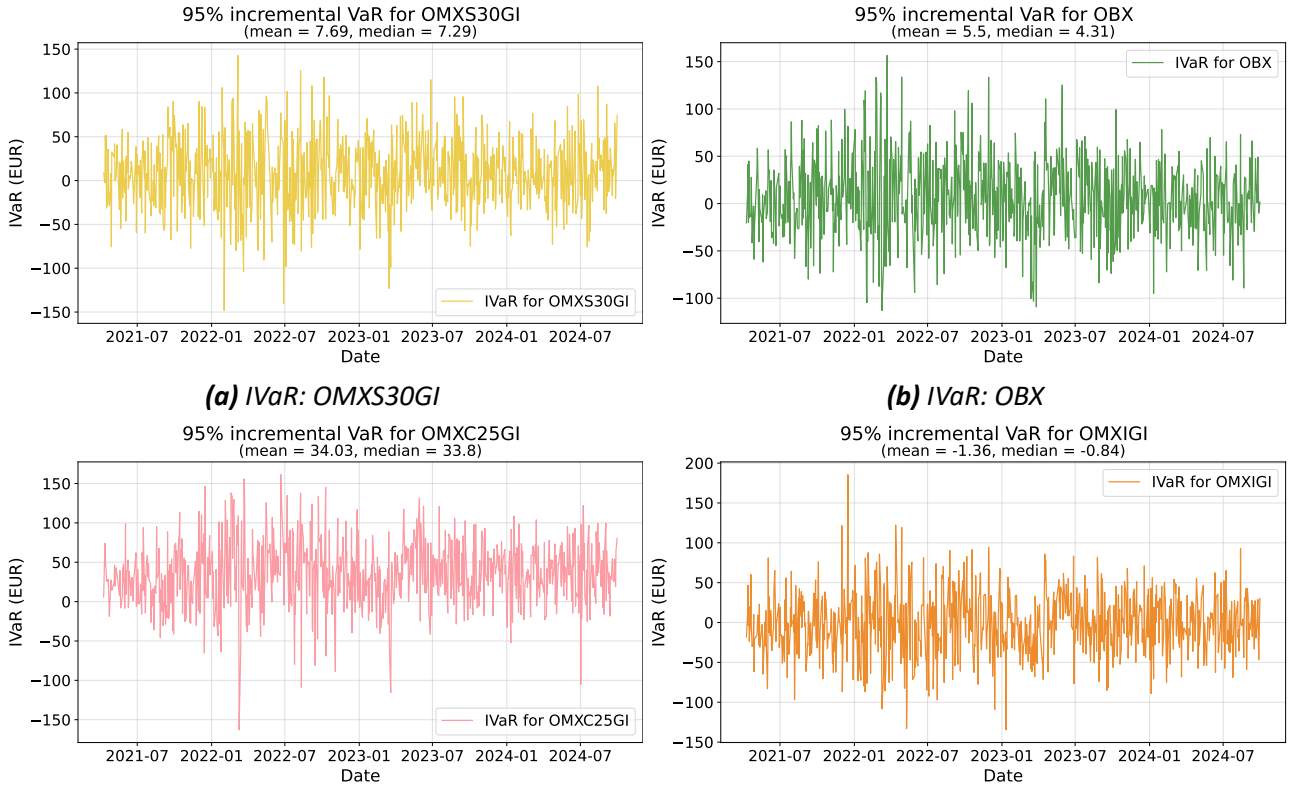
(a) *IVaR: OMXVGI* ***(b)*** *IVaR: OMXRGI*

(c) *IVaR: OMXTGI* ***(d)*** *IVaR: OMXH25GI*

Figure 16. *95% incremental VaR for the Baltic indices and OMXH25GI produced by the hierarchical Clayton copula model estimated on a calibration window of 1000 days*

The 95% IVaR series for the Scandinavian indices and Iceland's OMXIGI are shown in Figure 17. In this case, the exclusion of the OMXS30GI, OBX, and OMXC25GI indices from the portfolio

has a smaller impact on its risk, with IVaR fluctuating around zero over time, though, on average, having a positive effect on the portfolio's 95% VaR. However, in contrast to the Scandinavian indices, removing OMXIGI from the portfolio results in an average increase of risk. OMXIGI was the only asset whose uniform margins of standardised residuals from the calmer period displayed not only weakly positive, but also very small negative correlation with one of the other indices (see Figure 4a). Hence, this index stands out as a potential hedge in the portfolio at hand. Similar results can be observed for 97.5% and 99% confidence level IVaR series (see Figures 36 and 38 in the Appendix).



(a) *IVaR: OMXS30GI*

(b) *IVaR: OBX*

(c) *IVaR: OMXC25GI*

(d) *IVaR: OMXIGI*

Figure 17. *95% incremental VaR for the Scandinavian indices and OMXIGI produced by the hierarchical Clayton copula model estimated on a calibration window of 1000 days*

Results and conclusions

An extensive analysis of the financial risk by using Value-at-Risk and Expected Shortfall for a portfolio consisting of three Baltic and five Nordic stock indices during the period spanning from 2016-12-20 to 2024-09-30 is performed. In total, 20 models are estimated by using five types of estimation procedures for calibration windows corresponding to one, two, three, and four years. It is found that in general, using longer calibration windows for estimating the risk measures leads to more conservative forecasts that tend to pass the different traditional backtests for both VaR and ES more frequently. For each risk measure, three backtesting strategies are applied. The Basel Traffic Light and the Kupiec Proportion of Failures tests have similar outcomes across the different VaR confidence levels and model calibration windows. The mixed Kupiec test, assessing the independence of VaR violations, is passed only for some confidence levels by the age-weighted historical simulation, hierarchical Clayton and Frank copulas for the calibration window of 1000 days. When also considering traditional ES backtesting outcomes, the HCC procedure stands out as one of the best-performing methods, producing adequate estimates of both VaR and ES across different confidence levels, which is similar to the results achieved by Byun and Song on a smaller portfolio of equities [8]. Comparative backtesting also reveals the appropriateness of the HCC method for different confidence levels and both risk measures, although the age-weighted historical simulation approach also generally results in acceptable risk measure estimates. The combination of forecasts based on averaging, taking the minimum or the median of estimates produced by the best models (as indicated by backtesting results) shows that even just taking the simple average of the most appropriate forecasting procedures might lead to superior forecasts than of the individual models, especially for the highest confidence level of 99% for ES. Nevertheless, the minimum forecast combination strategy results in estimates that are too conservative, which would not be an issue from the regulatory perspective, but is not beneficial to the financial institution that estimates the risk measures for its capital requirements. Finally, the application of incremental VaR analysis is also shown in this thesis for the portfolio at hand, indicating that the risk tends to move in the same direction for most of the assets, except for OMXIGI, which stands out as a marginal hedge in the portfolio by displaying, on average, negative IVaR values over time.

Several directions could be explored in order to extend the current analysis. It is found that, in general, the MNIG distribution produces similar results to the basic historical simulation. It could be the case that applying a similar volatility filter as in the case of copula modelling for the univariate margins before estimating the multivariate NIG distribution would result in estimates that are more closely related to the market movements. Different forecast combination strategies can also be employed for the analysed methods and portfolio, e.g., weighting methods based on scoring functions. In addition, a portfolio consisting of more complex financial instruments could be assessed in terms of VaR and ES by using the methods explored in this thesis. In particular, the application of hierarchical Clayton copula with GARCH-filtered standardised return residuals with a fitted normal inverse Gaussian distribution as the copula's margins for VaR and ES displays good performance in terms of both goodness of fit and high forecast accuracy that is comparable or superior to other models.

References and sources

[1] History of the Basel Committee. (2014). https://www.bis.org/bcbs/history.htm.

[2] Artzner, P., Delbaen, F., Eber, J.-M. Heath, D. (1999). Coherent measures of eisk. *Mathematical Finance, 9*, 203-228. https://doi.org/10.1111/1467-9965.00068.

[3] Danielsson, J. , Jorgensen, B. N., Mandira, S., Samorodnitsky, G., de Vries, C. G. (2005). Subadditivity re–examined: the case for value-at-risk. *Financial Markets Group Discussion Papers (549)*. Financial Markets Group. https://www.fmg.ac.uk/publications/discussion-papers/subadditivity-re-examined-case-value-risk.

[4] Acerbi, C., Székely, B. (2014). BACKTESTING EXPECTED SHORTFALL Introducing three model-independent, non-parametric back-test methodologies for Expected Shortfall. https://www.msci.com/documents/10199/22aa9922-f874-4060-b77a-0f0e267a489b.

[5] Bellini, F, Di Bernardino, E. (2015). Risk management with Expectiles. *European Journal of Finance, 23*(6), 487–506. https://doi.org/10.1080/1351847X.2015.1052150

[6] Santos, S. S., Righi, M. B., Horta, E. de O. (2024). The limitations of comonotonic additive risk measures: A literature review (arXiv:2212.13864). arXiv. https://doi.org/10.48550/arXiv.2212.13864.

[7] European Banking Authority. (2024). *EBA report: Results from the 2023 market risk benchmarking exercise*. https://www.eba.europa.eu/sites/default/files/2024-02/9cb044f7-9508-41bc-9db4-ce50b8dc317c/EBA%20report%20results%20from%20the%202023%20market%20risk%20benchmarking%20exercise.pdf.

[8] Byun, K., Song, S. (2021). Value at Risk of portfolios using copulas.*Communications for Statistical Applications and Methods, 28*(1), 59–79. https://doi.org/10.29220/CSAM.2021.28.1.059.

[9] Lu, X. F., Lai, K. K., Liang, L. (2014). Portfolio value-at-risk estimation in energy futures markets with time-varying copula-GARCH model. *Annals of Operations Research, 219*, 333–357. https://doi.org/10.1007/s10479-011-0900-9.

[10] Radivojevic, N., Ćurčić, N., Vukajlović, D. (2017). Hull-White's value at risk model: case study of Baltic equities market. *Journal of Business Economics and Management, 18(5)*, 1023-1041. https://doi.org/10.3846/16111699.2017.1357049.

[11] Jurgilas, D. (2012). Measuring and forecasting Value at Risk for Nordic and Baltic stock indexes.[Manuscript]: Master Thesis: economics. Vilnius, ISM University of Management and Economics. https://www.nasdaqbaltic.com/files/vilnius/inv_sv/Master_Thesis_Darius_Jurgilas.pdf.

[12] Le, T. H. (2024). Forecasting value-at-risk and expected shortfall in emerging market: Does forecast combination help? *The Journal of Risk Finance, 25*(1), 160–177. https://doi.org/10.1108/JRF-06-2023-0137.

[13] European Central Bank. (2024). *ECB guide to internal models.* https://www.bankingsupervision.europa.eu/ecb/pub/pdf/ssm.supervisory_guides202402_internalmodels.en.pdf.

[14] Hyndman, R. J., Fan, Y. (1996). Sample quantiles in statistical packages. *The American Statistician, 50*(4), 361-365. https://doi.org/10.1080/00031305.1996.10473566.

[15] Hull, J. C. (2022). *Options, futures, and other derivatives*, 11th Edition, Global Edition. Pearson.

[16] Øigård, T. A.; Hanssen, A.; Hansen, R. E. (2004). The Multivariate Normal Inverse Gaussian distribution: EM-estimation and analysis of synthetic aperture sonar data. *12th European Signal Processing Conference*, Vienna, Austria, 2004, pp. 1433-1436. https://ieeexplore.ieee.org/document/7079951.

[17] McNeil, A. J., Frey, R., Embrechts, P. (2005). *Quantitative risk management: Concepts, techniques, and tools*. Princeton University Press, Princeton.

[18] Pfaff, B., McNeil, A. (2020). QRM: Provides R-language code to examine quantitative risk management concepts. [R package version 0.4-31]. https://CRAN.R-project.org/package=QRM.

[19] Weibel, M., Luethi, D., Breymann, W. (2023). ghyp: Generalized hyperbolic distribution and its special cases. [R package version 1.6.4]. https://CRAN.R-project.org/package=ghyp.

[20] Scott, D. (2023). GeneralizedHyperbolic: The generalized hyperbolic distribution. [R package version 0.8-4]. https://CRAN.R-project.org/package=GeneralizedHyperbolic.

[21] Nelsen, R. B. (n.d.) *Dependence modeling with Archimedean copulas*. https://web.pdx.edu/~fountair/seminar/arch.pdf.

[22] Boucher, J. P., Denuit, M. Guillen, M. (2008). Models of insurance claim counts with time dependence based on generalisation of Poisson and negative binomial distributions. *Variance, 2*(1), 135–162. Casualty Actuarial Society. https://www.casact.org/sites/default/files/2021-07/Models-Insurance-Claims-Boucher-Denuit-Guillen.pdf.

[23] Okhrin, O., Ristig, A. (2014). Hierarchical Archimedean Copulae: The HAC package. *Journal of Statistical Software, 58*(4), 1-20. https://doi.org/10.18637/jss.v058.i04.

[24] Okhrin, O., Okhrin, Y., Schmid, W. (2013). On the structure and estimation of hierarchical Archimedean copulas. *Journal of Econometrics, 173*(2), 189-204. https://doi.org/10.1016/j.jeconom.2012.12.001.

[25] Nolde, N., Ziegel, J. F. (2016). Elicitability and backtesting: Perspectives for banking regulation. *The Annals of Applied Statistics, 11*(4), 1833–1874. https://doi.org/10.1214/17-AOAS1041.

[26] Basle Committee on Banking Supervision. (1996). *Supervisory framework for the use of "Backtesting" in conjunction with the internal models approach to market risk capital requirements.* www.bis.org/publ/bcbs22.htm.

[27] Costanzino, N., Curran, M. (2018). A simple traffic light approach to backtesting Expected Shortfall. *Risks, 6*(1), 2. https://doi.org/10.3390/risks6010002.

[28] Basel Committee on Banking Supervision. (n.d.). Internal models approach: backtesting and P&L attribution test requirements. *Basel Framework.* https://www.bis.org/basel_framework/chapter/MAR/32.htm.

[29] Zhang, Y., Nadarajah, S. (2017). A review of backtesting for value at risk. *Communications in Statistics - Theory and Methods*, 1-24. https://doi.org/10.1080/03610926.2017.1361984.

[30] Holton, G. A. (2013). *Value-at-Risk: Theory and Practice*. Second edition. www.value-at-risk.net.

[31] Haas, M. (2001). *New methods in backtesting*. https://www.ime.usp.br/~rvicente/risco/haas.pdf.

[32] Kratz, M., Lok, Y. H., McNeil, A. J. (2018). Multinomial VaR backtests: A simple implicit approach to backtesting expected shortfall. *Journal of Banking & Finance, 88*, 393–407. https://doi.org/10.1016/j.jbankfin.2018.01.002.

[33] McNeil, A.J., Frey, R. (2000). Estimation of tail-related risk measures for heteroscedastic financial time series: an extreme value approach. *Journal of Empirical Finance, 7*(3-4), 271-300. https://doi.org/10.1016/S0927-5398(00)00012-8.

[34] Bayer, S., Dimitriadis, T. (2022). Regression-based Expected Shortfall backtesting. *Journal of Financial Econometrics, 20*(3), 437–471. https://doi.org/10.1093/jjfinec/nbaa013 (Supplementary Appendix).

[35] Efron, B., Tibshirani, R. J. (1994). *An introduction to the bootstrap*. Chapman & Hall. https://doi.org/10.1201/9780429246593.

[36] Spring, K. (2021). Backtesting the Expected Shortfall. *Junior Management Science, 6*(3), 590-636. https://doi.org/10.5282/jums/v6i3pp590-636.

[37] Diks, C., Panchenko, V., van Dijk, D. (2011). Likelihood-based scoring rules for comparing density forecasts in tails. *Journal of Econometrics, 163*(2), 215-230. https://doi.org/10.1016/j.jeconom.2011.04.001.

[38] Fissler, T., Ziegel, J. F., Gneiting, T. (2015). Expected Shortfall is jointly elicitable with Value at Risk - Implications for backtesting. Papers, arXiv.org. https://doi.org/10.48550/arXiv.1507.00244.

[39] Dendoncker, V., Lebègue, A. (2019). Comparative backtesting of the Expected Shortfall [White paper]. *Reacfin*. https://www.reacfin.com/index.php/2019/10/28/comparative-backtesting-of-the-expected-shortfall/.

[40] Koziorowska, K. (2012). The marginal, incremental and component Value at Risk. *Zeszyty Naukowe, 222*, 135–150. https://bazekon.uek.krakow.pl/zeszyty/171217959.

[41] Nasdaq Baltic. (n.d.). *About indexes*. https://nasdaqbaltic.com/market-information/about-indexes/.

[42] Nasdaq. (n.d.). *Overview for OMXRGI*. https://indexes.nasdaqomx.com/Index/Overview/OMXRGI.

[43] FTSE Russell. (n.d.). *Industry Classification Benchmark (ICB)*. London Stock Exchange Group. https://www.lseg.com/en/ftse-russell/industry-classification-benchmark-icb

[44] Nasdaq. (n.d.). *Breakdown for OMXRGI*. https://indexes.nasdaqomx.com/Index/Breakdown/OMXRGI

[45] Nasdaq. (n.d.). *Overview for OMXTGI*. https://indexes.nasdaqomx.com/Index/Overview/OMXTGI.

[46] Nasdaq. (n.d.). *Breakdown for OMXTGI*. https://indexes.nasdaqomx.com/Index/Breakdown/OMXTGI.

[47] Nasdaq. (n.d.). *Overview for OMXVGI*. https://indexes.nasdaqomx.com/Index/Overview/OMXVGI.

[48] Nasdaq. (n.d.). *Breakdown for OMXVGI*. https://indexes.nasdaqomx.com/Index/Breakdown/OMXVGI.

[49] Nasdaq. (n.d.). *OMXC25 fact sheet*. https://indexes.nasdaqomx.com/docs/FS_OMXC25.pdf.

[50] Nasdaq. (n.d.). *Overview for OMXH25GI*. https://indexes.nasdaqomx.com/Index/Overview/OMXH25GI.

[51] Morningstar. (n.d.). *NASDAQ OMX Helsinki 25 PR EUR (OMXH25) portfolio*. https://www.morningstar.com/indexes/xhel/omxh25/portfolio

[52] Nasdaq. (n.d.). *Overview for OMXIGI*. https://indexes.nasdaqomx.com/Index/Overview/OMXIGI.

[53] Nasdaq. (n.d.). *Breakdown for OMXIGI*. https://indexes.nasdaqomx.com/Index/Breakdown/OMXIGI.

[54] Nasdaq. (n.d.). *OMXS30 fact sheet*. https://indexes.nasdaqomx.com/docs/FS_OMXS30.pdf.

[55] Euronext. (n.d.). *OBX Total Return Index*. https://live.euronext.com/en/product/indices/NO0000000021-XOSL/market-information

[56] Nasdaq Nordic. (n.d.). *Nasdaq Nordic website*.https://www.nasdaqomxnordic.com/

Appendix

1 Graphs

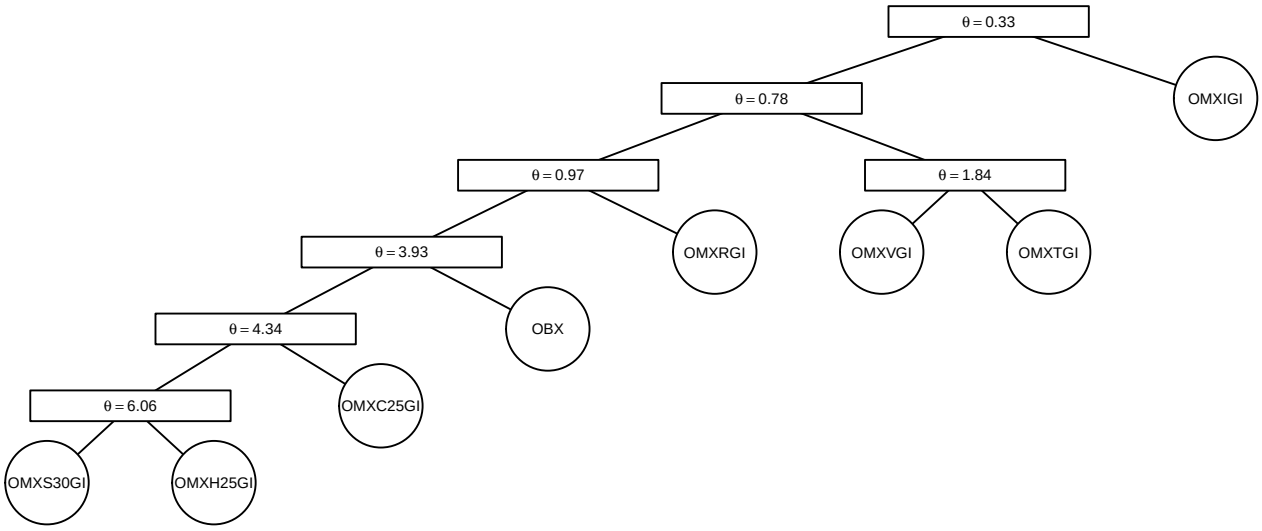


Figure 18. *Example of a partially nested Archimedean copula tree produced by fitting the hierarchical Frank copula to the uniform margins from standardised AR(1)-GARCH(1, 1) residuals from the first 500 observations in the dataset*

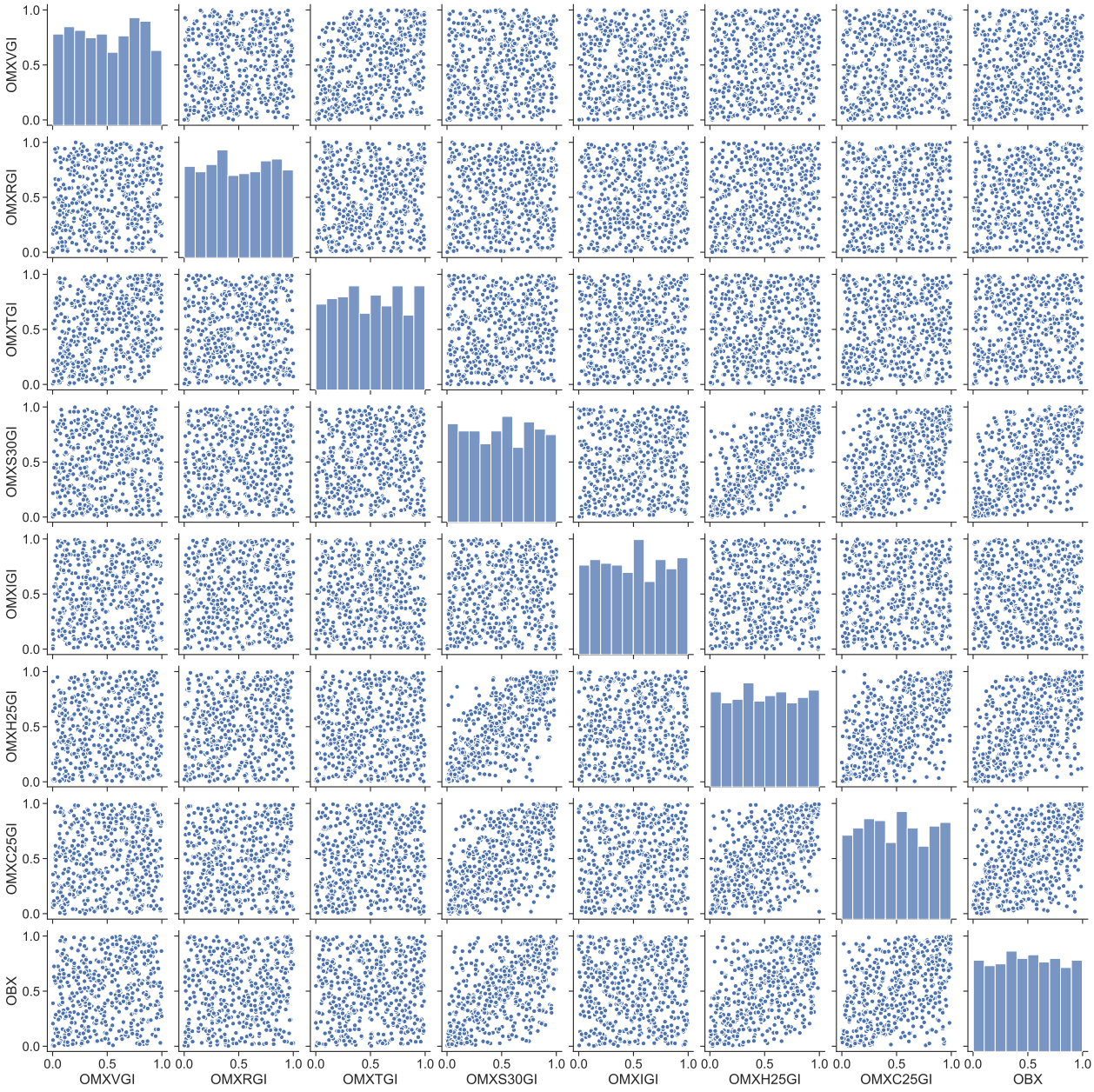


Figure 19. *Scatterplots of standardised residuals from univariate GARCH models transformed into uniform margins (500-day calibration window: 2016-12-20–2019-02-06)*

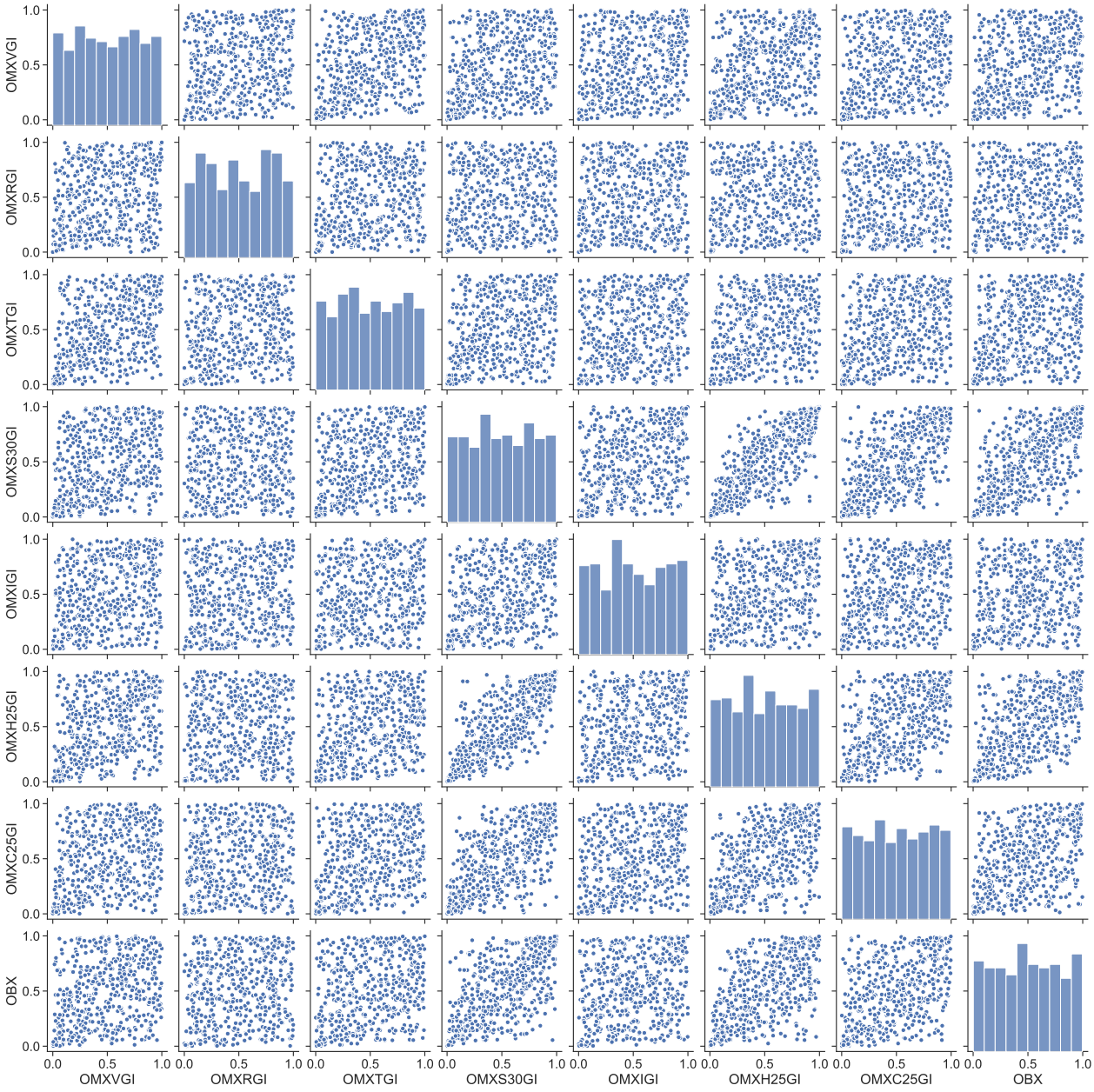
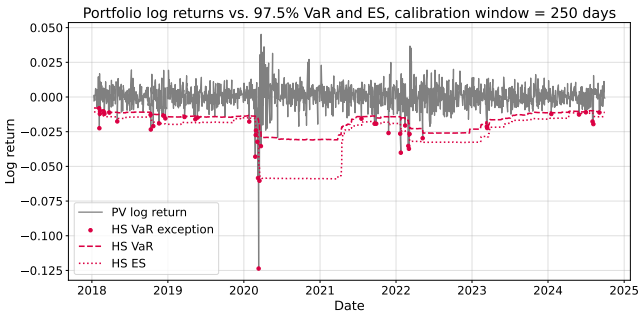
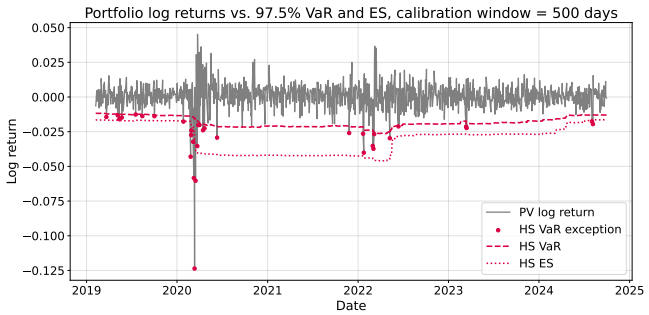


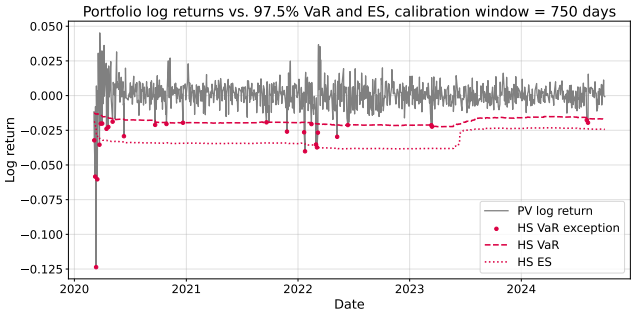
Figure 20. *Scatterplots of standardised residuals from univariate GARCH models transformed into uniform margins (500-day calibration window: 2019-12-17–2022-02-15)*



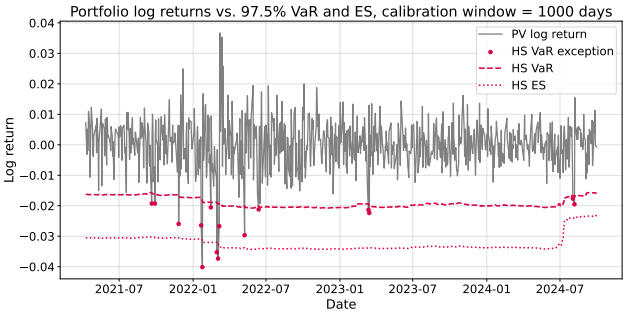
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

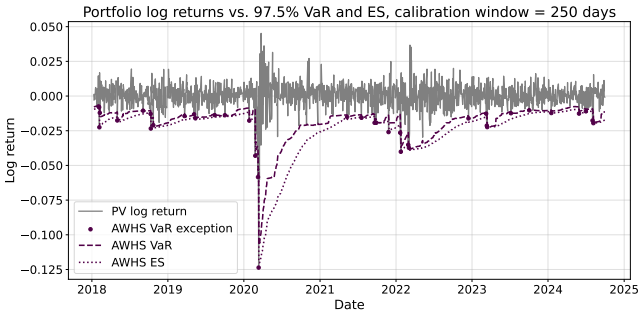


(c) *750-day calibration window, 2020-03-05–2024-09-30*

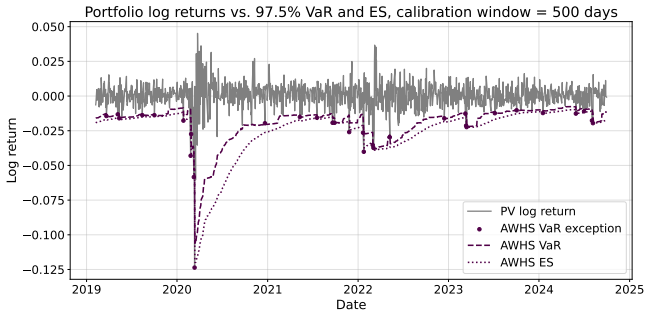


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

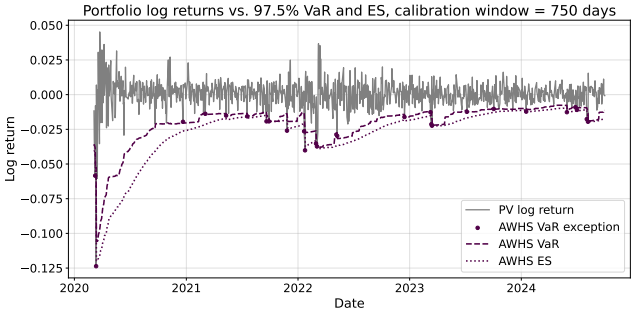
Figure 21. *97.5% basic historical simulation (HS) VaR and ES forecasts using different calibration windows*



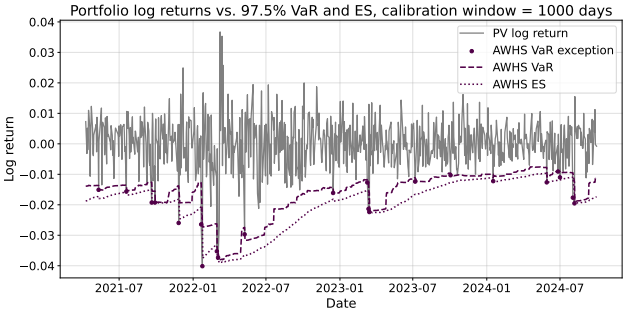
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

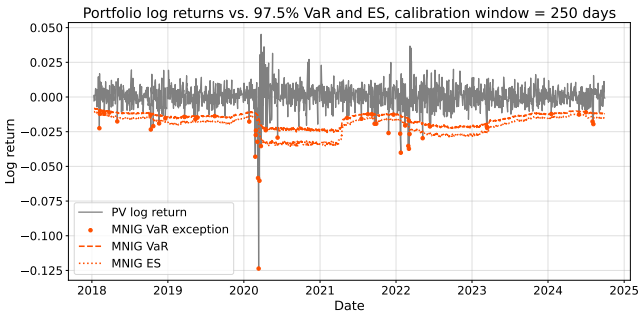


(c) *750-day calibration window, 2020-03-05–2024-09-30*

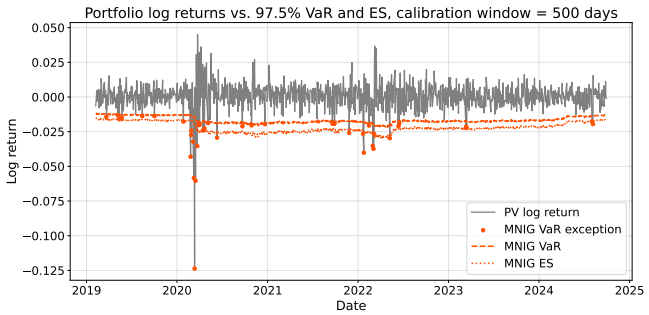


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

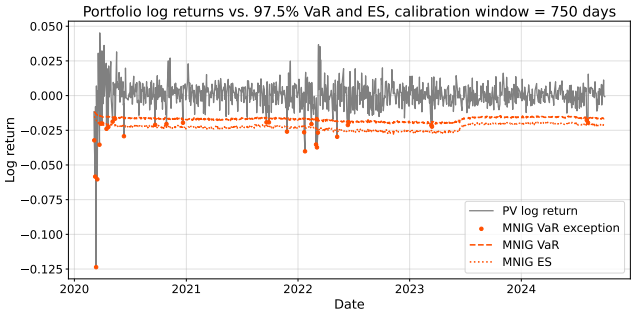
Figure 22. *97.5% age-weighted historical simulation (AWHS) VaR and ES forecasts using different calibration windows*



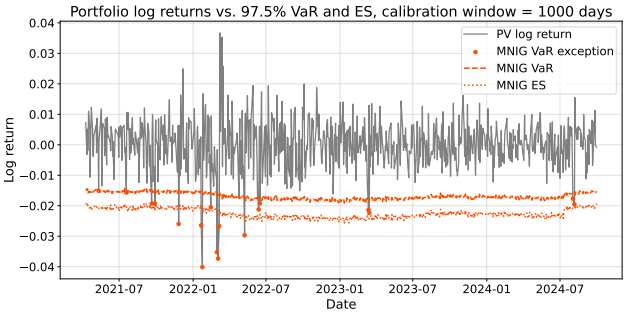
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

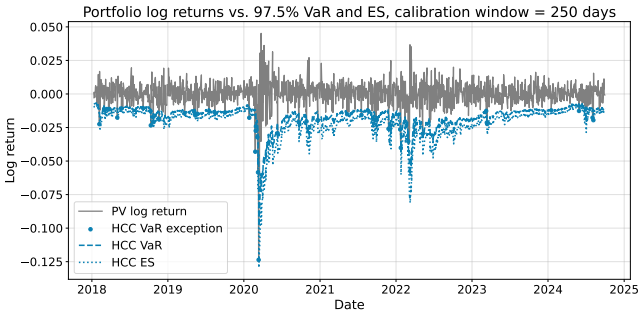


(c) *750-day calibration window, 2020-03-05–2024-09-30*

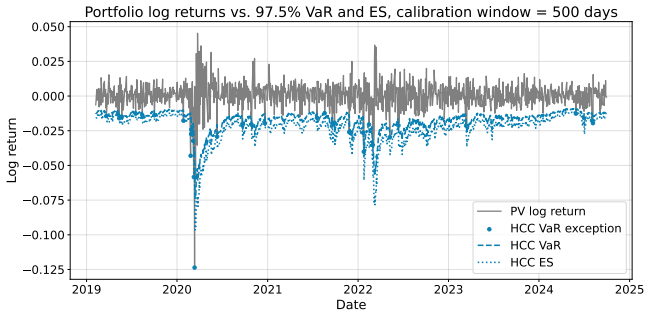


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

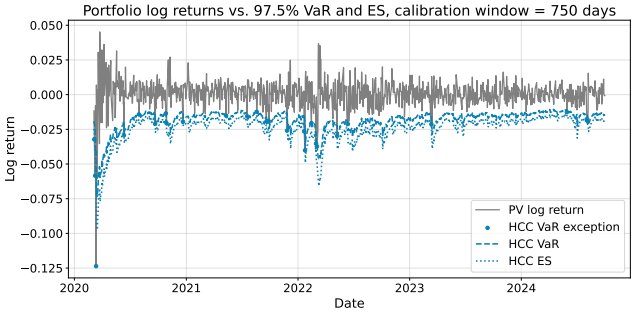
Figure 23. *97.5% multivariate normal inverse Gaussian distribution (MNIG) VaR and ES forecasts using different calibration windows*



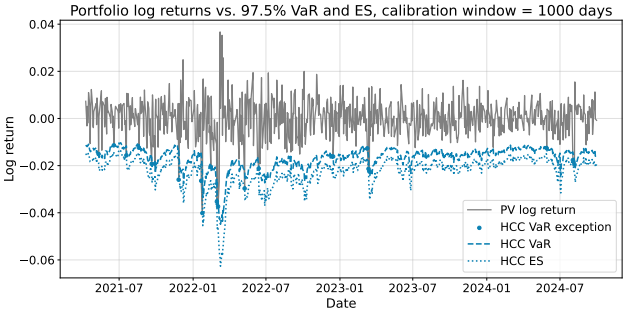
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

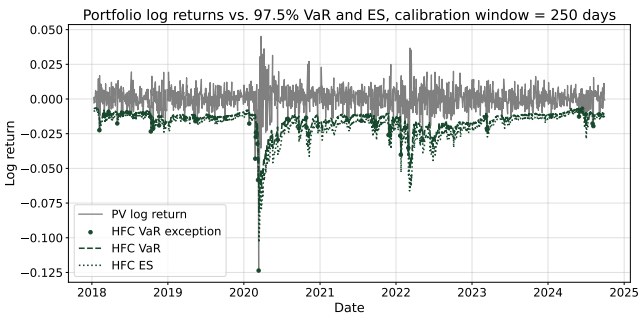


(c) *750-day calibration window, 2020-03-05–2024-09-30*

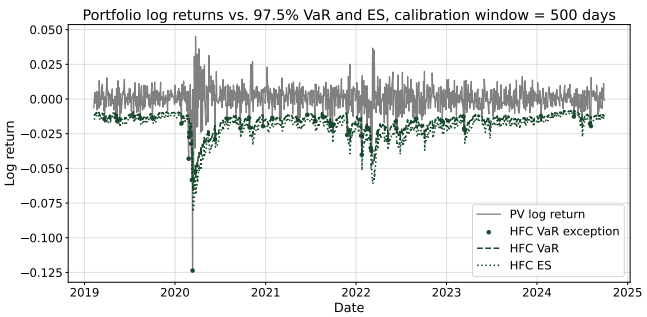


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

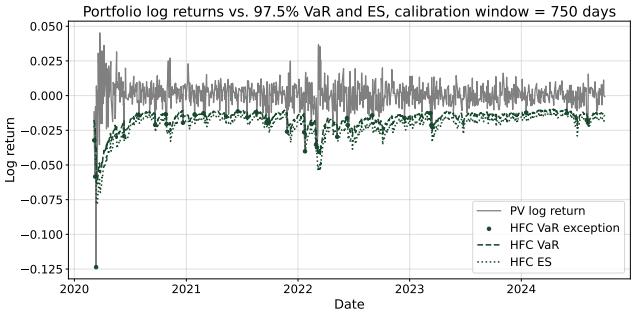
Figure 24. *97.5% hierarchical Clayton copula (HCC) VaR and ES forecasts using different calibration windows*



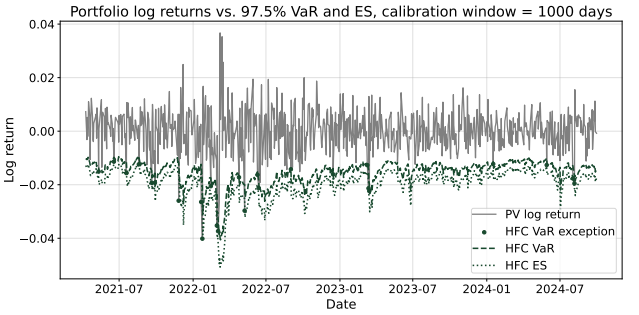
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

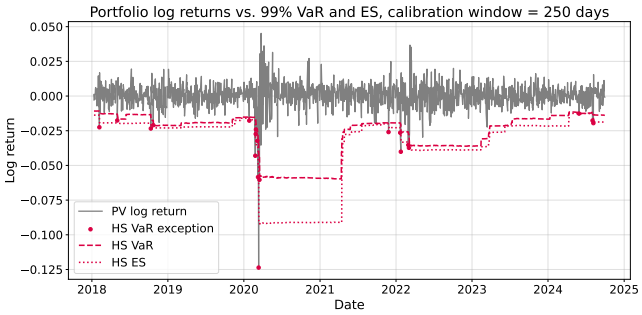


(c) *750-day calibration window, 2020-03-05–2024-09-30*

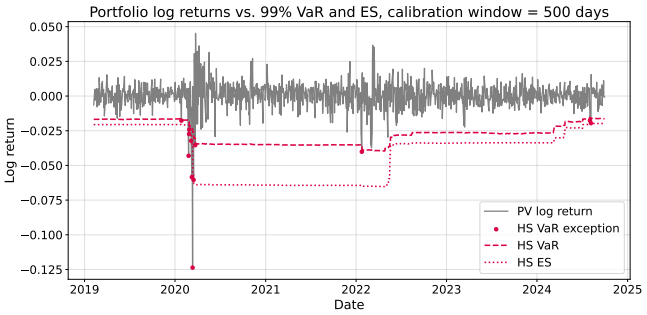


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

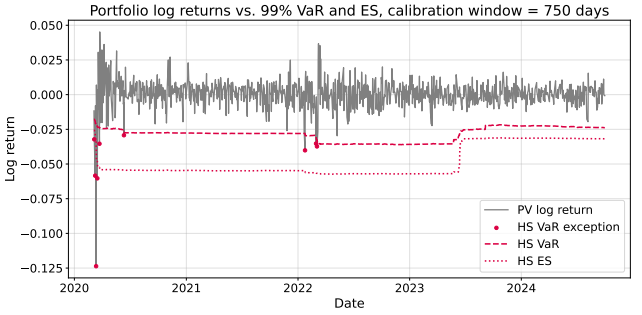
Figure 25. *97.5% hierarchical Frank copula (HFC) VaR and ES forecasts using different calibration windows*



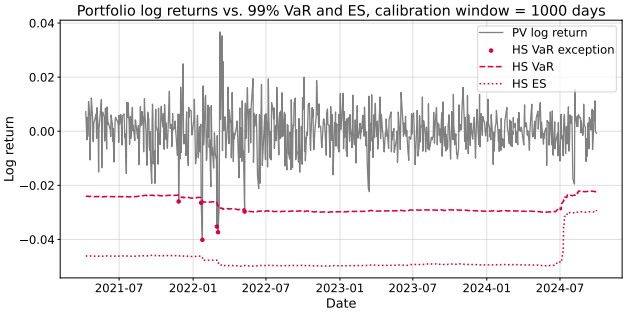
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

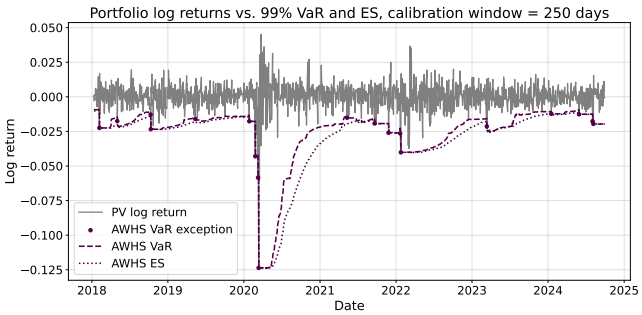


(c) *750-day calibration window, 2020-03-05–2024-09-30*

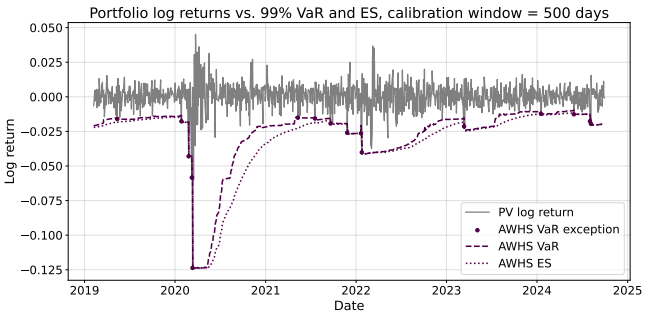


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

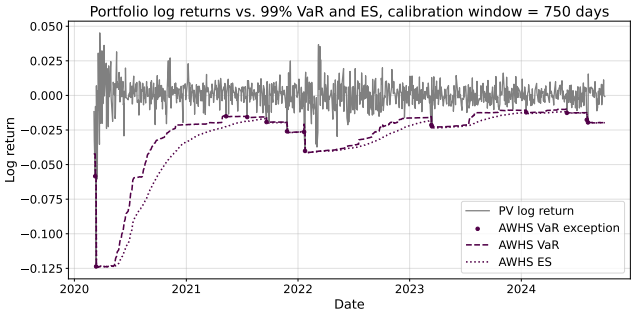
Figure 26. *99% basic historical simulation (HS) VaR and ES forecasts using different calibration windows*



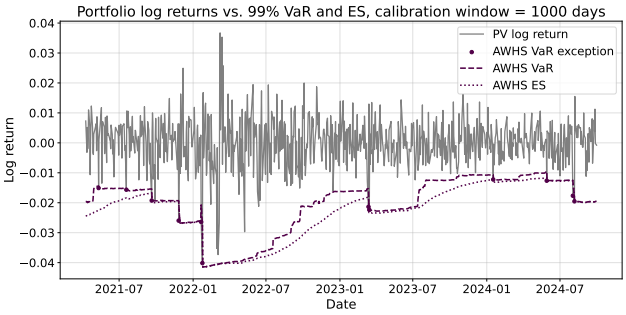
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

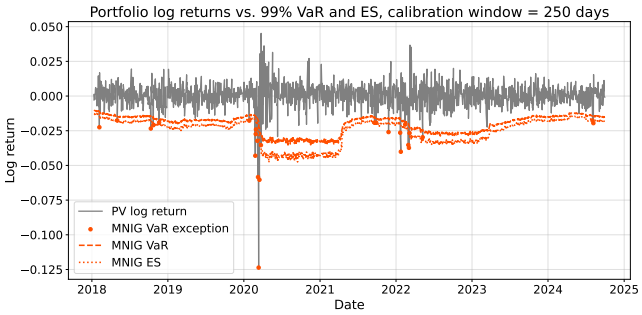


(c) *750-day calibration window, 2020-03-05–2024-09-30*

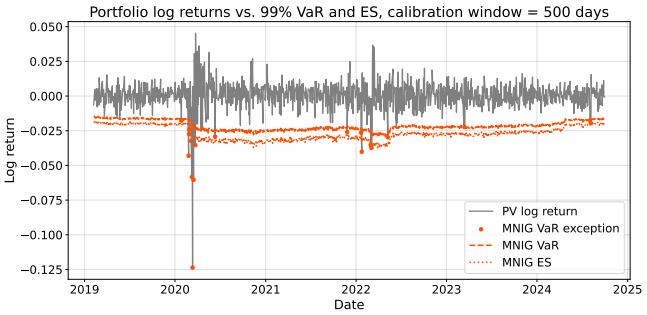


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

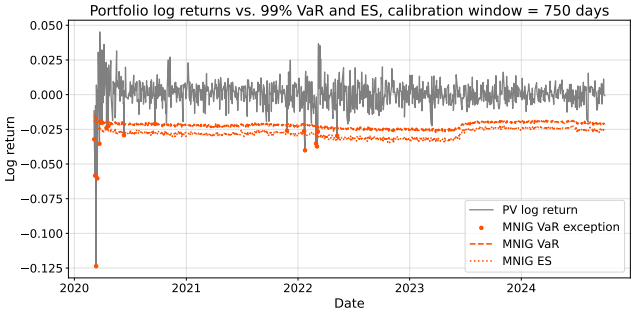
Figure 27. *99% age-weighted historical simulation (AWHS) VaR and ES forecasts using different calibration windows*



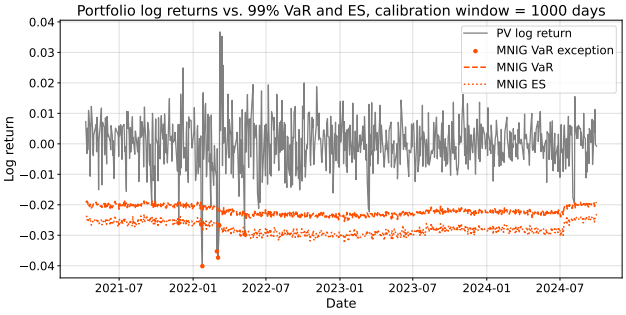
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

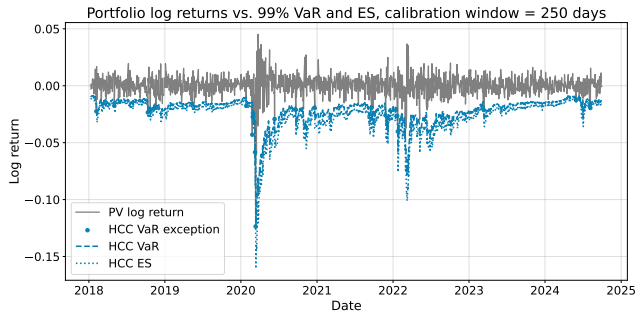


(c) *750-day calibration window, 2020-03-05–2024-09-30*

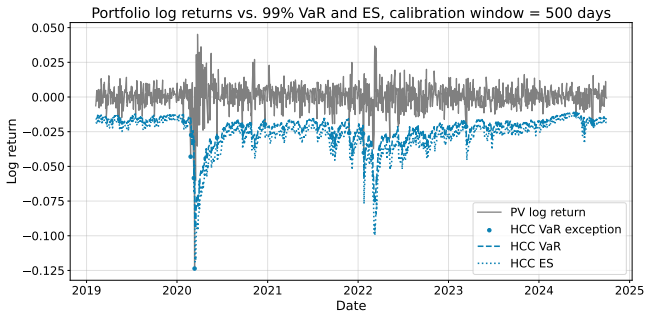


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

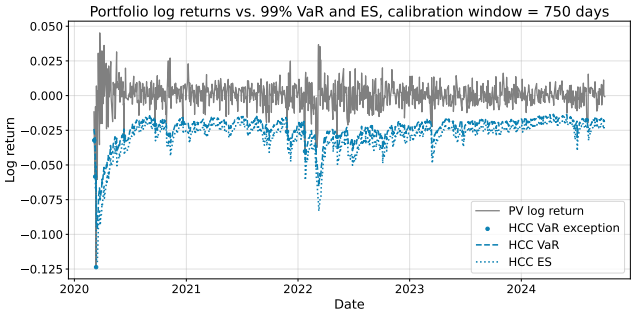
Figure 28. *99% multivariate normal inverse Gaussian distribution (MNIG) VaR and ES forecasts using different calibration windows*



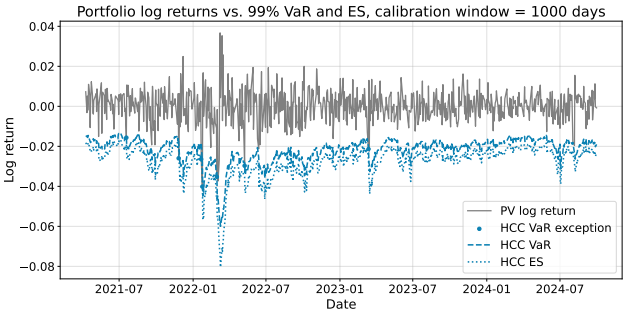
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*

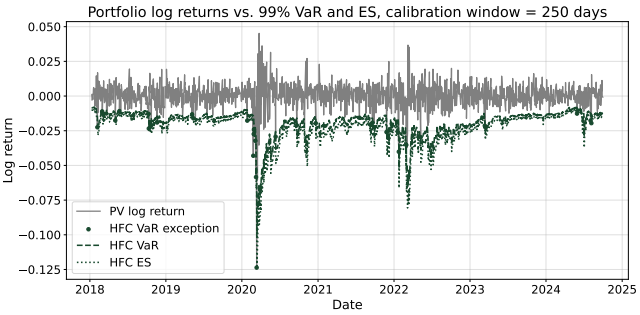


(c) *750-day calibration window, 2020-03-05–2024-09-30*

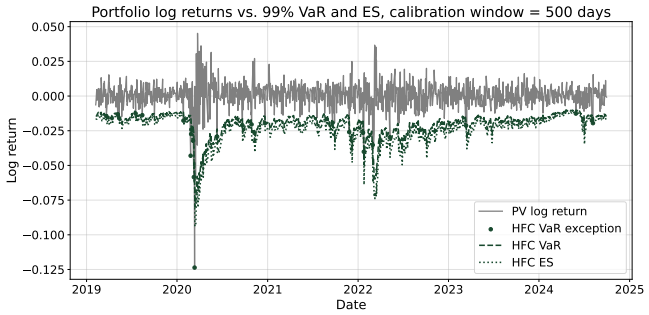


(d) *1000-day calibration window, 2021-04-09–2024-09-30*

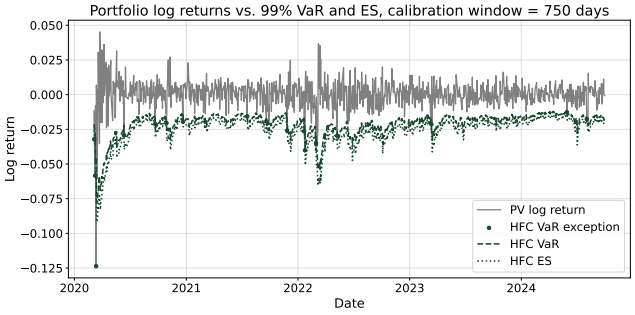
Figure 29. *99% hierarchical Clayton copula (HCC) VaR and ES forecasts using different calibration windows*



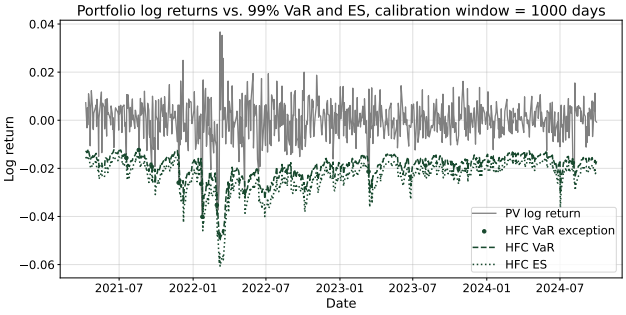
(a) *250-day calibration window, 2018-01-11–2024-09-30*



(b) *500-day calibration window, 2019-02-07–2024-09-30*



(c) *750-day calibration window, 2020-03-05–2024-09-30*



(d) *1000-day calibration window, 2021-04-09–2024-09-30*

Figure 30. *99% hierarchical Frank copula (HFC) VaR and ES forecasts using different calibration windows*

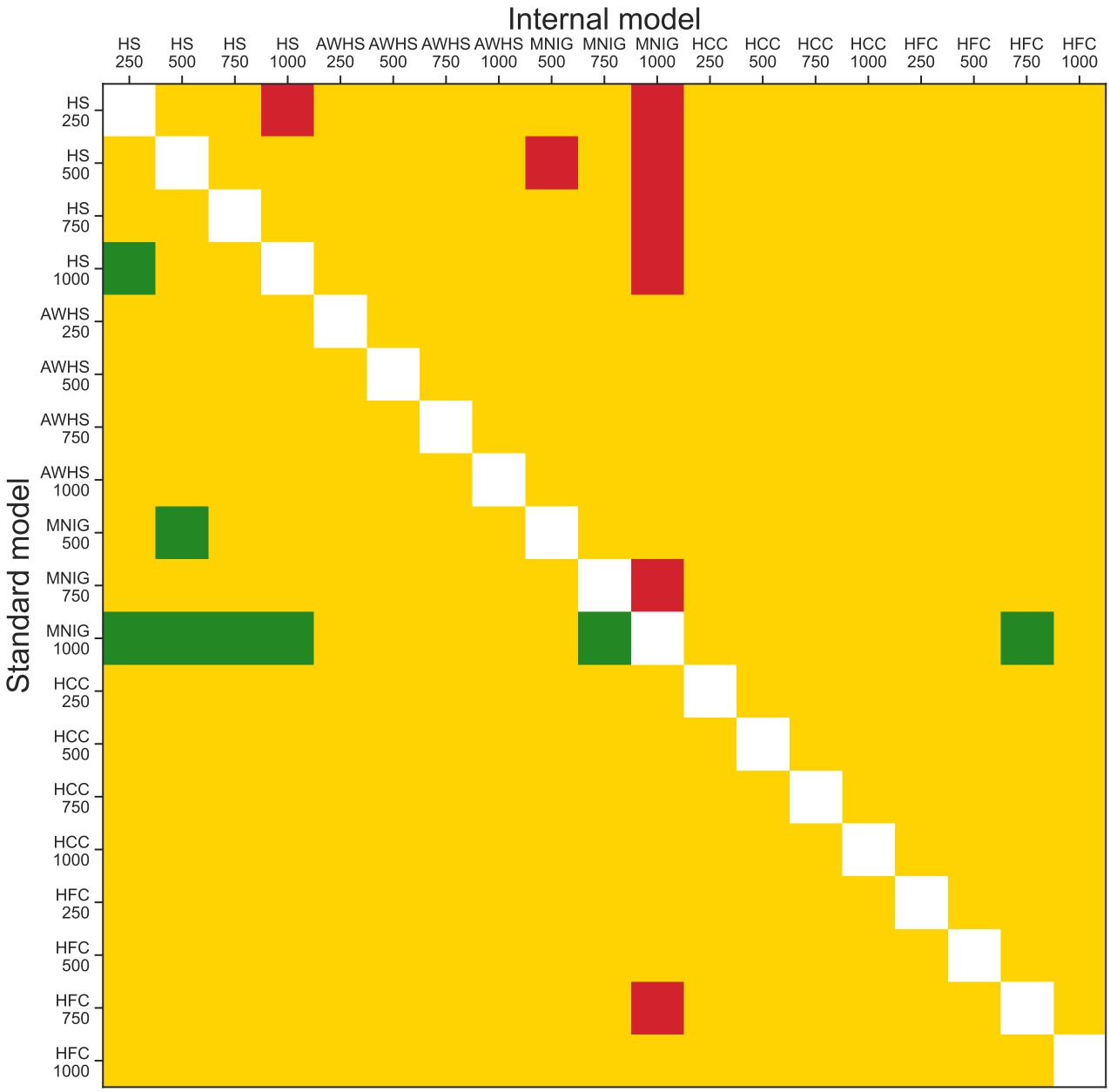


Figure 31. *Traffic light matrix for 95% VaR comparative backtesting results*

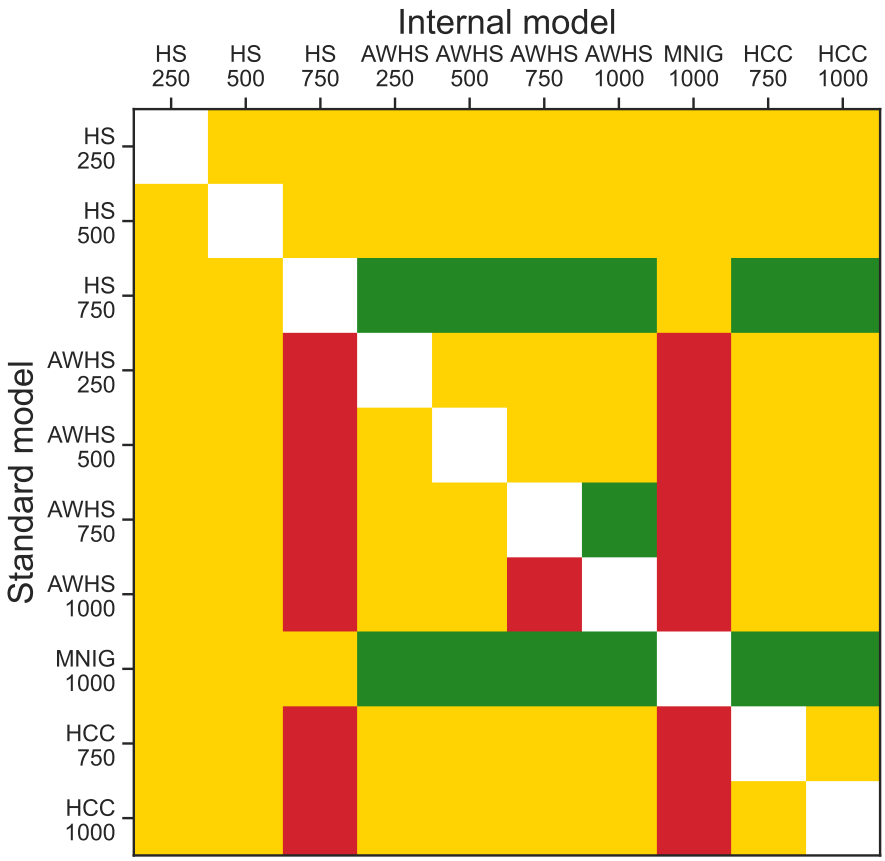
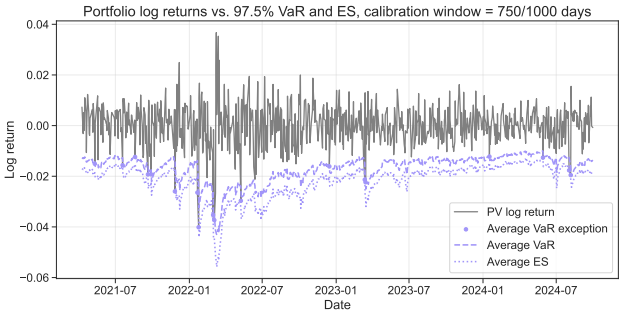
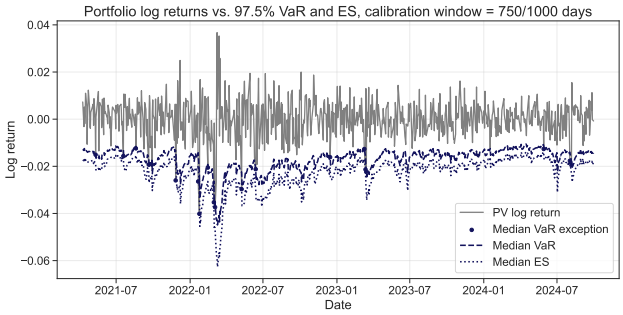


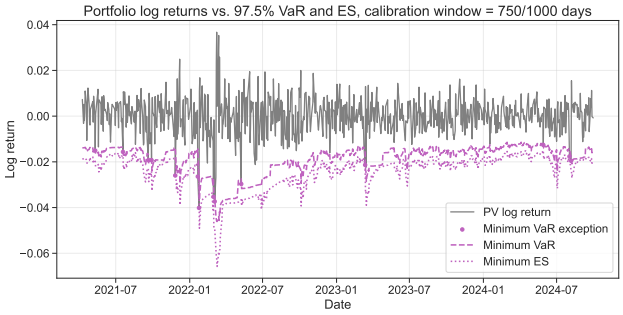
Figure 32. *Traffic light matrix for 95% ES comparative backtesting results*



(a) *97.5% VaR and ES produced by averaging forecasts*

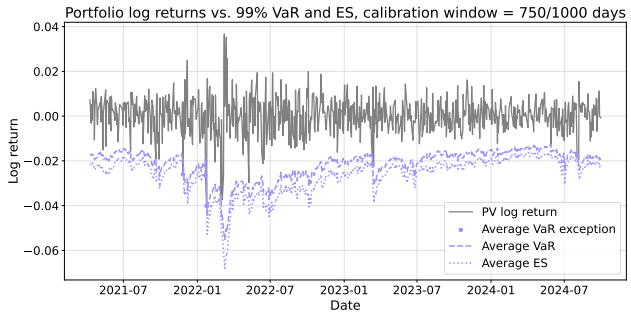


(b) *97.5% VaR and ES produced by the median of forecasts*

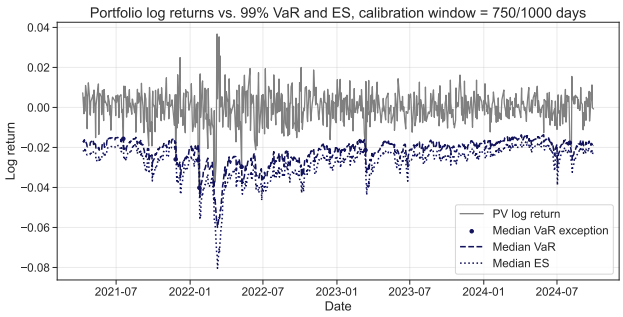


(c) *97.5% VaR and ES produced by the minimum of forecasts*

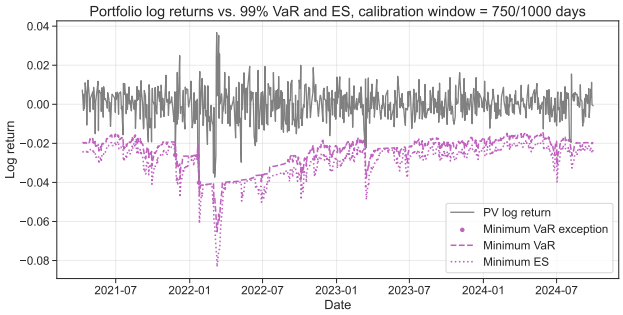
Figure 33. *97.5% VaR and ES from the combined forecasts produced by AWHS (w=1000), HCC (w=750), HCC (w=1000) models*



(a) *99% VaR and ES produced by averaging forecasts*

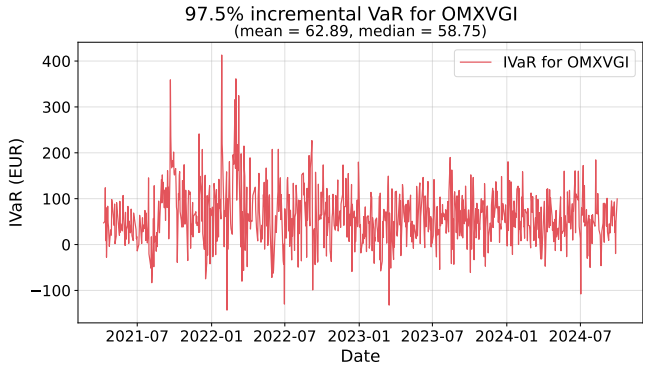


(b) *99% VaR and ES produced by the median of forecasts*

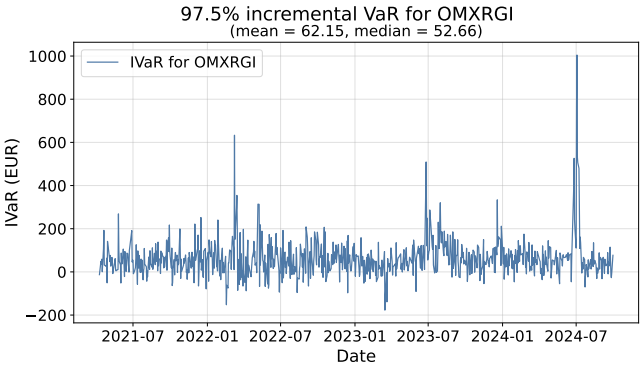


(c) *99% VaR and ES produced by the minimum of forecasts*

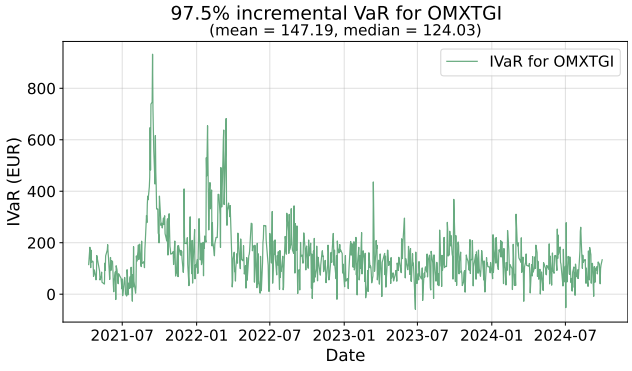
Figure 34. *99% VaR and ES from the combined forecasts produced by AWHS (w=1000), HCC (w=750), HCC (w=1000) models*



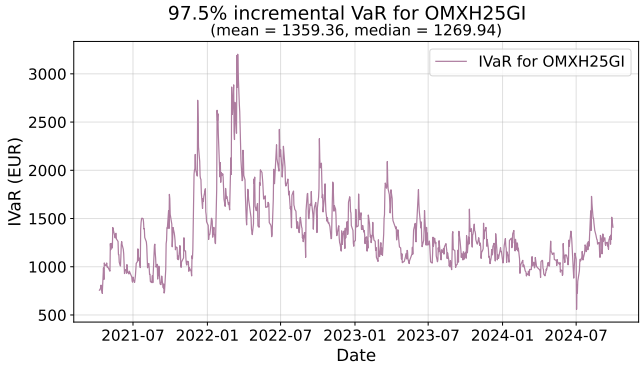
(a) *IVaR: OMXVGI*



(b) *IVaR: OMXRGI*

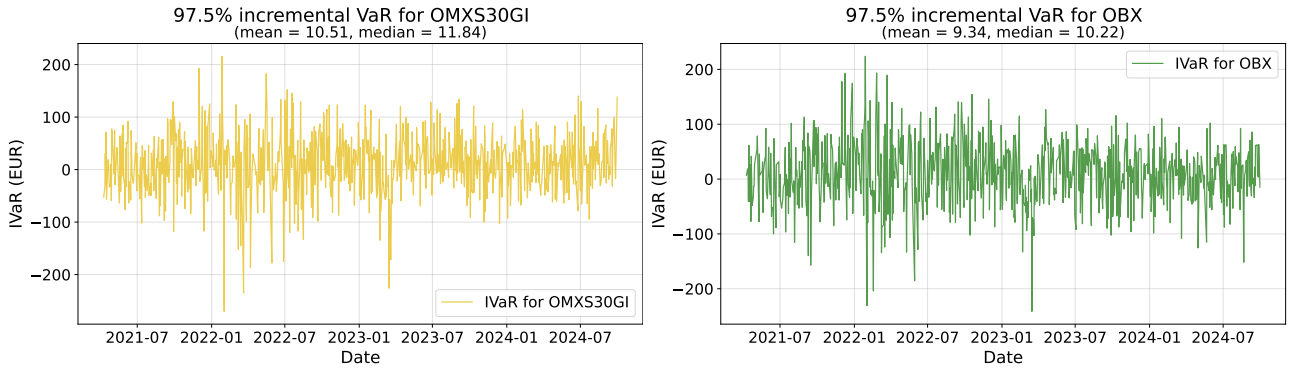


(c) *IVaR: OMXTGI*

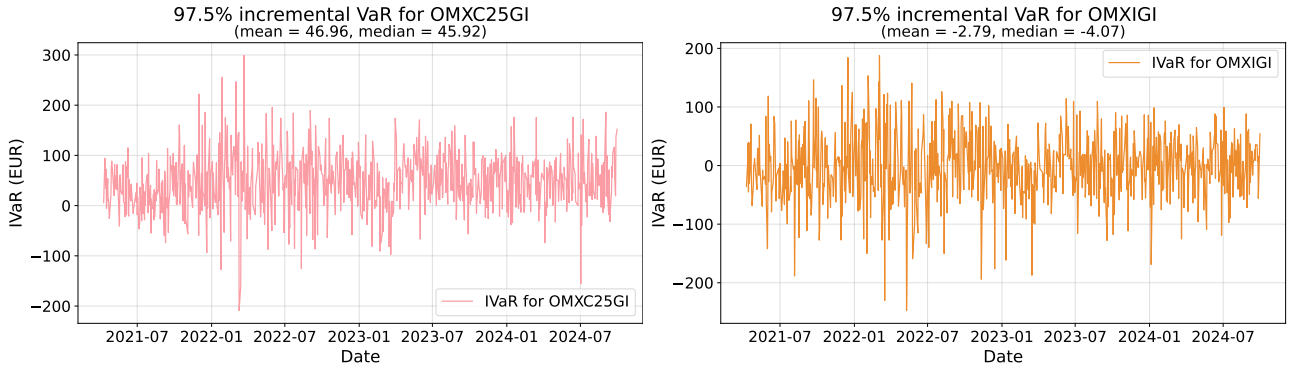


(d) *IVaR: OMXH25GI*

Figure 35. *97.5% incremental VaR for the Baltic indices and OMXH25GI produced by the hierarchical Clayton copula model estimated on a calibration window of 1000 days*

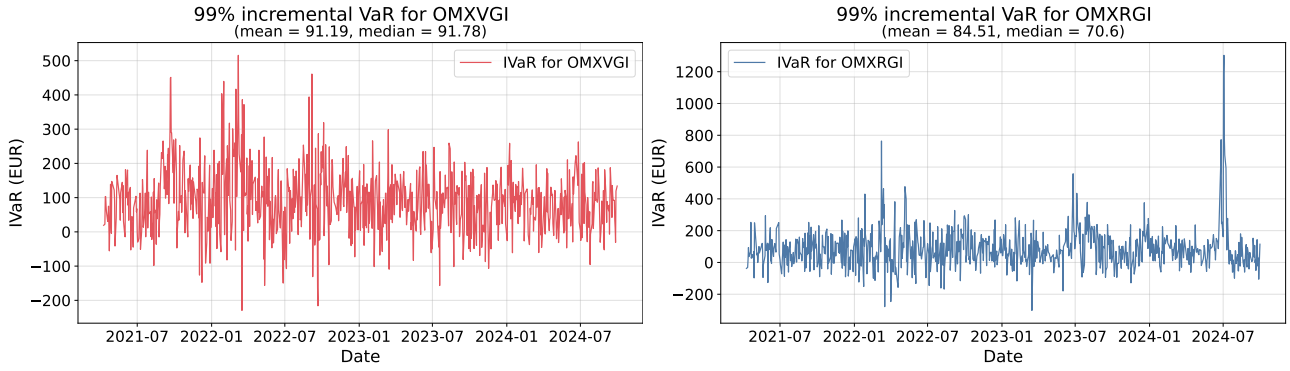


***(a)** IVaR: OMXS30GI* ***(b)** IVaR: OBX*

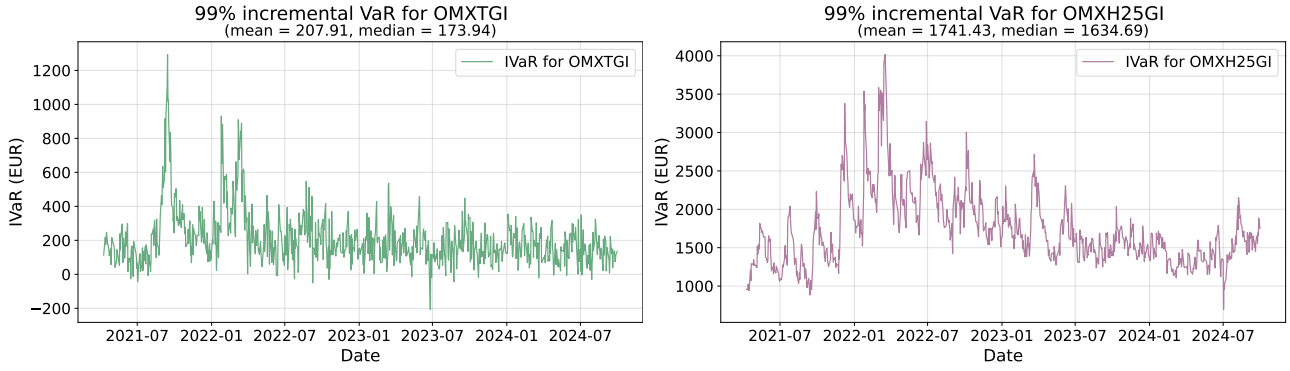


***(c)** IVaR: OMXC25GI* ***(d)** IVaR: OMXIGI*

Figure 36. *97.5% incremental VaR for the Scandinavian indices and OMXIGI produced by the hierarchical Clayton copula model estimated on a calibration window of 1000 days*

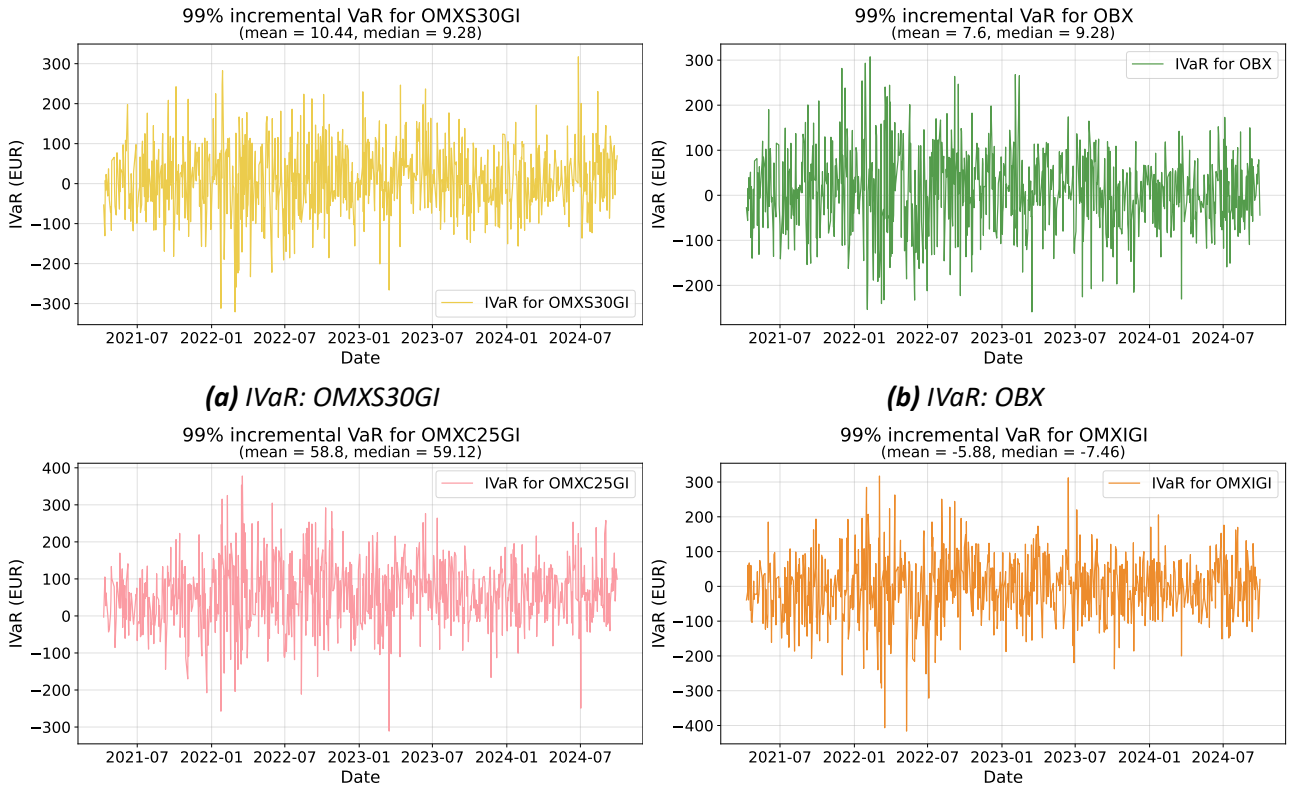


***(a)** IVaR: OMXVGI* ***(b)** IVaR: OMXRGI*



***(c)** IVaR: OMXTGI* ***(d)** IVaR: OMXH25GI*

Figure 37. *99% incremental VaR for the Baltic indices and OMXH25GI produced by the hierarchical Clayton copula model estimated on a calibration window of 1000 days*



(a) *IVaR: OMXS30GI*

(b) *IVaR: OBX*

(c) *IVaR: OMXC25GI*

(d) *IVaR: OMXIGI*

Figure 38. *99% incremental VaR for the Scandinavian indices and OMXIGI produced by the hierarchical Clayton copula model estimated on a calibration window of 1000 days*

2 Software code

In this thesis, the Python programming language was mainly used to perform data analysis and model estimation. However, by using the **rpy2** interface to R in Python, some key R packages have also been applied, namely **ghyp** and **QRM** for MNIG models, and **HAC**, **GeneralizedHyperbolic**, and **rugarch** for copula-based models.

```
import pandas as pd
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
from timeit import default_timer as timer
from scipy.stats import chi2
from scipy.stats import binom
from scipy.stats import norm
from scipy.stats import kurtosis
from scipy.stats import skew
from scipy.stats import jarque_bera
from statsmodels.tsa.stattools import adfuller
from statsmodels.stats.diagnostic import het_arch
from statsmodels.stats.diagnostic import acorr_ljungbox
import seaborn as sns
# Using R packages:
import rpy2.robjects as robjects
from rpy2.robjects.packages import importr
from rpy2.robjects.conversion import localconverter
import rpy2.robjects as ro
from rpy2.robjects import r
from rpy2.robjects import pandas2ri
from rpy2.robjects.vectors import FloatMatrix as FMat
from rpy2.rinterface_lib.embedded import RRuntimeError
set_seed = robjects.r('set.seed')
utils = importr('utils')
base = importr('base')
# For MNIG:
ghyp = importr('ghyp')
QRM = importr('QRM')
# For copulas:
genhyp = importr('GeneralizedHyperbolic')
HAC = importr('HAC')
rg = importr('rugarch')

index_prices_df = pd.read_excel('only_index_prices_eur_with_start.xlsx', index_col=0)
index_units = [10] * 8
index_prices_df['PV'] = index_prices_df.drop(columns='Date').mul(index_units).sum(axis=1)
index_prices_df['log_r_PV'] = (np.log(index_prices_df['PV']) -
                               np.log(index_prices_df['PV'].shift(1)))

log_returns_df = pd.read_excel('only_log_returns_eur.xlsx', index_col=0)

price_columns = list(index_prices_df.filter(regex=r'^Close').columns)
logr_columns = list(log_returns_df.filter(regex=r'_eur').columns)
logr_columns_dict = ['logr_vgi_eur', 'logr_rgi_eur', 'logr_tgi_eur', 'logr_h25gi_eur',
                     'logr_s30gi_eur', 'logr_igi_eur', 'logr_c25gi_eur', 'logr_obx_eur']
col_match = dict(zip(logr_columns_dict, price_columns))

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# ------------- EDA ------------- #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #

stats_list = ['min', 'mean', '50%', 'max', 'std']
```

```
descr_stats = (log_returns_df.describe().loc[stats_list, :]*100).copy()
descr_stats.index = ['Minimum', 'Mean', 'Median', 'Maximum',
                     'Standard deviation']
descr_stats.loc['Excess kurtosis', :] = kurtosis(log_returns_df[logr_columns])
descr_stats.loc['Skewness', :] = skew(log_returns_df[logr_columns])
descr_stats.loc["Q(10)", :] = log_returns_df[logr_columns].apply(
    lambda x: jarque_bera(x)[0], axis=0
)
descr_stats.loc["ADF test", :] = log_returns_df[logr_columns].apply(
    lambda x: adfuller(x)[0], axis=0
)
descr_stats.loc["ARCH test", :] = log_returns_df[logr_columns].apply(
    lambda x: het_arch(x, nlags=12)[0], axis=0
)
descr_stats.loc["Ljung-Box test", :] = log_returns_df[logr_columns].apply(
    lambda x: acorr_ljungbox(x).iloc[-1][0], axis=0
)
descr_stats.loc["LjungBox_pval", :] = log_returns_df[logr_columns].apply(
    lambda x: acorr_ljungbox(x).iloc[-1][1], axis=0
)
descr_stats.columns = ['OMXVGI', 'OMXRGI', 'OMXTGI', 'OMXS30GI',
                       'OMXIGI', 'OMXH25GI', 'OMXC25GI', 'OBX']
descr_stats.to_excel('descriptive_statistics_logr.xlsx')

stt = importr('stats')


def plot_unif_pairs(data, KStest=False):
    with localconverter(ro.default_converter + pandas2ri.converter):
        filtered_df_R = ro.conversion.py2rpy(data)

    # Estimating GARCH models:
    garch_vgi = fit_garch(filtered_df_R, 'logr_vgi_eur')
    garch_rgi = fit_garch(filtered_df_R, 'logr_rgi_eur')
    garch_tgi = fit_garch(filtered_df_R, 'logr_tgi_eur')
    garch_s30gi = fit_garch(filtered_df_R, 'logr_s30gi_eur')
    garch_igi = fit_garch(filtered_df_R, 'logr_igi_eur')
    garch_h25gi = fit_garch(filtered_df_R, 'logr_h25gi_eur')
    garch_c25gi = fit_garch(filtered_df_R, 'logr_c25gi_eur')
    garch_obx = fit_garch(filtered_df_R, 'logr_obx_eur')

    # Getting the standardised residuals:
    with localconverter(ro.default_converter + pandas2ri.converter):
        sr_vgi = FMat(np.divide(rg.residuals(garch_vgi), rg.sigma(garch_vgi)))
        sr_rgi = FMat(np.divide(rg.residuals(garch_rgi), rg.sigma(garch_rgi)))
        sr_tgi = FMat(np.divide(rg.residuals(garch_tgi), rg.sigma(garch_tgi)))
        sr_s30gi = FMat(np.divide(rg.residuals(garch_s30gi), rg.sigma(garch_s30gi)))
        sr_igi = FMat(np.divide(rg.residuals(garch_igi), rg.sigma(garch_igi)))
        sr_h25gi = FMat(np.divide(rg.residuals(garch_h25gi), rg.sigma(garch_h25gi)))
        sr_c25gi = FMat(np.divide(rg.residuals(garch_c25gi), rg.sigma(garch_c25gi)))
        sr_obx = FMat(np.divide(rg.residuals(garch_obx), rg.sigma(garch_obx)))

    # Fitting the NIG distribution to marginals:
    p_nig_vgi = genhyp.nigFit(sr_vgi)
    p_nig_rgi = genhyp.nigFit(sr_rgi)
    p_nig_tgi = genhyp.nigFit(sr_tgi)
    p_nig_s30gi = genhyp.nigFit(sr_s30gi)
    p_nig_igi = genhyp.nigFit(sr_igi)
    p_nig_h25gi = genhyp.nigFit(sr_h25gi)
    p_nig_c25gi = genhyp.nigFit(sr_c25gi)
    p_nig_obx = genhyp.nigFit(sr_obx)

    if not KStest:
```

```
        # Converting to uniform marginals:
        unif_vgi = genhyp.pghyp(sr_vgi, param=p_vector(p_nig_vgi))
        unif_rgi = genhyp.pghyp(sr_rgi, param=p_vector(p_nig_rgi))
        unif_tgi = genhyp.pghyp(sr_tgi, param=p_vector(p_nig_tgi))
        unif_s30gi = genhyp.pghyp(sr_s30gi, param=p_vector(p_nig_s30gi))
        unif_igi = genhyp.pghyp(sr_igi, param=p_vector(p_nig_igi))
        unif_h25gi = genhyp.pghyp(sr_h25gi, param=p_vector(p_nig_h25gi))
        unif_c25gi = genhyp.pghyp(sr_c25gi, param=p_vector(p_nig_c25gi))
        unif_obx = genhyp.pghyp(sr_obx, param=p_vector(p_nig_obx))

        unif_marg = base.cbind(
            unif_vgi=unif_vgi,
            unif_rgi=unif_rgi,
            unif_tgi=unif_tgi,
            unif_s30gi=unif_s30gi,
            unif_igi=unif_igi,
            unif_h25gi=unif_h25gi,
            unif_c25gi=unif_c25gi,
            unif_obx=unif_obx,
        )

        py_unif_marg = pd.DataFrame(np.array(unif_marg))
        py_unif_marg.columns = ['OMXVGI', 'OMXRGI', 'OMXTGI', 'OMXS30GI',
                                'OMXIGI', 'OMXH25GI', 'OMXC25GI', 'OBX']
        cor_matrix = py_unif_marg.corr()
        sns.set_theme(style='ticks')
        sns.set_context('notebook', font_scale=1.5)
        unif_pairplot = sns.pairplot(py_unif_marg, plot_kws=dict(linewidth=0.5))

        return py_unif_marg, cor_matrix, unif_pairplot

    elif KStest:
        pval_vgi = np.array(stt.ks_test(sr_vgi, genhyp.pghyp,
                                        param=p_vector(p_nig_vgi)).rx2('p.value'))[0]
        pval_rgi = np.array(stt.ks_test(sr_rgi, genhyp.pghyp,
                                        param=p_vector(p_nig_rgi)).rx2('p.value'))[0]
        pval_tgi = np.array(stt.ks_test(sr_tgi, genhyp.pghyp,
                                        param=p_vector(p_nig_tgi)).rx2('p.value'))[0]
        pval_s30gi = np.array(stt.ks_test(sr_s30gi, genhyp.pghyp,
                                          param=p_vector(p_nig_s30gi)).rx2('p.value'))[0]
        pval_igi = np.array(stt.ks_test(sr_igi, genhyp.pghyp,
                                        param=p_vector(p_nig_igi)).rx2('p.value'))[0]
        pval_h25gi = np.array(stt.ks_test(sr_h25gi, genhyp.pghyp,
                                          param=p_vector(p_nig_h25gi)).rx2('p.value'))[0]
        pval_c25gi = np.array(stt.ks_test(sr_c25gi, genhyp.pghyp,
                                          param=p_vector(p_nig_c25gi)).rx2('p.value'))[0]
        pval_obx = np.array(stt.ks_test(sr_obx, genhyp.pghyp,
                                        param=p_vector(p_nig_obx)).rx2('p.value'))[0]
        pvals = [pval_vgi, pval_rgi, pval_tgi, pval_s30gi,
                 pval_igi, pval_h25gi, pval_c25gi, pval_obx]
        return pd.DataFrame({'pvalues': pvals}, index=['OMXVGI', 'OMXRGI', 'OMXTGI',
                                                       'OMXS30GI', 'OMXIGI', 'OMXH25GI',
                                                       'OMXC25GI', 'OBX'])


# 2016-12-20 - 2019-02-06
df_start = log_returns_df.iloc[:500, :].copy()
df_start.set_index('Date', inplace=True)
unif_data_start, cor_start, unif_plot_start = plot_unif_pairs(df_start)
unif_data_start.to_excel('start_unif_data.xlsx')
cor_start.to_excel('start_unif_corr.xlsx')
unif_plot_start.figure.savefig('pairs_start_unif.pdf')
# 2019-12-17 - 2022-02-15
```

```
filtered_df = log_returns_df.iloc[700:1200, :].copy()
filtered_df.set_index('Date', inplace=True)
unif_data_covid, cor_covid, unif_plot_covid = plot_unif_pairs(filtered_df)
unif_data_covid.to_excel('covid_unif_data.xlsx')
cor_covid.to_excel('covid_unif_corr.xlsx')
unif_plot_covid.figure.savefig('pairs_covid_unif.pdf')

# Create a mask for the upper triangle:
mask = np.triu(np.ones_like(cor_covid, dtype=bool))
np.fill_diagonal(mask, False)
heatmap_cov = sns.heatmap(cor_covid, cmap='coolwarm', annot=True, fmt='.2f',
                          annot_kws={'fontsize': 12}, mask=mask, vmin=-1, vmax=1)
heatmap_cov.get_figure().savefig('corr_heatmap_covid.pdf', dpi=200, bbox_inches='tight')
heatmap_start = sns.heatmap(cor_start, cmap='coolwarm', annot=True, fmt='.2f',
                            annot_kws={'fontsize': 12}, mask=mask, vmin=-1, vmax=1)
heatmap_start.get_figure().savefig('corr_heatmap_start.pdf', dpi=200,
                                   bbox_inches='tight')


def get_aic(df, col):
    arma_spec_00 = base.list(armaOrder=base.c(0, 0))
    arma_spec_10 = base.list(armaOrder=base.c(1, 0))
    arma_spec_11 = base.list(armaOrder=base.c(1, 1))
    specs = {
        'ARMA(0,0)': arma_spec_00,
        'ARMA(1,0)': arma_spec_10,
        'ARMA(1,1)': arma_spec_11
        }
    garch_spec = base.list(model="sGARCH", garchOrder=base.c(1, 1))
    aic_vals = {}
    for arma_spec in specs:
        fit_r = rg.ugarchspec(mean_model=specs[arma_spec],
                              variance_model=garch_spec,
                              distribution_model="nig")
        res = rg.ugarchfit(data=df.rx2(col), spec=fit_r, solver='hybrid')
        aic_val = rg.infocriteria(res)[0]
        aic_vals[arma_spec] = aic_val

    return aic_vals


# 2016-12-20 - 2019-02-06
logr_start = log_returns_df.iloc[:500, :].copy()
logr_start.set_index('Date', inplace=True)
with localconverter(ro.default_converter + pandas2ri.converter):
    df_R_start = ro.conversion.py2rpy(logr_start)

aic_values = pd.DataFrame()
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_start, 'logr_vgi_eur'),
                                                 index=['OMXVGI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_start, 'logr_rgi_eur'),
                                                 index=['OMXRGI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_start, 'logr_tgi_eur'),
                                                 index=['OMXTGI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_start, 'logr_s30gi_eur'),
                                                 index=['OMXS30GI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_start, 'logr_igi_eur'),
                                                 index=['OMXIGI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_start, 'logr_h25gi_eur'),
                                                 index=['OMXH25GI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_start, 'logr_c25gi_eur'),
                                                 index=['OMXC25GI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_start, 'logr_obx_eur'),
```

```
                                                index=['OBX'])])
aic_values.T.to_excel('AIC_values_first_start.xlsx')

# 2019-12-17 - 2022-02-15
logr_cov = log_returns_df.iloc[700:1200, :].copy()
logr_cov.set_index('Date', inplace=True)
with localconverter(ro.default_converter + pandas2ri.converter):
    df_R_cov = ro.conversion.py2rpy(logr_cov)

aic_values = pd.DataFrame()
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_cov, 'logr_vgi_eur'),
                                                index=['OMXVGI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_cov, 'logr_rgi_eur'),
                                                index=['OMXRGI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_cov, 'logr_tgi_eur'),
                                                index=['OMXTGI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_cov, 'logr_s30gi_eur'),
                                                index=['OMXS30GI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_cov, 'logr_igi_eur'),
                                                index=['OMXIGI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_cov, 'logr_h25gi_eur'),
                                                index=['OMXH25GI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_cov, 'logr_c25gi_eur'),
                                                index=['OMXC25GI'])])
aic_values = pd.concat([aic_values, pd.DataFrame(get_aic(df_R_cov, 'logr_obx_eur'),
                                                index=['OBX'])])
aic_values.T.to_excel('AIC_values_first_cov.xlsx')


pvals_start = plot_unif_pairs(logr_start, KStest=True)
pvals_cov = plot_unif_pairs(logr_cov, KStest=True)
pvals_start['pvals_cov'] = pvals_cov
pvals_start.T.to_excel('KS_test_pvals_500d.xlsx')

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# ------------ Estimation ------------ #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


# A general function implementing the rolling window estimation with
# the estimation method passed as an argument:
def VaR_ES_estimation(estimation_func, results_df, calibration_window, index_prices_df,
                      return_df=None, random_seed=2024, method_in_R=False, c_type=None):

    results_df = pd.DataFrame(columns=['forecast_date', 'VaR_95', 'VaR_975', 'VaR_99',
                                       'VaR_9625', 'VaR_98125', 'VaR_9875', 'VaR_9925',
                                       'VaR_99375', 'VaR_995', 'VaR_9975',
                                       'ES_95', 'ES_975', 'ES_99', 'actual_log_r'])
    if method_in_R:
        set_seed(random_seed)
    else:
        np.random.seed(random_seed)

    if return_df is None:
        for study_date in index_prices_df['Date'][calibration_window:-1]:
            estimate_row = estimation_func(index_prices_df, calibration_window,
                                           study_date.strftime('%Y-%m-%d'))
            results_df = pd.concat([results_df, estimate_row])
    else:
        if estimation_func != VaR_ES_Copula:
            for study_date in index_prices_df['Date'][calibration_window:-1]:
                estimate_row = estimation_func(return_df, calibration_window,
                                               study_date.strftime('%Y-%m-%d'))
```

```
                results_df = pd.concat([results_df, estimate_row])
        elif estimation_func == VaR_ES_Copula:
            iter = 1
            for study_date in index_prices_df['Date'][calibration_window:-1]:
                estimate_row = estimation_func(return_df, calibration_window,
                                               study_date.strftime('%Y-%m-%d'),
                                               type=c_type)
                results_df = pd.concat([results_df, estimate_row])
                print(f'iter {iter} done')
                iter += 1

    # Main VaR level exceptions:
    results_df['VaR_95_breach'] = np.where(results_df['VaR_95'] >
                                           results_df['actual_log_r'], 1, 0)
    results_df['VaR_975_breach'] = np.where(results_df['VaR_975'] >
                                            results_df['actual_log_r'], 1, 0)
    results_df['VaR_99_breach'] = np.where(results_df['VaR_99'] >
                                           results_df['actual_log_r'], 1, 0)

    return results_df


def plot_VaR_ES(df, model_type, calibration_window, level=95, level_in_graph=None,
                save_fig=True, fig_title=None, legend_loc='lower left', borderpad=0.4,
                estimate_col='#db0043'):
    if level_in_graph is None:
        level_in_graph = level
    if fig_title is None:
        fig_title = f'{model_type}_{level}_{calibration_window}'
    plt.figure(figsize=(13, 6), dpi=200)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    plt.plot(df['forecast_date'], df['actual_log_r'], label="PV log return",
             color="grey", lw=1.65)
    plt.scatter(df['forecast_date'][df[f'VaR_{level}_breach'] == 1],
                df['actual_log_r'][df[f'VaR_{level}_breach'] == 1], color=estimate_col,
                label=f'{model_type} VaR exception', zorder=5, s=20)
    plt.plot(df['forecast_date'], df[f'VaR_{level}'], color=estimate_col,
             linestyle="--", lw=2, label=f'{model_type} VaR')
    plt.plot(df['forecast_date'], df[f'ES_{level}'], color=estimate_col,
             linestyle=":", lw=2, label=f'{model_type} ES')
    plt.title(f'Portfolio log returns vs. {level_in_graph}% VaR and ES, ' +
              f'calibration window = {calibration_window} days', fontsize=18.5)
    plt.xlabel('Date', fontsize=16.5)
    plt.ylabel('Log return', fontsize=16.5)
    plt.legend(fontsize=15, loc=legend_loc, borderpad=borderpad)
    plt.grid(alpha=0.4)
    if save_fig:
        plt.savefig(f'{fig_title}.pdf')
    plt.show()


# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# -------- Historical simulation -------- #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


def VaR_ES_HS(index_price_df, calibration_window, study_date):
    # (calibration_window + 1) is used in order to have
    # calibration_window number of returns:
    filtered_df = index_price_df[index_price_df['Date'] <=
                                 study_date].tail(calibration_window+1).copy()
    scenarios = pd.DataFrame()
```

```
    for ind in price_columns:
        # returns between i+1 and i multiplied by the nth price:
        scenarios[ind] = (filtered_df[ind] / filtered_df[ind].shift(1) *
                          filtered_df[ind].tail(1).iloc[0])
    scenarios['PV'] = scenarios.mul([10] * 8).sum(axis=1)
    scenarios.dropna(inplace=True)
    # Log returns between generated PVs and the actual PV on nth day:
    scenarios['log_r_PV'] = np.log(scenarios['PV'] / filtered_df['PV'].tail(1).iloc[0])
    est_q = 'median_unbiased'
    VaR_95 = np.percentile(scenarios['log_r_PV'], 100-95, method=est_q)
    VaR_975 = np.percentile(scenarios['log_r_PV'], 100-97.5, method=est_q)
    VaR_99 = np.percentile(scenarios['log_r_PV'], 100-99, method=est_q)
    # Additional estimates for multinomial backtesting:
    VaR_9625 = np.percentile(scenarios['log_r_PV'], 100-96.25, method=est_q)
    VaR_98125 = np.percentile(scenarios['log_r_PV'], 100-98.125, method=est_q)
    VaR_9875 = np.percentile(scenarios['log_r_PV'], 100-98.75, method=est_q)
    VaR_9925 = np.percentile(scenarios['log_r_PV'], 100-99.25, method=est_q)
    VaR_99375 = np.percentile(scenarios['log_r_PV'], 100-99.375, method=est_q)
    VaR_995 = np.percentile(scenarios['log_r_PV'], 100-99.5, method=est_q)
    VaR_9975 = np.percentile(scenarios['log_r_PV'], 100-99.75, method=est_q)
    date_filter = (index_prices_df['Date'] > study_date)
    estimates_row = {'forecast_date': index_price_df.loc[date_filter,
                                                         'Date'].head(1).iloc[0],
                     'VaR_95': VaR_95,
                     'VaR_975': VaR_975,
                     'VaR_99': VaR_99,
                     'VaR_9625': VaR_9625,
                     'VaR_98125': VaR_98125,
                     'VaR_9875': VaR_9875,
                     'VaR_9925': VaR_9925,
                     'VaR_99375': VaR_99375,
                     'VaR_995': VaR_995,
                     'VaR_9975': VaR_9975,
                     'ES_95': np.mean(scenarios.loc[scenarios['log_r_PV'] <= VaR_95,
                                                    'log_r_PV']),
                     'ES_975': np.mean(scenarios.loc[scenarios['log_r_PV'] <= VaR_975,
                                                     'log_r_PV']),
                     'ES_99': np.mean(scenarios.loc[scenarios['log_r_PV'] <= VaR_99,
                                                    'log_r_PV']),
                     'actual_log_r': index_price_df.loc[date_filter,
                                                        'log_r_PV'].head(1).iloc[0]
                     }

    return pd.Series(estimates_row).to_frame().T


estimates_HS_250 = pd.DataFrame()
estimates_HS_250 = VaR_ES_estimation(VaR_ES_HS, estimates_HS_250,
                                     250, index_prices_df)
plot_VaR_ES(estimates_HS_250, 'HS', 250, 95)
plot_VaR_ES(estimates_HS_250, 'HS', 250, 975, level_in_graph=97.5)
plot_VaR_ES(estimates_HS_250, 'HS', 250, 99)
estimates_HS_250.to_excel('Estimated_VaR_ES_HS_250d_median_unbiased.xlsx')

estimates_HS_500 = pd.DataFrame()
estimates_HS_500 = VaR_ES_estimation(VaR_ES_HS, estimates_HS_500,
                                     500, index_prices_df)
plot_VaR_ES(estimates_HS_500, 'HS', 500, 95, legend_loc='lower right')
plot_VaR_ES(estimates_HS_500, 'HS', 500, 975, level_in_graph=97.5,
            legend_loc='lower right')
plot_VaR_ES(estimates_HS_500, 'HS', 500, 99, legend_loc='lower right')
estimates_HS_500.to_excel('Estimated_VaR_ES_HS_500d_median_unbiased.xlsx')
```

```
estimates_HS_750 = pd.DataFrame()
estimates_HS_750 = VaR_ES_estimation(VaR_ES_HS, estimates_HS_750,
                                     750, index_prices_df)
plot_VaR_ES(estimates_HS_750, 'HS', 750, 95, legend_loc='lower right')
plot_VaR_ES(estimates_HS_750, 'HS', 750, 975, level_in_graph=97.5,
            legend_loc='lower right')
plot_VaR_ES(estimates_HS_750, 'HS', 750, 99, legend_loc='lower right')
estimates_HS_750.to_excel('Estimated_VaR_ES_HS_750d_median_unbiased.xlsx')

estimates_HS_1000 = pd.DataFrame()
estimates_HS_1000 = VaR_ES_estimation(VaR_ES_HS, estimates_HS_1000,
                                      1000, index_prices_df)
plot_VaR_ES(estimates_HS_1000, 'HS', 1000, 95, legend_loc='upper right')
plot_VaR_ES(estimates_HS_1000, 'HS', 1000, 975, level_in_graph=97.5,
            legend_loc='upper right', borderpad=0.2)
plot_VaR_ES(estimates_HS_1000, 'HS', 1000, 99, legend_loc='upper right', borderpad=0.15)
estimates_HS_1000.to_excel('Estimated_VaR_ES_HS_1000d_median_unbiased.xlsx')

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# -------- Age-weighted historical simulation -------- #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


def weigh_obs(n, i, lambd=0.98):
    weight = lambd**(n-i)*(1-lambd) / (1-lambd**n)
    return weight


def age_weighted_HS(sorted_df, measure_level, c_weights_col='cumul_weights',
                    ret_col='log_r_PV', ES=False, weights='weights'):

    if sorted_df.loc[sorted_df[c_weights_col] <= (1-measure_level), ret_col].empty:
        VaR_est = sorted_df[ret_col][0]
        # In case of very large risk measure levels, VaR is assumed to be equal to ES,
        # the same value will be returned for both measures
        print(f'Min logr ({VaR_est}) for VaR is used at level {measure_level}, ES?={ES}')

    elif not sorted_df.loc[sorted_df[c_weights_col] <= (1-measure_level), ret_col].empty:
        left_index = sorted_df.loc[sorted_df[c_weights_col] <=
                                   (1-measure_level), ret_col].idxmax()
        probs = sorted_df.loc[left_index:left_index+1, c_weights_col].values
        prob_ret = sorted_df.loc[left_index:left_index+1, ret_col].values
        VaR_est = np.interp((1-measure_level), probs, prob_ret)

        if ES:
            if sorted_df.loc[left_index, c_weights_col] < (1 - measure_level):
                ret_w = np.array(list(sorted_df.loc[:left_index, weights]) +
                                 [(1-measure_level) -
                                  np.sum(sorted_df.loc[:left_index, weights])])
                ret_val = np.array(list(sorted_df.loc[:left_index, ret_col]) + [VaR_est])
            elif (sorted_df.loc[left_index, c_weights_col] == (1 - measure_level)):
                ret_w = sorted_df.loc[:left_index, weights]
                ret_val = sorted_df.loc[:left_index, ret_col]
            ES_est = np.sum(ret_w*ret_val)/(1-measure_level)

            return ES_est

    return VaR_est


def VaR_ES_WHS(index_price_df, calibration_window, study_date):
    # (calibration_window + 1) is used in order to have
    # calibration_window number of returns:
```

```
    filtered_df = index_price_df[index_price_df['Date'] <=
                                 study_date].tail(calibration_window+1).copy()
    scenarios = pd.DataFrame()
    for ind in price_columns:
        # returns between i+1 and i multiplied by the nth price:
        nth_price = filtered_df[ind].tail(1).iloc[0]
        scenarios[ind] = (filtered_df[ind] / filtered_df[ind].shift(1)) * nth_price
    scenarios['PV'] = scenarios.mul([10] * 8).sum(axis=1)
    scenarios.dropna(inplace=True)
    # log returns between generated PVs and the actual PV on nth day:
    scenarios['log_r_PV'] = np.log(scenarios['PV'] / filtered_df['PV'].tail(1).iloc[0])

    weights = [weigh_obs(len(scenarios), i) for i in range(1, len(scenarios)+1)]
    # weights = [weigh_obs(len(scenarios), i) for i in scenarios.index.values]
    scenarios['weights'] = weights
    scenarios_sorted = scenarios.sort_values(by='log_r_PV').copy()
    scenarios_sorted['cumul_weights'] = scenarios_sorted['weights'].cumsum()
    scenarios_sorted.reset_index(inplace=True)

    VaR_95 = age_weighted_HS(scenarios_sorted, 0.95)
    VaR_975 = age_weighted_HS(scenarios_sorted, 0.975)
    VaR_99 = age_weighted_HS(scenarios_sorted, 0.99)
    # Additional estimates for multinomial backtesting:
    VaR_9625 = age_weighted_HS(scenarios_sorted, 0.9625)
    VaR_98125 = age_weighted_HS(scenarios_sorted, 0.98125)
    VaR_9875 = age_weighted_HS(scenarios_sorted, 0.9875)
    VaR_9925 = age_weighted_HS(scenarios_sorted, 0.9925)
    VaR_99375 = age_weighted_HS(scenarios_sorted, 0.99375)
    VaR_995 = age_weighted_HS(scenarios_sorted, 0.995)
    VaR_9975 = age_weighted_HS(scenarios_sorted, 0.9975)
    date_filter = (index_price_df['Date'] > study_date)
    estimates_row = {'forecast_date': index_price_df.loc[date_filter,
                                                         'Date'].head(1).iloc[0],
                     'VaR_95': VaR_95,
                     'VaR_975': VaR_975,
                     'VaR_99': VaR_99,
                     'VaR_9625': VaR_9625,
                     'VaR_98125': VaR_98125,
                     'VaR_9875': VaR_9875,
                     'VaR_9925': VaR_9925,
                     'VaR_99375': VaR_99375,
                     'VaR_995': VaR_995,
                     'VaR_9975': VaR_9975,
                     'ES_95': age_weighted_HS(scenarios_sorted, 0.95, ES=True),
                     'ES_975': age_weighted_HS(scenarios_sorted, 0.975, ES=True),
                     'ES_99': age_weighted_HS(scenarios_sorted, 0.99, ES=True),
                     'actual_log_r': index_price_df.loc[date_filter,
                                                        'log_r_PV'].head(1).iloc[0]
                     }

    return pd.Series(estimates_row).to_frame().T


estimates_WHS_250 = pd.DataFrame()
estimates_WHS_250 = VaR_ES_estimation(VaR_ES_WHS, estimates_WHS_250,
                                      250, index_prices_df)
plot_VaR_ES(estimates_WHS_250, 'AWHS', 250, 95, estimate_col='#510048')
plot_VaR_ES(estimates_WHS_250, 'AWHS', 250, 975, level_in_graph=97.5,
            estimate_col='#510048')
plot_VaR_ES(estimates_WHS_250, 'AWHS', 250, 99, estimate_col='#510048')
estimates_WHS_250.to_excel('Estimated_VaR_ES_WHS_250d.xlsx')

estimates_WHS_500 = pd.DataFrame()
```

```
estimates_WHS_500 = VaR_ES_estimation(VaR_ES_WHS, estimates_WHS_500,
                                      500, index_prices_df)
plot_VaR_ES(estimates_WHS_500, 'AWHS', 500, 95, estimate_col='#510048',
            legend_loc='lower right')
plot_VaR_ES(estimates_WHS_500, 'AWHS', 500, 975, level_in_graph=97.5,
            estimate_col='#510048', legend_loc='lower right')
plot_VaR_ES(estimates_WHS_500, 'AWHS', 500, 99, estimate_col='#510048',
            legend_loc='lower right')
estimates_WHS_500.to_excel('Estimated_VaR_ES_WHS_500d.xlsx')

estimates_WHS_750 = pd.DataFrame()
estimates_WHS_750 = VaR_ES_estimation(VaR_ES_WHS, estimates_WHS_750,
                                      750, index_prices_df)
plot_VaR_ES(estimates_WHS_750, 'AWHS', 750, 95, estimate_col='#510048',
            legend_loc='lower right')
plot_VaR_ES(estimates_WHS_750, 'AWHS', 750, 975, level_in_graph=97.5,
            estimate_col='#510048', legend_loc='lower right')
plot_VaR_ES(estimates_WHS_750, 'AWHS', 750, 99, estimate_col='#510048',
            legend_loc='lower right')
estimates_WHS_750.to_excel('Estimated_VaR_ES_WHS_750d.xlsx')

estimates_WHS_1000 = pd.DataFrame()
estimates_WHS_1000 = VaR_ES_estimation(VaR_ES_WHS, estimates_WHS_1000,
                                       1000, index_prices_df)
plot_VaR_ES(estimates_WHS_1000, 'AWHS', 1000, 95, estimate_col='#510048',
            legend_loc='upper right', borderpad=0.2)
plot_VaR_ES(estimates_WHS_1000, 'AWHS', 1000, 975, level_in_graph=97.5,
            estimate_col='#510048', legend_loc='upper right', borderpad=0.2)
plot_VaR_ES(estimates_WHS_1000, 'AWHS', 1000, 99, estimate_col='#510048',
            legend_loc='upper right', borderpad=0.2)
estimates_WHS_1000.to_excel('Estimated_VaR_ES_WHS_1000d.xlsx')

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# ------------ MNIG simulation ------------ #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


def VaR_ES_MNIG(return_df, calibration_window, study_date):

    filtered_df = return_df[return_df['Date'] <=
                            study_date].tail(calibration_window).copy()
    filtered_df.set_index('Date', inplace=True)
    with localconverter(ro.default_converter + pandas2ri.converter):
        filtered_df_R = ro.conversion.py2rpy(filtered_df)
    mnig_estimates = QRM.fit_mNH(filtered_df_R, symmetric=r("FALSE"), case="NIG")
    mNIGparams = ghyp.NIG(chi=mnig_estimates[0][1], psi=mnig_estimates[0][2],
                          mu=mnig_estimates[1], sigma=mnig_estimates[2],
                          gamma=mnig_estimates[3])
    logr_scenarios = ghyp.rghyp(n=10000, object=mNIGparams)
    logr_scenarios_df = pd.DataFrame(np.array(logr_scenarios), columns=logr_columns)

    PV_scenarios = pd.DataFrame()
    time_filter = (index_prices_df['Date'] <= study_date)
    for ind in logr_columns:
        # returns between i+1 and i multiplied by the nth price:
        nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
        PV_scenarios[ind] = np.exp(logr_scenarios_df[ind]) * nth_price
    PV_scenarios['PV'] = PV_scenarios.mul([10] * 8).sum(axis=1)
    # scenarios.dropna(inplace=True)
    # Log returns between generated PVs and the actual PV on nth day:
    nth_PV = index_prices_df.loc[time_filter, 'PV'].tail(1).iloc[0]
    PV_scenarios['log_r_PV'] = np.log(PV_scenarios['PV'] / nth_PV)
    est_q = 'median_unbiased'
```

```
    VaR_95 = np.percentile(PV_scenarios['log_r_PV'], 100-95, method=est_q)
    VaR_975 = np.percentile(PV_scenarios['log_r_PV'], 100-97.5, method=est_q)
    VaR_99 = np.percentile(PV_scenarios['log_r_PV'], 100-99, method=est_q)
    # Additional estimates for multinomial backtesting:
    VaR_9625 = np.percentile(PV_scenarios['log_r_PV'], 100-96.25, method=est_q)
    VaR_98125 = np.percentile(PV_scenarios['log_r_PV'], 100-98.125, method=est_q)
    VaR_9875 = np.percentile(PV_scenarios['log_r_PV'], 100-98.75, method=est_q)
    VaR_9925 = np.percentile(PV_scenarios['log_r_PV'], 100-99.25, method=est_q)
    VaR_99375 = np.percentile(PV_scenarios['log_r_PV'], 100-99.375, method=est_q)
    VaR_995 = np.percentile(PV_scenarios['log_r_PV'], 100-99.5, method=est_q)
    VaR_9975 = np.percentile(PV_scenarios['log_r_PV'], 100-99.75, method=est_q)
    date_filter = (index_prices_df['Date'] > study_date)
    estimates_row = {'forecast_date': return_df.loc[return_df['Date'] > study_date,
                                                    'Date'].head(1).iloc[0],
                     'VaR_95': VaR_95,
                     'VaR_975': VaR_975,
                     'VaR_99': VaR_99,
                     'VaR_9625': VaR_9625,
                     'VaR_98125': VaR_98125,
                     'VaR_9875': VaR_9875,
                     'VaR_9925': VaR_9925,
                     'VaR_99375': VaR_99375,
                     'VaR_995': VaR_995,
                     'VaR_9975': VaR_9975,
                     'ES_95': np.mean(PV_scenarios.loc[PV_scenarios['log_r_PV'] <=
                                                       VaR_95, 'log_r_PV']),
                     'ES_975': np.mean(PV_scenarios.loc[PV_scenarios['log_r_PV'] <=
                                                        VaR_975, 'log_r_PV']),
                     'ES_99': np.mean(PV_scenarios.loc[PV_scenarios['log_r_PV'] <=
                                                       VaR_99, 'log_r_PV']),
                     'actual_log_r': index_prices_df.loc[date_filter,
                                                         'log_r_PV'].head(1).iloc[0]
                     }

    return pd.Series(estimates_row).to_frame().T


estimates_MNIG_250 = pd.DataFrame()
estimates_MNIG_250 = VaR_ES_estimation(VaR_ES_MNIG, estimates_MNIG_250,
                                       250, index_prices_df=index_prices_df,
                                       return_df=log_returns_df, method_in_R=True)
plot_VaR_ES(estimates_MNIG_250, 'MNIG', 250, 95, estimate_col='#ff4f00')
plot_VaR_ES(estimates_MNIG_250, 'MNIG', 250, 975, level_in_graph=97.5,
            estimate_col='#ff4f00')
plot_VaR_ES(estimates_MNIG_250, 'MNIG', 250, 99, estimate_col='#ff4f00')
estimates_MNIG_250.to_excel('Estimated_VaR_ES_MNIG_250d.xlsx')

estimates_MNIG_500 = pd.DataFrame()
estimates_MNIG_500 = VaR_ES_estimation(VaR_ES_MNIG, estimates_MNIG_500,
                                       500, index_prices_df=index_prices_df,
                                       return_df=log_returns_df, method_in_R=True)
plot_VaR_ES(estimates_MNIG_500, 'MNIG', 500, 95, estimate_col='#ff4f00',
            legend_loc='lower right')
plot_VaR_ES(estimates_MNIG_500, 'MNIG', 500, 975, level_in_graph=97.5,
            estimate_col='#ff4f00', legend_loc='lower right')
plot_VaR_ES(estimates_MNIG_500, 'MNIG', 500, 99, estimate_col='#ff4f00',
            legend_loc='lower right')
estimates_MNIG_500.to_excel('Estimated_VaR_ES_MNIG_500d.xlsx')

estimates_MNIG_750 = pd.DataFrame()
estimates_MNIG_750 = VaR_ES_estimation(VaR_ES_MNIG, estimates_MNIG_750,
                                       750, index_prices_df=index_prices_df,
                                       return_df=log_returns_df, method_in_R=True)
```

```
plot_VaR_ES(estimates_MNIG_750, 'MNIG', 750, 95, estimate_col='#ff4f00',
            legend_loc='lower right')
plot_VaR_ES(estimates_MNIG_750, 'MNIG', 750, 975, level_in_graph=97.5,
            estimate_col='#ff4f00', legend_loc='lower right')
plot_VaR_ES(estimates_MNIG_750, 'MNIG', 750, 99, estimate_col='#ff4f00',
            legend_loc='lower right')
estimates_MNIG_750.to_excel('Estimated_VaR_ES_MNIG_750d.xlsx')

estimates_MNIG_1000 = pd.DataFrame()
estimates_MNIG_1000 = VaR_ES_estimation(VaR_ES_MNIG, estimates_MNIG_1000,
                                        1000, index_prices_df=index_prices_df,
                                        return_df=log_returns_df, method_in_R=True)
plot_VaR_ES(estimates_MNIG_1000, 'MNIG', 1000, 95, estimate_col='#ff4f00',
            legend_loc='upper right', borderpad=0.2)
plot_VaR_ES(estimates_MNIG_1000, 'MNIG', 1000, 975, level_in_graph=97.5,
            estimate_col='#ff4f00', legend_loc='upper right', borderpad=0.2)
plot_VaR_ES(estimates_MNIG_1000, 'MNIG', 1000, 99, estimate_col='#ff4f00',
            legend_loc='upper right', borderpad=0.2)
estimates_MNIG_1000.to_excel('Estimated_VaR_ES_MNIG_1000d.xlsx')

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# ----------- Copulas ----------- #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


# Function for collecting fitted parameters of marginal distributions:
def p_vector(fit):
    param_vector = base.c(
        fit.rx2("param").rx2("mu"),
        fit.rx2("param").rx2("delta"),
        fit.rx2("param").rx2("alpha"),
        fit.rx2("param").rx2("beta"),
        -1 / 2,  # lambda = -1/2 for NIG distribution
    )
    return param_vector


def fit_garch(df, col):
    # AR(1)-GARCH(1,1) specification:
    arma_spec = base.list(armaOrder=base.c(1, 0))
    garch_spec = base.list(model="sGARCH", garchOrder=base.c(1, 1))
    fit_r = rg.ugarchspec(mean_model=arma_spec,
                          variance_model=garch_spec,
                          distribution_model="nig")
    try:
        res = rg.ugarchfit(data=df.rx2(col), spec=fit_r, solver='hybrid')
    except RRuntimeError as e:
        msg = str(e)
        print(e)
        if 'Error in robustvcv' in msg:
            print(f'Could not fit an AR(1)-GARCH(1,1) model for {col}')
            autoOrder = rg.autoarfima(data=df.rx2(col), criterion='AIC',
                                      include_mean=r('TRUE'), method='partial',
                                      distribution_model='nig')
            auto_fit = autoOrder.rx2('fit')
            ar_p = auto_fit.slots['model'].rx2('modelinc').rx2('ar')
            ma_q = auto_fit.slots['model'].rx2('modelinc').rx2('ma')
            arma_spec = base.list(armaOrder=base.c(ar_p, ma_q))
            fit_r_upd = rg.ugarchspec(mean_model=arma_spec,
                                      variance_model=garch_spec,
                                      distribution_model="nig")
            try:
                print(f'Fitting ARMA({str(ar_p[0])[0]},{str(ma_q[0])[0]})-GARCH(1,1):')
```

```
                res = rg.ugarchfit(data=df.rx2(col), spec=fit_r_upd, solver='hybrid')
            except Exception as e:
                print(e)
                print('Automatic order failed, fitting a constant model:')
                arma_spec = base.list(armaOrder=base.c(0, 0))
                garch_spec = base.list(model="sGARCH", garchOrder=base.c(0, 0))
                fit_r_c = rg.ugarchspec(mean_model=arma_spec,
                                        variance_model=garch_spec,
                                        distribution_model="nig")
                res = rg.ugarchfit(data=df.rx2(col), spec=fit_r_c, solver='hybrid')
    except Exception as e:
        print(e)
        print('Fitting a constant model:')
        arma_spec = base.list(armaOrder=base.c(0, 0))
        garch_spec = base.list(model="sGARCH", garchOrder=base.c(0, 0))
        fit_r_c = rg.ugarchspec(mean_model=arma_spec,
                                variance_model=garch_spec,
                                distribution_model="nig")
        res = rg.ugarchfit(data=df.rx2(col), spec=fit_r_c, solver='hybrid')

    return res


def VaR_ES_Copula(return_df, calibration_window, study_date, type='Clayton', nsim=10000):

    filtered_df = return_df[return_df['Date'] <=
                            study_date].tail(calibration_window).copy()
    filtered_df.set_index('Date', inplace=True)
    with localconverter(ro.default_converter + pandas2ri.converter):
        filtered_df_R = ro.conversion.py2rpy(filtered_df)

    # Estimating GARCH models:
    garch_vgi = fit_garch(filtered_df_R, 'logr_vgi_eur')
    garch_rgi = fit_garch(filtered_df_R, 'logr_rgi_eur')
    garch_tgi = fit_garch(filtered_df_R, 'logr_tgi_eur')
    garch_s30gi = fit_garch(filtered_df_R, 'logr_s30gi_eur')
    garch_igi = fit_garch(filtered_df_R, 'logr_igi_eur')
    garch_h25gi = fit_garch(filtered_df_R, 'logr_h25gi_eur')
    garch_c25gi = fit_garch(filtered_df_R, 'logr_c25gi_eur')
    garch_obx = fit_garch(filtered_df_R, 'logr_obx_eur')

    # Extracting one-day conditional mean and volatility predictions:
    forecast_vgi = rg.ugarchforecast(garch_vgi, n_ahead=1)
    mu_vgi = base.as_numeric(forecast_vgi.slots['forecast'].rx2('seriesFor'))
    sigma_vgi = base.as_numeric(forecast_vgi.slots['forecast'].rx2('sigmaFor'))

    forecast_rgi = rg.ugarchforecast(garch_rgi, n_ahead=1)
    mu_rgi = base.as_numeric(forecast_rgi.slots['forecast'].rx2('seriesFor'))
    sigma_rgi = base.as_numeric(forecast_rgi.slots['forecast'].rx2('sigmaFor'))

    forecast_tgi = rg.ugarchforecast(garch_tgi, n_ahead=1)
    mu_tgi = base.as_numeric(forecast_tgi.slots['forecast'].rx2('seriesFor'))
    sigma_tgi = base.as_numeric(forecast_tgi.slots['forecast'].rx2('sigmaFor'))

    forecast_s30gi = rg.ugarchforecast(garch_s30gi, n_ahead=1)
    mu_s30gi = base.as_numeric(forecast_s30gi.slots['forecast'].rx2('seriesFor'))
    sigma_s30gi = base.as_numeric(forecast_s30gi.slots['forecast'].rx2('sigmaFor'))

    forecast_igi = rg.ugarchforecast(garch_igi, n_ahead=1)
    mu_igi = base.as_numeric(forecast_igi.slots['forecast'].rx2('seriesFor'))
    sigma_igi = base.as_numeric(forecast_igi.slots['forecast'].rx2('sigmaFor'))

    forecast_h25gi = rg.ugarchforecast(garch_h25gi, n_ahead=1)
```

```
mu_h25gi = base.as_numeric(forecast_h25gi.slots['forecast'].rx2('seriesFor'))
sigma_h25gi = base.as_numeric(forecast_h25gi.slots['forecast'].rx2('sigmaFor'))

forecast_c25gi = rg.ugarchforecast(garch_c25gi, n_ahead=1)
mu_c25gi = base.as_numeric(forecast_c25gi.slots['forecast'].rx2('seriesFor'))
sigma_c25gi = base.as_numeric(forecast_c25gi.slots['forecast'].rx2('sigmaFor'))

forecast_obx = rg.ugarchforecast(garch_obx, n_ahead=1)
mu_obx = base.as_numeric(forecast_obx.slots['forecast'].rx2('seriesFor'))
sigma_obx = base.as_numeric(forecast_obx.slots['forecast'].rx2('sigmaFor'))
# These values will be used for calculating simulated log returns:
# log_r = mu_index + sigma_index * simulated residual

# Getting the standardised residuals:
with localconverter(ro.default_converter + pandas2ri.converter):
    sr_vgi = FMat(np.divide(rg.residuals(garch_vgi), rg.sigma(garch_vgi)))
    sr_rgi = FMat(np.divide(rg.residuals(garch_rgi), rg.sigma(garch_rgi)))
    sr_tgi = FMat(np.divide(rg.residuals(garch_tgi), rg.sigma(garch_tgi)))
    sr_s30gi = FMat(np.divide(rg.residuals(garch_s30gi), rg.sigma(garch_s30gi)))
    sr_igi = FMat(np.divide(rg.residuals(garch_igi), rg.sigma(garch_igi)))
    sr_h25gi = FMat(np.divide(rg.residuals(garch_h25gi), rg.sigma(garch_h25gi)))
    sr_c25gi = FMat(np.divide(rg.residuals(garch_c25gi), rg.sigma(garch_c25gi)))
    sr_obx = FMat(np.divide(rg.residuals(garch_obx), rg.sigma(garch_obx)))

# Fitting the NIG distribution to marginals:
p_nig_vgi = genhyp.nigFit(sr_vgi)
p_nig_rgi = genhyp.nigFit(sr_rgi)
p_nig_tgi = genhyp.nigFit(sr_tgi)
p_nig_s30gi = genhyp.nigFit(sr_s30gi)
p_nig_igi = genhyp.nigFit(sr_igi)
p_nig_h25gi = genhyp.nigFit(sr_h25gi)
p_nig_c25gi = genhyp.nigFit(sr_c25gi)
p_nig_obx = genhyp.nigFit(sr_obx)

# Converting to uniform marginals:
unif_vgi = genhyp.pghyp(sr_vgi, param=p_vector(p_nig_vgi))
unif_rgi = genhyp.pghyp(sr_rgi, param=p_vector(p_nig_rgi))
unif_tgi = genhyp.pghyp(sr_tgi, param=p_vector(p_nig_tgi))
unif_s30gi = genhyp.pghyp(sr_s30gi, param=p_vector(p_nig_s30gi))
unif_igi = genhyp.pghyp(sr_igi, param=p_vector(p_nig_igi))
unif_h25gi = genhyp.pghyp(sr_h25gi, param=p_vector(p_nig_h25gi))
unif_c25gi = genhyp.pghyp(sr_c25gi, param=p_vector(p_nig_c25gi))
unif_obx = genhyp.pghyp(sr_obx, param=p_vector(p_nig_obx))

unif_marg = base.cbind(unif_vgi=unif_vgi, unif_rgi=unif_rgi, unif_tgi=unif_tgi,
                       unif_s30gi=unif_s30gi, unif_igi=unif_igi,
                       unif_h25gi=unif_h25gi, unif_c25gi=unif_c25gi,
                       unif_obx=unif_obx)
unif_mat = base.as_matrix(unif_marg)

# Fitting copulas and simulating from them:
if type == 'Clayton':
    fit_HAC_Clayton = HAC.estimate_copula(unif_mat, type=3)
    hac_obj = HAC.hac(type=3, tree=fit_HAC_Clayton.rx2('tree'))
elif type == 'Frank':
    fit_HAC_Frank = HAC.estimate_copula(unif_mat, type=5)
    hac_obj = HAC.hac(type=5, tree=fit_HAC_Frank.rx2('tree'))

sim_unif = HAC.rHAC(nsim, hac_obj)
sim_unif_df = base.as_data_frame(sim_unif)

# Converting the simulated values to standardised residuals:
sim_vgi = genhyp.qghyp(sim_unif_df.rx2("unif_vgi"), param=p_vector(p_nig_vgi))
```

```
sim_rgi = genhyp.qghyp(sim_unif_df.rx2("unif_rgi"), param=p_vector(p_nig_rgi))
sim_tgi = genhyp.qghyp(sim_unif_df.rx2("unif_tgi"), param=p_vector(p_nig_tgi))
sim_s30gi = genhyp.qghyp(sim_unif_df.rx2("unif_s30gi"), param=p_vector(p_nig_s30gi))
sim_igi = genhyp.qghyp(sim_unif_df.rx2("unif_igi"), param=p_vector(p_nig_igi))
sim_h25gi = genhyp.qghyp(sim_unif_df.rx2("unif_h25gi"), param=p_vector(p_nig_h25gi))
sim_c25gi = genhyp.qghyp(sim_unif_df.rx2("unif_c25gi"), param=p_vector(p_nig_c25gi))
sim_obx = genhyp.qghyp(sim_unif_df.rx2("unif_obx"), param=p_vector(p_nig_obx))

# Simulated log returns:
res_vgi = r['+'](r['*'](sim_vgi, sigma_vgi), mu_vgi)
res_rgi = r['+'](r['*'](sim_rgi, sigma_rgi), mu_rgi)
res_tgi = r['+'](r['*'](sim_tgi, sigma_tgi), mu_tgi)
res_s30gi = r['+'](r['*'](sim_s30gi, sigma_s30gi), mu_s30gi)
res_igi = r['+'](r['*'](sim_igi, sigma_igi), mu_igi)
res_h25gi = r['+'](r['*'](sim_h25gi, sigma_h25gi), mu_h25gi)
res_c25gi = r['+'](r['*'](sim_c25gi, sigma_c25gi), mu_c25gi)
res_obx = r['+'](r['*'](sim_obx, sigma_obx), mu_obx)
logr_scenarios = base.cbind(logr_vgi_eur=res_vgi, logr_rgi_eur=res_rgi,
                            logr_tgi_eur=res_tgi, logr_s30gi_eur=res_s30gi,
                            logr_igi_eur=res_igi, logr_h25gi_eur=res_h25gi,
                            logr_c25gi_eur=res_c25gi, logr_obx_eur=res_obx)

logr_scenarios_df = pd.DataFrame(np.array(logr_scenarios), columns=logr_columns)

PV_scenarios = pd.DataFrame()
time_filter = (index_prices_df['Date'] <= study_date)
for ind in logr_columns:
    # returns between i+1 and i multiplied by the nth price:
    nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
    PV_scenarios[ind] = np.exp(logr_scenarios_df[ind]) * nth_price
PV_scenarios['PV'] = PV_scenarios.mul([10] * 8).sum(axis=1)
# Log returns between generated PVs and the actual PV on nth day:
nth_PV = index_prices_df.loc[time_filter, 'PV'].tail(1).iloc[0]
PV_scenarios['log_r_PV'] = np.log(PV_scenarios['PV'] / nth_PV)
est_q = 'median_unbiased'
VaR_95 = np.percentile(PV_scenarios['log_r_PV'], 100-95, method=est_q)
VaR_975 = np.percentile(PV_scenarios['log_r_PV'], 100-97.5, method=est_q)
VaR_99 = np.percentile(PV_scenarios['log_r_PV'], 100-99, method=est_q)
# Additional estimates for multinomial backtesting:
VaR_9625 = np.percentile(PV_scenarios['log_r_PV'], 100-96.25, method=est_q)
VaR_98125 = np.percentile(PV_scenarios['log_r_PV'], 100-98.125, method=est_q)
VaR_9875 = np.percentile(PV_scenarios['log_r_PV'], 100-98.75, method=est_q)
VaR_9925 = np.percentile(PV_scenarios['log_r_PV'], 100-99.25, method=est_q)
VaR_99375 = np.percentile(PV_scenarios['log_r_PV'], 100-99.375, method=est_q)
VaR_995 = np.percentile(PV_scenarios['log_r_PV'], 100-99.5, method=est_q)
VaR_9975 = np.percentile(PV_scenarios['log_r_PV'], 100-99.75, method=est_q)
date_filter = (index_prices_df['Date'] > study_date)
estimates_row = {
    'forecast_date': return_df.loc[return_df['Date'] > study_date,
                                   'Date'].head(1).iloc[0],
    'VaR_95': VaR_95,
    'VaR_975': VaR_975,
    'VaR_99': VaR_99,
    'VaR_9625': VaR_9625,
    'VaR_98125': VaR_98125,
    'VaR_9875': VaR_9875,
    'VaR_9925': VaR_9925,
    'VaR_99375': VaR_99375,
    'VaR_995': VaR_995,
    'VaR_9975': VaR_9975,
    'ES_95': np.mean(PV_scenarios.loc[PV_scenarios['log_r_PV'] <= VaR_95,
                                      'log_r_PV']),
    'ES_975': np.mean(PV_scenarios.loc[PV_scenarios['log_r_PV'] <= VaR_975,
```

```
                                          'log_r_PV']),
        'ES_99': np.mean(PV_scenarios.loc[PV_scenarios['log_r_PV'] <= VaR_99,
                                          'log_r_PV']),
        'actual_log_r': index_prices_df.loc[date_filter, 'log_r_PV'].head(1).iloc[0]
        }

    return pd.Series(estimates_row).to_frame().T


# Hierarchical Clayton copula:
estimates_Clayton_250 = pd.DataFrame()
start = timer()
estimates_Clayton_250 = VaR_ES_estimation(VaR_ES_Copula, estimates_Clayton_250,
                                          250, index_prices_df=index_prices_df,
                                          return_df=log_returns_df, method_in_R=True,
                                          c_type='Clayton')
print(f'{timer()-start} seconds passed')  # 13325 seconds -> ~3.7 hours
plot_VaR_ES(estimates_Clayton_250, 'HCC', 250, 95, estimate_col='#0B81B3')
plot_VaR_ES(estimates_Clayton_250, 'HCC', 250, 975, level_in_graph=97.5,
            estimate_col='#0B81B3')
plot_VaR_ES(estimates_Clayton_250, 'HCC', 250, 99, estimate_col='#0B81B3')
estimates_Clayton_250.to_excel('Estimated_VaR_ES_Copula_Clayton_250d.xlsx')

estimates_Clayton_500 = pd.DataFrame()
start = timer()
estimates_Clayton_500 = VaR_ES_estimation(VaR_ES_Copula, estimates_Clayton_500,
                                          500, index_prices_df=index_prices_df,
                                          return_df=log_returns_df, method_in_R=True,
                                          c_type='Clayton')
print(f'{timer()-start} seconds passed')
plot_VaR_ES(estimates_Clayton_500, 'HCC', 500, 95, estimate_col='#0B81B3',
            legend_loc='lower right')
plot_VaR_ES(estimates_Clayton_500, 'HCC', 500, 975, level_in_graph=97.5,
            estimate_col='#0B81B3', legend_loc='lower right')
plot_VaR_ES(estimates_Clayton_500, 'HCC', 500, 99, estimate_col='#0B81B3',
            legend_loc='lower right')
estimates_Clayton_500.to_excel('Estimated_VaR_ES_Copula_Clayton_500d.xlsx')

estimates_Clayton_750 = pd.DataFrame()
start = timer()
estimates_Clayton_750 = VaR_ES_estimation(VaR_ES_Copula, estimates_Clayton_750,
                                          750, index_prices_df=index_prices_df,
                                          return_df=log_returns_df, method_in_R=True,
                                          c_type='Clayton')
print(f'{timer()-start} seconds passed')   # 13204.92 seconds -> ~3.67 hours
plot_VaR_ES(estimates_Clayton_750, 'HCC', 750, 95, estimate_col='#0B81B3',
            legend_loc='lower right')
plot_VaR_ES(estimates_Clayton_750, 'HCC', 750, 975, level_in_graph=97.5,
            estimate_col='#0B81B3', legend_loc='lower right')
plot_VaR_ES(estimates_Clayton_750, 'HCC', 750, 99, estimate_col='#0B81B3',
            legend_loc='lower right')
estimates_Clayton_750.to_excel('Estimated_VaR_ES_Copula_Clayton_750d.xlsx')

estimates_Clayton_1000 = pd.DataFrame()
start = timer()
estimates_Clayton_1000 = VaR_ES_estimation(VaR_ES_Copula, estimates_Clayton_1000,
                                           1000, index_prices_df=index_prices_df,
                                           return_df=log_returns_df, method_in_R=True,
                                           c_type='Clayton')
print(f'{timer()-start} seconds passed')  # 13388.5 seconds -> ~3.72 hours
plot_VaR_ES(estimates_Clayton_1000, 'HCC', 1000, 95, estimate_col='#0B81B3',
            legend_loc='lower right', borderpad=0.3)
plot_VaR_ES(estimates_Clayton_1000, 'HCC', 1000, 975, level_in_graph=97.5,
```

```
            estimate_col='#0B81B3', legend_loc='lower right')
plot_VaR_ES(estimates_Clayton_1000, 'HCC', 1000, 99, estimate_col='#0B81B3',
            legend_loc='lower right')
estimates_Clayton_1000.to_excel('Estimated_VaR_ES_Copula_Clayton_1000d.xlsx')

# Hierarchical Frank copula:
estimates_Frank_250 = pd.DataFrame()
start = timer()
estimates_Frank_250 = VaR_ES_estimation(VaR_ES_Copula, estimates_Frank_250,
                                        250, index_prices_df=index_prices_df,
                                        return_df=log_returns_df, method_in_R=True,
                                        c_type='Frank')
print(f'{timer()-start} seconds passed')  # 14437.22 seconds -> ~4 hours
plot_VaR_ES(estimates_Frank_250, 'HFC', 250, 95, estimate_col='#18492E')
plot_VaR_ES(estimates_Frank_250, 'HFC', 250, 975, level_in_graph=97.5,
            estimate_col='#18492E')
plot_VaR_ES(estimates_Frank_250, 'HFC', 250, 99, estimate_col='#18492E')
estimates_Frank_250.to_excel('Estimated_VaR_ES_Copula_Frank_250d.xlsx')

estimates_Frank_500 = pd.DataFrame()
start = timer()
estimates_Frank_500 = VaR_ES_estimation(VaR_ES_Copula, estimates_Frank_500,
                                        500, index_prices_df=index_prices_df,
                                        return_df=log_returns_df, method_in_R=True,
                                        c_type='Frank')
print(f'{timer()-start} seconds passed')
plot_VaR_ES(estimates_Frank_500, 'HFC', 500, 95, estimate_col='#18492E',
            legend_loc='lower right')
plot_VaR_ES(estimates_Frank_500, 'HFC', 500, 975, level_in_graph=97.5,
            estimate_col='#18492E', legend_loc='lower right')
plot_VaR_ES(estimates_Frank_500, 'HFC', 500, 99, estimate_col='#18492E',
            legend_loc='lower right')
estimates_Frank_500.to_excel('Estimated_VaR_ES_Copula_Frank_500d.xlsx')

estimates_Frank_750 = pd.DataFrame()
start = timer()
estimates_Frank_750 = VaR_ES_estimation(VaR_ES_Copula, estimates_Frank_750,
                                        750, index_prices_df=index_prices_df,
                                        return_df=log_returns_df, method_in_R=True,
                                        c_type='Frank')
print(f'{timer()-start} seconds passed')
plot_VaR_ES(estimates_Frank_750, 'HFC', 750, 95, estimate_col='#18492E',
            legend_loc='lower right')
plot_VaR_ES(estimates_Frank_750, 'HFC', 750, 975, level_in_graph=97.5,
            estimate_col='#18492E', legend_loc='lower right')
plot_VaR_ES(estimates_Frank_750, 'HFC', 750, 99, estimate_col='#18492E',
            legend_loc='lower right')
estimates_Frank_750.to_excel('Estimated_VaR_ES_Copula_Frank_750d.xlsx')

estimates_Frank_1000 = pd.DataFrame()
start = timer()
estimates_Frank_1000 = VaR_ES_estimation(VaR_ES_Copula, estimates_Frank_1000,
                                         1000, index_prices_df=index_prices_df,
                                         return_df=log_returns_df, method_in_R=True,
                                         c_type='Frank')
print(f'{timer()-start} seconds passed')  # 12823 seconds -> ~3.6 hours
plot_VaR_ES(estimates_Frank_1000, 'HFC', 1000, 95, estimate_col='#18492E',
            legend_loc='upper right', borderpad=0.15)
plot_VaR_ES(estimates_Frank_1000, 'HFC', 1000, 975, level_in_graph=97.5,
            estimate_col='#18492E', legend_loc='lower right', borderpad=0.1)
plot_VaR_ES(estimates_Frank_1000, 'HFC', 1000, 99, estimate_col='#18492E',
            legend_loc='lower right', borderpad=0.1)
estimates_Frank_1000.to_excel('Estimated_VaR_ES_Copula_Frank_1000d.xlsx')
```

```
# ------------ Collecting results ----------- #

df_dict_250 = {
    'HS': estimates_HS_250,
    'AWHS': estimates_WHS_250,
    'MNIG': estimates_MNIG_250,
    'HCC': estimates_Clayton_250,
    'HFC': estimates_Frank_250,
}
df_dict_500 = {
    'HS': estimates_HS_500,
    'AWHS': estimates_WHS_500,
    'MNIG': estimates_MNIG_500,
    'HCC': estimates_Clayton_500,
    'HFC': estimates_Frank_500,
}
df_dict_750 = {
    'HS': estimates_HS_750,
    'AWHS': estimates_WHS_750,
    'MNIG': estimates_MNIG_750,
    'HCC': estimates_Clayton_750,
    'HFC': estimates_Frank_750,
}
df_dict_1000 = {
    'HS': estimates_HS_1000,
    'AWHS': estimates_WHS_1000,
    'MNIG': estimates_MNIG_1000,
    'HCC': estimates_Clayton_1000,
    'HFC': estimates_Frank_1000,
}

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# ------------ VaR Backtesting ------------ #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


def basel_traffic_light(no_of_exc, sample_size, VaR_level):
    cum_prob = binom.cdf(no_of_exc, sample_size, 1-VaR_level)
    if cum_prob < 0.95:
        zone = 'green'
    elif (cum_prob >= 0.95) & (cum_prob < 0.9999):
        zone = 'yellow'
    elif cum_prob >= 0.9999:
        zone = 'red'
    return cum_prob, zone


def basel_TL_res(df, model, window):
    res_95 = basel_traffic_light(np.sum(df['VaR_95_breach']), len(df), 0.95)
    res_975 = basel_traffic_light(np.sum(df['VaR_975_breach']), len(df), 0.975)
    res_99 = basel_traffic_light(np.sum(df['VaR_99_breach']), len(df), 0.99)
    return {
        'mod_w_level': [f'{model}_{window}_95', f'{model}_{window}_975',
                        f'{model}_{window}_99'],
        'cum_probs': [res_95[0], res_975[0], res_99[0]],
        'color': [res_95[1], res_975[1], res_99[1]]
        }


bt_basel_250 = pd.DataFrame()
for df in df_dict_250:
```

```
    res = basel_TL_res(df_dict_250[df], df, 250)
    bt_basel_250 = pd.concat([bt_basel_250, pd.DataFrame(res)])
bt_basel_250.to_excel('backtesting_results_Basel_TL_250.xlsx')

bt_basel_500 = pd.DataFrame()
for df in df_dict_500:
    res = basel_TL_res(df_dict_500[df], df, 500)
    bt_basel_500 = pd.concat([bt_basel_500, pd.DataFrame(res)])
bt_basel_500.to_excel('backtesting_results_Basel_TL_500.xlsx')

bt_basel_750 = pd.DataFrame()
for df in df_dict_750:
    res = basel_TL_res(df_dict_750[df], df, 750)
    bt_basel_750 = pd.concat([bt_basel_750, pd.DataFrame(res)])
bt_basel_750.to_excel('backtesting_results_Basel_TL_750.xlsx')

bt_basel_1000 = pd.DataFrame()
for df in df_dict_1000:
    res = basel_TL_res(df_dict_1000[df], df, 1000)
    bt_basel_1000 = pd.concat([bt_basel_1000, pd.DataFrame(res)])
bt_basel_1000.to_excel('backtesting_results_Basel_TL_1000.xlsx')


def kupiec_backtest(N, T, VaR_level, test_level=0.95):
    # N - number of exceptions
    # T - sample size
    p = 1 - VaR_level
    # p - 1-VaR level
    LR_uc = -2 * np.log((1-p)**(T-N) * p**N) + 2 * np.log((1-(N/T))**(T-N) * (N/T)**N)
    if LR_uc >= chi2.ppf(test_level, 1):
        result = 'fail'
    else:
        result = 'pass'
    p_val = 1 - chi2.cdf(LR_uc, 1)
    return result, LR_uc, p_val


def kupiec_res(df, model, window):
    res_95 = kupiec_backtest(np.sum(df['VaR_95_breach']), len(df), 0.95)
    res_975 = kupiec_backtest(np.sum(df['VaR_975_breach']), len(df), 0.975)
    res_99 = kupiec_backtest(np.sum(df['VaR_99_breach']), len(df), 0.99)
    return {
        'mod_w_level': [f'{model}_{window}_95', f'{model}_{window}_975',
                        f'{model}_{window}_99'],
        'result': [res_95[0], res_975[0], res_99[0]],
        'LR_uc': [res_95[1], res_975[1], res_99[1]],
        'p_val': [res_95[2], res_975[2], res_99[2]],
        }


bt_kupiec_250 = pd.DataFrame()
for df in df_dict_250:
    res = kupiec_res(df_dict_250[df], df, 250)
    bt_kupiec_250 = pd.concat([bt_kupiec_250, pd.DataFrame(res)])
bt_kupiec_250.to_excel('backtesting_results_Kupiec_250.xlsx')

bt_kupiec_500 = pd.DataFrame()
for df in df_dict_500:
    res = kupiec_res(df_dict_500[df], df, 500)
    bt_kupiec_500 = pd.concat([bt_kupiec_500, pd.DataFrame(res)])
bt_kupiec_500.to_excel('backtesting_results_Kupiec_500.xlsx')

bt_kupiec_750 = pd.DataFrame()
```

```
for df in df_dict_750:
    res = kupiec_res(df_dict_750[df], df, 750)
    bt_kupiec_750 = pd.concat([bt_kupiec_750, pd.DataFrame(res)])
bt_kupiec_750.to_excel('backtesting_results_Kupiec_750.xlsx')

bt_kupiec_1000 = pd.DataFrame()
for df in df_dict_1000:
    res = kupiec_res(df_dict_1000[df], df, 1000)
    bt_kupiec_1000 = pd.concat([bt_kupiec_1000, pd.DataFrame(res)])
bt_kupiec_1000.to_excel('backtesting_results_Kupiec_1000.xlsx')


def mixed_kupiec_backtest(data, exception_col, date_col, VaR_level, test_level=0.95):

    p = 1 - VaR_level
    exception_dates = data[data[exception_col] == 1][[date_col, exception_col]].copy()
    except_col_ind = exception_dates.columns.get_loc(exception_col)
    first_exc_date = exception_dates.iloc[0][date_col]
    exception_dates.iloc[0, except_col_ind] = len(data[data[date_col] < first_exc_date])
    previous_date = first_exc_date

    for curr_date in exception_dates[date_col][1:]:
        date_index = (exception_dates[date_col] == curr_date)
        day_distance = len(data[(data[date_col] < curr_date) &
                                (data[date_col] >= previous_date)])
        exception_dates.loc[date_index, exception_col] = day_distance
        previous_date = curr_date

    nu_1 = (exception_dates[exception_col]-1).copy()
    nu_div = (1/exception_dates[exception_col]).copy()

    # LR_TUFF = -2 * np.log((p*(1-p)**(nu_1.iloc[0])) /
    #                       (nu_div.iloc[0]*(1-nu_div.iloc[0])**(nu_1.iloc[0])))
    LR_ind = np.sum(-2 * np.log((p*(1-p)**(nu_1)) / (nu_div * (1-nu_div)**nu_1)))
    LR_pof = kupiec_backtest(len(exception_dates), len(data), VaR_level)[1]
    LR_mix = LR_ind + LR_pof
    if LR_mix >= chi2.ppf(test_level, len(exception_dates)+1):
        result = 'fail'
    else:
        result = 'pass'
    p_val = 1 - chi2.cdf(LR_mix, len(exception_dates)+1)
    return result, LR_mix, p_val


def mixed_kupiec_res(df, model, window):
    res_95 = mixed_kupiec_backtest(df, 'VaR_95_breach', 'forecast_date', 0.95)
    res_975 = mixed_kupiec_backtest(df, 'VaR_975_breach', 'forecast_date', 0.975)
    res_99 = mixed_kupiec_backtest(df, 'VaR_99_breach', 'forecast_date', 0.99)
    return {
        'mod_w_level': [f'{model}_{window}_95', f'{model}_{window}_975',
                        f'{model}_{window}_99'],
        'result': [res_95[0], res_975[0], res_99[0]],
        'LR_mix': [res_95[1], res_975[1], res_99[1]],
        'p_val': [res_95[2], res_975[2], res_99[2]],
        }


bt_mixed_kupiec_250 = pd.DataFrame()
for df in df_dict_250:
    res = mixed_kupiec_res(df_dict_250[df], df, 250)
    bt_mixed_kupiec_250 = pd.concat([bt_mixed_kupiec_250, pd.DataFrame(res)])
bt_mixed_kupiec_250.to_excel('backtesting_results_mixed_Kupiec_250.xlsx')
```

```
bt_mixed_kupiec_500 = pd.DataFrame()
for df in df_dict_500:
    res = mixed_kupiec_res(df_dict_500[df], df, 500)
    bt_mixed_kupiec_500 = pd.concat([bt_mixed_kupiec_500, pd.DataFrame(res)])
bt_mixed_kupiec_500.to_excel('backtesting_results_mixed_Kupiec_500.xlsx')

bt_mixed_kupiec_750 = pd.DataFrame()
for df in df_dict_750:
    res = mixed_kupiec_res(df_dict_750[df], df, 750)
    bt_mixed_kupiec_750 = pd.concat([bt_mixed_kupiec_750, pd.DataFrame(res)])
bt_mixed_kupiec_750.to_excel('backtesting_results_mixed_Kupiec_750.xlsx')

bt_mixed_kupiec_1000 = pd.DataFrame()
for df in df_dict_1000:
    res = mixed_kupiec_res(df_dict_1000[df], df, 1000)
    bt_mixed_kupiec_1000 = pd.concat([bt_mixed_kupiec_1000, pd.DataFrame(res)])
bt_mixed_kupiec_1000.to_excel('backtesting_results_mixed_Kupiec_1000.xlsx')

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# ------------ ES Backtesting ------------ #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


def alpha_levels(alpha, N=4):
    levels = [alpha + (j-1)/N*(1-alpha) for j in range(1, N+1)]
    return levels


# Determining the required VaR levels in order to implicitly backtest ES at
# 95%, 97.5%, and 99% levels:
all_levels = alpha_levels(0.95) + alpha_levels(0.975) + alpha_levels(0.99)
all_levels = set([round(x, 5) for x in all_levels])

# creating VaR exception columns:
breach_cols = ['VaR_9625_breach', 'VaR_98125_breach', 'VaR_9875_breach',
               'VaR_9925_breach', 'VaR_99375_breach', 'VaR_995_breach',
               'VaR_9975_breach']

for df in df_dict_250:
    for level in breach_cols:
        df_dict_250[df][level] = np.where(df_dict_250[df][level.split('_breach')[0]] >
                                          df_dict_250[df]['actual_log_r'], 1, 0)

for df in df_dict_500:
    for level in breach_cols:
        df_dict_500[df][level] = np.where(df_dict_500[df][level.split('_breach')[0]] >
                                          df_dict_500[df]['actual_log_r'], 1, 0)

for df in df_dict_750:
    for level in breach_cols:
        df_dict_750[df][level] = np.where(df_dict_750[df][level.split('_breach')[0]] >
                                          df_dict_750[df]['actual_log_r'], 1, 0)

for df in df_dict_1000:
    for level in breach_cols:
        df_dict_1000[df][level] = np.where(df_dict_1000[df][level.split('_breach')[0]] >
                                           df_dict_1000[df]['actual_log_r'], 1, 0)


# Implicit traffic light test for ES:
def multinomial_backtest(data, measure_level, N=4):

    if measure_level == 0.95:
```

```
        breach_columns = ['VaR_95_breach', 'VaR_9625_breach',
                          'VaR_975_breach', 'VaR_9875_breach']
    elif measure_level == 0.975:
        breach_columns = ['VaR_975_breach', 'VaR_98125_breach',
                          'VaR_9875_breach', 'VaR_99375_breach']
    elif measure_level == 0.99:
        breach_columns = ['VaR_99_breach', 'VaR_9925_breach',
                          'VaR_995_breach', 'VaR_9975_breach']

    n = len(data)
    E_S_N = N
    alphas = [0] + alpha_levels(measure_level) + [1]
    alpha_j_diff = np.diff(np.array(alphas))
    var_S_N = 2*N - (N**2+4*N+1)/n + (1/n)*sum(1/alpha_j_diff)
    c = (2 * E_S_N) / var_S_N

    # X_t column:
    X_t = data[breach_columns].sum(axis=1)

    # Observed cell counts, j = 0, ..., N:
    vector_oj = np.zeros(N+1)
    for j in range(N+1):
        vector_oj[j] = np.sum(X_t == j)

    S_denom = n * alpha_j_diff
    S_N = np.sum((vector_oj - S_denom)**2 / S_denom)
    cS_N = c * S_N
    df = c * E_S_N

    cum_prob = chi2.cdf(cS_N, df)
    if cum_prob < 0.95:
        zone = 'green'
    elif (cum_prob >= 0.95) & (cum_prob < 0.9999):
        zone = 'yellow'
    elif cum_prob >= 0.9999:
        zone = 'red'
    p_val = 1 - chi2.cdf(cS_N, df)

    return zone, cS_N, p_val


def multinomial_res(df, model, window):
    res_95 = multinomial_backtest(df, 0.95)
    res_975 = multinomial_backtest(df, 0.975)
    res_99 = multinomial_backtest(df, 0.99)
    return {
        'mod_w_level': [f'{model}_{window}_95', f'{model}_{window}_975',
                        f'{model}_{window}_99'],
        'color': [res_95[0], res_975[0], res_99[0]],
        'cS_N': [res_95[1], res_975[1], res_99[1]],
        'p_val': [res_95[2], res_975[2], res_99[2]],
        }


bt_multinomial_250 = pd.DataFrame()
for df in df_dict_250:
    res = multinomial_res(df_dict_250[df], df, 250)
    bt_multinomial_250 = pd.concat([bt_multinomial_250, pd.DataFrame(res)])
bt_multinomial_250.to_excel('backtesting_results_multinomial_250.xlsx')

bt_multinomial_500 = pd.DataFrame()
for df in df_dict_500:
    res = multinomial_res(df_dict_500[df], df, 500)
```

```
    bt_multinomial_500 = pd.concat([bt_multinomial_500, pd.DataFrame(res)])
bt_multinomial_500.to_excel('backtesting_results_multinomial_500.xlsx')

bt_multinomial_750 = pd.DataFrame()
for df in df_dict_750:
    res = multinomial_res(df_dict_750[df], df, 750)
    bt_multinomial_750 = pd.concat([bt_multinomial_750, pd.DataFrame(res)])
bt_multinomial_750.to_excel('backtesting_results_multinomial_750.xlsx')

bt_multinomial_1000 = pd.DataFrame()
for df in df_dict_1000:
    res = multinomial_res(df_dict_1000[df], df, 1000)
    bt_multinomial_1000 = pd.concat([bt_multinomial_1000, pd.DataFrame(res)])
bt_multinomial_1000.to_excel('backtesting_results_multinomial_1000.xlsx')


def er_ES_backtest(data, VaR_col, ES_col, M=10000):
    er = (data['actual_log_r'] - data[ES_col])[data['actual_log_r'] <= data[VaR_col]]
    t_stat = np.mean(er) / np.std(er) * np.sqrt(len(er))
    # z tilde:
    adj_er = er - np.mean(er)
    r_i_stat = np.zeros(M)
    for i in range(0, M):
        r_i_sample = np.random.choice(adj_er, size=len(adj_er), replace=True)
        r_i_stat[i] = np.mean(r_i_sample) / np.std(r_i_sample) * np.sqrt(len(r_i_sample))
    # One-sided H1 is that mean(er) < 0 (which implies that the ES estimates do not
    # outweigh the losses (both negative)):
    one_sided_p_val = 1/M * np.sum(r_i_stat < t_stat)
    two_sided_p_val = 1/M * np.sum(abs(r_i_stat) > abs(t_stat))
    return two_sided_p_val, one_sided_p_val


def er_ES_res(df, model, window):
    res_95 = er_ES_backtest(df, 'VaR_95', 'ES_95')
    res_975 = er_ES_backtest(df, 'VaR_975', 'ES_975')
    res_99 = er_ES_backtest(df, 'VaR_99', 'ES_99')
    return {
        'mod_w_level': [f'{model}_{window}_95', f'{model}_{window}_975',
                        f'{model}_{window}_99'],
        'two_sided_H0': [res_95[0], res_975[0], res_99[0]],
        'one-sided_H0': [res_95[1], res_975[1], res_99[1]],
        }


np.random.seed(2024)
bt_er_ES_250 = pd.DataFrame()
for df in df_dict_250:
    res = er_ES_res(df_dict_250[df], df, 250)
    bt_er_ES_250 = pd.concat([bt_er_ES_250, pd.DataFrame(res)])
bt_er_ES_250.to_excel('backtesting_results_er_ES_250.xlsx')

bt_er_ES_500 = pd.DataFrame()
for df in df_dict_500:
    res = er_ES_res(df_dict_500[df], df, 500)
    bt_er_ES_500 = pd.concat([bt_er_ES_500, pd.DataFrame(res)])
bt_er_ES_500.to_excel('backtesting_results_er_ES_500.xlsx')

bt_er_ES_750 = pd.DataFrame()
for df in df_dict_750:
    res = er_ES_res(df_dict_750[df], df, 750)
    bt_er_ES_750 = pd.concat([bt_er_ES_750, pd.DataFrame(res)])
bt_er_ES_750.to_excel('backtesting_results_er_ES_750.xlsx')
```

```
bt_er_ES_1000 = pd.DataFrame()
for df in df_dict_1000:
    res = er_ES_res(df_dict_1000[df], df, 1000)
    bt_er_ES_1000 = pd.concat([bt_er_ES_1000, pd.DataFrame(res)])
bt_er_ES_1000.to_excel('backtesting_results_er_ES_1000.xlsx')

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# ---- Violation rate graphs ------- #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


col_dict = {
    'HS': '#db0043',
    'AWHS': '#510048',
    'MNIG': '#ff4f00',
    'HCC': '#0B81B3',
    'HFC': '#18492E',
}


def create_VR_dict(df_dict):
    levels = ['VaR_95', 'VaR_9625', 'VaR_975', 'VaR_98125', 'VaR_9875',
              'VaR_99', 'VaR_9925', 'VaR_99375', 'VaR_995', 'VaR_9975']
    vr_dict = {}
    for df in df_dict:
        vr_dict[df] = {}
        vr_dict[df]["vr"] = df_dict[df][levels].apply(
            lambda x: (x > df_dict[df]["actual_log_r"]).mean()
        )

    return vr_dict


vr_dict_250 = create_VR_dict(df_dict_250)
vr_dict_500 = create_VR_dict(df_dict_500)
vr_dict_750 = create_VR_dict(df_dict_750)
vr_dict_1000 = create_VR_dict(df_dict_1000)


def plot_VR(conf_levels, expected_vr, vr_dict, col_dict, window,
            legend_loc='upper right', save_fig=True, fig_title=None):
    if fig_title is None:
        fig_title = f'Violation_rates_{window}d'
    plt.figure(figsize=(6.8, 6), dpi=200)
    plt.xticks(fontsize=13)
    plt.yticks(fontsize=13)
    plt.plot(conf_levels, expected_vr, label='Expected rate', color='grey',
             linewidth=2.5)
    for model in vr_dict:
        plt.plot(conf_levels, vr_dict[model]['vr'], 'o--', label=f'{model}',
                 color=col_dict[model], alpha=0.8, markersize=5)
    plt.xlabel('Confidence level', fontsize=15)
    plt.ylabel('Violation rate', fontsize=15)
    plt.legend(fontsize=13, loc=legend_loc)
    plt.grid(alpha=0.4)
    plt.title('Comparison of VaR violation rates for models\n' +
              f'estimated using a {window}-day calibration window', fontsize=16)
    if save_fig:
        plt.savefig(f'{fig_title}.pdf')
    plt.show()


all_levels = alpha_levels(0.95) + alpha_levels(0.975) + alpha_levels(0.99)
```

```
all_levels = set([round(x, 5) for x in all_levels])
conf_levels = sorted(list(all_levels))
expected_vr = [round(1-cl, 5) for cl in conf_levels]

plot_VR(conf_levels, expected_vr, vr_dict_250, col_dict, 250)
plot_VR(conf_levels, expected_vr, vr_dict_500, col_dict, 500)
plot_VR(conf_levels, expected_vr, vr_dict_750, col_dict, 750)
plot_VR(conf_levels, expected_vr, vr_dict_1000, col_dict, 1000)


# Expected numbers of breaches:
breach_expectations = pd.DataFrame(index=['250', '500', '750', '1000'],
                                   columns=['95', '97.5', '99'])
for col in breach_expectations.columns:
    for ind in breach_expectations.index:
        breach_expectations.loc[ind, col] = (1 - np.float64(col) / 100) * (
            1811 - np.float64(ind)
        )

breach_expectations.astype(float).to_excel('expected_no_of_breaches.xlsx')

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# Combining forecasts of AWHS1000, HCC750, HCC1000 #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #

comb_est_1 = estimates_Clayton_750.tail(811).copy()
comb_est_2 = estimates_Clayton_1000.copy()
comb_est_3 = estimates_WHS_1000.copy()
avg_estimates = comb_est_1[['forecast_date', 'actual_log_r']].copy()
min_estimates = comb_est_1[['forecast_date', 'actual_log_r']].copy()
median_estimates = comb_est_1[['forecast_date', 'actual_log_r']].copy()
forecast_cols = ['VaR_95', 'VaR_975', 'VaR_99', 'VaR_9625', 'VaR_98125',
                 'VaR_9875', 'VaR_9925', 'VaR_99375', 'VaR_995',
                 'VaR_9975', 'ES_95', 'ES_975', 'ES_99']

for col in forecast_cols:
    avg_estimates[col] = pd.concat([comb_est_1[col], comb_est_2[col],
                                    comb_est_3[col]], axis=1).mean(axis=1)
    min_estimates[col] = pd.concat([comb_est_1[col], comb_est_2[col],
                                    comb_est_3[col]], axis=1).min(axis=1)
    median_estimates[col] = pd.concat([comb_est_1[col], comb_est_2[col],
                                       comb_est_3[col]], axis=1).median(axis=1)


# creating VaR exception columns:
breach_cols = ['VaR_95_breach', 'VaR_975_breach', 'VaR_99_breach', 'VaR_9625_breach',
               'VaR_98125_breach', 'VaR_9875_breach', 'VaR_9925_breach',
               'VaR_99375_breach', 'VaR_995_breach', 'VaR_9975_breach']

for level in breach_cols:
    avg_estimates[level] = np.where(avg_estimates[level.split('_breach')[0]] >
                                    avg_estimates['actual_log_r'], 1, 0)
    min_estimates[level] = np.where(min_estimates[level.split('_breach')[0]] >
                                    min_estimates['actual_log_r'], 1, 0)
    median_estimates[level] = np.where(median_estimates[level.split('_breach')[0]] >
                                       median_estimates['actual_log_r'], 1, 0)

plot_VaR_ES(avg_estimates, 'Average', '750/1000', 95, legend_loc='lower right',
            fig_title='average_combination_AWHS1000_HCC750_HCC1000_95',
            estimate_col='#a195f9')
plot_VaR_ES(avg_estimates, 'Average', '750/1000', level=975, level_in_graph=97.5,
            estimate_col='#a195f9', legend_loc='lower right',
            fig_title='average_combination_AWHS1000_HCC750_HCC1000_975')
```

```
plot_VaR_ES(avg_estimates, 'Average', '750/1000', level=99,
            estimate_col='#a195f9', legend_loc='lower right',
            fig_title='average_combination_AWHS1000_HCC750_HCC1000_99')

plot_VaR_ES(min_estimates, 'Minimum', '750/1000', 95, legend_loc='lower right',
            fig_title='Minimum_combination_AWHS1000_HCC750_HCC1000_95',
            estimate_col='#bf63bf')
plot_VaR_ES(min_estimates, 'Minimum', '750/1000', level=975, level_in_graph=97.5,
            estimate_col='#bf63bf', legend_loc='lower right',
            fig_title='Minimum_combination_AWHS1000_HCC750_HCC1000_975')
plot_VaR_ES(min_estimates, 'Minimum', '750/1000', level=99,
            estimate_col='#bf63bf', legend_loc='lower right',
            fig_title='Minimum_combination_AWHS1000_HCC750_HCC1000_99')

plot_VaR_ES(median_estimates, 'Median', '750/1000', 95, legend_loc='lower right',
            fig_title='Median_combination_AWHS1000_HCC750_HCC1000_95',
            estimate_col='#13135d')
plot_VaR_ES(median_estimates, 'Median', '750/1000', level=975, level_in_graph=97.5,
            estimate_col='#13135d', legend_loc='lower right',
            fig_title='Median_combination_AWHS1000_HCC750_HCC1000_975')
plot_VaR_ES(median_estimates, 'Median', '750/1000', level=99,
            estimate_col='#13135d', legend_loc='lower right',
            fig_title='Median_combination_AWHS1000_HCC750_HCC1000_99')

avg_estimates.to_excel('Estimated_VaR_ES_Average_AWHS1000_HCC750_HCC1000.xlsx')
min_estimates.to_excel('Estimated_VaR_ES_Minimum_AWHS1000_HCC750_HCC1000.xlsx')
median_estimates.to_excel('Estimated_VaR_ES_Median_AWHS1000_HCC750_HCC1000.xlsx')

# Backtesting the combined estimates:

df_dict_comb = {
    'Mean': avg_estimates,
    'Minimum': min_estimates,
    'Median': median_estimates,
}

bt_basel_comb = pd.DataFrame()
for df in df_dict_comb:
    res = basel_TL_res(df_dict_comb[df], df, '750_1000')
    bt_basel_comb = pd.concat([bt_basel_comb, pd.DataFrame(res)])
bt_basel_comb.to_excel('backtesting_results_Basel_TL_comb.xlsx')

bt_kupiec_comb = pd.DataFrame()
for df in df_dict_comb:
    res = kupiec_res(df_dict_comb[df], df, '750_1000')
    bt_kupiec_comb = pd.concat([bt_kupiec_comb, pd.DataFrame(res)])
bt_kupiec_comb.to_excel('backtesting_results_Kupiec_comb.xlsx')

bt_mixed_kupiec_comb = pd.DataFrame()
for df in df_dict_comb:
    res = mixed_kupiec_res(df_dict_comb[df], df, '750_1000')
    bt_mixed_kupiec_comb = pd.concat([bt_mixed_kupiec_comb, pd.DataFrame(res)])
bt_mixed_kupiec_comb.to_excel('backtesting_results_mixed_Kupiec_comb.xlsx')

bt_multinomial_comb = pd.DataFrame()
for df in df_dict_comb:
    res = multinomial_res(df_dict_comb[df], df, '750_1000')
    bt_multinomial_comb = pd.concat([bt_multinomial_comb, pd.DataFrame(res)])
bt_multinomial_comb.to_excel('backtesting_results_multinomial_comb.xlsx')

np.random.seed(2024)
bt_er_ES_comb = pd.DataFrame()
for df in df_dict_comb:
```

```
    res = er_ES_res(df_dict_comb[df], df, '750_1000')
    bt_er_ES_comb = pd.concat([bt_er_ES_comb, pd.DataFrame(res)])
bt_er_ES_comb.to_excel('backtesting_results_er_ES_comb.xlsx')

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# ------------- Comparative backtesting ------------- #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


# 0-homogeneous loss scoring function for comparative ES backtesting:
def scoring_func(x, VaR, ES, breach_col, alpha=0.05):
    # alpha is 1 - risk measure confidence level
    if len(x) == 1:
        S_val = (int(x < VaR) * (x - VaR)/ES +
                 alpha * (VaR/ES - 1 + np.log(-ES)))
    elif len(x) > 1:
        S_val = (breach_col * (x - VaR)/ES +
                 alpha * (VaR/ES - 1 + np.log(-ES.astype(float))))
    return S_val


# 0-homogeneous loss scoring function for comparative VaR backtesting:
def scoring_func_VaR(x, VaR, breach_col=None, alpha=0.05):
    # alpha is 1 - risk measure confidence level
    if len(x) == 1:
        S_val = (alpha - int(x < VaR)) * np.log(-VaR) + \
            np.where(int(x < VaR) == 1, int(x < VaR) * np.log(-x), 0)
    elif len(x) > 1:
        S_val = (alpha - breach_col) * np.log((-VaR).astype(float)) + \
            np.where(breach_col == 1, breach_col * np.log((-x).astype(float)), 0)
    return S_val


# Calculate autocovariance at a certain lag for the HAC estimator:
def autocovariance(delta_S, delta_n_S, lag, n):

    auto_cov = 0
    for i in np.arange(0, n-lag):
        auto_cov += ((delta_S[i+lag])-delta_n_S)*(delta_S[i]-delta_n_S)

    return (1/(n-1)) * auto_cov


# HAC estimator of the asymptotic variance:
def hac_sigma_est(scores_internal, scores_standard):

    delta_S = scores_internal - scores_standard
    delta_n_S = np.mean(delta_S)
    n = len(delta_S)
    K = np.floor(n**(1/4))
    auto_cov_sum = 0
    auto_cov_sum += autocovariance(delta_S, delta_n_S, 0, n)
    for lag in np.arange(1, K):
        auto_cov_sum += 2 * autocovariance(delta_S, delta_n_S, lag, n) * (1-lag/K)

    return auto_cov_sum


def comparative_backtest(df_int, df_stnd, risk_measure, risk_m_level, eta=0.05):
    if risk_m_level == 0.95:
        level_name = '95'
    elif risk_m_level == 0.975:
        level_name = '975'
```

```
    elif risk_m_level == 0.99:
        level_name = '99'
    VaR_col = 'VaR_' + level_name
    br_col = 'VaR_' + level_name + '_breach'
    alpha = 1 - risk_m_level
    n = len(df_int)
    if risk_measure == 'ES':
        ES_col = 'ES_' + level_name
        S_int = scoring_func(df_int['actual_log_r'], df_int[VaR_col], df_int[ES_col],
                             df_int[br_col], alpha).reset_index(drop=True)
        S_stnd = scoring_func(df_stnd['actual_log_r'], df_stnd[VaR_col], df_stnd[ES_col],
                              df_stnd[br_col], alpha).reset_index(drop=True)
    elif risk_measure == 'VaR':
        S_int = scoring_func_VaR(df_int['actual_log_r'], df_int[VaR_col],
                                 df_int[br_col], alpha).reset_index(drop=True)
        S_stnd = scoring_func_VaR(df_stnd['actual_log_r'], df_stnd[VaR_col],
                                  df_stnd[br_col], alpha).reset_index(drop=True)
    delta_n_S = np.mean(S_int - S_stnd)
    sigma_hat = hac_sigma_est(S_int, S_stnd)
    print(f'sigma_hat is {sigma_hat}')
    T_statistic = delta_n_S / np.sqrt(sigma_hat / n)
    # The test is passed when H0+ is rejected:
    if norm.cdf(T_statistic) <= eta:
        result = 'pass: internal model is in the green region'
    # The test is failed when H0- is rejected:
    elif 1 - norm.cdf(T_statistic) <= eta:
        result = 'fail: internal model is in the red region'
    else:
        result = 'inconclusive: internal model is in the yellow region'

    return result, T_statistic, norm.cdf(T_statistic)


all_results = {
    'HS_250': estimates_HS_250,
    'AWHS_250': estimates_WHS_250,
    'MNIG_250': estimates_MNIG_250,
    'HCC_250': estimates_Clayton_250,
    'HFC_250': estimates_Frank_250,
    'HS_500': estimates_HS_500,
    'AWHS_500': estimates_WHS_500,
    'MNIG_500': estimates_MNIG_500,
    'HCC_500': estimates_Clayton_500,
    'HFC_500': estimates_Frank_500,
    'HS_750': estimates_HS_750,
    'AWHS_750': estimates_WHS_750,
    'MNIG_750': estimates_MNIG_750,
    'HCC_750': estimates_Clayton_750,
    'HFC_750': estimates_Frank_750,
    'HS_1000': estimates_HS_1000,
    'AWHS_1000': estimates_WHS_1000,
    'MNIG_1000': estimates_MNIG_1000,
    'HCC_1000': estimates_Clayton_1000,
    'HFC_1000': estimates_Frank_1000,
    'Average': avg_estimates,
    'Median': median_estimates,
    'Minimum': min_estimates,
}

# ES comparative backtesting:
comp_99 = ['HS_500', 'HS_750', 'HCC_750', 'HCC_1000', 'HFC_1000']
comp_99_df = pd.DataFrame(index=comp_99, columns=comp_99)
for model in comp_99:
```

```
    for index in comp_99:
        if index != model:
            comp_99_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811), all_results[index].tail(811), 'ES', 0.99
            )[0]


comp_95 = ['HS_250', 'HS_500', 'HS_750', 'AWHS_250', 'AWHS_500', 'AWHS_750', 'AWHS_1000',
           'MNIG_1000', 'HCC_750', 'HCC_1000']
comp_95_df = pd.DataFrame(index=comp_95, columns=comp_95)
for model in comp_95:
    for index in comp_95:
        if index != model:
            comp_95_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811), all_results[index].tail(811), 'ES', 0.95
            )[0]

comp_975 = ['HS_500', 'HS_750', 'AWHS_1000', 'MNIG_1000',
            'HCC_750', 'HCC_1000', 'HFC_1000']
comp_975_df = pd.DataFrame(index=comp_975, columns=comp_975)
for model in comp_975:
    for index in comp_975:
        if index != model:
            comp_975_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811), all_results[index].tail(811), 'ES', 0.975
            )[0]


plot_map = {
    'fail: internal model is in the red region': -1,
    'inconclusive: internal model is in the yellow region': 0,
    'pass: internal model is in the green region': 1,
    np.nan: 2,
    }


def plot_traffic_light_comp_bt(matrix, labels, model, level, size=(6, 6), fontsize=None,
                               title=None, savefig=True):
    if title is None:
        title = f'comparative_BT_{model}_{level}.pdf'
    colors = ['#D2222D', '#FFD301', '#238823', 'white']
    cmap = matplotlib.colors.ListedColormap(colors, name='colors', N=None)
    fig, ax = plt.subplots(figsize=size)
    ax.tick_params(top=True, labeltop=True, bottom=False, labelbottom=False)
    if fontsize is None:
        ax.set_title('Internal model')
        ax.set_ylabel('Standard model')
    elif fontsize is not None:
        ax.set_title('Internal model', fontsize=fontsize)
        ax.set_ylabel('Standard model', fontsize=fontsize)
    ax.set_xticks(np.arange(0, len(labels)-0.5, 1))
    ax.set_yticks(np.arange(0, len(labels)-0.5, 1))
    ax.set_xticklabels(labels)
    ax.set_yticklabels(labels)
    ax.tick_params(axis='both', which='major', labelsize=12)
    ax.imshow(matrix, cmap=cmap, vmin=-1, vmax=2)
    if savefig:
        plt.savefig(title, dpi=200, bbox_inches='tight')
    plt.show()


plot_comp_975 = np.array(comp_975_df.replace(plot_map).copy())
comp_975_plot = ['HS\n500', 'HS\n750', 'AWHS\n1000', 'MNIG\n1000',
```

```
                'HCC\n750', 'HCC\n1000', 'HFC\n1000']
plot_traffic_light_comp_bt(plot_comp_975, comp_975_plot, model='ES', level=975)

plot_comp_99 = np.array(comp_99_df.replace(plot_map).copy())
comp_99_plot = ['HS\n500', 'HS\n750', 'HCC\n750', 'HCC\n1000', 'HFC\n1000']
plot_traffic_light_comp_bt(plot_comp_99, comp_99_plot, model='ES', level=99)

plot_comp_95 = np.array(comp_95_df.replace(plot_map).copy())
comp_95_plot = ['HS\n250', 'HS\n500', 'HS\n750', 'AWHS\n250', 'AWHS\n500',
                'AWHS\n750', 'AWHS\n1000', 'MNIG\n1000', 'HCC\n750', 'HCC\n1000']
plot_traffic_light_comp_bt(plot_comp_95, comp_95_plot, model='ES', level=95, size=(7, 7))


# VaR comparative backtesting:
comp_99_VaR = ['HS_250', 'HS_500', 'HS_750', 'HS_1000',
               'AWHS_250', 'AWHS_500', 'AWHS_750', 'AWHS_1000',
               'MNIG_1000', 'HCC_500', 'HCC_750', 'HCC_1000', 'HFC_1000']
comp_99_VaR_df = pd.DataFrame(index=comp_99_VaR, columns=comp_99_VaR)
for model in comp_99_VaR:
    for index in comp_99_VaR:
        if index != model:
            comp_99_VaR_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811),
                all_results[index].tail(811),
                risk_measure='VaR',
                risk_m_level=0.99,
            )[0]
plot_comp_99_VaR = np.array(comp_99_VaR_df.replace(plot_map).copy())
comp_99_VaR_plot = ['HS\n250', 'HS\n500', 'HS\n750', 'HS\n1000',
                    'AWHS\n250', 'AWHS\n500', 'AWHS\n750', 'AWHS\n1000',
                    'MNIG\n1000', 'HCC\n500', 'HCC\n750', 'HCC\n1000', 'HFC\n1000']
plot_traffic_light_comp_bt(plot_comp_99_VaR, comp_99_VaR_plot,
                           size=(9, 9), model='VaR', level=99)


comp_975_VaR = ['HS_250', 'HS_500', 'HS_750', 'HS_1000',
                'AWHS_250', 'AWHS_500', 'AWHS_750', 'AWHS_1000',
                'MNIG_500', 'MNIG_750', 'MNIG_1000',
                'HCC_250', 'HCC_500', 'HCC_750', 'HCC_1000', 'HFC_1000']
comp_975_VaR_df = pd.DataFrame(index=comp_975_VaR, columns=comp_975_VaR)
for model in comp_975_VaR:
    for index in comp_975_VaR:
        if index != model:
            comp_975_VaR_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811),
                all_results[index].tail(811),
                risk_measure='VaR',
                risk_m_level=0.975,
            )[0]
plot_comp_975_VaR = np.array(comp_975_VaR_df.replace(plot_map).copy())
comp_975_VaR_plot = ['HS\n250', 'HS\n500', 'HS\n750', 'HS\n1000',
                     'AWHS\n250', 'AWHS\n500', 'AWHS\n750', 'AWHS\n1000',
                     'MNIG\n500', 'MNIG\n750', 'MNIG\n1000',
                     'HCC\n250', 'HCC\n500', 'HCC\n750', 'HCC\n1000', 'HFC\n1000']
plot_traffic_light_comp_bt(plot_comp_975_VaR, comp_975_VaR_plot,
                           size=(11, 11), model='VaR', level=975)


comp_95_VaR = ['HS_250', 'HS_500', 'HS_750', 'HS_1000',
               'AWHS_250', 'AWHS_500', 'AWHS_750', 'AWHS_1000',
               'MNIG_500', 'MNIG_750', 'MNIG_1000',
               'HCC_250', 'HCC_500', 'HCC_750', 'HCC_1000',
               'HFC_250', 'HFC_500', 'HFC_750', 'HFC_1000']
```

```
comp_95_VaR_df = pd.DataFrame(index=comp_95_VaR, columns=comp_95_VaR)
for model in comp_95_VaR:
    for index in comp_95_VaR:
        if index != model:
            comp_95_VaR_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811),
                all_results[index].tail(811),
                risk_measure='VaR',
                risk_m_level=0.95,
            )[0]
plot_comp_95_VaR = np.array(comp_95_VaR_df.replace(plot_map).copy())
comp_95_VaR_plot = ['HS\n250', 'HS\n500', 'HS\n750', 'HS\n1000',
                    'AWHS\n250', 'AWHS\n500', 'AWHS\n750', 'AWHS\n1000',
                    'MNIG\n500', 'MNIG\n750', 'MNIG\n1000',
                    'HCC\n250', 'HCC\n500', 'HCC\n750', 'HCC\n1000',
                    'HFC\n250', 'HFC\n500', 'HFC\n750', 'HFC\n1000']
plot_traffic_light_comp_bt(plot_comp_95_VaR, comp_95_VaR_plot, fontsize=22,
                           size=(13, 13), model='VaR', level=95)

# Comparative backtesting for combined forecasts (mean and median) and
# models that are included in them:

comp_comb_99 = ['AWHS_1000', 'HCC_750', 'HCC_1000', 'Average', 'Median']
comp_comb_99_df = pd.DataFrame(index=comp_comb_99, columns=comp_comb_99)
for model in comp_comb_99:
    for index in comp_comb_99:
        if index != model:
            comp_comb_99_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811), all_results[index].tail(811), "ES", 0.99
            )[0]


comp_comb_95 = ['AWHS_1000', 'HCC_750', 'HCC_1000', 'Average', 'Median']
comp_comb_95_df = pd.DataFrame(index=comp_comb_95, columns=comp_comb_95)
for model in comp_comb_95:
    for index in comp_comb_95:
        if index != model:
            comp_comb_95_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811), all_results[index].tail(811), "ES", 0.95
            )[0]

comp_comb_975 = ['AWHS_1000', 'HCC_750', 'HCC_1000', 'Average', 'Median', 'Minimum']
comp_comb_975_df = pd.DataFrame(index=comp_comb_975, columns=comp_comb_975)
for model in comp_comb_975:
    for index in comp_comb_975:
        if index != model:
            comp_comb_975_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811), all_results[index].tail(811), "ES", 0.975
            )[0]

plot_comp_comb_975 = np.array(comp_comb_975_df.replace(plot_map).copy())
comp_comb_975_plot = ['AWHS\n1000', 'HCC\n750', 'HCC\n1000', 'Average',
                      'Median', 'Minimum']
plot_traffic_light_comp_bt(plot_comp_comb_975, comp_comb_975_plot,
                           model='Comb_ES', level=975)

plot_comp_comb_99 = np.array(comp_comb_99_df.replace(plot_map).copy())
comp_comb_99_plot = ['AWHS\n1000', 'HCC\n750', 'HCC\n1000', 'Average', 'Median']
plot_traffic_light_comp_bt(plot_comp_comb_99, comp_comb_99_plot,
                           model='Comb_ES', level=99)

plot_comp_comb_95 = np.array(comp_comb_95_df.replace(plot_map).copy())
comp_comb_95_plot = ['AWHS\n1000', 'HCC\n750', 'HCC\n1000', 'Average', 'Median']
```

```
plot_traffic_light_comp_bt(plot_comp_comb_95, comp_comb_95_plot, model='Comb_ES',
                           level=95, size=(7, 7))

# VaR:
comp_comb_99_VaR = ['AWHS_1000', 'HCC_750', 'HCC_1000', 'Average', 'Median']
comp_comb_99_VaR_df = pd.DataFrame(index=comp_comb_99_VaR, columns=comp_comb_99_VaR)
for model in comp_comb_99_VaR:
    for index in comp_comb_99_VaR:
        if index != model:
            comp_comb_99_VaR_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811),
                all_results[index].tail(811),
                risk_measure='VaR',
                risk_m_level=0.99,
            )[0]
plot_comp_comb_99_VaR = np.array(comp_comb_99_VaR_df.replace(plot_map).copy())
comp_comb_99_VaR_plot = ['AWHS\n1000', 'HCC\n750', 'HCC\n1000',
                         'Average', 'Median']
plot_traffic_light_comp_bt(plot_comp_comb_99_VaR, comp_comb_99_VaR_plot,
                           size=(6, 6), model='Comb_VaR', level=99)


comp_comb_975_VaR = ['AWHS_1000', 'HCC_750', 'HCC_1000', 'Average', 'Median', 'Minimum']
comp_comb_975_VaR_df = pd.DataFrame(index=comp_comb_975_VaR, columns=comp_comb_975_VaR)
for model in comp_comb_975_VaR:
    for index in comp_comb_975_VaR:
        if index != model:
            comp_comb_975_VaR_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811),
                all_results[index].tail(811),
                risk_measure='VaR',
                risk_m_level=0.975,
            )[0]
plot_comp_comb_975_VaR = np.array(comp_comb_975_VaR_df.replace(plot_map).copy())
comp_comb_975_VaR_plot = ['AWHS\n1000', 'HCC\n750', 'HCC\n1000',
                          'Average', 'Median', 'Minimum']
plot_traffic_light_comp_bt(plot_comp_comb_975_VaR, comp_comb_975_VaR_plot,
                           size=(6, 6), model='Comb_VaR', level=975)

comp_comb_95_VaR = ['AWHS_1000', 'HCC_750', 'HCC_1000', 'Average', 'Median']
comp_comb_95_VaR_df = pd.DataFrame(index=comp_comb_95_VaR, columns=comp_comb_95_VaR)
for model in comp_comb_95_VaR:
    for index in comp_comb_95_VaR:
        if index != model:
            comp_comb_95_VaR_df.loc[index, model] = comparative_backtest(
                all_results[model].tail(811),
                all_results[index].tail(811),
                risk_measure='VaR',
                risk_m_level=0.95,
            )[0]
plot_comp_comb_95_VaR = np.array(comp_comb_95_VaR_df.replace(plot_map).copy())
comp_comb_95_VaR_plot = ['AWHS\n1000', 'HCC\n750', 'HCC\n1000', 'Average', 'Median']
plot_traffic_light_comp_bt(plot_comp_comb_95_VaR, comp_comb_95_VaR_plot,
                           size=(6, 6), model='Comb_VaR', level=95)

# >>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>>> #
# ------- Incremental VaR analysis ------- #
# <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<< #


def get_name(col):
    return col.split('_')[1]
```

```
result = pd.DataFrame(
    {
        f'PV_no_{get_name(col)}': (
            index_prices_df[price_columns].drop(columns=[col]).sum(axis=1)
        )
        * 10
        for col in price_columns
    }
)


def without(list, val):
    return [item for item in list if item != val]


incr_prices_df = pd.concat([index_prices_df, result], axis=1).copy()


# VaR and ES estimation in case of incremental analysis:
def VaR_ES_estimation_incr(estimation_func, results_df, calibration_window,
                           index_prices_df, return_df=None, random_seed=2024,
                           method_in_R=False, c_type=None):
    rm_cols = ['forecast_date', 'VaR_95', 'VaR_975', 'VaR_99', 'VaR_9625',
               'VaR_98125', 'VaR_9875', 'VaR_9925', 'VaR_99375', 'VaR_995',
               'VaR_9975', 'ES_95', 'ES_975', 'ES_99', 'actual_log_r']
    results_df_no_vgi = pd.DataFrame(columns=rm_cols)
    results_df_no_rgi = pd.DataFrame(columns=rm_cols)
    results_df_no_tgi = pd.DataFrame(columns=rm_cols)
    results_df_no_s30gi = pd.DataFrame(columns=rm_cols)
    results_df_no_igi = pd.DataFrame(columns=rm_cols)
    results_df_no_h25gi = pd.DataFrame(columns=rm_cols)
    results_df_no_c25gi = pd.DataFrame(columns=rm_cols)
    results_df_no_obx = pd.DataFrame(columns=rm_cols)

    if method_in_R:
        set_seed(random_seed)
    else:
        np.random.seed(random_seed)

    if return_df is None:
        for study_date in index_prices_df['Date'][calibration_window:-1]:
            estimate_row = estimation_func(index_prices_df, calibration_window,
                                           study_date.strftime('%Y-%m-%d'))
            results_df = pd.concat([results_df, estimate_row])
    else:
        if estimation_func != VaR_ES_Copula:
            for study_date in index_prices_df['Date'][calibration_window:-1]:
                estimate_row = estimation_func(return_df, calibration_window,
                                               study_date.strftime('%Y-%m-%d'))
                results_df = pd.concat([results_df, estimate_row])
        elif estimation_func == VaR_ES_Copula:
            iter = 1
            for study_date in index_prices_df['Date'][calibration_window:-1]:
                (
                    estimate_row_vgi,
                    estimate_row_rgi,
                    estimate_row_tgi,
                    estimate_row_s30gi,
                    estimate_row_igi,
                    estimate_row_h25gi,
                    estimate_row_c25gi,
                    estimate_row_obx,
```

```
                ) = estimation_func(
                    return_df,
                    calibration_window,
                    study_date.strftime("%Y-%m-%d"),
                    index_prices_df=index_prices_df,
                    type=c_type
                )
                results_df_no_vgi = pd.concat([results_df_no_vgi, estimate_row_vgi])
                results_df_no_rgi = pd.concat([results_df_no_rgi, estimate_row_rgi])
                results_df_no_tgi = pd.concat([results_df_no_tgi, estimate_row_tgi])
                results_df_no_s30gi = pd.concat([results_df_no_s30gi,
                                                 estimate_row_s30gi])
                results_df_no_igi = pd.concat([results_df_no_igi, estimate_row_igi])
                results_df_no_h25gi = pd.concat([results_df_no_h25gi,
                                                 estimate_row_h25gi])
                results_df_no_c25gi = pd.concat([results_df_no_c25gi,
                                                 estimate_row_c25gi])
                results_df_no_obx = pd.concat([results_df_no_obx, estimate_row_obx])
                print(f'iter {iter} done')
                iter += 1

            results_dict = {
                "no_vgi": results_df_no_vgi,
                "no_rgi": results_df_no_rgi,
                "no_tgi": results_df_no_tgi,
                "no_s30gi": results_df_no_s30gi,
                "no_igi": results_df_no_igi,
                "no_h25gi": results_df_no_h25gi,
                "no_c25gi": results_df_no_c25gi,
                "no_obx": results_df_no_obx,
            }
            return results_dict

    return results_df


def VaR_ES_Copula(return_df, calibration_window, study_date, index_prices_df,
                  type='Clayton', nsim=10000):

    filtered_df = return_df[return_df['Date'] <=
                            study_date].tail(calibration_window).copy()
    filtered_df.set_index('Date', inplace=True)
    with localconverter(ro.default_converter + pandas2ri.converter):
        filtered_df_R = ro.conversion.py2rpy(filtered_df)

    # Estimating GARCH models:
    garch_vgi = fit_garch(filtered_df_R, 'logr_vgi_eur')
    garch_rgi = fit_garch(filtered_df_R, 'logr_rgi_eur')
    garch_tgi = fit_garch(filtered_df_R, 'logr_tgi_eur')
    garch_s30gi = fit_garch(filtered_df_R, 'logr_s30gi_eur')
    garch_igi = fit_garch(filtered_df_R, 'logr_igi_eur')
    garch_h25gi = fit_garch(filtered_df_R, 'logr_h25gi_eur')
    garch_c25gi = fit_garch(filtered_df_R, 'logr_c25gi_eur')
    garch_obx = fit_garch(filtered_df_R, 'logr_obx_eur')

    # Extracting one-day conditional mean and volatility predictions:
    forecast_vgi = rg.ugarchforecast(garch_vgi, n_ahead=1)
    mu_vgi = base.as_numeric(forecast_vgi.slots['forecast'].rx2('seriesFor'))
    sigma_vgi = base.as_numeric(forecast_vgi.slots['forecast'].rx2('sigmaFor'))

    forecast_rgi = rg.ugarchforecast(garch_rgi, n_ahead=1)
    mu_rgi = base.as_numeric(forecast_rgi.slots['forecast'].rx2('seriesFor'))
    sigma_rgi = base.as_numeric(forecast_rgi.slots['forecast'].rx2('sigmaFor'))
```

```
forecast_tgi = rg.ugarchforecast(garch_tgi, n_ahead=1)
mu_tgi = base.as_numeric(forecast_tgi.slots['forecast'].rx2('seriesFor'))
sigma_tgi = base.as_numeric(forecast_tgi.slots['forecast'].rx2('sigmaFor'))

forecast_s30gi = rg.ugarchforecast(garch_s30gi, n_ahead=1)
mu_s30gi = base.as_numeric(forecast_s30gi.slots['forecast'].rx2('seriesFor'))
sigma_s30gi = base.as_numeric(forecast_s30gi.slots['forecast'].rx2('sigmaFor'))

forecast_igi = rg.ugarchforecast(garch_igi, n_ahead=1)
mu_igi = base.as_numeric(forecast_igi.slots['forecast'].rx2('seriesFor'))
sigma_igi = base.as_numeric(forecast_igi.slots['forecast'].rx2('sigmaFor'))

forecast_h25gi = rg.ugarchforecast(garch_h25gi, n_ahead=1)
mu_h25gi = base.as_numeric(forecast_h25gi.slots['forecast'].rx2('seriesFor'))
sigma_h25gi = base.as_numeric(forecast_h25gi.slots['forecast'].rx2('sigmaFor'))

forecast_c25gi = rg.ugarchforecast(garch_c25gi, n_ahead=1)
mu_c25gi = base.as_numeric(forecast_c25gi.slots['forecast'].rx2('seriesFor'))
sigma_c25gi = base.as_numeric(forecast_c25gi.slots['forecast'].rx2('sigmaFor'))

forecast_obx = rg.ugarchforecast(garch_obx, n_ahead=1)
mu_obx = base.as_numeric(forecast_obx.slots['forecast'].rx2('seriesFor'))
sigma_obx = base.as_numeric(forecast_obx.slots['forecast'].rx2('sigmaFor'))
# These values will be used for calculating simulated log returns:
# log_r = mu_index + sigma_index * simulated residual

# Getting the standardised residuals:
with localconverter(ro.default_converter + pandas2ri.converter):
    sr_vgi = FMat(np.divide(rg.residuals(garch_vgi), rg.sigma(garch_vgi)))
    sr_rgi = FMat(np.divide(rg.residuals(garch_rgi), rg.sigma(garch_rgi)))
    sr_tgi = FMat(np.divide(rg.residuals(garch_tgi), rg.sigma(garch_tgi)))
    sr_s30gi = FMat(np.divide(rg.residuals(garch_s30gi), rg.sigma(garch_s30gi)))
    sr_igi = FMat(np.divide(rg.residuals(garch_igi), rg.sigma(garch_igi)))
    sr_h25gi = FMat(np.divide(rg.residuals(garch_h25gi), rg.sigma(garch_h25gi)))
    sr_c25gi = FMat(np.divide(rg.residuals(garch_c25gi), rg.sigma(garch_c25gi)))
    sr_obx = FMat(np.divide(rg.residuals(garch_obx), rg.sigma(garch_obx)))

# Fitting the NIG distribution to marginals:
p_nig_vgi = genhyp.nigFit(sr_vgi)
p_nig_rgi = genhyp.nigFit(sr_rgi)
p_nig_tgi = genhyp.nigFit(sr_tgi)
p_nig_s30gi = genhyp.nigFit(sr_s30gi)
p_nig_igi = genhyp.nigFit(sr_igi)
p_nig_h25gi = genhyp.nigFit(sr_h25gi)
p_nig_c25gi = genhyp.nigFit(sr_c25gi)
p_nig_obx = genhyp.nigFit(sr_obx)

# Converting to uniform marginals:
unif_vgi = genhyp.pghyp(sr_vgi, param=p_vector(p_nig_vgi))
unif_rgi = genhyp.pghyp(sr_rgi, param=p_vector(p_nig_rgi))
unif_tgi = genhyp.pghyp(sr_tgi, param=p_vector(p_nig_tgi))
unif_s30gi = genhyp.pghyp(sr_s30gi, param=p_vector(p_nig_s30gi))
unif_igi = genhyp.pghyp(sr_igi, param=p_vector(p_nig_igi))
unif_h25gi = genhyp.pghyp(sr_h25gi, param=p_vector(p_nig_h25gi))
unif_c25gi = genhyp.pghyp(sr_c25gi, param=p_vector(p_nig_c25gi))
unif_obx = genhyp.pghyp(sr_obx, param=p_vector(p_nig_obx))

unif_marg_no_vgi = base.cbind(unif_rgi=unif_rgi, unif_tgi=unif_tgi,
                              unif_s30gi=unif_s30gi, unif_igi=unif_igi,
                              unif_h25gi=unif_h25gi, unif_c25gi=unif_c25gi,
                              unif_obx=unif_obx)
unif_mat_no_vgi = base.as_matrix(unif_marg_no_vgi)
```

```
unif_marg_no_rgi = base.cbind(unif_vgi=unif_vgi, unif_tgi=unif_tgi,
                              unif_s30gi=unif_s30gi, unif_igi=unif_igi,
                              unif_h25gi=unif_h25gi, unif_c25gi=unif_c25gi,
                              unif_obx=unif_obx)
unif_mat_no_rgi = base.as_matrix(unif_marg_no_rgi)

unif_marg_no_tgi = base.cbind(unif_vgi=unif_vgi, unif_rgi=unif_rgi,
                              unif_s30gi=unif_s30gi, unif_igi=unif_igi,
                              unif_h25gi=unif_h25gi, unif_c25gi=unif_c25gi,
                              unif_obx=unif_obx)
unif_mat_no_tgi = base.as_matrix(unif_marg_no_tgi)

unif_marg_no_s30gi = base.cbind(unif_vgi=unif_vgi, unif_rgi=unif_rgi,
                                unif_tgi=unif_tgi, unif_igi=unif_igi,
                                unif_h25gi=unif_h25gi, unif_c25gi=unif_c25gi,
                                unif_obx=unif_obx)
unif_mat_no_s30gi = base.as_matrix(unif_marg_no_s30gi)

unif_marg_no_igi = base.cbind(unif_vgi=unif_vgi, unif_rgi=unif_rgi,
                              unif_tgi=unif_tgi, unif_s30gi=unif_s30gi,
                              unif_h25gi=unif_h25gi, unif_c25gi=unif_c25gi,
                              unif_obx=unif_obx)
unif_mat_no_igi = base.as_matrix(unif_marg_no_igi)

unif_marg_no_h25gi = base.cbind(unif_vgi=unif_vgi, unif_rgi=unif_rgi,
                                unif_tgi=unif_tgi, unif_s30gi=unif_s30gi,
                                unif_igi=unif_igi, unif_c25gi=unif_c25gi,
                                unif_obx=unif_obx)
unif_mat_no_h25gi = base.as_matrix(unif_marg_no_h25gi)

unif_marg_no_c25gi = base.cbind(unif_vgi=unif_vgi, unif_rgi=unif_rgi,
                                unif_tgi=unif_tgi,
                                unif_s30gi=unif_s30gi, unif_igi=unif_igi,
                                unif_h25gi=unif_h25gi, unif_obx=unif_obx)
unif_mat_no_c25gi = base.as_matrix(unif_marg_no_c25gi)

unif_marg_no_obx = base.cbind(unif_vgi=unif_vgi, unif_rgi=unif_rgi,
                              unif_tgi=unif_tgi,
                              unif_s30gi=unif_s30gi, unif_igi=unif_igi,
                              unif_h25gi=unif_h25gi, unif_c25gi=unif_c25gi)
unif_mat_no_obx = base.as_matrix(unif_marg_no_obx)

# Fitting copulas and simulating from them:
if type == 'Clayton':
    fit_HAC_Clayton_no_vgi = HAC.estimate_copula(unif_mat_no_vgi, type=3)
    hac_obj_no_vgi = HAC.hac(type=3, tree=fit_HAC_Clayton_no_vgi.rx2('tree'))

    fit_HAC_Clayton_no_rgi = HAC.estimate_copula(unif_mat_no_rgi, type=3)
    hac_obj_no_rgi = HAC.hac(type=3, tree=fit_HAC_Clayton_no_rgi.rx2('tree'))

    fit_HAC_Clayton_no_tgi = HAC.estimate_copula(unif_mat_no_tgi, type=3)
    hac_obj_no_tgi = HAC.hac(type=3, tree=fit_HAC_Clayton_no_tgi.rx2('tree'))

    fit_HAC_Clayton_no_s30gi = HAC.estimate_copula(unif_mat_no_s30gi, type=3)
    hac_obj_no_s30gi = HAC.hac(type=3, tree=fit_HAC_Clayton_no_s30gi.rx2('tree'))

    fit_HAC_Clayton_no_igi = HAC.estimate_copula(unif_mat_no_igi, type=3)
    hac_obj_no_igi = HAC.hac(type=3, tree=fit_HAC_Clayton_no_igi.rx2('tree'))

    fit_HAC_Clayton_no_h25gi = HAC.estimate_copula(unif_mat_no_h25gi, type=3)
    hac_obj_no_h25gi = HAC.hac(type=3, tree=fit_HAC_Clayton_no_h25gi.rx2('tree'))
```

```
    fit_HAC_Clayton_no_c25gi = HAC.estimate_copula(unif_mat_no_c25gi, type=3)
    hac_obj_no_c25gi = HAC.hac(type=3, tree=fit_HAC_Clayton_no_c25gi.rx2('tree'))

    fit_HAC_Clayton_no_obx = HAC.estimate_copula(unif_mat_no_obx, type=3)
    hac_obj_no_obx = HAC.hac(type=3, tree=fit_HAC_Clayton_no_obx.rx2('tree'))

sim_unif_no_vgi = HAC.rHAC(nsim, hac_obj_no_vgi)
sim_unif_df_no_vgi = base.as_data_frame(sim_unif_no_vgi)

sim_unif_no_rgi = HAC.rHAC(nsim, hac_obj_no_rgi)
sim_unif_df_no_rgi = base.as_data_frame(sim_unif_no_rgi)

sim_unif_no_tgi = HAC.rHAC(nsim, hac_obj_no_tgi)
sim_unif_df_no_tgi = base.as_data_frame(sim_unif_no_tgi)

sim_unif_no_s30gi = HAC.rHAC(nsim, hac_obj_no_s30gi)
sim_unif_df_no_s30gi = base.as_data_frame(sim_unif_no_s30gi)

sim_unif_no_igi = HAC.rHAC(nsim, hac_obj_no_igi)
sim_unif_df_no_igi = base.as_data_frame(sim_unif_no_igi)

sim_unif_no_h25gi = HAC.rHAC(nsim, hac_obj_no_h25gi)
sim_unif_df_no_h25gi = base.as_data_frame(sim_unif_no_h25gi)

sim_unif_no_c25gi = HAC.rHAC(nsim, hac_obj_no_c25gi)
sim_unif_df_no_c25gi = base.as_data_frame(sim_unif_no_c25gi)

sim_unif_no_obx = HAC.rHAC(nsim, hac_obj_no_obx)
sim_unif_df_no_obx = base.as_data_frame(sim_unif_no_obx)

# Converting the simulated values to standardised residuals:
sim_rgi_no_vgi = genhyp.qghyp(sim_unif_df_no_vgi.rx2("unif_rgi"),
                              param=p_vector(p_nig_rgi))
sim_tgi_no_vgi = genhyp.qghyp(sim_unif_df_no_vgi.rx2("unif_tgi"),
                              param=p_vector(p_nig_tgi))
sim_s30gi_no_vgi = genhyp.qghyp(sim_unif_df_no_vgi.rx2("unif_s30gi"),
                                param=p_vector(p_nig_s30gi))
sim_igi_no_vgi = genhyp.qghyp(sim_unif_df_no_vgi.rx2("unif_igi"),
                              param=p_vector(p_nig_igi))
sim_h25gi_no_vgi = genhyp.qghyp(sim_unif_df_no_vgi.rx2("unif_h25gi"),
                                param=p_vector(p_nig_h25gi))
sim_c25gi_no_vgi = genhyp.qghyp(sim_unif_df_no_vgi.rx2("unif_c25gi"),
                                param=p_vector(p_nig_c25gi))
sim_obx_no_vgi = genhyp.qghyp(sim_unif_df_no_vgi.rx2("unif_obx"),
                              param=p_vector(p_nig_obx))

sim_vgi_no_rgi = genhyp.qghyp(sim_unif_df_no_rgi.rx2("unif_vgi"),
                              param=p_vector(p_nig_vgi))
sim_tgi_no_rgi = genhyp.qghyp(sim_unif_df_no_rgi.rx2("unif_tgi"),
                              param=p_vector(p_nig_tgi))
sim_s30gi_no_rgi = genhyp.qghyp(sim_unif_df_no_rgi.rx2("unif_s30gi"),
                                param=p_vector(p_nig_s30gi))
sim_igi_no_rgi = genhyp.qghyp(sim_unif_df_no_rgi.rx2("unif_igi"),
                              param=p_vector(p_nig_igi))
sim_h25gi_no_rgi = genhyp.qghyp(sim_unif_df_no_rgi.rx2("unif_h25gi"),
                                param=p_vector(p_nig_h25gi))
sim_c25gi_no_rgi = genhyp.qghyp(sim_unif_df_no_rgi.rx2("unif_c25gi"),
                                param=p_vector(p_nig_c25gi))
sim_obx_no_rgi = genhyp.qghyp(sim_unif_df_no_rgi.rx2("unif_obx"),
                              param=p_vector(p_nig_obx))

sim_vgi_no_tgi = genhyp.qghyp(sim_unif_df_no_tgi.rx2("unif_vgi"),
                              param=p_vector(p_nig_vgi))
```

```
sim_rgi_no_tgi = genhyp.qghyp(sim_unif_df_no_tgi.rx2("unif_rgi"),
                              param=p_vector(p_nig_rgi))
sim_s30gi_no_tgi = genhyp.qghyp(sim_unif_df_no_tgi.rx2("unif_s30gi"),
                                param=p_vector(p_nig_s30gi))
sim_igi_no_tgi = genhyp.qghyp(sim_unif_df_no_tgi.rx2("unif_igi"),
                              param=p_vector(p_nig_igi))
sim_h25gi_no_tgi = genhyp.qghyp(sim_unif_df_no_tgi.rx2("unif_h25gi"),
                                param=p_vector(p_nig_h25gi))
sim_c25gi_no_tgi = genhyp.qghyp(sim_unif_df_no_tgi.rx2("unif_c25gi"),
                                param=p_vector(p_nig_c25gi))
sim_obx_no_tgi = genhyp.qghyp(sim_unif_df_no_tgi.rx2("unif_obx"),
                              param=p_vector(p_nig_obx))

sim_vgi_no_s30gi = genhyp.qghyp(sim_unif_df_no_s30gi.rx2("unif_vgi"),
                                param=p_vector(p_nig_vgi))
sim_rgi_no_s30gi = genhyp.qghyp(sim_unif_df_no_s30gi.rx2("unif_rgi"),
                                param=p_vector(p_nig_rgi))
sim_tgi_no_s30gi = genhyp.qghyp(sim_unif_df_no_s30gi.rx2("unif_tgi"),
                                param=p_vector(p_nig_tgi))
sim_igi_no_s30gi = genhyp.qghyp(sim_unif_df_no_s30gi.rx2("unif_igi"),
                                param=p_vector(p_nig_igi))
sim_h25gi_no_s30gi = genhyp.qghyp(sim_unif_df_no_s30gi.rx2("unif_h25gi"),
                                  param=p_vector(p_nig_h25gi))
sim_c25gi_no_s30gi = genhyp.qghyp(sim_unif_df_no_s30gi.rx2("unif_c25gi"),
                                  param=p_vector(p_nig_c25gi))
sim_obx_no_s30gi = genhyp.qghyp(sim_unif_df_no_s30gi.rx2("unif_obx"),
                                param=p_vector(p_nig_obx))

sim_vgi_no_igi = genhyp.qghyp(sim_unif_df_no_igi.rx2("unif_vgi"),
                              param=p_vector(p_nig_vgi))
sim_rgi_no_igi = genhyp.qghyp(sim_unif_df_no_igi.rx2("unif_rgi"),
                              param=p_vector(p_nig_rgi))
sim_tgi_no_igi = genhyp.qghyp(sim_unif_df_no_igi.rx2("unif_tgi"),
                              param=p_vector(p_nig_tgi))
sim_s30gi_no_igi = genhyp.qghyp(sim_unif_df_no_igi.rx2("unif_s30gi"),
                                param=p_vector(p_nig_s30gi))
sim_h25gi_no_igi = genhyp.qghyp(sim_unif_df_no_igi.rx2("unif_h25gi"),
                                param=p_vector(p_nig_h25gi))
sim_c25gi_no_igi = genhyp.qghyp(sim_unif_df_no_igi.rx2("unif_c25gi"),
                                param=p_vector(p_nig_c25gi))
sim_obx_no_igi = genhyp.qghyp(sim_unif_df_no_igi.rx2("unif_obx"),
                              param=p_vector(p_nig_obx))

sim_vgi_no_h25gi = genhyp.qghyp(sim_unif_df_no_h25gi.rx2("unif_vgi"),
                                param=p_vector(p_nig_vgi))
sim_rgi_no_h25gi = genhyp.qghyp(sim_unif_df_no_h25gi.rx2("unif_rgi"),
                                param=p_vector(p_nig_rgi))
sim_tgi_no_h25gi = genhyp.qghyp(sim_unif_df_no_h25gi.rx2("unif_tgi"),
                                param=p_vector(p_nig_tgi))
sim_s30gi_no_h25gi = genhyp.qghyp(sim_unif_df_no_h25gi.rx2("unif_s30gi"),
                                  param=p_vector(p_nig_s30gi))
sim_igi_no_h25gi = genhyp.qghyp(sim_unif_df_no_h25gi.rx2("unif_igi"),
                                param=p_vector(p_nig_igi))
sim_c25gi_no_h25gi = genhyp.qghyp(sim_unif_df_no_h25gi.rx2("unif_c25gi"),
                                  param=p_vector(p_nig_c25gi))
sim_obx_no_h25gi = genhyp.qghyp(sim_unif_df_no_h25gi.rx2("unif_obx"),
                                param=p_vector(p_nig_obx))

sim_vgi_no_c25gi = genhyp.qghyp(sim_unif_df_no_c25gi.rx2("unif_vgi"),
                                param=p_vector(p_nig_vgi))
sim_rgi_no_c25gi = genhyp.qghyp(sim_unif_df_no_c25gi.rx2("unif_rgi"),
                                param=p_vector(p_nig_rgi))
sim_tgi_no_c25gi = genhyp.qghyp(sim_unif_df_no_c25gi.rx2("unif_tgi"),
```

```
                                param=p_vector(p_nig_tgi))
sim_s30gi_no_c25gi = genhyp.qghyp(sim_unif_df_no_c25gi.rx2("unif_s30gi"),
                                  param=p_vector(p_nig_s30gi))
sim_igi_no_c25gi = genhyp.qghyp(sim_unif_df_no_c25gi.rx2("unif_igi"),
                                param=p_vector(p_nig_igi))
sim_h25gi_no_c25gi = genhyp.qghyp(sim_unif_df_no_c25gi.rx2("unif_h25gi"),
                                  param=p_vector(p_nig_h25gi))
sim_obx_no_c25gi = genhyp.qghyp(sim_unif_df_no_c25gi.rx2("unif_obx"),
                                param=p_vector(p_nig_obx))

sim_vgi_no_obx = genhyp.qghyp(sim_unif_df_no_obx.rx2("unif_vgi"),
                              param=p_vector(p_nig_vgi))
sim_rgi_no_obx = genhyp.qghyp(sim_unif_df_no_obx.rx2("unif_rgi"),
                              param=p_vector(p_nig_rgi))
sim_tgi_no_obx = genhyp.qghyp(sim_unif_df_no_obx.rx2("unif_tgi"),
                              param=p_vector(p_nig_tgi))
sim_s30gi_no_obx = genhyp.qghyp(sim_unif_df_no_obx.rx2("unif_s30gi"),
                                param=p_vector(p_nig_s30gi))
sim_igi_no_obx = genhyp.qghyp(sim_unif_df_no_obx.rx2("unif_igi"),
                              param=p_vector(p_nig_igi))
sim_h25gi_no_obx = genhyp.qghyp(sim_unif_df_no_obx.rx2("unif_h25gi"),
                                param=p_vector(p_nig_h25gi))
sim_c25gi_no_obx = genhyp.qghyp(sim_unif_df_no_obx.rx2("unif_c25gi"),
                                param=p_vector(p_nig_c25gi))

# Simulated log returns:
res_rgi_no_vgi = r['+'](r['*'](sim_rgi_no_vgi, sigma_rgi), mu_rgi)
res_tgi_no_vgi = r['+'](r['*'](sim_tgi_no_vgi, sigma_tgi), mu_tgi)
res_s30gi_no_vgi = r['+'](r['*'](sim_s30gi_no_vgi, sigma_s30gi), mu_s30gi)
res_igi_no_vgi = r['+'](r['*'](sim_igi_no_vgi, sigma_igi), mu_igi)
res_h25gi_no_vgi = r['+'](r['*'](sim_h25gi_no_vgi, sigma_h25gi), mu_h25gi)
res_c25gi_no_vgi = r['+'](r['*'](sim_c25gi_no_vgi, sigma_c25gi), mu_c25gi)
res_obx_no_vgi = r['+'](r['*'](sim_obx_no_vgi, sigma_obx), mu_obx)
logr_scenarios_no_vgi = base.cbind(
    logr_rgi_eur=res_rgi_no_vgi, logr_tgi_eur=res_tgi_no_vgi,
    logr_s30gi_eur=res_s30gi_no_vgi, logr_igi_eur=res_igi_no_vgi,
    logr_h25gi_eur=res_h25gi_no_vgi, logr_c25gi_eur=res_c25gi_no_vgi,
    logr_obx_eur=res_obx_no_vgi,
)

res_vgi_no_rgi = r['+'](r['*'](sim_vgi_no_rgi, sigma_vgi), mu_vgi)
res_tgi_no_rgi = r['+'](r['*'](sim_tgi_no_rgi, sigma_tgi), mu_tgi)
res_s30gi_no_rgi = r['+'](r['*'](sim_s30gi_no_rgi, sigma_s30gi), mu_s30gi)
res_igi_no_rgi = r['+'](r['*'](sim_igi_no_rgi, sigma_igi), mu_igi)
res_h25gi_no_rgi = r['+'](r['*'](sim_h25gi_no_rgi, sigma_h25gi), mu_h25gi)
res_c25gi_no_rgi = r['+'](r['*'](sim_c25gi_no_rgi, sigma_c25gi), mu_c25gi)
res_obx_no_rgi = r['+'](r['*'](sim_obx_no_rgi, sigma_obx), mu_obx)
logr_scenarios_no_rgi = base.cbind(
    logr_vgi_eur=res_vgi_no_rgi, logr_tgi_eur=res_tgi_no_rgi,
    logr_s30gi_eur=res_s30gi_no_rgi, logr_igi_eur=res_igi_no_rgi,
    logr_h25gi_eur=res_h25gi_no_rgi, logr_c25gi_eur=res_c25gi_no_rgi,
    logr_obx_eur=res_obx_no_rgi,
)

res_vgi_no_tgi = r['+'](r['*'](sim_vgi_no_tgi, sigma_vgi), mu_vgi)
res_rgi_no_tgi = r['+'](r['*'](sim_rgi_no_tgi, sigma_rgi), mu_rgi)
res_s30gi_no_tgi = r['+'](r['*'](sim_s30gi_no_tgi, sigma_s30gi), mu_s30gi)
res_igi_no_tgi = r['+'](r['*'](sim_igi_no_tgi, sigma_igi), mu_igi)
res_h25gi_no_tgi = r['+'](r['*'](sim_h25gi_no_tgi, sigma_h25gi), mu_h25gi)
res_c25gi_no_tgi = r['+'](r['*'](sim_c25gi_no_tgi, sigma_c25gi), mu_c25gi)
res_obx_no_tgi = r['+'](r['*'](sim_obx_no_tgi, sigma_obx), mu_obx)
logr_scenarios_no_tgi = base.cbind(
    logr_vgi_eur=res_vgi_no_tgi, logr_rgi_eur=res_rgi_no_tgi,
```

```
    logr_s30gi_eur=res_s30gi_no_tgi, logr_igi_eur=res_igi_no_tgi,
    logr_h25gi_eur=res_h25gi_no_tgi, logr_c25gi_eur=res_c25gi_no_tgi,
    logr_obx_eur=res_obx_no_tgi,
)

res_vgi_no_s30gi = r['+'](r['*'](sim_vgi_no_s30gi, sigma_vgi), mu_vgi)
res_rgi_no_s30gi = r['+'](r['*'](sim_rgi_no_s30gi, sigma_rgi), mu_rgi)
res_tgi_no_s30gi = r['+'](r['*'](sim_tgi_no_s30gi, sigma_tgi), mu_tgi)
res_igi_no_s30gi = r['+'](r['*'](sim_igi_no_s30gi, sigma_igi), mu_igi)
res_h25gi_no_s30gi = r['+'](r['*'](sim_h25gi_no_s30gi, sigma_h25gi), mu_h25gi)
res_c25gi_no_s30gi = r['+'](r['*'](sim_c25gi_no_s30gi, sigma_c25gi), mu_c25gi)
res_obx_no_s30gi = r['+'](r['*'](sim_obx_no_s30gi, sigma_obx), mu_obx)
logr_scenarios_no_s30gi = base.cbind(
    logr_vgi_eur=res_vgi_no_s30gi, logr_rgi_eur=res_rgi_no_s30gi,
    logr_tgi_eur=res_tgi_no_s30gi, logr_igi_eur=res_igi_no_s30gi,
    logr_h25gi_eur=res_h25gi_no_s30gi, logr_c25gi_eur=res_c25gi_no_s30gi,
    logr_obx_eur=res_obx_no_s30gi,
)

res_vgi_no_igi = r['+'](r['*'](sim_vgi_no_igi, sigma_vgi), mu_vgi)
res_rgi_no_igi = r['+'](r['*'](sim_rgi_no_igi, sigma_rgi), mu_rgi)
res_tgi_no_igi = r['+'](r['*'](sim_tgi_no_igi, sigma_tgi), mu_tgi)
res_s30gi_no_igi = r['+'](r['*'](sim_s30gi_no_igi, sigma_s30gi), mu_s30gi)
res_h25gi_no_igi = r['+'](r['*'](sim_h25gi_no_igi, sigma_h25gi), mu_h25gi)
res_c25gi_no_igi = r['+'](r['*'](sim_c25gi_no_igi, sigma_c25gi), mu_c25gi)
res_obx_no_igi = r['+'](r['*'](sim_obx_no_igi, sigma_obx), mu_obx)
logr_scenarios_no_igi = base.cbind(
    logr_vgi_eur=res_vgi_no_igi, logr_rgi_eur=res_rgi_no_igi,
    logr_tgi_eur=res_tgi_no_igi, logr_s30gi_eur=res_s30gi_no_igi,
    logr_h25gi_eur=res_h25gi_no_igi, logr_c25gi_eur=res_c25gi_no_igi,
    logr_obx_eur=res_obx_no_igi,
)

res_vgi_no_h25gi = r['+'](r['*'](sim_vgi_no_h25gi, sigma_vgi), mu_vgi)
res_rgi_no_h25gi = r['+'](r['*'](sim_rgi_no_h25gi, sigma_rgi), mu_rgi)
res_tgi_no_h25gi = r['+'](r['*'](sim_tgi_no_h25gi, sigma_tgi), mu_tgi)
res_s30gi_no_h25gi = r['+'](r['*'](sim_s30gi_no_h25gi, sigma_s30gi), mu_s30gi)
res_igi_no_h25gi = r['+'](r['*'](sim_igi_no_h25gi, sigma_igi), mu_igi)
res_c25gi_no_h25gi = r['+'](r['*'](sim_c25gi_no_h25gi, sigma_c25gi), mu_c25gi)
res_obx_no_h25gi = r['+'](r['*'](sim_obx_no_h25gi, sigma_obx), mu_obx)
logr_scenarios_no_h25gi = base.cbind(
    logr_vgi_eur=res_vgi_no_h25gi, logr_rgi_eur=res_rgi_no_h25gi,
    logr_tgi_eur=res_tgi_no_h25gi, logr_s30gi_eur=res_s30gi_no_h25gi,
    logr_igi_eur=res_igi_no_h25gi, logr_c25gi_eur=res_c25gi_no_h25gi,
    logr_obx_eur=res_obx_no_h25gi,
)

res_vgi_no_c25gi = r['+'](r['*'](sim_vgi_no_c25gi, sigma_vgi), mu_vgi)
res_rgi_no_c25gi = r['+'](r['*'](sim_rgi_no_c25gi, sigma_rgi), mu_rgi)
res_tgi_no_c25gi = r['+'](r['*'](sim_tgi_no_c25gi, sigma_tgi), mu_tgi)
res_s30gi_no_c25gi = r['+'](r['*'](sim_s30gi_no_c25gi, sigma_s30gi), mu_s30gi)
res_igi_no_c25gi = r['+'](r['*'](sim_igi_no_c25gi, sigma_igi), mu_igi)
res_h25gi_no_c25gi = r['+'](r['*'](sim_h25gi_no_c25gi, sigma_h25gi), mu_h25gi)
res_obx_no_c25gi = r['+'](r['*'](sim_obx_no_c25gi, sigma_obx), mu_obx)
logr_scenarios_no_c25gi = base.cbind(
    logr_vgi_eur=res_vgi_no_c25gi, logr_rgi_eur=res_rgi_no_c25gi,
    logr_tgi_eur=res_tgi_no_c25gi, logr_s30gi_eur=res_s30gi_no_c25gi,
    logr_igi_eur=res_igi_no_c25gi, logr_h25gi_eur=res_h25gi_no_c25gi,
    logr_obx_eur=res_obx_no_c25gi,
)

res_vgi_no_obx = r['+'](r['*'](sim_vgi_no_obx, sigma_vgi), mu_vgi)
res_rgi_no_obx = r['+'](r['*'](sim_rgi_no_obx, sigma_rgi), mu_rgi)
```

```
res_tgi_no_obx = r['+'](r['*'](sim_tgi_no_obx, sigma_tgi), mu_tgi)
res_s30gi_no_obx = r['+'](r['*'](sim_s30gi_no_obx, sigma_s30gi), mu_s30gi)
res_igi_no_obx = r['+'](r['*'](sim_igi_no_obx, sigma_igi), mu_igi)
res_h25gi_no_obx = r['+'](r['*'](sim_h25gi_no_obx, sigma_h25gi), mu_h25gi)
res_c25gi_no_obx = r['+'](r['*'](sim_c25gi_no_obx, sigma_c25gi), mu_c25gi)
logr_scenarios_no_obx = base.cbind(
    logr_vgi_eur=res_vgi_no_obx, logr_rgi_eur=res_rgi_no_obx,
    logr_tgi_eur=res_tgi_no_obx, logr_s30gi_eur=res_s30gi_no_obx,
    logr_igi_eur=res_igi_no_obx, logr_h25gi_eur=res_h25gi_no_obx,
    logr_c25gi_eur=res_c25gi_no_obx,
)

logr_scenarios_df_no_vgi = pd.DataFrame(np.array(logr_scenarios_no_vgi),
                                        columns=without(logr_columns,
                                                        'logr_vgi_eur'))
logr_scenarios_df_no_rgi = pd.DataFrame(np.array(logr_scenarios_no_rgi),
                                        columns=without(logr_columns,
                                                        'logr_rgi_eur'))
logr_scenarios_df_no_tgi = pd.DataFrame(np.array(logr_scenarios_no_tgi),
                                        columns=without(logr_columns,
                                                        'logr_tgi_eur'))
logr_scenarios_df_no_s30gi = pd.DataFrame(np.array(logr_scenarios_no_s30gi),
                                          columns=without(logr_columns,
                                                          'logr_s30gi_eur'))
logr_scenarios_df_no_igi = pd.DataFrame(np.array(logr_scenarios_no_igi),
                                        columns=without(logr_columns,
                                                        'logr_igi_eur'))
logr_scenarios_df_no_h25gi = pd.DataFrame(np.array(logr_scenarios_no_h25gi),
                                          columns=without(logr_columns,
                                                          'logr_h25gi_eur'))
logr_scenarios_df_no_c25gi = pd.DataFrame(np.array(logr_scenarios_no_c25gi),
                                          columns=without(logr_columns,
                                                          'logr_c25gi_eur'))
logr_scenarios_df_no_obx = pd.DataFrame(np.array(logr_scenarios_no_obx),
                                        columns=without(logr_columns,
                                                        'logr_obx_eur'))

PV_scenarios_no_vgi = pd.DataFrame()
PV_scenarios_no_rgi = pd.DataFrame()
PV_scenarios_no_tgi = pd.DataFrame()
PV_scenarios_no_s30gi = pd.DataFrame()
PV_scenarios_no_igi = pd.DataFrame()
PV_scenarios_no_h25gi = pd.DataFrame()
PV_scenarios_no_c25gi = pd.DataFrame()
PV_scenarios_no_obx = pd.DataFrame()
time_filter = (index_prices_df['Date'] <= study_date)

for ind in without(logr_columns, 'logr_vgi_eur'):
    # returns between i+1 and i multiplied by the nth price:
    nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
    PV_scenarios_no_vgi[ind] = np.exp(logr_scenarios_df_no_vgi[ind]) * nth_price
PV_scenarios_no_vgi['PV'] = PV_scenarios_no_vgi.mul([10] * 7).sum(axis=1)
# Log returns between generated PVs and the actual PV on nth day:
nth_PV = index_prices_df.loc[time_filter, 'PV_no_vgi'].tail(1).iloc[0]
PV_scenarios_no_vgi['log_r_PV'] = np.log(PV_scenarios_no_vgi['PV'] / nth_PV)

for ind in without(logr_columns, 'logr_rgi_eur'):
    # returns between i+1 and i multiplied by the nth price:
    nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
    PV_scenarios_no_rgi[ind] = np.exp(logr_scenarios_df_no_rgi[ind]) * nth_price
PV_scenarios_no_rgi['PV'] = PV_scenarios_no_rgi.mul([10] * 7).sum(axis=1)
# Log returns between generated PVs and the actual PV on nth day:
nth_PV = index_prices_df.loc[time_filter, 'PV_no_rgi'].tail(1).iloc[0]
```

```
PV_scenarios_no_rgi['log_r_PV'] = np.log(PV_scenarios_no_rgi['PV'] / nth_PV)

for ind in without(logr_columns, 'logr_tgi_eur'):
    # returns between i+1 and i multiplied by the nth price:
    nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
    PV_scenarios_no_tgi[ind] = np.exp(logr_scenarios_df_no_tgi[ind]) * nth_price
PV_scenarios_no_tgi['PV'] = PV_scenarios_no_tgi.mul([10] * 7).sum(axis=1)
# Log returns between generated PVs and the actual PV on nth day:
nth_PV = index_prices_df.loc[time_filter, 'PV_no_tgi'].tail(1).iloc[0]
PV_scenarios_no_tgi['log_r_PV'] = np.log(PV_scenarios_no_tgi['PV'] / nth_PV)

for ind in without(logr_columns, 'logr_s30gi_eur'):
    # returns between i+1 and i multiplied by the nth price:
    nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
    PV_scenarios_no_s30gi[ind] = np.exp(logr_scenarios_df_no_s30gi[ind]) * nth_price
PV_scenarios_no_s30gi['PV'] = PV_scenarios_no_s30gi.mul([10] * 7).sum(axis=1)
# Log returns between generated PVs and the actual PV on nth day:
nth_PV = index_prices_df.loc[time_filter, 'PV_no_s30gi'].tail(1).iloc[0]
PV_scenarios_no_s30gi['log_r_PV'] = np.log(PV_scenarios_no_s30gi['PV'] / nth_PV)

for ind in without(logr_columns, 'logr_igi_eur'):
    # returns between i+1 and i multiplied by the nth price:
    nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
    PV_scenarios_no_igi[ind] = np.exp(logr_scenarios_df_no_igi[ind]) * nth_price
PV_scenarios_no_igi['PV'] = PV_scenarios_no_igi.mul([10] * 7).sum(axis=1)
# Log returns between generated PVs and the actual PV on nth day:
nth_PV = index_prices_df.loc[time_filter, 'PV_no_igi'].tail(1).iloc[0]
PV_scenarios_no_igi['log_r_PV'] = np.log(PV_scenarios_no_igi['PV'] / nth_PV)

for ind in without(logr_columns, 'logr_h25gi_eur'):
    # returns between i+1 and i multiplied by the nth price:
    nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
    PV_scenarios_no_h25gi[ind] = np.exp(logr_scenarios_df_no_h25gi[ind]) * nth_price
PV_scenarios_no_h25gi['PV'] = PV_scenarios_no_h25gi.mul([10] * 7).sum(axis=1)
# Log returns between generated PVs and the actual PV on nth day:
nth_PV = index_prices_df.loc[time_filter, 'PV_no_h25gi'].tail(1).iloc[0]
PV_scenarios_no_h25gi['log_r_PV'] = np.log(PV_scenarios_no_h25gi['PV'] / nth_PV)

for ind in without(logr_columns, 'logr_c25gi_eur'):
    # returns between i+1 and i multiplied by the nth price:
    nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
    PV_scenarios_no_c25gi[ind] = np.exp(logr_scenarios_df_no_c25gi[ind]) * nth_price
PV_scenarios_no_c25gi['PV'] = PV_scenarios_no_c25gi.mul([10] * 7).sum(axis=1)
# Log returns between generated PVs and the actual PV on nth day:
nth_PV = index_prices_df.loc[time_filter, 'PV_no_c25gi'].tail(1).iloc[0]
PV_scenarios_no_c25gi['log_r_PV'] = np.log(PV_scenarios_no_c25gi['PV'] / nth_PV)

for ind in without(logr_columns, 'logr_obx_eur'):
    # returns between i+1 and i multiplied by the nth price:
    nth_price = index_prices_df.loc[time_filter, col_match[ind]].tail(1).iloc[0]
    PV_scenarios_no_obx[ind] = np.exp(logr_scenarios_df_no_obx[ind]) * nth_price
PV_scenarios_no_obx['PV'] = PV_scenarios_no_obx.mul([10] * 7).sum(axis=1)
# Log returns between generated PVs and the actual PV on nth day:
nth_PV = index_prices_df.loc[time_filter, 'PV_no_obx'].tail(1).iloc[0]
PV_scenarios_no_obx['log_r_PV'] = np.log(PV_scenarios_no_obx['PV'] / nth_PV)

def produce_est_row(PV_scenarios, return_df, index_prices_df, study_date):
    est_q = 'median_unbiased'
    VaR_95 = np.percentile(PV_scenarios['log_r_PV'], 100-95, method=est_q)
    VaR_975 = np.percentile(PV_scenarios['log_r_PV'], 100-97.5, method=est_q)
    VaR_99 = np.percentile(PV_scenarios['log_r_PV'], 100-99, method=est_q)
    # Additional estimates for multinomial backtesting:
    VaR_9625 = np.percentile(PV_scenarios['log_r_PV'], 100-96.25, method=est_q)
```

```
        VaR_98125 = np.percentile(PV_scenarios['log_r_PV'], 100-98.125, method=est_q)
        VaR_9875 = np.percentile(PV_scenarios['log_r_PV'], 100-98.75, method=est_q)
        VaR_9925 = np.percentile(PV_scenarios['log_r_PV'], 100-99.25, method=est_q)
        VaR_99375 = np.percentile(PV_scenarios['log_r_PV'], 100-99.375, method=est_q)
        VaR_995 = np.percentile(PV_scenarios['log_r_PV'], 100-99.5, method=est_q)
        VaR_9975 = np.percentile(PV_scenarios['log_r_PV'], 100-99.75, method=est_q)
        date_filter = (index_prices_df['Date'] > study_date)
        estimates_row = {
            'forecast_date': return_df.loc[return_df['Date'] > study_date,
                                           'Date'].head(1).iloc[0],
            'VaR_95': VaR_95,
            'VaR_975': VaR_975,
            'VaR_99': VaR_99,
            'VaR_9625': VaR_9625,
            'VaR_98125': VaR_98125,
            'VaR_9875': VaR_9875,
            'VaR_9925': VaR_9925,
            'VaR_99375': VaR_99375,
            'VaR_995': VaR_995,
            'VaR_9975': VaR_9975,
            'ES_95': np.mean(PV_scenarios.loc[PV_scenarios['log_r_PV'] <= VaR_95,
                                              'log_r_PV']),
            'ES_975': np.mean(PV_scenarios.loc[PV_scenarios['log_r_PV'] <= VaR_975,
                                               'log_r_PV']),
            'ES_99': np.mean(PV_scenarios.loc[PV_scenarios['log_r_PV'] <= VaR_99,
                                              'log_r_PV']),
            'actual_log_r': index_prices_df.loc[date_filter, 'log_r_PV'].head(1).iloc[0]
            }
        return pd.Series(estimates_row).to_frame().T

    est_no_vgi = produce_est_row(PV_scenarios_no_vgi, return_df, index_prices_df,
                                 study_date)
    est_no_rgi = produce_est_row(PV_scenarios_no_rgi, return_df, index_prices_df,
                                 study_date)
    est_no_tgi = produce_est_row(PV_scenarios_no_tgi, return_df, index_prices_df,
                                 study_date)
    est_no_s30gi = produce_est_row(PV_scenarios_no_s30gi, return_df, index_prices_df,
                                   study_date)
    est_no_igi = produce_est_row(PV_scenarios_no_igi, return_df, index_prices_df,
                                 study_date)
    est_no_h25gi = produce_est_row(PV_scenarios_no_h25gi, return_df, index_prices_df,
                                   study_date)
    est_no_c25gi = produce_est_row(PV_scenarios_no_c25gi, return_df, index_prices_df,
                                   study_date)
    est_no_obx = produce_est_row(PV_scenarios_no_obx, return_df, index_prices_df,
                                 study_date)

    return (
        est_no_vgi,
        est_no_rgi,
        est_no_tgi,
        est_no_s30gi,
        est_no_igi,
        est_no_h25gi,
        est_no_c25gi,
        est_no_obx,
    )


start = timer()
estimates_incr_VaR_HCC_1000 = VaR_ES_estimation_incr(
    VaR_ES_Copula, pd.DataFrame(),
    1000, index_prices_df=incr_prices_df,
```

```
    return_df=log_returns_df, method_in_R=True, c_type="Clayton",
)
print(f'{timer()-start} seconds passed')

for df in estimates_incr_VaR_HCC_1000:
    estimates_incr_VaR_HCC_1000[df].to_excel(f'Incremental_VaR_ES_HCC1000_{df}.xlsx')

no_c25gi = pd.read_excel("Incremental_VaR_ES_HCC1000_no_c25gi.xlsx", index_col=0)
no_h25gi = pd.read_excel("Incremental_VaR_ES_HCC1000_no_h25gi.xlsx", index_col=0)
no_igi = pd.read_excel("Incremental_VaR_ES_HCC1000_no_igi.xlsx", index_col=0)
no_obx = pd.read_excel("Incremental_VaR_ES_HCC1000_no_obx.xlsx", index_col=0)
no_rgi = pd.read_excel("Incremental_VaR_ES_HCC1000_no_rgi.xlsx", index_col=0)
no_s30gi = pd.read_excel("Incremental_VaR_ES_HCC1000_no_s30gi.xlsx", index_col=0)
no_tgi = pd.read_excel("Incremental_VaR_ES_HCC1000_no_tgi.xlsx", index_col=0)
no_vgi = pd.read_excel("Incremental_VaR_ES_HCC1000_no_vgi.xlsx", index_col=0)


def create_VaR_ES_df(df, PV_col):
    res = pd.concat(
        [
            df[["forecast_date"]].reset_index(drop=True),
            np.exp(df[[f"VaR_{95}"]]).reset_index(drop=True),
            np.exp(df[[f"VaR_{975}"]]).reset_index(drop=True),
            np.exp(df[[f"VaR_{99}"]]).reset_index(drop=True),
            incr_prices_df[[PV_col]].tail(812).head(811).reset_index(drop=True),
        ],
        axis=1,
    ).copy()
    for level in [95, 975, 99]:
        res[f'VaR_{level}_EUR'] = res[f'VaR_{level}'] * res[PV_col] - res[PV_col]
    return res


no_vgi_df_VaR = create_VaR_ES_df(no_vgi, 'PV_no_vgi')
no_rgi_df_VaR = create_VaR_ES_df(no_rgi, 'PV_no_rgi')
no_tgi_df_VaR = create_VaR_ES_df(no_tgi, 'PV_no_tgi')
no_s30gi_df_VaR = create_VaR_ES_df(no_s30gi, 'PV_no_s30gi')
no_igi_df_VaR = create_VaR_ES_df(no_igi, 'PV_no_igi')
no_h25gi_df_VaR = create_VaR_ES_df(no_h25gi, 'PV_no_h25gi')
no_c25gi_df_VaR = create_VaR_ES_df(no_c25gi, 'PV_no_c25gi')
no_obx_df_VaR = create_VaR_ES_df(no_obx, 'PV_no_obx')
HCC_1000_VaR = create_VaR_ES_df(estimates_Clayton_1000, 'PV')


def plot_IVaR(df_orig, df_asset, asset, level=95, level_in_graph=None,
              save_fig=True, fig_title=None, legend_loc='upper left', borderpad=0.4,
              estimate_col='#db0043'):
    if level_in_graph is None:
        level_in_graph = level
    if fig_title is None:
        fig_title = f'Incremental_VaR_{level}_{asset}'
    plt.figure(figsize=(10, 5), dpi=200)
    plt.xticks(fontsize=15)
    plt.yticks(fontsize=15)
    diff = (np.abs(df_orig[f'VaR_{level}_EUR']) - np.abs(df_asset[f'VaR_{level}_EUR']))
    plt.suptitle(f'{level_in_graph}% incremental VaR for {asset}', fontsize=18.5)
    plt.title(f'(mean = {round(np.mean(diff), 2)}, median = {round(np.median(diff),2)})',
              fontsize=15)
    plt.plot(df_orig['forecast_date'], diff, color=estimate_col,
             linestyle="-", lw=1.2, label=f'IVaR for {asset}')
    plt.xlabel('Date', fontsize=16.5)
    plt.ylabel('IVaR (EUR)', fontsize=16.5)
    plt.legend(fontsize=15, loc=legend_loc, borderpad=borderpad)
```

```
    plt.grid(alpha=0.4)
    if save_fig:
        plt.savefig(f'{fig_title}.pdf')
    plt.show()


plot_IVaR(HCC_1000_VaR, no_vgi_df_VaR, 'OMXVGI', legend_loc='upper right',
          estimate_col='#e3545b', level=95)
plot_IVaR(HCC_1000_VaR, no_vgi_df_VaR, 'OMXVGI', legend_loc='upper right',
          estimate_col='#e3545b', level=975, level_in_graph=97.5)
plot_IVaR(HCC_1000_VaR, no_vgi_df_VaR, 'OMXVGI', legend_loc='upper right',
          estimate_col='#e3545b', level=99)

plot_IVaR(HCC_1000_VaR, no_rgi_df_VaR, 'OMXRGI',
          estimate_col='#4e79a7', level=95)
plot_IVaR(HCC_1000_VaR, no_rgi_df_VaR, 'OMXRGI',
          estimate_col='#4e79a7', level=975, level_in_graph=97.5)
plot_IVaR(HCC_1000_VaR, no_rgi_df_VaR, 'OMXRGI',
          estimate_col='#4e79a7', level=99)

plot_IVaR(HCC_1000_VaR, no_tgi_df_VaR, 'OMXTGI', legend_loc='upper right',
          estimate_col='#68ab80', level=95)
plot_IVaR(HCC_1000_VaR, no_tgi_df_VaR, 'OMXTGI', legend_loc='upper right',
          estimate_col='#68ab80', level=975, level_in_graph=97.5)
plot_IVaR(HCC_1000_VaR, no_tgi_df_VaR, 'OMXTGI', legend_loc='upper right',
          estimate_col='#68ab80', level=99)

plot_IVaR(HCC_1000_VaR, no_s30gi_df_VaR, 'OMXS30GI', legend_loc='lower right',
          estimate_col='#eccc4c', level=95)
plot_IVaR(HCC_1000_VaR, no_s30gi_df_VaR, 'OMXS30GI', legend_loc='lower right',
          estimate_col='#eccc4c', level=975, level_in_graph=97.5)
plot_IVaR(HCC_1000_VaR, no_s30gi_df_VaR, 'OMXS30GI', legend_loc='lower right',
          estimate_col='#eccc4c', level=99)

plot_IVaR(HCC_1000_VaR, no_igi_df_VaR, 'OMXIGI', legend_loc='upper right',
          estimate_col='#ed8c2c', level=95)
plot_IVaR(HCC_1000_VaR, no_igi_df_VaR, 'OMXIGI', legend_loc='upper right',
          estimate_col='#ed8c2c', level=975, level_in_graph=97.5)
plot_IVaR(HCC_1000_VaR, no_igi_df_VaR, 'OMXIGI', legend_loc='upper right',
          estimate_col='#ed8c2c', level=99)

plot_IVaR(HCC_1000_VaR, no_h25gi_df_VaR, 'OMXH25GI', legend_loc='upper right',
          estimate_col='#ae7da0', level=95)
plot_IVaR(HCC_1000_VaR, no_h25gi_df_VaR, 'OMXH25GI', legend_loc='upper right',
          estimate_col='#ae7da0', level=975, level_in_graph=97.5)
plot_IVaR(HCC_1000_VaR, no_h25gi_df_VaR, 'OMXH25GI', legend_loc='upper right',
          estimate_col='#ae7da0', level=99)

plot_IVaR(HCC_1000_VaR, no_c25gi_df_VaR, 'OMXC25GI', legend_loc='lower right',
          estimate_col='#fb9ba3', level=95)
plot_IVaR(HCC_1000_VaR, no_c25gi_df_VaR, 'OMXC25GI', legend_loc='lower right',
          estimate_col='#fb9ba3', level=975, level_in_graph=97.5)
plot_IVaR(HCC_1000_VaR, no_c25gi_df_VaR, 'OMXC25GI', legend_loc='upper right',
          estimate_col='#fb9ba3', level=99)

plot_IVaR(HCC_1000_VaR, no_obx_df_VaR, 'OBX', legend_loc='upper right',
          estimate_col='#549c4c', level=95)
plot_IVaR(HCC_1000_VaR, no_obx_df_VaR, 'OBX', legend_loc='upper right',
          estimate_col='#549c4c', level=975, level_in_graph=97.5)
plot_IVaR(HCC_1000_VaR, no_obx_df_VaR, 'OBX', legend_loc='upper right',
          estimate_col='#549c4c', level=99)
```

R code snippet for the estimated HAC tree graphs and examples of bivariate copulas:

```
# The tree plots for HCC and HFC are based on the fitted copulas
# for the uniform margins of standardised AR(1)-GARCH(1, 1) residuals from
# the first 500 observations in the dataset:
colnames(unif_mat) <- c(
    "OMXVGI", "OMXRGI", "OMXTGI", "OMXS30GI",
    "OMXIGI", "OMXH25GI", "OMXC25GI", "OBX"
)
fit_HAC_Frank <- HAC::estimate.copula(unif_mat, type = 5)
plot(fit_HAC_Frank)
fit_HAC_Clayton <- HAC::estimate.copula(unif_mat, type = 3)
plot(fit_HAC_Clayton)

# Examples of bivariate copulas:
fc <- copula::frankCopula(6.06, dim = 2)
cc <- copula::claytonCopula(1.38, dim = 2)
set.seed(2024)
par(mfrow = c(1, 2))
plot(cc,
    n = 4000, main = "Bivariate Clayton copula with θ = 1.38",
    pch = 20, cex = 0.9, col = "#0B81B3"
)
plot(fc,
    n = 4000, main = "Bivariate Frank copula with θ = 6.06",
    pch = 20, cex = 0.9, col = "#18492E"
)
```