FIN 417: QUANTITATIVE RISK MANAGEMENT PROJECT I

Market Risk: VaR, ES, and Copulas

Philipp Mayer	**Rafael Barroso**	**Roberto Magno**
SCIPER: 329758	**SCIPER: 406607**	**SCIPER: 41717**
Ingenierie Mathématique	Mathématiques	Data Science

École Polytechnique Fédérale de Lausanne

November 28, 2025

Contents

Abstract

This report studies the market risk of an equally weighted portfolio of AAPL, META and JPM using Value at Risk (VaR) and Expected Shortfall (ES) at confidence levels 95% and 99%. We first document standard stylized facts of equity returns and then estimate five univariate loss models for each asset: Historical Simulation, Gaussian, Student-t, conditional AR–GARCH and Filtered Historical Simulation (FHS). These specifications are used to generate one–day–ahead portfolio VaR and ES forecasts under an equal–weight strategy. In a second step, we extend the analysis by incorporating Gaussian and Student copulas to model cross–asset dependence.

Model performance is evaluated by a rolling–window backtesting scheme. For VaR we implement the Kupiec Proportion–of–Failures test and the Christoffersen independence and conditional coverage tests; for ES we use the Acerbi–Székely Z_1 backtest with Monte Carlo p–values. Our results show that simple Gaussian (and, to a lesser extent, Gaussian–GARCH) models systematically underestimate tail risk, leading to frequent rejections, especially at the 99% level. In contrast, Historical Simulation, Student-t and FHS—as well as the copula–based extensions—provide more conservative and backtest–consistent risk measures, particularly during the volatility episode at the end of the sample. Overall, the evidence highlights the importance of heavy tails, time–varying volatility and dependence modelling in equity portfolio risk management.

1 Empirical Stylized Facts

In this section we analyze the daily *log-returns* of three large-cap U.S. stocks; AAPL, META, and JPM, over the time period from January 1, 2023 to June 30, 2025. Adjusted daily closing prices were fetched directly from Yahoo Finance via the `yfinance` Python package using the provided template.

Let P_t denote the adjusted close price on day t. We construct log-returns as

$$R_t = \log_n\left(\frac{P_t}{P_{t-1}}\right), \qquad t = 1, \ldots, T,$$

after aligning dates across assets and dropping days with missing observations.

Note that in comparison to *simple returns* (i.e. $sr_t = \frac{P_t}{P_{t-1}}$), log-returns are easier to work with since logarithms have the following additive property:

$$\log_n\left(\frac{P_t}{P_{t-1}}\right) + \log_n\left(\frac{P_{t-1}}{P_{t-2}}\right) = \log_n\left(\frac{P_t}{P_{t-1}} \cdot \frac{P_{t-1}}{P_{t-2}}\right) = \log_n\left(\frac{P_t}{P_{t-2}}\right)$$

Another useful fact to note is that log-returns are more likely to be normally distributed than simple returns; which helps for risk models (e.g., VaR, Sharpe ratio).

1.1 Return Series and Volatility Patterns

As a preliminary visualization, we provide a plot of the closing prices for each of the stocks to be analyzed as shown in Figure 2. Next, we perform a vectorized calculation using the formula provided above over the closing prices which in turn yields a time series of log-returns for each of the stocks, i.e. AAPL, META, and JPM. Once the time series is calculated, we perform a plot of each individual return series where we highlight in red the **negative** returns (i.e. those below the dashed line denoting neutral returns). Figure 1 demonstrates the behavior of the returns regarding the stocks being analyzed. Also, AAPL and META show visibly larger and more frequent deviations from the zero line compared to JPM, indicating higher volatility. All three assets—AAPL, META, and JPM—display non-zero, fluctuating returns, confirming regular variability in price levels.

As explained throughout the lectures, daily log-returns exhibit alternating positive and negative spikes, corresponding respectively to gains and losses. These spikes indicate periods of unusually large market movements, while values close to zero simply reflect small day-to-day price changes.
In financial terms, volatility measures the dispersion of returns around their mean and is formally quantified via the standard deviation

$$\sigma = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(r_i - \bar{r})^2}, \quad \bar{r} = \frac{1}{n}\sum_{i=1}^{n} r_i.$$

Under the simplifying assumption of independent returns, variances scale linearly with time, implying

$$\mathrm{Var}(r_{\text{period}}) = \text{period} \cdot \mathrm{Var}(r_{\text{daily}}), \qquad \sigma(r_{\text{period}}) = \sqrt{\text{period}} \cdot \sigma(r_{\text{daily}}).$$

Volatility over any rolling window of fixed length can similarly be obtained from

$$\sigma_t^{\text{window}} = \sqrt{\frac{1}{\text{window}} \sum_{i=t-(\text{window}-1)}^{t} (r_i - \bar{r})^2}.$$

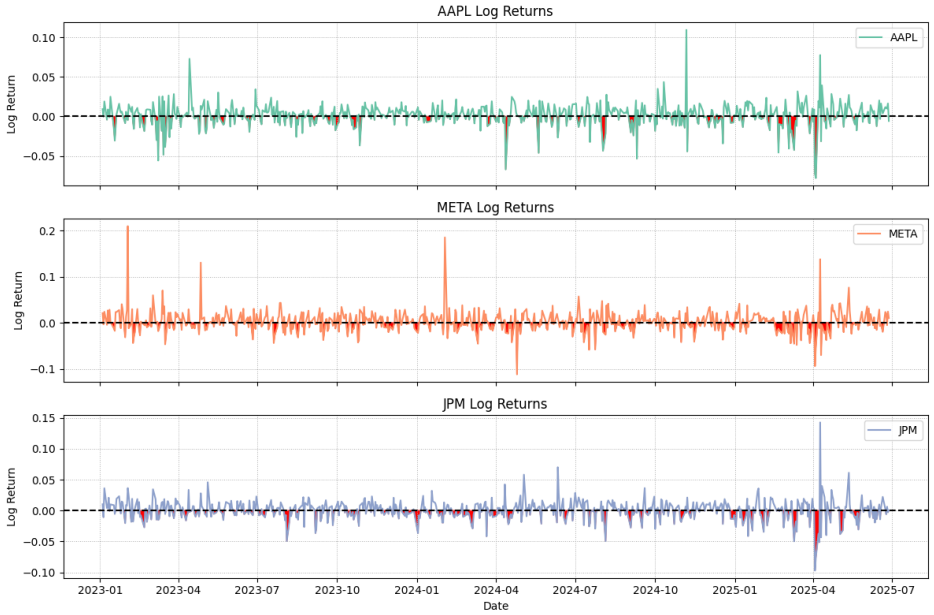

Figure 1: Logarithmic returns of AAPL, META and JPM over selected time period.

In what follows, we compute and compare both directly estimated and time-scaled volatility to evaluate the empirical adequacy of the scaling relation. Also, large shocks tend to occur in clusters: periods of elevated volatility are followed by further large fluctuations, while calmer periods persist for some time. This time-varying volatility, or *volatility clustering*, is particularly pronounced in META and AAPL, while JPM shows somewhat milder clustering consistent with the behavior of financial-sector stocks.

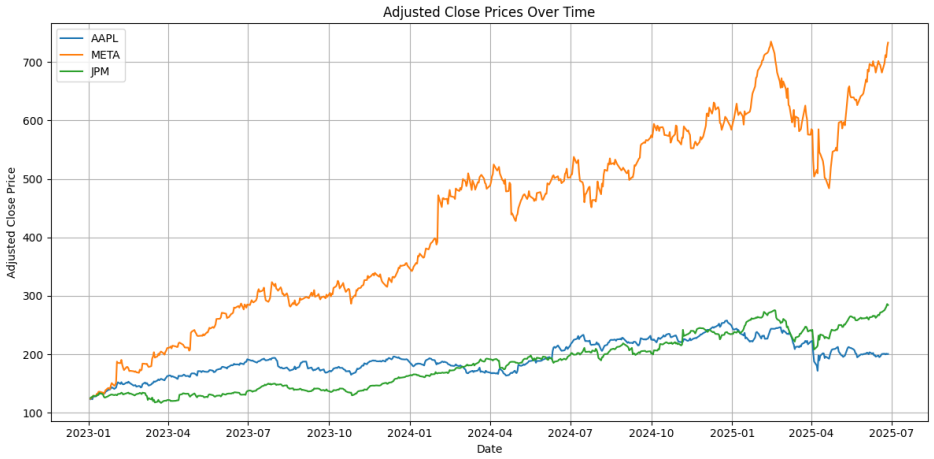

Figure 2: Daily closing prices of AAPL, META, and JPM from January 2023 to June 2025.

After computing annualized volatility and statistical metrics for the log-returns, we note the following observations (Figure 3); The dashed blue line represents the theoretical annualized volatility obtained by scaling the daily volatility with the factor $\sqrt{252}$, an approach valid only under the assumption of independent and identically distributed daily returns. The orange curve reports the empirical rolling volatility, which reflects how risk actually evolved over time.

When both measures are close, the $\sqrt{252}$ rule provides a reasonable approximation. However, noticeable deviations between the two indicate periods of volatility clustering—episodes of sustained turbulence or

calm that the simple scaling rule cannot capture. In our results, the rolling volatility exhibits pronounced spikes, while the scaled volatility remains comparatively smooth, demonstrating that market volatility is not constant over time but tends to persist once elevated.

Summary statistics further highlight cross-asset differences. META exhibits the highest risk profile, with an average volatility of approximately 35% and peak values exceeding 80%. JPM follows with an average of 23% and maximum volatility near 78%. AAPL displays the lowest risk, with typical volatility remaining below 30%. These observations confirm that META behaves as the most volatile asset, whereas AAPL and JPM show comparatively more stable dynamics.

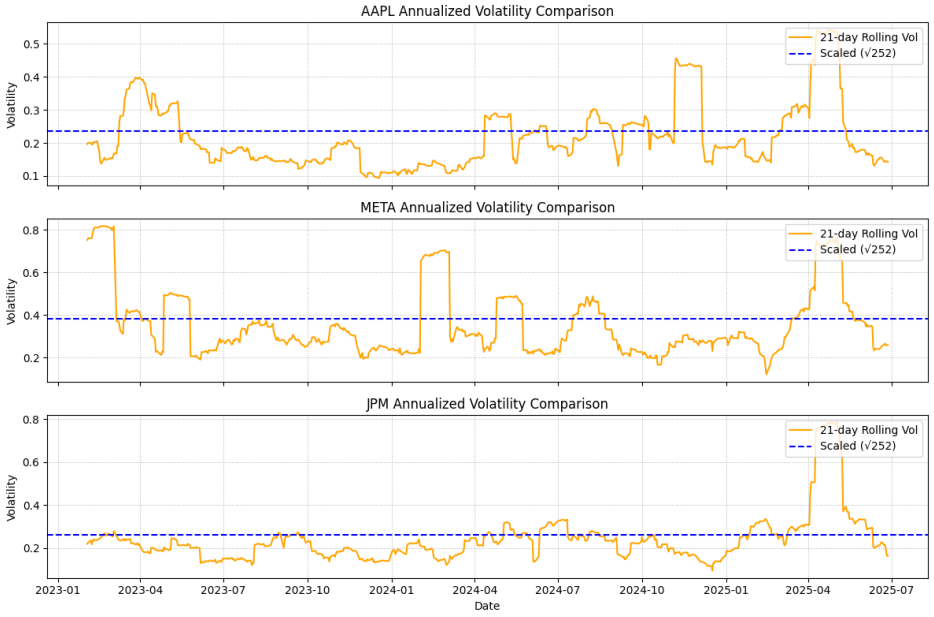

Figure 3: Annualized volatility comparison of stocks.

1.2 Autocorrelation and Cross-Correlation Structure

To analyse temporal and cross-sectional dependence in the return series, we compute both cross-correlation and auto-correlation functions for raw and absolute returns. For two assets n and m, the *cross-correlation function* (CCF) of raw returns at lag h is defined as

$$\mathrm{Corr}(R_{t,n}, R_{t-h,m}) = \frac{\mathrm{Cov}(R_{t,n}, R_{t-h,m})}{\sqrt{\mathrm{Var}(R_{t,n})\ \mathrm{Var}(R_{t-h,m})}}, \qquad h = 0, \ldots, 25.$$

This quantity measures whether returns of asset m at time $t-h$ are linearly associated with returns of asset n at time t.

- For $n = m$, this reduces to the *auto-correlation function* (ACF), which quantifies whether an asset's own past returns contain information about its future returns. In liquid and informationally efficient markets, raw-return ACFs are typically close to zero for $h \geq 1$.
- For $n \neq m$, the CCF assesses co-movement across assets and may reveal contemporaneous linkages ($h = 0$) or delayed reactions ($h > 0$) between the series.

To capture persistence in volatility rather than direction, we compute analogous correlations for absolute returns:

$$\mathrm{Corr}(|R_{t,n}|, |R_{t-h,m}|).$$

Absolute-return correlations are central to identifying second-order dependence:

- The *auto-correlation of absolute returns* detects *volatility clustering*, the empirical phenomenon in which large (or small) absolute returns tend to be followed by further periods of elevated (or subdued) volatility.
- The *cross-correlation of absolute returns* reflects potential *volatility spillovers* across assets, where periods of high volatility in one asset coincide with or precede heightened volatility in another.

In summary, raw-return correlations quantify linear co-movement and predictability in returns themselves, whereas absolute-return correlations reveal persistence and propagation of volatility across assets—both essential for understanding dependence structures relevant for risk modeling.

The ACF of raw returns (Figure 4) shows that autocorrelations decay immediately to values close to zero for all three stocks. This behavior is consistent with the widespread empirical observation that daily returns are approximately *serially uncorrelated*, supporting the common modeling assumption of zero autocorrelation in the mean dynamics.

This hints at the validity of the *efficient market hypothesis* (EMH), which in brief, states that financial markets incorporate all available information about asset price immediately. In other words, prices of stocks adjust so fast that it is "impossible" to predict future values only from historical data. Formally,

$$\mathbb{E}[R_{t,n} | R_{t-1,n}, R_{t-2,n}, \ldots] = 0,$$

which implies that the autocorrelation returns is approximately **zero** for all lags greater than zero. This is also known as the *weak-form* EMH, which is clearly supported by our raw autocorrelation plots.

In contrast, the ACF of absolute returns (Figure 5) exhibits significant positive dependence at multiple lags for all assets. This slow decay of autocorrelations is a hallmark of volatility clustering and motivates the use of conditional heteroskedasticity models such as GARCH$(1,1)$ in later sections.

The cross-correlations between raw returns show weak but non-negligible dependence, particularly between AAPL and META, reflecting common exposure to the technology sector. Cross-correlations of absolute returns are stronger, suggesting co-movement in volatility regimes across assets, a key motivation for copula modeling in Section 4.

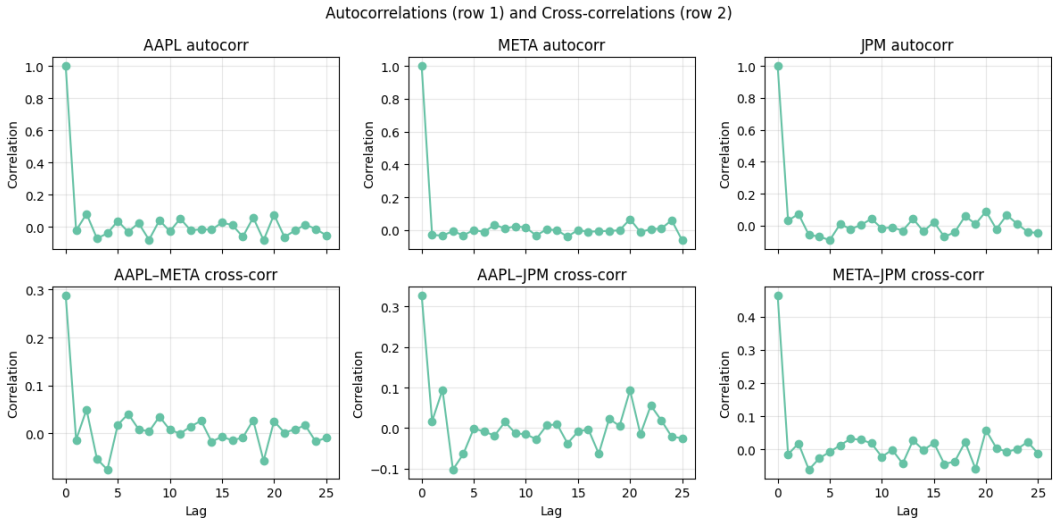

Figure 4: Autocorrelation functions of raw returns for lags $h = 0$–25.

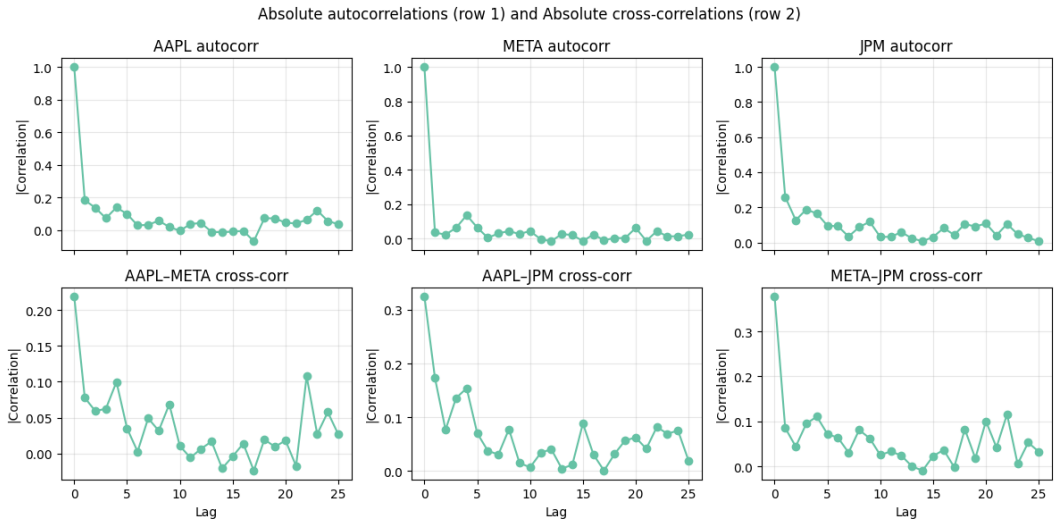

Figure 5: Autocorrelation functions of absolute returns for lags $h = 0$–25.

1.3 Normality Assessment: Q-Q Plots and Jarque–Bera Test

We now examine the similarities that the returns of each stock might share with a normal distribution by making use of *quantiles*.

It is well known that financial returns exhibit heavy tails relative to the Gaussian distribution. To illustrate this, Figure 6 presents Q-Q plots of returns against the standard normal distribution. All three assets show clear deviations from normality: the empirical quantiles diverge substantially from the normal quantiles in both tail regions, indicating excess kurtosis and occasional extreme shocks.

A quantile is a number which divides a distribution into equal parts, indicating the relative position of a data point within it e.g. the 50th quantile (median) divides the data in half.

A quantile to quantile plot (Q-Q plot) compares the quantiles of empirical data with those of a theoretical distribution (such as the normal). If the data follows the theoretical distribution, the points lie close to a 45°line. Systematic deviations from this line reveal differences in shape:

- **Curved tails** indicate heavier or lighter tails than normal.
- **Asymmetric deviations** suggest skewness (a certain asymmetry about the mean of the data).

To formally support these findings, we perform the Jarque–Bera (JB) test on each return series.

The JB test combines sample skewness and kurtosis into the statistic

$$\mathrm{JB} = \frac{T}{6}\left(S^2 + \frac{(K-3)^2}{4}\right),$$

where S is the sample skewness and K is the sample kurtosis. Under the null hypothesis of Gaussianity, $\mathrm{JB} \sim \chi^2_2$ asymptotically.

The test strongly rejects normality for all assets at the 1% significance level: p-values are effectively zero. This confirms heavy-tailed behavior and motivates the use of fat-tailed parametric models, such as the Student-t distribution, and nonparametric techniques such as historical simulation.

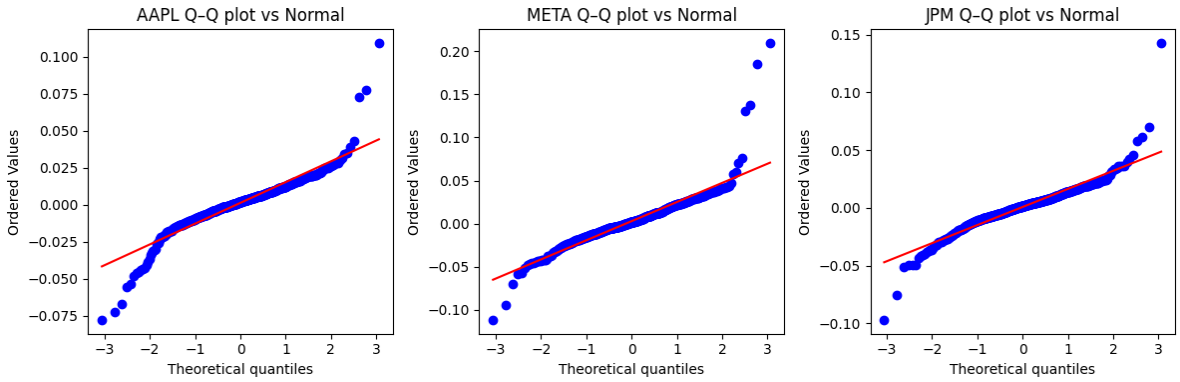

Figure 6: Q-Q plots of daily log-returns vs. the normal distribution. Heavy tails and deviations from Gaussianity are evident for all assets.

Summary. (i) Daily log-returns exhibit negligible linear autocorrelation, with ACF values fluctuating around zero beyond lag 1. This behavior is consistent with the weak-form Efficient Market Hypothesis and indicates that returns themselves possess minimal short-term linear predictability.

(ii) The *autocorrelation structure* of absolute returns displays clear persistence across multiple lags, forming slowly decaying ACF profiles indicative of volatility clustering. Episodes of heightened volatility observed in the time series plots (particularly for META and JPM) align with this finding. Such persistence is characteristic of conditional heteroskedasticity and motivates the use of GARCH-type models in Section 2 of the project, as recommended by the project guidelines

(iii) The *cross-correlation analysis* reveals non-trivial, though moderate, interdependence across assets, especially at lag 0 and the immediate succeeding lags. These results suggest that shocks to one asset can propagate rapidly to others, reflecting market-wide reactions or sectoral co-movement. Cross-asset linkages

are particularly visible in the absolute-return correlations, supporting the need for dependence models and copula-based approaches later in the project.

(iv) Graphical normality assessment via Q–Q plots and the Jarque–Bera test reveals strong deviations from Gaussianity. All assets display heavy tails, positive outliers, and asymmetry, resulting in large departures from the theoretical normal quantiles. Such non-normal behavior implies elevated tail risk and justifies the consideration of flexible fat-tailed distributions (e.g., Student-t) and non-parametric methods in subsequent VaR and ES estimation.

2 Single window modeling

In this section, we aim to understand the distributions of the losses; which in turn correspond to future negative returns, on a single window of length $W = 252$ days (approximately one trading year) separately for each asset JPM, META and AAPL at different levels $\alpha = 95\%$ and $\alpha = 99\%$ (confidence levels). Thus, we approximate these distributions through different methodologies and evaluate them through the use of VaR_α and ES_α for each distribution.

Value-at-Risk at confidence level α is the α-quantile of the loss distribution L. It represents the smallest number q_α such that losses exceed q_α with probability $1 - \alpha$:

$$\text{VaR}_\alpha(L) \;=\; \inf\{x \in \mathbb{R} : \mathbb{P}(L \le x) \ge \alpha\}.$$

Intuitively, VaR_α is the loss level that will not be exceeded on $(100\alpha)\%$ of days.

Expected Shortfall at confidence level α measures the *expected loss in the tail* beyond the VaR threshold. It is defined as the conditional expectation of losses that exceed VaR_α:

$$\text{ES}_\alpha(L) \;=\; \mathbb{E}[\, L \,|\, L > \text{VaR}_\alpha(L)]\,.$$

While VaR gives a quantile, ES captures the *average severity* of extreme losses and is therefore a coherent and more informative tail-risk measure.

The first model we use in order to estimate the loss distribution is called *Historical Simulation*, which may be utilized as follows.

2.1 Models

- **Historical Simulation**: The historical simulation procedure directly computes the value at risk at a given level α as the empirical α-quantile of the empirical (observed) distribution of the losses.

  The expected shortfall is defined as the conditional expectation of the losses, given that the value at risk exceeds the level α.

  Given the losses $L_1, ..., L_W$, we let

  $$\widehat{F}_L(t) = \frac{1}{W}\sum_{i=1}^{W} \mathbb{1}_{L_i \le t}$$

  and compute

  $$\widehat{\text{VaR}_\alpha} = \inf\{t : \widehat{F}_L(t) \ge \alpha\} = q_\alpha(\widehat{F}_L) = L_{(\lceil \alpha W \rceil)}, \text{ and}$$

  $$\widehat{\text{ES}_\alpha} = \frac{1}{(1-\alpha)W}\sum_{i=1}^{W} L_i \mathbb{1}_{L_i \ge \widehat{\text{VaR}_\alpha}}$$

This approach presents a major flaw; past anomalies will propagate. In fact, if we assume for instance that the first window was subject to significant market shocks that strongly impacted the distribution of the losses. Then, the distribution of the losses will present heavier tails, whereas the future market may not posses such behavior. Thus, we may state that this method does not provide a robust fit to capture enough of the market's nature.

In order to try and counter this weak fit, we utilize the second methodology proposed; the *Gaussian* model.

- **Gaussian distribution fit**:

  Given the losses $L_1, ..., L_W$, we fit a Gaussian distribution to the losses. That is, we assume that the the losses follow $L \sim \mathcal{N}(\mu, \sigma^2)$ and use the closed-form expressions for the VaR_α and ES_α of a $\mathcal{N}(\mu, \sigma^2)$ distribution as estimates.

  Having estimated μ and σ^2, the closed formulas for the value at risk and the expected shortfall are given by:

$$\widehat{\text{VaR}_\alpha} = \widehat{\mu} + \widehat{\sigma} z_\alpha, \text{ and}$$
$$\widehat{\text{ES}_\alpha} = \widehat{\mu} + \widehat{\sigma} \frac{\varphi(z_\alpha)}{1-\alpha}$$

  where z_α is the α-quantile of the $\mathcal{N}(0,1)$ distribution and φ is the PDF of $\mathcal{N}(0,1)$.

The Gaussian model assumes that losses over the estimation window are i.i.d. and follow a normal distribution $L \sim \mathcal{N}(\mu, \sigma^2)$, where (μ, σ) are estimated from the sample mean and variance. This specification provides a smooth parametric approximation of the loss distribution and reduces sensitivity to outliers. Its main limitation is the normality assumption; financial returns typically exhibit excess kurtosis and tail heaviness that a Gaussian model cannot capture, which may lead to systematic underestimation of extreme losses.

If we wish to grasp the 'fat-tailed' behavior of the market, i.e. the extreme cases happening, we implement the the *Student-t* fit. This method should be 'better' suited for such cases, it is mainly described as follows:

- **Student-t distribution fit**:

  Given the losses $L_1, ..., L_W$, we fit a Student t distribution to the losses. That is, we assume that the the losses follow $L \sim t_\nu(\mu, \sigma^2)$ and use the closed-form expressions for the VaR_α and ES_α of a $t_\nu(\mu, \sigma^2)$ distribution as estimates.

  Having estimated ν, μ and σ^2, the closed formulas for the value at risk and the expected shortfall are given by:

$$\widehat{\text{VaR}_\alpha} = \widehat{\mu} + \widehat{\sigma}\, t_{\widehat{\nu}}^{-1}(\alpha), \text{ and}$$
$$\widehat{\text{ES}_\alpha} = \widehat{\mu} + \widehat{\sigma}\, \frac{\widehat{\nu} + \left(t_{\widehat{\nu}}^{-1}(\alpha)\right)^2}{\widehat{\nu} - 1} \frac{f_{\widehat{\nu}}\left(t_{\widehat{\nu}}^{-1}(\alpha)\right)}{1-\alpha}$$

  where $t_{\widehat{\nu}}^{-1}$ is the α-quantile $t_{\widehat{\nu}}(0,1)$ distribution and $f_{\widehat{\nu}}$ is the PDF of $t_{\widehat{\nu}}(0,1)$.

Similarly to the Gaussian fit, this method presents the same inconvenience such as distribution assumption and the assumption that the losses are *i.i.d.*. Note that as $\nu \to \infty$, the $t_\nu(0,1)$ distribution approaches that of a $\mathcal{N}(0,1)$, whereas whenever ν gets closer to 1, the $t_\nu(0,1)$ becomes heavier tailed in comparison to the $\mathcal{N}(0,1)$ distribution. This is an advantage as, as previously mentioned, market losses often present heavier tails in comparison to a Gaussian assumption.

In the single–window setting, a "one-step-ahead forecast" refers to the model's implied distribution for the next-day loss L_{W+1} based on information from the estimation window $\{L_1, \ldots, L_W\}$. Since no dynamic updating occurs at this stage, the entire predictive distribution is taken to be identical to the distribution estimated from the past window. In other words, the model does not attempt to forecast time-varying parameters but instead treats the fitted window-distribution as the best available proxy for the next-day distribution. Thus, the "forecast" is simply the past estimated distribution itself, evaluated as if it were the conditional law of L_{W+1}.

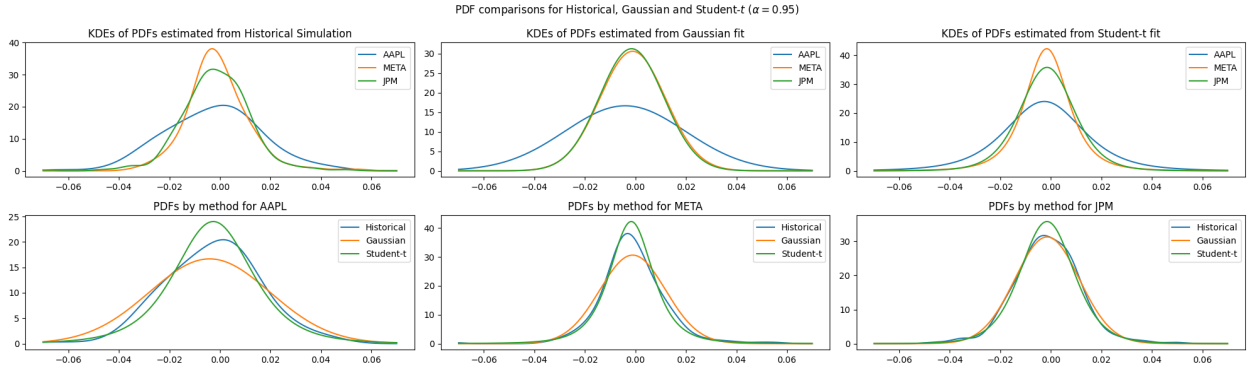

Figure 7: Loss distribution approximations of losses using Historical Simulation, Gaussian and Student-t approaches at 95% confidence level.

Comparison of loss distributions across methods. Figures 7 and 8 compare the single–window loss distributions obtained from Historical Simulation, the Gaussian fit, and the Student-t fit for AAPL, META, and JPM. The top row of each figure shows KDEs (Kernel Density Estimations, i.e. estimation of the data set's underlying probability density function) of the model-implied densities for each asset separately, while the bottom row overlays the three methods asset by asset. Across all assets, the fitted Gaussian and Student-t distributions are centered close to zero and provide smooth parametric approximations of the empirical (historical) densities. However, the Gaussian fit systematically exhibits a lower peak and thinner tails than both the historical and Student-t densities, indicating that it spreads probability mass too evenly across the center and underweights extreme outcomes. In contrast, the Student-t fit is more sharply peaked and has visibly heavier tails, closely tracking the empirical density in both the body and the tails of the distribution.

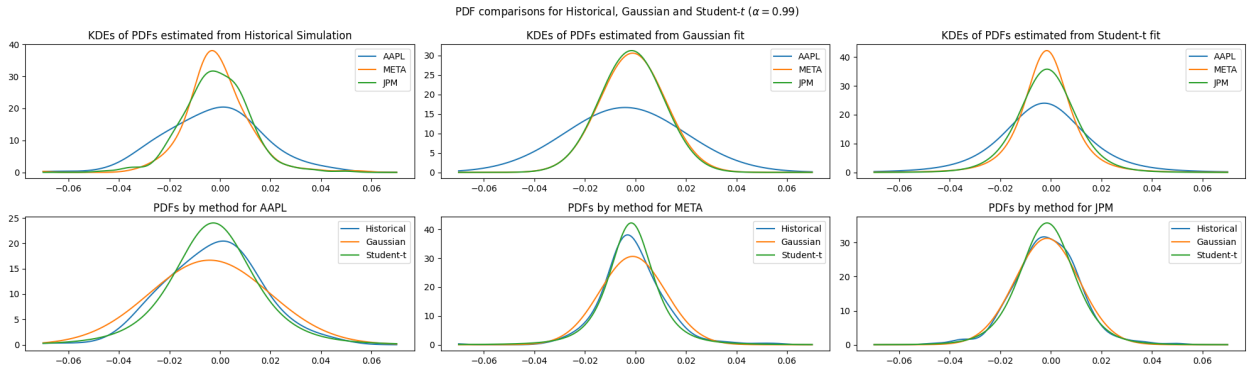

Figure 8: Loss distribution approximations of losses using Historical Simulation, Gaussian and Student-t approaches at 99% confidence level.

Asset-specific patterns. For AAPL, the three densities are relatively close, but the historical and Student-t curves display slightly fatter tails than the Gaussian, especially on the loss side. This suggests that, even for a comparatively liquid and well–diversified large-cap stock, the Gaussian assumption may underestimate tail risk. For META, which shows the highest volatility, the discrepancy is more pronounced: the Gaussian fit is clearly too light–tailed relative to the empirical density, while the Student-t distribution captures both the sharp central peak and the elevated probability of large losses. For JPM, the three densities are again broadly aligned around the center, but the Student-t fit most closely matches the historical shape in the tails, whereas the Gaussian density decays too fast.

Implications for VaR and ES at 95% and 99%. The underlying fitted distributions do not depend on the confidence level α, the figures labeled for $\alpha = 0.95$ and $\alpha = 0.99$.

As previously mentioned, both Gaussian and Student-t fit approaches present the inconvenience of assuming that the losses are *i.i.d.*. In order to prevent this, we present the following conditional parametric approach which proposes an alternative.

- **Conditional Parametric**: Consider losses $L_1, ..., L_W$. As mentionned above, these will not be assumed *i.i.d.* anymore and will be treated as a time series.

  The mean μ is assumed to follow an autoregressive AR(p) process (the order p will be chosen with help of the ACF plots of part 1. The volatility σ will be estimated based on a GARCH$(1,1)$ model. Note that the residuals are deviations from the mean, which is estimated following an AR(p) model. We proceed as follows:

  - **Fit the** AR(p) **model**: L_t is assumed to be expressed as

    $$L_t = \mu_t + \epsilon_t = \phi_0 + \sum_{j=1}^{p} \phi_j L_{t-j} + \epsilon_t$$

    This allows one to obtain the residuals as $\epsilon_t = L_t - \mu_t$.
  - **Fit the** GARCH$(1,1)$ **model**: We assume that the residuals ϵ_t follow a GARCH$(1,1)$ model, that is $\epsilon_t = \sigma_t Z_t$, where $Z_t \sim \mathcal{N}(0,1)$ are *i.i.d.* and σ_t is assumed to be described as

    $$\sigma_t^2 = \psi_0 + \psi_1 \epsilon_{t-1}^2 + \psi_2 \sigma_{t-1}^2$$

  Under model assumptions, we are able to propose the following value at risk and expected shortfall forecasts:

  $$\widehat{\mathrm{VaR}}_{t+1,\alpha} = \widehat{\mu}_{t+1} + \widehat{\sigma}_{t+1} z_\alpha, \text{ and}$$

  $$\widehat{\mathrm{ES}}_{t+1,\alpha} = \widehat{\mu}_{t+1} + \widehat{\sigma}_{t+1} \frac{\varphi(z_\alpha)}{\alpha}$$

  where ϕ and z_α have been defined previously.

Remark: Following the suggestions, M is chosen as 1000. For p, we observe using Part 1 it is reasonable to pick $p = 7, 7, 5$ for AAPL, META and JPM respectively. This will always be the default values used for the AR(p)+GARCH(1,1), which we will implicitly denote by AR+GARCH.

As already mentioned, this model is more robust and takes the past (with model assumptions) to produce a forecast of the future values of VaR_α and ES_α. This suggests an alternative to the *i.i.d.* assumption, with the constraint of still being a parametric model.

Finally, we propose a non-parametric approach to VaR_α and ES_α forecasting, that does not assume *i.i.d.* losses.

- **Filtered Historical Simulation (FHS)**: The FHS method models residuals as AR(p) + GARCH(1, 1) and uses a non-parametric Bootstrap method to obtain standardized residuals forecasts. More precisely,
  - **Fit** AR(p) + GARCH(1, 1) **model to losses**:
    We proceed similarly to the conditional parametric approach to obtain a forecast $\widehat{\mu}_{t+1}$ and $\widehat{\sigma}_{t+1}$ and we keep the residuals $\widehat{\epsilon}_t = L_t - \widehat{\mu}_t$.
  - **Compute standardized residuals**: Given the residuals $\widehat{\epsilon}_t$, we compute $\widetilde{\epsilon}_t = \frac{\widehat{\epsilon}_t}{\widehat{\sigma}_t}$.
  - **Bootstrap for the innovations**: We have computed $\widehat{\epsilon}_1, ..., \widehat{\epsilon}_W$ above and now obtain innovations as
    $$\widetilde{\epsilon}_{t+1}^{(m)} \sim \text{Bootstrap}\{\widetilde{\epsilon}_1, ..., \widetilde{\epsilon}_W\} \quad \text{for} \quad m = 1, ..., M$$
  - **Compute forecasts for the loss**: Recall that with the AR(p) + GARCH(1, 1) assumptions, the losses are modeled as
    $$L_t = \mu_t + \epsilon_t \quad \text{where} \quad \epsilon_t = \sigma_t Z_t \quad \text{for} \quad Z_t \sim \mathcal{N}(0, 1)$$
    Given the bootstrapped, we obtain 1-step ahead forecasts as
    $$\widehat{L}_{t+1}^{(m)} = \widehat{\mu}_{t+1} + \widehat{\sigma}_{t+1}\widetilde{\epsilon}_{t+1}^{(m)} \quad \text{for} \quad m = 1, ..., M$$

  Finally, the value at risk and expected shortfall forecasts are obtained as
  - $\widehat{\text{VaR}}_{t+1,\alpha}$ = Empirical α-quantile of $\widehat{L}_{t+1}^{(1)}, ..., \widehat{L}_{t+1}^{(M)}$.
  - $\widehat{\text{ES}}_{t+1,\alpha}$ = Mean of the $\widehat{L}_{t+1}^{(m)}$ that exceed $\widehat{\text{VaR}}_{t+1,\alpha}$.

Remark: p will always be chosen as for the respective AR+GARCH model, that is 7,7,5 for AAPL, META and JPM respectively as mentioned previously

This approach takes inspiration from the modeling approach of he conditional parametric method presented below, with its advantages (notably the forecasting feature coming from the model assumptions) but adds the non-parametric bootstrap approach to capture additional feature of the past losses/returns. This method appears strongly robust in comparison to the other featured forecasting methods.

Comparison of AR–GARCH and FHS loss distributions. Figures 9, and 10 illustrate how the AR–GARCH model and the Filtered Historical Simulation (FHS) approach approximate the one-step-ahead loss distribution for AAPL, META, and JPM. The AR–GARCH model yields smooth Gaussian densities centered around the conditional mean and scaled by the forecasted conditional volatility. In contrast, FHS produces densities obtained from bootstrapped standardized residuals, allowing the simulated distribution to inherit the empirical skewness, kurtosis, and tail behavior observed in the estimation window.

Across all assets, the two methods agree closely near the center of the distribution, where most probability mass lies. Differences become more visible in the tails: FHS exhibits sharper peaks and slightly heavier tails, reflecting the non-Gaussian features present in the data, while AR–GARCH maintains the symmetric, lighter-tailed shape implied by Gaussian innovations. These deviations are more pronounced for the more volatile stocks, particularly META. Overall, the comparison highlights that AR–GARCH captures conditional volatility dynamics well, but FHS provides additional flexibility in modeling tail risk by preserving empirical residual structure.

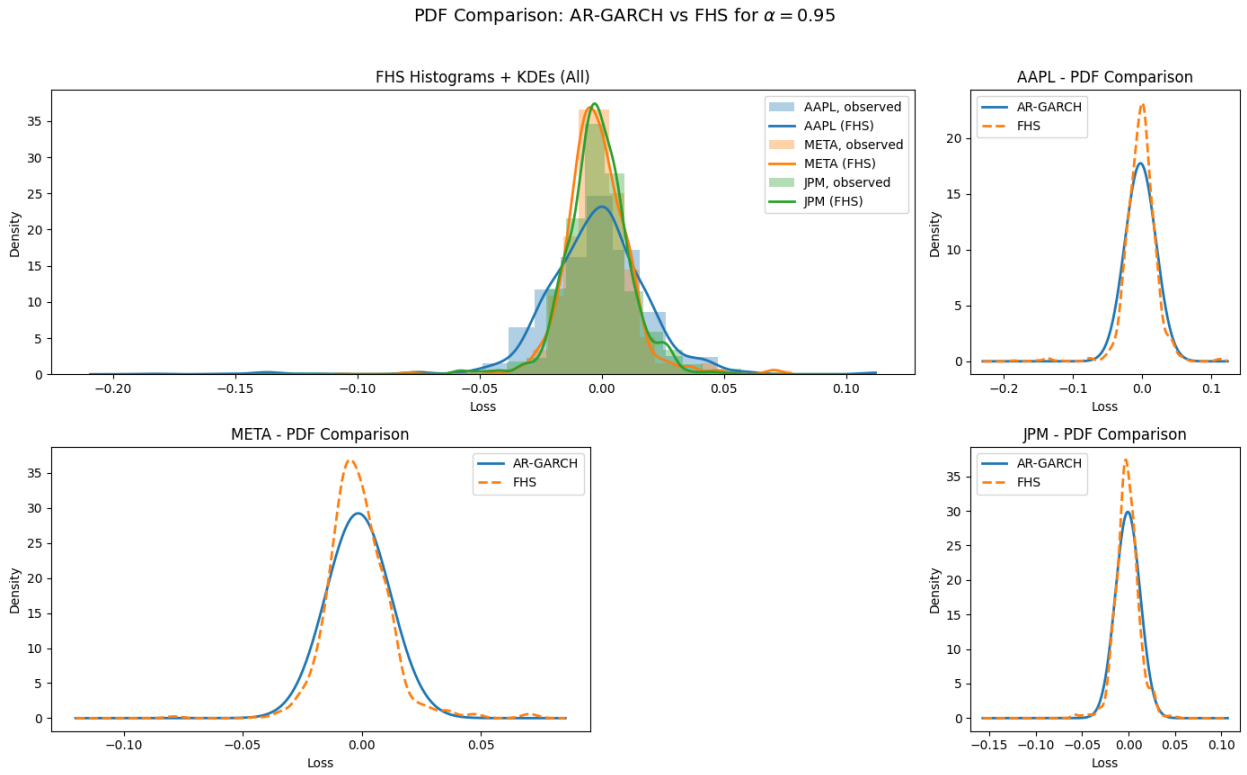

Figure 9: Loss distribution approximations of losses using FHS and AR-GARCH approaches at 95% confidence level.

3 Backtesting VaR and ES: Rolling–Window Analysis

In this section, we evaluate the out–of–sample forecasting performance of the five risk–measurement procedures introduced in Section 2:

- Historical Simulation (HS),
- Gaussian parametric fit,
- Student–t parametric fit,
- Conditional Gaussian AR–GARCH,
- Filtered Historical Simulation (FHS).

For each individual asset (`AAPL`, `META`, `JPM`), we compute rolling one–day–ahead forecasts of $\mathrm{VaR}_{\alpha,t+1}$ and $\mathrm{ES}_{\alpha,t+1}$ at the confidence levels $\alpha \in \{95\%, 99\%\}$ using a rolling window of $W = 252$ daily observations (approximately one trading year). Throughout, losses are defined as $L_t = -R_t$, so that large positive values correspond to large negative returns.

The objective is to assess whether each model is capable of providing statistically adequate risk forecasts when compared to the realized losses. This is performed using formal backtesting methods: the Kupiec (1995) Proportion–of–Failures test, the Christoffersen (1998) independence and conditional coverage tests, and the ES backtest of Acerbi and Székely (2014).

3.1 Rolling–Window Forecasting Methodology

For each asset, we perform the following steps to generate the out-of-sample forecasts.

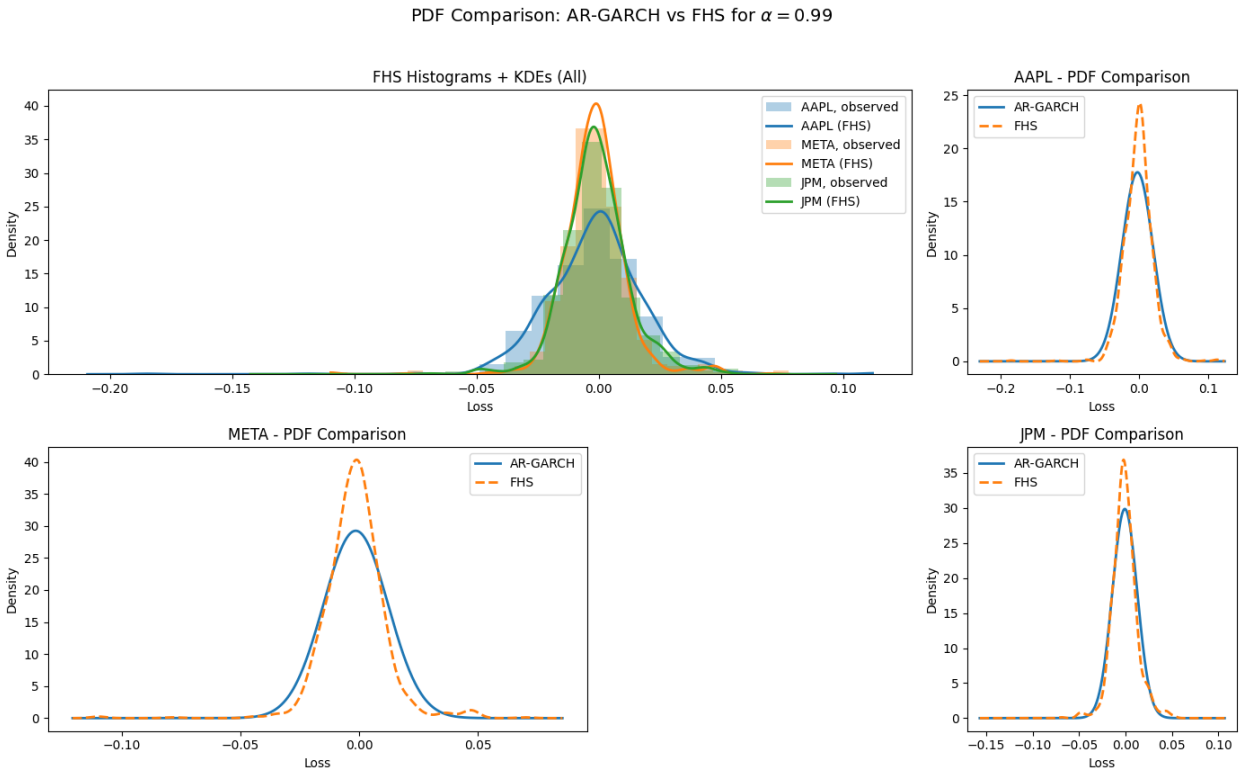

Figure 10: Loss distribution approximations of losses using FHS and AR-GARCH approaches at 99% confidence level.

Let the full sample of losses be $\{L_1, \ldots, L_T\}$. For each time point $t = W, W+1, \ldots, T-1$:

1. **Estimation window:** Collect the most recent W losses:
$$\{L_{t-W+1}, \ldots, L_t\}.$$
2. **Fit each model** on this window:
   - **HS:** Compute the empirical cdf $\widehat{F}_L$;
   - **Gaussian:** Estimate $(\widehat{\mu}_t, \widehat{\sigma}_t)$ via MLE;
   - **Student–t:** Estimate $(\widehat{\mu}_t, \widehat{\sigma}_t, \widehat{\nu}_t)$ via MLE;
   - **AR–GARCH(1,1):** Fit the volatility model to obtain $(\widehat{\mu}_{t+1}, \widehat{\sigma}_{t+1})$;
   - **FHS:** Filter returns by GARCH volatility, compute standardized residuals, and use bootstrap innovation for the forecast.
3. **Compute 1–step–ahead forecasts:**
$$\mathrm{VaR}^{(m)}_{\alpha,t+1}, \qquad \mathrm{ES}^{(m)}_{\alpha,t+1}, \qquad m \in \{\mathrm{HS}, \mathrm{Gauss}, t-\mathrm{Student}, \mathrm{GARCH}, \mathrm{FHS}\}.$$
4. **Advance the window** to $t+1$ and repeat.

This procedure generates time series $\{\mathrm{VaR}^{(m)}_{\alpha,t+1}\}$ and $\{\mathrm{ES}^{(m)}_{\alpha,t+1}\}$ aligned with the realized losses $\{L_{t+1}\}$.

3.2 VaR Backtesting Framework

To formally evaluate the adequacy of the VaR forecasts, we define the "hit" or exception sequence indicator I_t as:

$$I_t = \mathbb{1}\{L_t > \text{VaR}_{\alpha,t}\}.$$

Let $N = \sum_{t=1}^{T^*} I_t$ denote the total number of VaR violations observed over the out-of-sample period T^*. We employ the three standard tests outlined below.

3.2.1 Kupiec Proportion-Of-Failures (POF) Test

The Kupiec test checks the **unconditional coverage** property: essentially, does the model fail as often as expected?

Objective: Test whether the observed exception rate $\hat{\pi} = N/T^*$ matches the theoretical confidence level $1 - \alpha$.
Hypotheses:

$$H_0 : \pi = 1 - \alpha \quad \text{(Correct Coverage)} \qquad \text{vs} \qquad H_1 : \pi \neq 1 - \alpha$$

Test Statistic: The test is a Likelihood Ratio (LR) test based on the Binomial distribution. The statistic is defined as:

$$\text{LR}_{\text{POF}} = -2 \ln \left(\frac{L(1-\alpha)}{L(\hat{\pi})} \right)$$

where the likelihood function is $L(p) = (1-p)^{T^*-N} p^N$. Explicitly:

$$\text{LR}_{\text{POF}} = -2 \left[N \ln(1-\alpha) + (T^* - N) \ln(\alpha) - N \ln(\hat{\pi}) - (T^* - N) \ln(1-\hat{\pi}) \right]$$

Under H_0, $\text{LR}_{\text{POF}} \sim \chi^2(1)$ asymptotically.
Decision Rule (at 5% level):

- If p-value < 0.05, reject H_0.
- **Interpretation:** If $\hat{\pi} \gg 1 - \alpha$, the model *underestimates* risk (too many exceptions). If $\hat{\pi} \ll 1 - \alpha$, the model is too conservative.

3.2.2 Christoffersen (1998) Independence Test

A model might satisfy unconditional coverage (e.g., 5% failures on average) but fail to capture volatility clustering (e.g., all failures happen in a single week). The Independence test evaluates the **timing** of exceptions.

Objective: Test whether a VaR exception today increases the probability of an exception tomorrow (clustering).
Transition Counts: Let n_{ij} be the number of days where state i is followed by state j (where $0 =$ no violation, $1 =$ violation).

$$n_{00}, n_{01}, n_{10}, n_{11}$$

We estimate the conditional probabilities:

$$\hat{\pi}_{01} = \frac{n_{01}}{n_{00} + n_{01}} \quad \text{and} \quad \hat{\pi}_{11} = \frac{n_{11}}{n_{10} + n_{11}}$$

Test Statistic:

$$\text{LR}_{\text{ind}} = -2 \ln \left(\frac{L(\hat{\pi})}{L(\hat{\pi}_{01}, \hat{\pi}_{11})} \right) \sim \chi^2(1)$$

where the numerator assumes independence ($\pi_{01} = \pi_{11} = \pi$) and the denominator allows for Markov dependence.
Interpretation: If $\hat{\pi}_{11} > \hat{\pi}_{01}$, exceptions appear in clusters. A rejection ($p < 0.05$) implies the model has failed to filter the volatility dynamics (common in static Historical/Gaussian models).

3.2.3 Conditional Coverage Test

The Christoffersen Conditional Coverage (CC) test is a joint test of both properties:

$$\text{LR}_{\text{CC}} = \text{LR}_{\text{POF}} + \text{LR}_{\text{ind}} \sim \chi^2(2)$$

A model is only considered adequate if it passes this joint test (i.e., correct frequency *and* no clustering).

3.3 ES Backtesting Framework: Acerbi and Székely

Since Expected Shortfall is a conditional expectation rather than a quantile, it cannot be backtested using simple "hit" ratios. We utilize the Z_1 test, which checks if the forecasted ES correctly estimates the magnitude of tail losses.

Objective: Recall that the expected shortfall is given by the conditional expectation $\mathrm{ES}_{\alpha,t} = \mathbb{E}\left[L_t \mid L_t > \mathrm{VaR}_{\alpha,t}\right]$. If both $\mathrm{VaR}_{\alpha,t}$ and $\mathrm{ES}_{\alpha,t}$ are correctly specified, then the null hypothesis asks weather

$$\mathbb{E}\left[\frac{L_t}{\widehat{\mathrm{ES}}_{\alpha,t}} - 1 \middle| L_t > \widehat{\mathrm{VaR}_{\alpha,t}}\right] = 0$$

holds.
The Acerby test proposes the following Z_1 statistic. Given $N = \sum_{t=1}^{T} \mathbb{1}\left(L_t > \widehat{\mathrm{VaR}_{\alpha,t}}\right)$

$$Z_1 = \frac{1}{N}\sum_{t=1}^{T} \frac{L_t \mathbb{1}\left(L_t > \widehat{\mathrm{VaR}_{\alpha,t}}\right)}{\widehat{\mathrm{ES}_{\alpha,t}}}$$

Unlike in the VaR cases, the test statistic's distribution can not be expressed in closed form. Acerby suggests a simulation approach. The implementation we have done follows the method suggested in the Appendix. The exact implementation is roughly described in Figure 11
Implementation Details:

- As mentioned, we follow the standard procedure to simulating Z_1 and obtaining p-values. The strategy is to generate, given a model and α, $M = 1000$ future (simulated) one step-ahead forecasts of the losses $L_t^{(1)}, ..., L_t^{(M)}$. This allows us build instances $Z_1^{(1)}, ..., Z_1^{(M)}$. $Z_1^{(m)}$ approximate the distribution of Z_1 under H_0. We perform the one step-ahead forecasts at time t based on a rolling window scheme and compute the one-sided p-value

$$p = \frac{1}{M}\sum_{m=1}^{M} \mathbb{1}\left(Z_1^{(m)} \geq Z_1^{\mathrm{obs}}\right),$$

- **Decision:** Reject H_0 if p-value < 0.05. This indicates the model systematically underestimates tail risk.

  Note: Simply discarding the values of N that are equal to zero is an erroneous strategy because of the fact that the equation yields unbalanced results. Thus, we subtract those instances to the value of M in order to keep the theory sane.

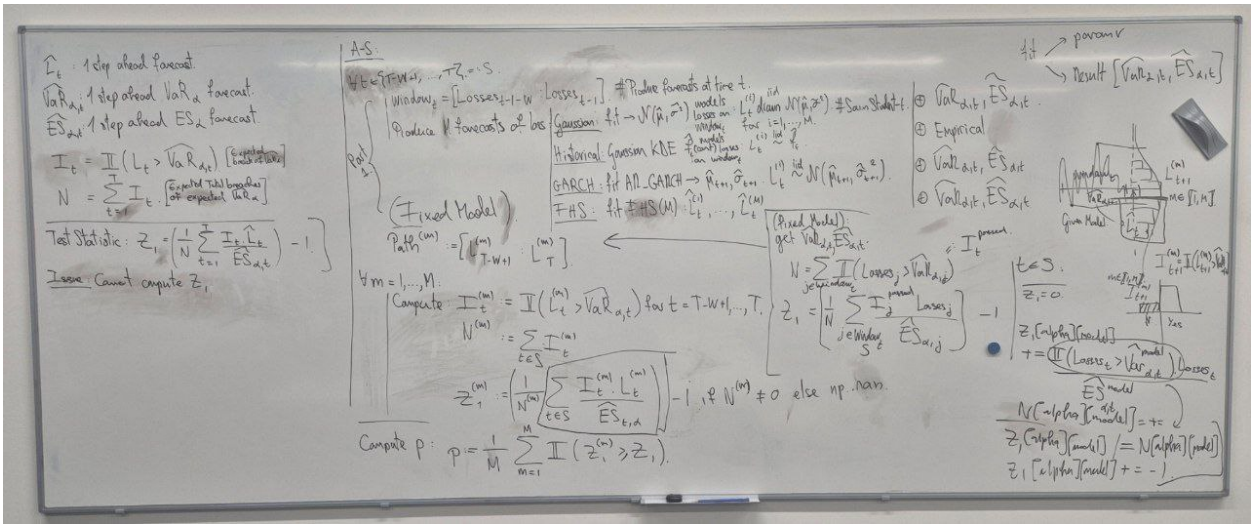

Figure 11: Implementation flow of Acerby test on whiteboard

3.4 Backtesting the Value at Risk

3.4.1 Visual Inspection of VaR Forecasts

We begin with observing the path of the one-step-ahead forecasts of the value at risk.

For each stock (here AAPL, JPM and META), the models do not capture sudden unexpected market shocks. More precisely, sudden loss spikes in low volatility regions almost always lead to a VaR breach. Furthermore, we observe that the increased volatility period between March 2025 and April 2025 lead to a strong reaction of the AR+GARCH models and the FHS models. On the other hand, though reactions can be observed for the other models, they do not capture the sudden change in volatility as well. Note however that except for AAPL, no breaches occurred following the increased volatility period around April 2025 as the losses where negative (i.e profits were made).

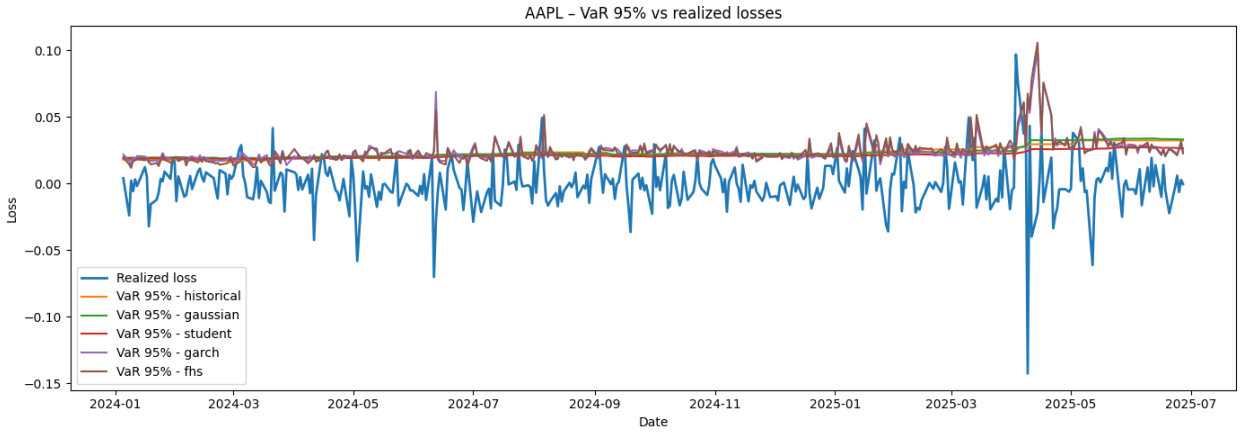

Figure 12: AAPL VaR forecasts at $\alpha = 0.95$

In more details, we see that all models have had troubles to react at the loss spike of June 2025, following the increased volatility cluster. The models are similar in stable periods (f.ex. September 2024 to January 2025).

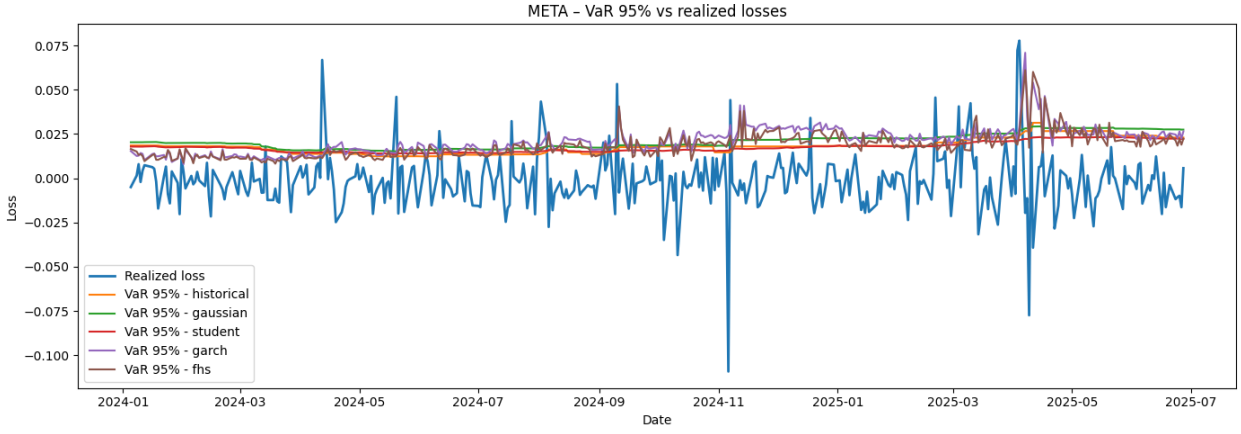

Figure 13: META VaR forecasts at $\alpha = 0.95$

META is presents more loss spikes outside high volatility clusters (interpreted as unexpected market shocks) in comparison to AAPL. This is in line with the observations in the first art where we established that META was the more volatile of the three stocks over this period.

Our observations translate to the JPM stock - except for specific-stock related observations such as market shocks and breaches.

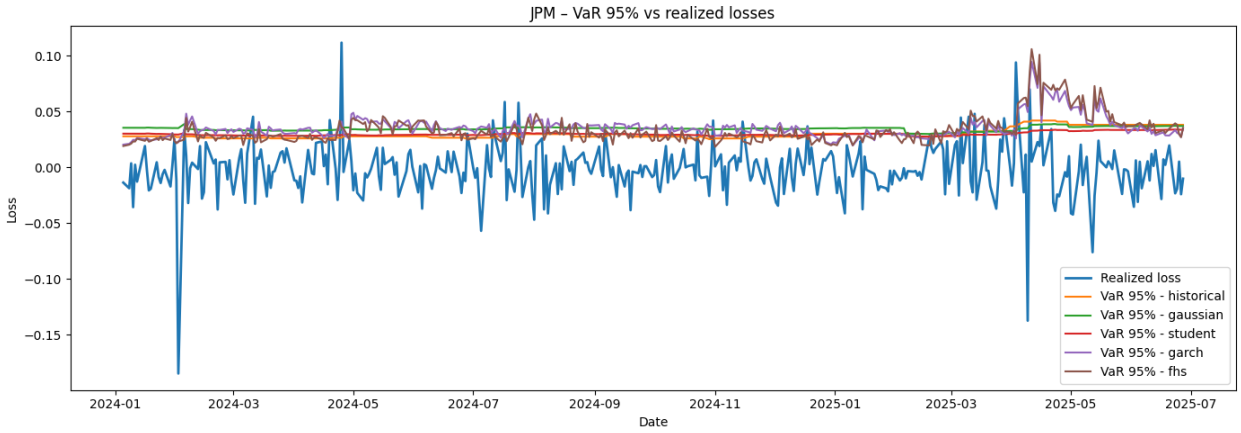

Figure 14: JPM VaR forecasts at $\alpha = 0.95$

Our first observations highlight the following points:

- As expected, no model is able to react to sudden (unexpected) market shocks fast enough. This can lead to dangerous capital exposure to the market. The expected quantitative shortfall backtest will confirm this intuition for a number of Value at Risk models.
- Dynamic time series-based models such as AR+GARCH and FHS react better to high volatility clusters, but with (expected) delay. This is a desirable property, but potential gains may have been compromised around June 2025.

3.4.2 Quantitative Results

Recall the various tests (POF, Christofferesen and CC tests) presented previously.

The backtest yields the following results presented in Tables 1, 2, 3 for $\alpha = 0.95$ and Tables 4, 5, 6 for $\alpha = 0.99$.

We base the rejections on the Conditional Coverage test (CC) and deem a model Accepted if $p_{\text{model, CC}} \geq 5\% = 0.05$ and Rejected otherwise (see previous discussions and explanations).

We observe the following:

- Unanimously the models are rejected for $\alpha = 0.95$ for AAPL. We observe that the models fail the POF test but succeed on the CI test, that is, informally, that the frequency is not matched but clustering is informative.
- Unanimously the models are Accepted for $\alpha = 0.95$ for META.
- Unanimously the models are Rejected for $\alpha = 0.95$ for JPM.
- At $\alpha = 0.99$, the results are not uniform. Historical and FHS appear to be consistently Accepted, while Student-t is only Rejected for META (which is the more volatile stock in the portfolio).

3.5 Backtesting the expected shortfall

3.5.1 Quantitative Results

For the Expected Shortfall backtest, form Figure 15 we observe that at $\alpha = 0.95$, the following models are uniformly rejected accross all stocks:

- AR+GARCH
- Gaussian

Table 1: AAPL VaR backtest for $\alpha = 0.95$

AAPL VaR backtest for $\alpha = 0.95$					
Model	Exceptions	p_{POF}	p_{CI}	p_{CC}	Test status
Historical	33	0.00174	0.0789	0.00159	Rejected
Gaussian	31	0.00632	0.141	0.00814	Rejected
Student	40	0.00001	0.0694	0.00001	Rejected
AR+GARCH	31	0.00632	0.793	0.02325	Rejected
FHS	31	0.00632	0.793	0.02325	Rejected

Table 2: META VaR backtest for $\alpha = 0.95$

META VaR backtest for $\alpha = 0.95$					
Model	Exceptions	p_{POF}	p_{CI}	p_{CC}	Test status
Historical	23	0.30023	0.05335	0.09044	Accepted
Gaussian	19	0.90546	0.07529	0.20415	Accepted
Student	23	0.30023	0.05335	0.09044	Accepted
AR+GARCH	21	0.55902	0.13260	0.27210	Accepted
FHS	26	0.09091	0.03187	0.02395	Accepted

Table 3: JPM VaR backtest for $\alpha = 0.95$

JPM VaR backtest for $\alpha = 0.95$					
Model	Exceptions	p_{POF}	p_{CI}	p_{CC}	Test status
Historical	33	0.00174	0.07887	0.00158	Rejected
Gaussian	31	0.00632	0.14110	0.00814	Rejected
Student	40	0.00000	0.06939	0.00000	Rejected
AR+GARCH	30	0.01150	0.70420	0.03819	Rejected
FHS	33	0.00174	0.52059	0.00604	Rejected

Table 4: AAPL VaR backtest for $\alpha = 0.99$

META VaR backtest for $\alpha = 0.99$					
Model	Exceptions	p_{POF}	p_{CI}	p_{CC}	Test status
Historical	6	0.27022	0.65605	0.49313	Accepted
Gaussian	9	0.01926	0.20357	0.02881	Rejected
Student	5	0.51910	0.71091	0.75842	Accepted
AR+GARCH	5	0.51910	0.71091	0.75842	Accepted
FHS	6	0.27022	0.65605	0.49313	Accepted

Table 5: META VaR backtest for $\alpha = 0.99$

META VaR backtest for $\alpha = 0.99$					
Model	Exceptions	p_{POF}	p_{CI}	p_{CC}	Test status
Historical	6	0.27022	0.07624	0.11311	Accepted
Gaussian	13	0.00015	0.46821	0.00060	Rejected
Student	12	0.00058	0.39329	0.00187	Rejected
AR+GARCH	13	0.00015	0.46821	0.00060	Rejected
FHS	8	0.05160	0.15384	0.05441	Accepted

Table 6: JPM VaR backtest for $\alpha = 0.99$

JPM VaR backtest for $\alpha = 0.99$					
Model	Exceptions	p_{POF}	p_{CI}	p_{CC}	Test status
Historical	7	0.12481	0.11138	0.08670	Accepted
Gaussian	9	0.01925	0.20357	0.02881	Rejected
Student	7	0.12481	0.11138	0.08670	Accepted
AR+GARCH	13	0.00015	0.46821	0.00060	Rejected
FHS	7	0.12481	0.11138	0.08670	Accepted

and at $\alpha = 0.99$, the models that ar uniformly rejected are:

- AR+GARCH

while the Gaussian model is rejected for AAPL and META as well.

The takeaway is clear: Assuming Gaussianity drastically underestimates the Expected Shortfall. This can easily be interpreted as the Market returns (or equivalently losses) present heavier tails than a Gaussian distribution. The Expected Shortfall forecast (and/or computation) specifically relies on the tails. We observe that the Historical method performs well. This is surprising at first, as we don't expect pas returns to inform on the market's future. In this context, the rolling window being $W = 252$ allows to capture the heaviness of the returns/losses tails. Moreover, we do not observe a long trading period and as a consequence, volatility periods are relatively short in comparison to the rolling windows. The FHS model performs well as expected. It incorporates the reactivity of the GARCH model but does not assume Gaussianity on the forecasts and instead informs on past data - similarly to the Historical model. Similarly to the GARCH, the Gaussian method is not able to capture the heaviness of the tails - in contrary to the Student-t model.

The models that perform across Value at Risk and Expected Shortfall backtests are mainly historical and FHS. Both methods are similar. We believe that in a long period, the FHS will outperform the historical method. Indeed we can expect longer volatility clusters. There, the FHS model will react quickly, whereas we expected the historical approach to take around 50% of the rolling window's time to react.

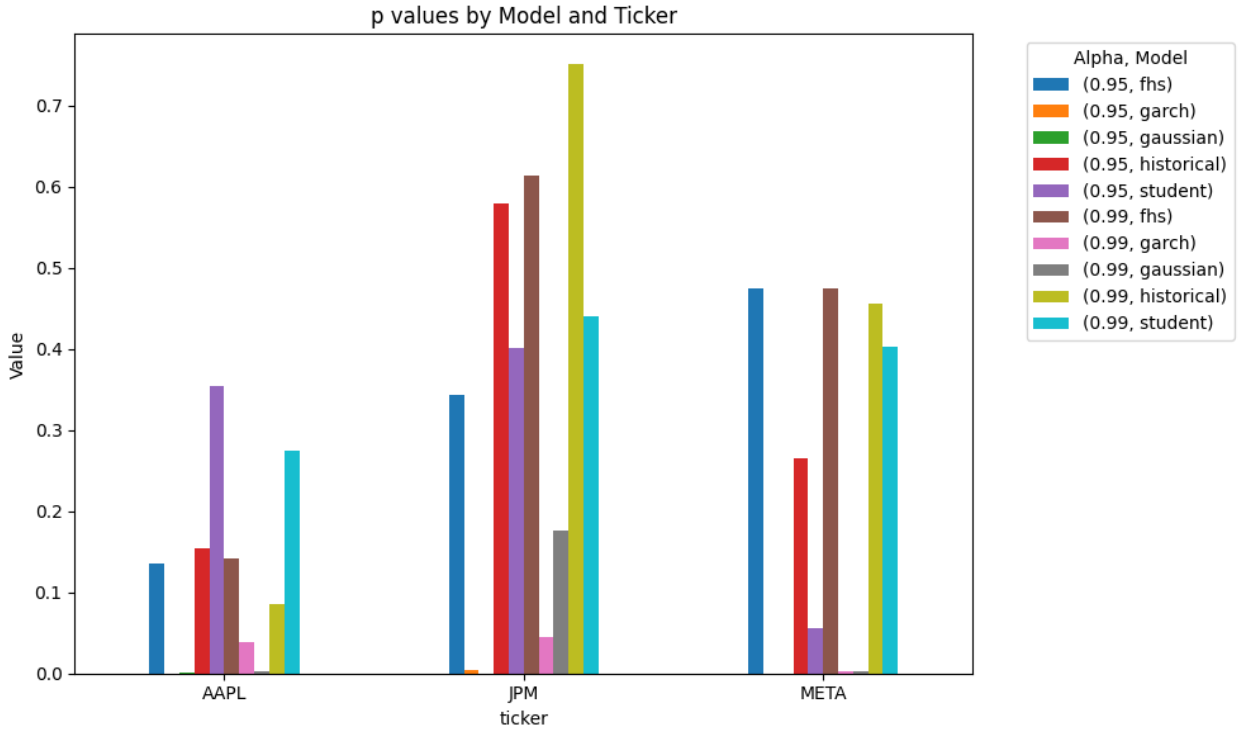

Figure 15: p values for the Acerby test

4 Copula fitting

In order to grasp a better understanding of the joint distribution of our assets, it is not enough to only have an intuition about marginal distributions. Hence, we implement the methodology of copulas, which consist on linking one-dimensional marginal distributions in order to form a multivariate distribution, while preserving the dependency structure between variables.

A copula $C : [0,1]^d \to [0,1]$ is a CDF is uniform marginals on $[0,1]$. Consider random variables $X_i \sim F_i$, where F_i is the CDF of the i-th random variable $(i = 1, .., n)$. Moreover, let F be the CDF of the random vector, that is $(X_1, ..., X_n) \sim F$. **Skar's theorem** guarantees that the (unique) existence of a copula $C : [0,1]^d \to [0,1]$ such that

$$F(x_1, ..., x_n) = C(F_1(x_1), ..., F_n(x_n))$$

Classical families of copulas are the Gaussian copulas and Student-t copulas.

Given a copula C, we can generate from C as we would generate from any CDF. having obtained $U_1, ..., U_m \sim C$, we can retrieve marginals by inverse CDF transform $X_i \sim F_i^-(U_i)$. This guarantees that the generated samples have the correct marginals, with the additional copula dependence.

In this part, we will use Gaussian and Student-t distributions, fitted with marginals being the returns of each asset in the portfolio.

4.1 Observing dependencies

Denote by $i \in \{\text{JPM}, \text{META}, \text{AAPL}\}$ an asset. We compute the empirical quantiles (pseudo-observations) as

$$U_{t,i} = \frac{\text{rank}(R_{t,i})}{W+1} \quad \text{where Rank}(R_{t,i}) \text{ is the position in the ordered return samples}$$

We start of by observing dependencies in both return and pseudo-observation spaces.

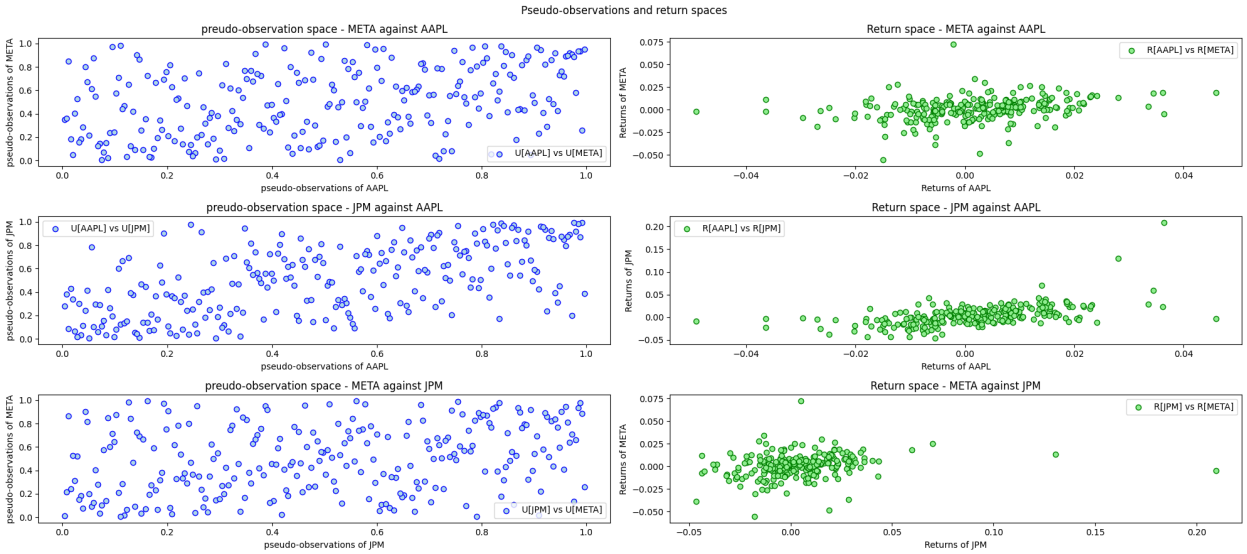

Figure 16: On the left are scatter plots representing $(U_{t,i}, U_{t,j})$ for $i \neq j$. The right side of the plot represents scatter plots of $(R_{t,i}, R_{t,j})$ for $i \neq j$.

Let us look first at META vs AAPL. The return space reveals a noticeable correlation between the two returns. Some spread is observed. The pseudo-observation space appears to present a slightly linear trend with a slight clustering in the top right and bottom left corners in comparison to top left and bottom right corners.

Looking at JPM vs AAPL, we observe a strong linear dependence (indicating correlation) in the return space. The correlation is more pronounced in comparison to META vs AAPL. A larger spread is observed in the right tail, while a slight drift from the linear dependence can be observed in the left tails. In the pseudo-observation space, the linear dependence is more noticeable in comparison to the case of META vs AAPL. We observe a stronger concentration in the top right and bottom left corners, while a noticeable absence of points in the bottom right and top left corners. Moreover the linear dependence is clearly noticeable in comparison to the already present linear dependence in META vs AAPL.

Taking a look at META vs JPM, we observe less pronounced linear dependence in the return space in comparison to META vs AAPL and JPM against AAPL. While still present in the bulk, the linear dependence appears to drift in the right tail. The pseudo-observation space presents a slight linear trend from the bottom left corner to the top right corner. Stronger concentration of scatter points is also observed in the bottom left and top right corners, while a light absence of scatter points are observed in the top left and bottom right corners.

In hindsight, we observe similar behaviors (sometimes more pronounced) for all the assets. A linear trend in the pseudo-observation space with light clustering in the bottom left and top right corners as well as a linear dependence in the return space. In all three cases, the bulk in the return space is relatively symmetric around its mean.

This justifies the use of a copula. The linear trend in both spaces and the in-majority symmetry of the

returns justifies the use of a Gaussian copula. The light concentrations in the tails in the pseudo-observation space justify the use of a Student-t copula as well to capture heavier tails.

4.2 Copula fitting

Given our observations and previous deductions, we now fit a Gaussian copula and Student-t copula to our data.

Let $C_\theta : [0,1]^3 \to [0,1]$ be a copula parametrized by a parameter $\theta \in \Theta$. Consider F_1, F_2, F_3 CDFs representing the distributions of the returns for, say, META, JPM, AAPL receptively. This is why $d = 3$ was chosen above.

We aim, as mentioned previously, to represent the joint distribution of returns $(R_{t,1}, R_{t,2}, R_{t,3}) \sim F$ which is unknown and not observable. Given the justifications of Part 4.1, we will consider a Gaussian copula and a Student-t copula which are both parametric copulas. Let any of those be C_θ.

The copula C_θ is fitted (usually through MLE for Gaussian and Student-t) to $(U_{t,1}, U_{t,2}, U_{t,3})$. That is, $\theta \in \Theta$ is estimated via Maximum Likelihood Estimation (recall that C is a CDF) to the data $(U_{t,1}, U_{t,2}, U_{t,3})$.

The copula will capture the patterns in the pseudo-observation space.

Copulas Table		
Parameters	**Gaussian**	**Student**
$\widehat{\boldsymbol{\rho}}$	$\begin{pmatrix} 1.000 & 0.349 & 0.600 \\ & 1.000 & 0.302 \\ & & 1.000 \end{pmatrix}$	$\begin{pmatrix} 1.000 & 0.343 & 0.598 \\ & 1.000 & 0.287 \\ & & 1.000 \end{pmatrix}$
$\widehat{\nu}$	$\emptyset$	13.051

Table 7: Parameters of Gaussian and Student fitted copulas. The numbers have been rounded to the third decimal

We observe similar correlation estimates between Gaussian and Student copulas (as expected since we are fitting to the same data and both distributions are relatively similar in their properties). However, the estimated degree of freedom of the Student-t distribution, which captures the heaviness of the tails, is $\widehat{\nu} \approx 13$. To illustrate this difference, Figure 17 displays a one dimensional example. Figure 17 represents the PDF of a Gaussian $\mathcal{N}(0,1)$ distribution, that of a $t_{13}(0,1)$ and finally the difference between the two distributions, that is $\text{diff}(x) = t_{13}(x; 13, 0, 1) - \mathcal{N}(x; 0, 1)$.

We observe that for the Student-t ($\nu = 13$), the distribution heavier tailed in comparison to the gaussian. This can be a desirable feature for financial returns.

4.3 Simulating from the obtained copulas

Having fitted copulas, we can now generate joint returns from the three assets, JPM, AAPL and META. This allows us to model a portfolio consisting of these three assets.

In this project, we will consider an equal-weight portfolio of the three assets, that is

$$R_t^{\text{Portfolio}} = \frac{1}{3} R_t^{\text{JPM}} + \frac{1}{3} R_t^{\text{AAPL}} + \frac{1}{3} R_t^{\text{META}}$$

We start of by visualizing the equally-weighted portfolio's value across time on the first window $W = 252$ in Figure 18.

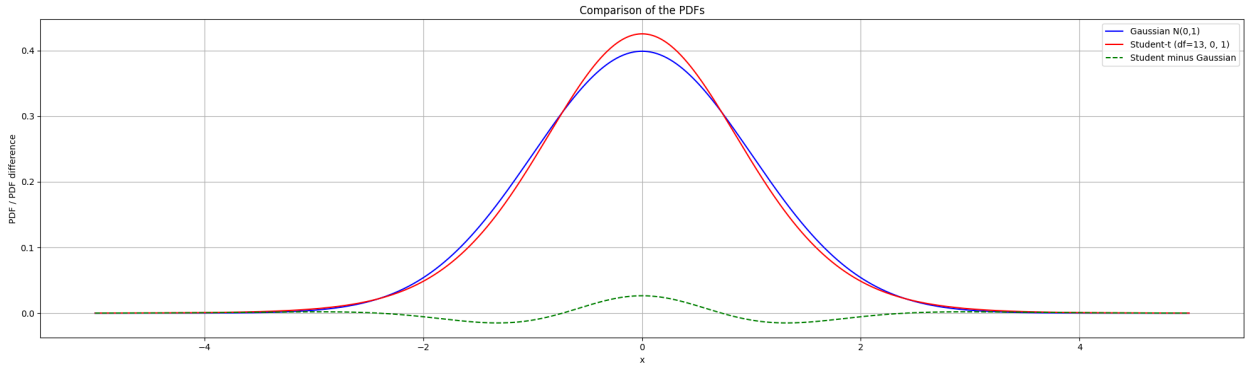

Figure 17: One dimensional PDF comparisons. Dashed green represents $\text{diff}(x) = t_{13}(x; 13, 0, 1) - \mathcal{N}(x; 0, 1)$, solid red is $t_{13}(x; 0, 1)$ and solid blue displays $\mathcal{N}(x; 0, 1)$ as $x \in [-5, 5]$.

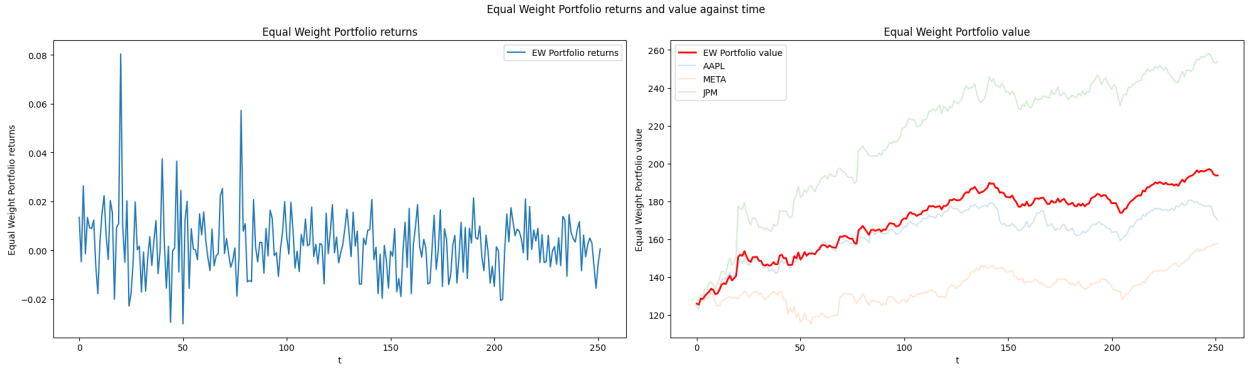

Figure 18: Portfolio value in the first window $W = 252$.

Our goal in this section is to model the joint dependencies between the returns of all the assets (JPM, META, AAPL). We will use the previously fitted copulas for this matter.

Consider a copula $C : [0,1]^3 \to [0,1]$ (with $d = 3$ as before) modeling the joint returns of the asset forming our portfolio.
Recall that here C is either Gaussian or Student-t (multivariate) and we have estimated parameters of $C = C_\theta$ in Part 4, with results reported (rounded up to the third decimal) in Table 4.2.

As previously mentioned, given a copula C, we can generate from C as we would generate from any CDF. having obtained $U_1, ..., U_m \sim C$, we can retrieve marginals by inverse CDF transform $R_{t,i} \sim F_i^-(U_i)$. This guarantees that the generated samples have the correct marginals, with the additional copula dependence.
We can thus generate $T \in \mathbb{N}$ joint returns as follows:

Step 0: Fit the copula as described in Part 4 and estimate $\widehat{\theta}$ so that $C \equiv C_{\widehat{\theta}}$

Step 1: Generate $(U_{t,1}, U_{t,2}, U_{t,3}) \sim C$ for $t = 1, ..., T$

Step 2: Revert back to the marginal distributions using the empirical CDFs: , we set

$$\tilde{R}_{t,i} := F_i^-(U_{t,i}) \quad \text{for} \quad t = 1, ..., T \quad \text{and} \quad i = 1, 2, 3$$

Step 3: Return $(\tilde{R}_{t,1}, \tilde{R}_{t,2}, \tilde{R}_{t,3})$ for $t = 1, ..., T$

(Optional) Step 4: To return realizations of $\tilde{R}_t^{\text{Portfolio}}$, return

$$\tilde{R}_t^{\text{Portfolio}} = \frac{1}{3}\tilde{R}_{t,1} + \frac{1}{3}\tilde{R}_{t,2} + \frac{1}{3}\tilde{R}_{t,3}$$

We now generate $T = 622$ samples of returns using the fitted copulas (Student-t and Gaussian).

We begin with scatter plots. Figure 19 represents the generations of $U_{t,i}$ for $t = 1, ..., T$ and $i = 1, 2, 3$ (indexes of the stocks) on the left, while the right represents the recovered (generated) returns. For the returns, we also display the observed returns to compare.

We display the scatter plots for both Fitted Gaussian and fitted Student-t copulas, for every (non-redundant) pair of assets.

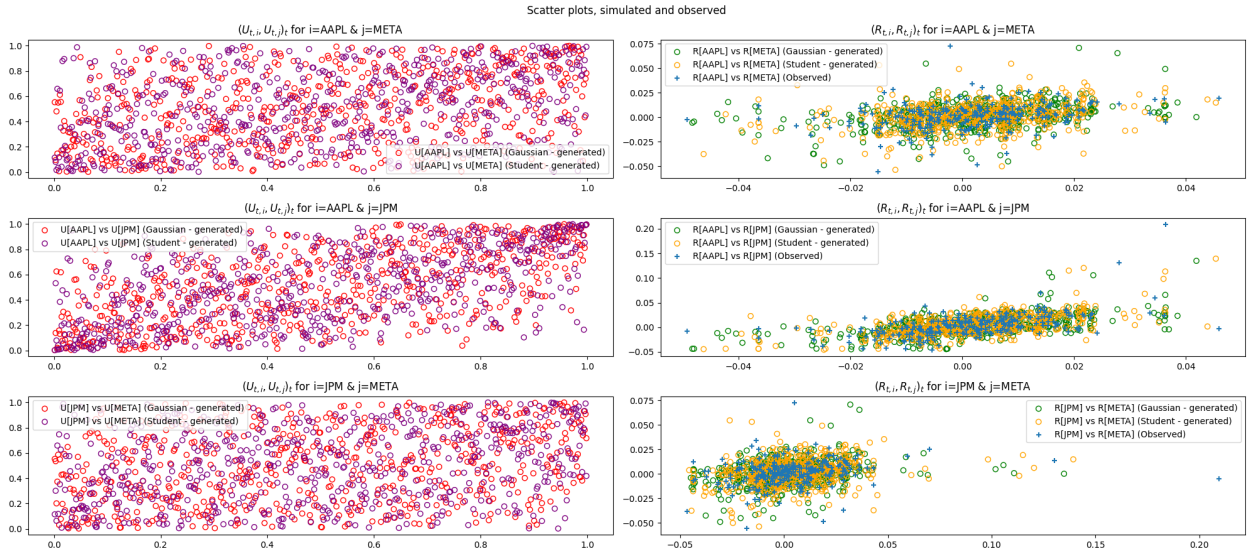

Figure 19: Scatter plots for generated data from both copulas (Gaussian and Student-t) and comparison with observed data.

We observe that the generated pseudo-observations are aligned with what we observed in Figure 16.

The increased number of samples $T > W$ highlight the clustering that we observed initially. This confirms that the copulas captured the dependence structured that we guessed initially. In the return space, the dependence structure is conserved and consistent.

Figure 20 represents the generated returns as a time series, as well as the true observed returns. We observe consistency and the frequency of extreme events appears to be truthful. The apparent frequency

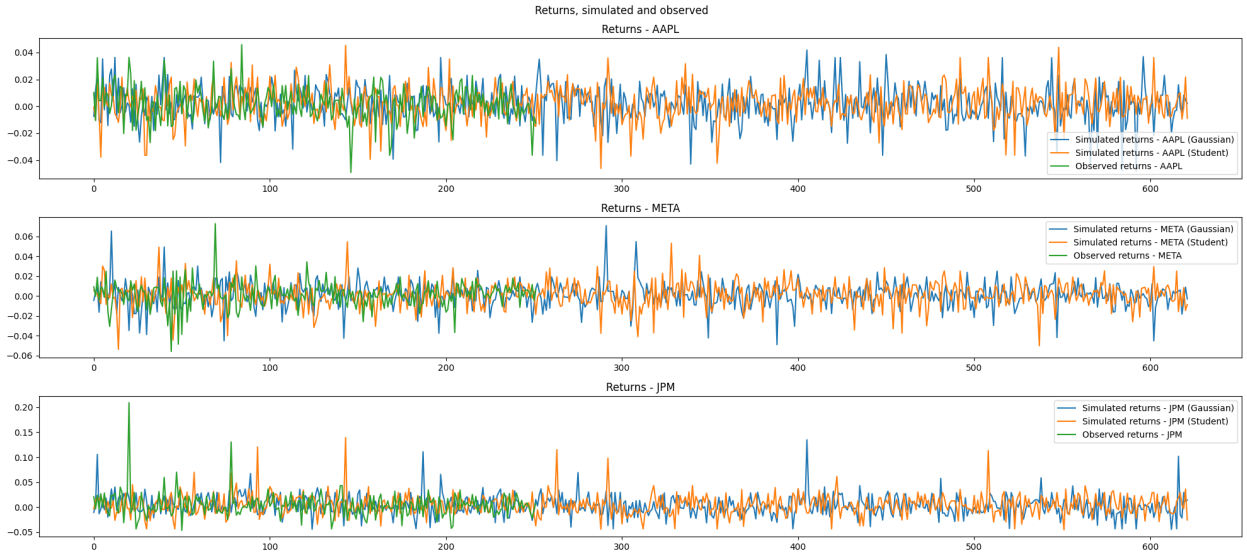

Figure 20: Generated return as a time series from each copula, compared with the observed time series

confirms our beliefs that the Gaussian and Student-t should capture the dependence structure well enough (at least for our purpose).

We finish with a histogram representation of the simulated returns, with the observed returns for each stock, represented in Figure21. We observe no significant discrepancy.

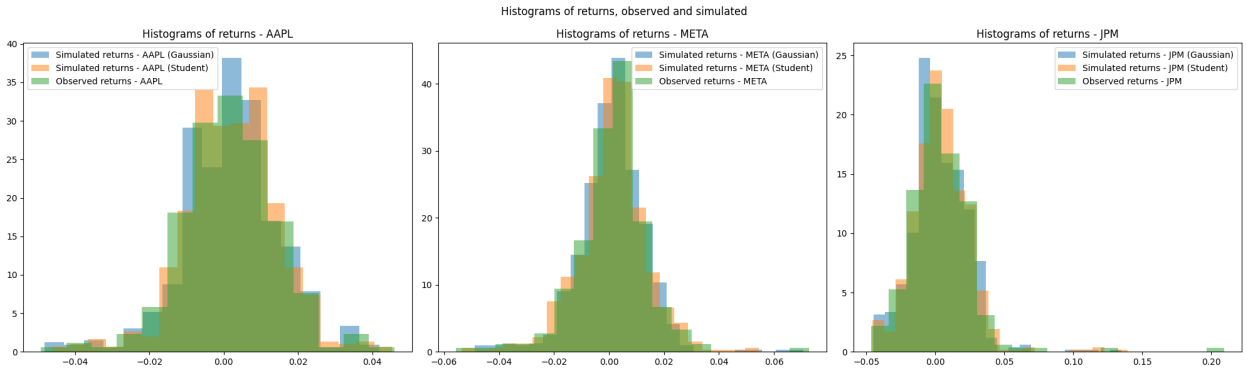

Figure 21: Histograms of generated returns as well as histogram of observed returns

In conclusion, our copula fits appear to capture the dependence structure well (as observed in Figure 19) and are able to recover the marginals as seen in Figure 20 and 21

We conclude this section with the visualization of simulated price paths of each asset using the copula (we simulate log-returns and revert back to prices) in Figure 22. We perform this simulation only on the first window $W = 252$ for illustration purposes. We also display the observed paths on top.

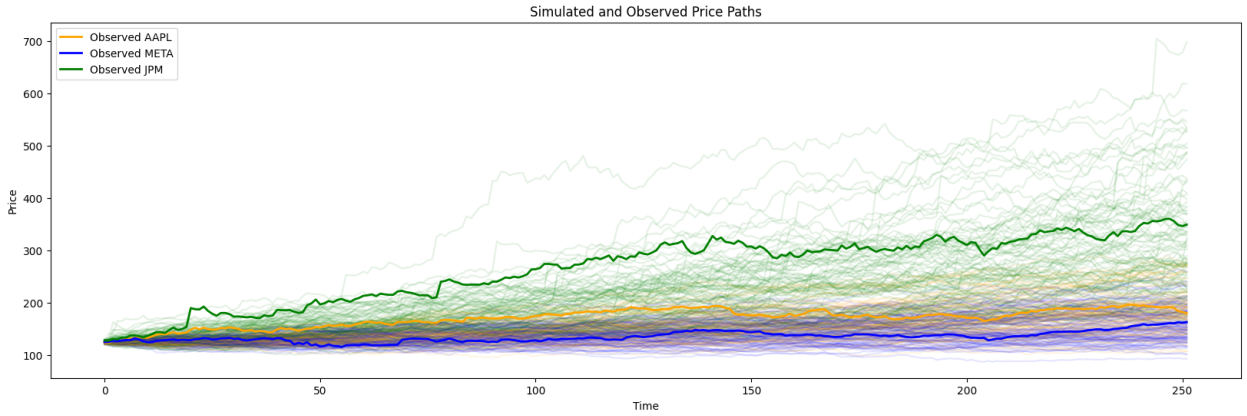

Figure 22: Simulated price paths. Light-colored are simulated paths and in bold are the observed paths on the window $W = 252$

4.3.1 Further analysis

In order to verify weather or not our assets contain monotonic relationships, we implemented Spearman's rank correlation r_s for each asset pair on the first window $W = 252$ (see Table 8).

Spearnman's rank correlation r_s is defined as Pearson's correlation coefficient ρ between the *rank* variables. These may be computed by converting each pair of raw scores $R_{t,i}, R_{t,j}$ into ranks $\text{rank}_{t,i}(R_{t,i}), \text{rank}_{t,j}(R_{t,j})$. Then, r_s may be computed as follows:

$$r_s = \rho[\text{rank}_{t,i}(R_{t,i}), \text{rank}_{t,j}(R_{t,j})] = \frac{\text{Cov}[\text{rank}_{\text{t,i}}(\text{R}_{\text{t,i}}), \text{rank}_{\text{t,j}}(\text{R}_{\text{t,j}})]}{\sigma_{\text{rank}_{t,i}(R_{t,i})}\sigma_{\text{rank}_{t,j}(R_{t,j})}}.$$

Note that only when all ranks are distinct integers (they have no ties), r_s may then be computed by using the proper formula:

$$r_s = 1 - \frac{6\ \Sigma d_i^2}{n(n^2-1)}$$

where n denotes the number of observations, and $d_i = \text{rank}_{t,i}(R_{t,i}) - \text{rank}_{t,j}(R_{t,j})$ is the difference between the two ranks of each observation.

Also, since we wish to measure correspondence of rankings, we compute Kendall's τ is. All coefficients are positive and highly significant (p-values $\ll 1\%$), with the strongest dependence between META and JPM.

Pair	r_s	$p(\rho_S)$	τ	$p(\tau)$
AAPL–META	0.279434	6.7285×10^{-6}	0.19585	3.6321×10^{-6}
AAPL–JPM	0.589580	0	0.41744	0
META–JPM	0.343454	2.1909×10^{-8}	0.23645	2.2516×10^{-8}

Table 8: Spearman's ρ_S and Kendall's τ for asset return pairs on the first window $W = 252$.

Kendall's τ measures rank correlation by comparing all pairs of observations. Formaly,

$$\tau = \frac{C - D}{\frac{1}{2}n(n-1)},$$

where C denotes the number of concordant pairs $(x_i - x_j)(y_i - y_j) > 0$ and D represents the number of discordant pairs $(x_i - x_j)(y_i - y_j) < 0$.

In order to compute it, we first compare every pair $i < j$, classify as concordant or discordant, count C and D, plug into formula.

Note that the range is $-1 \leq \tau \leq 1$

5 Portfolio Backtesting for Value at Risk and Expected Shortfall

5.1 Setup and link to Section 3

In this section we revisit the five univariate risk–measurement procedures introduced in Section 2:

- Historical Simulation (HS),
- Gaussian parametric fit,
- Student–t parametric fit,
- conditional Gaussian AR–GARCH,
- Filtered Historical Simulation (FHS).

We work with daily losses $L_t = -R_t$ so that large positive values correspond to large negative returns. One–day–ahead forecasts of $\mathrm{VaR}_{\alpha,t+1}$ and $\mathrm{ES}_{\alpha,t+1}$ at levels $\alpha \in \{0.95, 0.99\}$ are obtained using a rolling window of $W = 252$ trading days (approximately one trading year), exactly as described in Section 3.

The evaluation framework is also the same as in Section 3:

- **VaR backtests:** the exception indicators $I_t = \mathbb{1}\{L_t > \mathrm{VaR}_{\alpha,t}\}$ are used to compute the Kupiec Proportion–of–Failures (POF) test for unconditional coverage and the Christoffersen independence and conditional coverage tests.
- **ES backtest:** Expected Shortfall forecasts are assessed using the Acerbi–Székely Z_1 test, based on realized tail losses and a Monte Carlo approximation of the null distribution.

To avoid redundancy, we do not repeat the formal definitions of the tests here, and instead focus on the empirical behaviour of the models for the three assets AAPL, META and JPM.

5.2 Visual inspection of VaR forecasts

We first visually inspect the one–step–ahead $\mathrm{VaR}_{0.95}$ forecasts for each model and asset. Figures 12–14 overlay the realized returns with the corresponding $\mathrm{VaR}_{0.95}$ curves (converted back to return space).

Across all three stocks, none of the models is able to react instantaneously to sudden, isolated loss spikes. Large negative returns occurring in otherwise low–volatility periods often lead to VaR breaches, regardless of the specification used. This highlights the intrinsic difficulty of predicting rare, idiosyncratic shocks with purely historical or parametric methods.

In contrast, extended periods of elevated volatility are better captured by the dynamic models. Around the volatility increase observed between March and April 2025, the AR–GARCH and FHS specifications adjust their forecasted VaR levels more strongly and more persistently than the static Historical, Gaussian and Student–t fits. The latter do react, but with less pronounced adjustments.

There are also some asset–specific patterns. META exhibits more loss spikes outside high–volatility clusters than AAPL and JPM, which is consistent with the preliminary analysis in Part 1, where META was identified as the most volatile stock. For JPM, the qualitative behaviour of the models is similar, but the timing and magnitude of individual shocks differ, leading to different breach patterns across assets.

Overall, the visual inspection suggests:

- sudden, isolated shocks are hard to anticipate for all models;
- dynamic specifications (AR–GARCH, FHS) react more clearly to volatility clusters, albeit with some delay;
- static specifications (HS, Gaussian, Student–t) adjust more slowly to changes in the volatility regime.

5.3 Quantitative results for VaR backtests

We now turn to the quantitative VaR backtests. For each asset, model and confidence level we compute the number of exceptions N, the Kupiec POF, Christoffersen independence (CI) and conditional coverage (CC) statistics, together with their p–values. The results are reported in Tables 1–6.

Following the discussion in Section 3.2, we primarily base our decision on the CC test and classify a model as *Accepted* if the CC p–value exceeds 5%, and as *Rejected* otherwise. The main patterns are:

- **AAPL, $\alpha = 0.95$:** all models are rejected. The POF test typically indicates too many exceptions, i.e. the models underestimate risk at this level. The CI test is less problematic, suggesting that the timing of exceptions is not grossly misspecified but the overall frequency is.
- **META, $\alpha = 0.95$:** all models are accepted. Here the observed exception rates are closer to the nominal 5% level and clustering is less severe, so unconditional and conditional coverage are both satisfactory.
- **JPM, $\alpha = 0.95$:** all models are again rejected. As in the AAPL case, the issue is mainly with unconditional coverage: observed failure rates deviate significantly from the nominal level.
- **All assets, $\alpha = 0.99$:** results are more mixed. Historical Simulation and FHS are accepted for most assets, whereas the Gaussian model is often rejected, especially for META. The Student–t and AR–GARCH specifications sit in between: they improve upon the Gaussian model but still fail the CC test in some cases.

Taken together, the VaR backtests confirm that:

- plain Gaussian assumptions tend to underestimate tail risk, especially at the 99% level;
- allowing for heavier tails (Student–t) or time–varying volatility (AR–GARCH, FHS) generally improves VaR performance;
- Historical Simulation can perform reasonably well when the rolling window is long enough to capture tail behaviour, but it remains sensitive to the specific sample of past returns.

5.4 Quantitative results for ES backtests

We now consider the Expected Shortfall forecasts. As detailed in Section **??**, we use the Acerbi–Székely Z_1 statistic together with a Monte Carlo approximation of its null distribution. Figure 15 reports the resulting p–values for each asset, model and confidence level.

At the $\alpha = 0.95$ level, the ES backtest shows that:

- the Gaussian and AR–GARCH specifications are uniformly rejected across all three stocks, indicating systematic *underestimation* of tail losses;
- Historical Simulation, Student–t and FHS generally perform better, with Z_1 statistics that are not significantly positive for most asset–model combinations.

At the more extreme $\alpha = 0.99$ level, the test becomes stricter because it focuses on very few extreme observations. In this case:

- AR–GARCH is rejected for all assets;
- the Gaussian specification is rejected for AAPL and META, and is borderline for JPM;
- FHS and Student–t continue to perform comparatively better, although even these models occasionally underestimate the most extreme losses.

The strong rejection of Gaussian ES forecasts is consistent with the visual impression that empirical losses display heavier tails than a normal distribution. The good performance of Historical Simulation is less obvious a priori, but can be explained by the relatively long rolling window ($W = 252$) and the limited sample period, which together allow the empirical distribution to capture tail behaviour reasonably well. FHS combines the volatility–adaptation of GARCH with a non–parametric treatment of residuals, which helps it track both changing volatility and heavy tails.

5.5 Overall comparison of models

Combining the VaR and ES backtests yields a coherent picture:

- Simple i.i.d. Gaussian models are clearly inadequate for equity returns at both 95% and 99% confidence levels. They typically underestimate tail risk and are frequently rejected by both VaR and ES backtests.
- Allowing for heavy tails (Student–t) or time–varying volatility (AR–GARCH) improves performance, but does not completely eliminate underestimation of extreme losses, especially at 99%.
- Historical Simulation and FHS are the most robust across both VaR and ES criteria. HS benefits from a sufficiently long window to capture empirical tail behaviour, while FHS adds a dynamic component that reacts to volatility clustering without imposing a fully parametric tail shape.

In a longer sample with more pronounced volatility regimes, we would expect FHS to gradually outperform pure Historical Simulation: the GARCH filter should allow faster adjustment to new volatility clusters, whereas HS requires a substantial fraction of the window to be updated before the empirical tail distribution reflects the new regime. Within the present sample, however, both HS and FHS provide comparatively reliable risk forecasts, especially at the 95% confidence level.

5.5.1 Univariate models

The following Figures (Figure 23 and Figure 24) represent the backtests of the Value at Risk visually. For now, discard the 'Gaussian copula' and 'Student copula' curves. We observe similar behavior to what we observed previously. While the portfolio appears to be more stable from the fact that its somewhat diversified, we again observe that the models are not able to prevent market shocks (as expected). However this time the shocks can come form each underlying stock, which increases the number of potential shocks (even if it is somewhat diversified).

Both $\alpha = 0.95$ and $\alpha = 0.99$ cases behave similarly to what we previously observed.

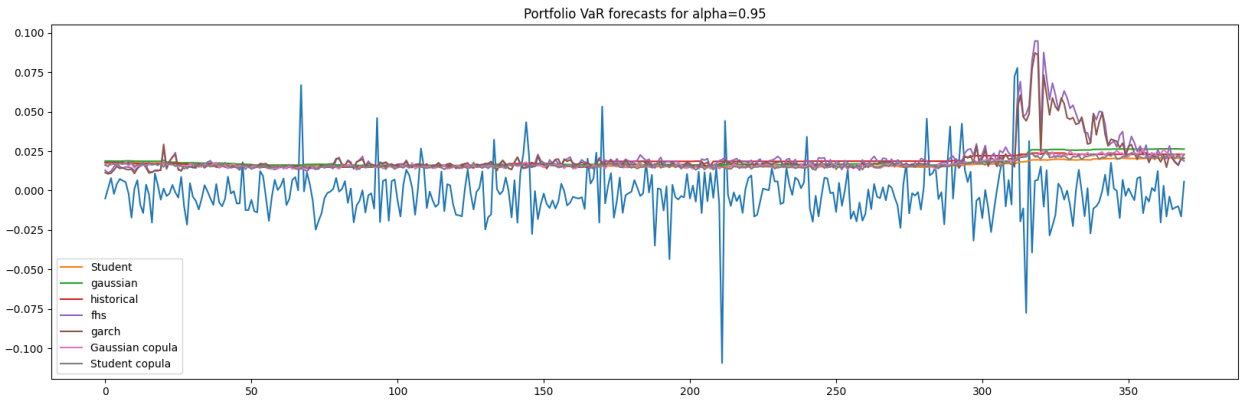

Figure 23: VaR Backtest visualization with copulas for $\alpha = 0.95$

When performing the Value at Risk Backtest we obtain the following results for $\alpha = 0.95$ in Table 9 and Table 10 presents results for $\alpha = 0.99$.

We observe that for $\alpha = 0.95$, all the univariate models are Accepted. On the other hand, as we tighten the risk management to $\alpha = 0.99$, we have that most models dramatically struggle on the POF frequency test and in turn only the FHS model passes the test.

For the Expected Shortfall Backtest, see Figure 25 we observe that for the only models that are not rejected are Student-t for both $\alpha = 0.95$ and $\alpha = 0.99$, as well as the historical for $\alpha = 0.99$.

We believe that no model is able to capture at the same time the tail behaviors and the increased risks due to cross-asset correlations. The student distribution is able to capture the tails on the other hand, and so is the historical approach (but as mentioned previously, this is a single run Backtest and the Historical model is not robust)

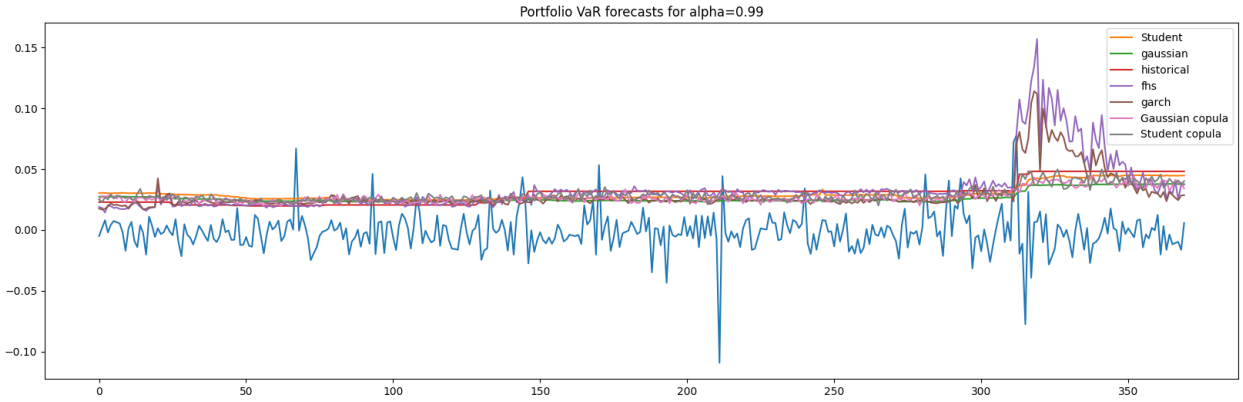

Figure 24: VaR Backtest visualization with copulas for $\alpha = 0.99$

Table 9: Equal Weight Portfolio VaR backtest for $\alpha = 0.95$

JPM VaR backtest for $\alpha = 0.95$					
Model	Exceptions	p_{POF}	p_{CI}	p_{CC}	Test status
Historical	25	0.14016	0.32417	0.20719	Accepted
Gaussian	24	0.20874	0.26608	0.24452	Accepted
Student	27	0.05699	0.45996	0.12438	Accepted
AR+GARCH	25	0.14016	0.32417	0.20719	Accepted
FHS	24	0.20874	0.13118	0.42501	Accepted

Table 10: Equal Weight Portfolio VaR backtest for $\alpha = 0.99$

JPM VaR backtest for $\alpha = 0.99$					
Model	Exceptions	p_{POF}	p_{CI}	p_{CC}	Test status
Historical	10	0.00654	0.26035	0.01316	Rejected
Gaussian	13	0.00015	0.46821	0.00060	Rejected
Student	11	0.00203	0.32377	0.00527	Rejected
AR+GARCH	11	0.00203	0.41093	0.00611	Rejected
FHS	9	0.01926	0.50231	0.05163	Accepted

5.5.2 Adding the copula

Shifting our focus now to the copula plots, we observe that the copulas behave similarly to the historical simulation. This is reasonable, as the copula method essentially resamples from fitted copulas built on the empirical CDFs of the marginal losses.

From a qualitative point of view, we expect the copula–based models to perform well in terms of Expected Shortfall: by modelling the joint distribution explicitly, they account for "snowball" effects in which large losses in several names occur simultaneously.

Note however that the historical simulation observes the cross–asset correlation effects and potential tail event realizations *only to the extent that these specific joint moves actually appear in the window.* In contrast, once the copula has been fitted, we can generate many additional synthetic scenarios that preserve

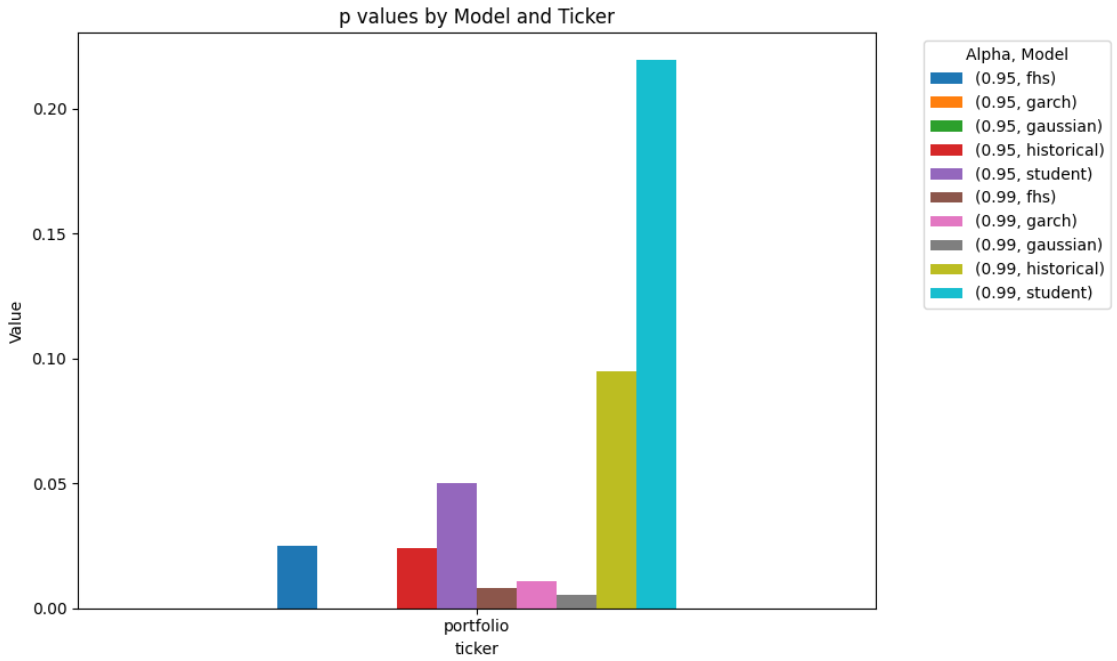

Figure 25: ES Backtest for equal weight portfolio

the estimated dependence structure and thus explore a richer set of joint tail outcomes. This should, in principle, give the copula approach an advantage for portfolio tail–risk measurement, at the cost of some additional model risk if the copula family is misspecified.

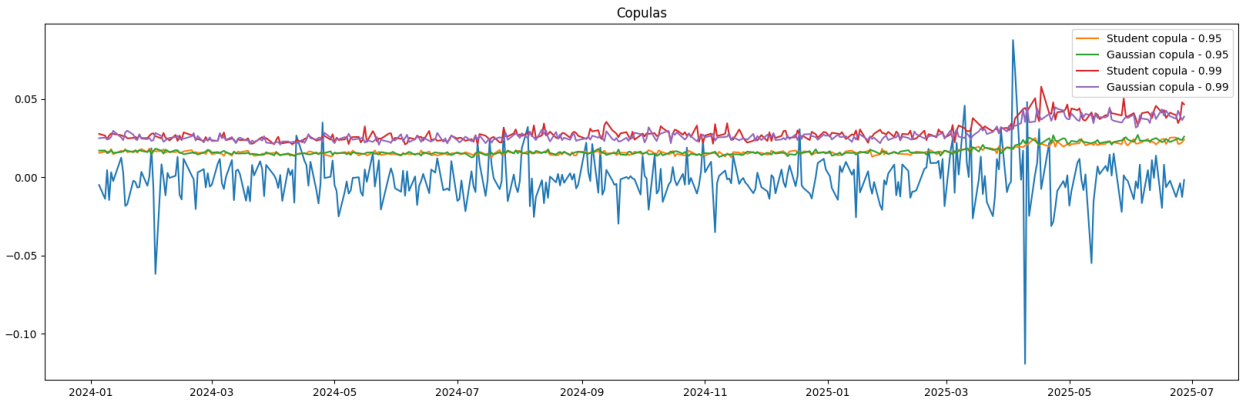

Figure 26: Copula VaR Backtest visualization for Portfolio

From a quantitative perspective, Figure 25 shows the p–values of the Z_1 ES backtest for the equal–weight portfolio. At $\alpha = 0.95$, all models except the Student–t and (marginally) the historical method yield very small p–values, indicating systematic *underestimation* of portfolio tail losses. The Student–t model sits right at the 5% threshold, while Historical and FHS are borderline but still clearly less extreme than Gaussian and AR–GARCH.

At the tighter $\alpha = 0.99$ level the contrast becomes more notable: Gaussian and AR–GARCH are strongly rejected, and FHS also falls below the 5% cutoff, whereas the historical and Student–t models produce substantially larger p–values (around 0.10 and 0.22 respectively). In other words, only these two specifications avoid clear evidence of ES underestimation for the portfolio at very high confidence.

This pattern is consistent with our visual impressions. In the last part of the sample the portfolio

experiences a pronounced volatility spike. The historical method, by construction, places substantial weight on these recent large losses and therefore delivers relatively high ES forecasts; the Z_1 test, which is designed to penalize *under*–estimation of tail risk more than conservative over–estimation, tends to "reward" this behavior. The Student–t model benefits from a similar mechanism through its heavy–tailed parametric form, which allows it to accommodate both the marginal tail behavior and the increased joint risk induced by cross–asset dependence. Together, these results reinforce the view that, for this portfolio and sample, models that either explicitly incorporate heavy tails (Student–t) or mechanically track recent extreme losses (Historical) provide the most reliable ES forecasts.

5.5.3 Comparison and analysis

Figure 27 summarizes all portfolio VaR forecasts on a single plot, for both confidence levels and for all univariate and copula–based models. During the long calm period of 2024, all curves are fairly close: the different specifications essentially produce a band of VaR lines just above the realized losses, with the 99% levels (as expected) lying slightly above the corresponding 95% levels. In this region the choice of model makes little difference from a risk–measurement point of view.

The picture changes around March–May 2025, when the portfolio goes through a pronounced volatility spike. Here the models separate more clearly. The FHS and historical specifications produce the strongest increase in VaR, followed by the Student and copula–based models, while the Gaussian and AR–GARCH VaR remain comparatively flat. This is exactly the behaviour reflected in the backtests:

- In the VaR backtests for the portfolio, all univariate models pass at $\alpha = 0.95$, but at $\alpha = 0.99$ only FHS is clearly accepted by the CC test; the other models, especially Gaussian, tend to generate too many exceptions.
- In the ES backtests, the Student and historical models obtain the largest Z_1 p–values, with FHS close behind. Gaussian and AR–GARCH are systematically rejected, in line with their relatively muted reaction to the volatility spike.
- Copula–based VaR forecasts track the historical/FHS curves fairly closely: they rise when the joint dependence structure amplifies losses, but do not radically change the backtesting picture compared to the best univariate models.

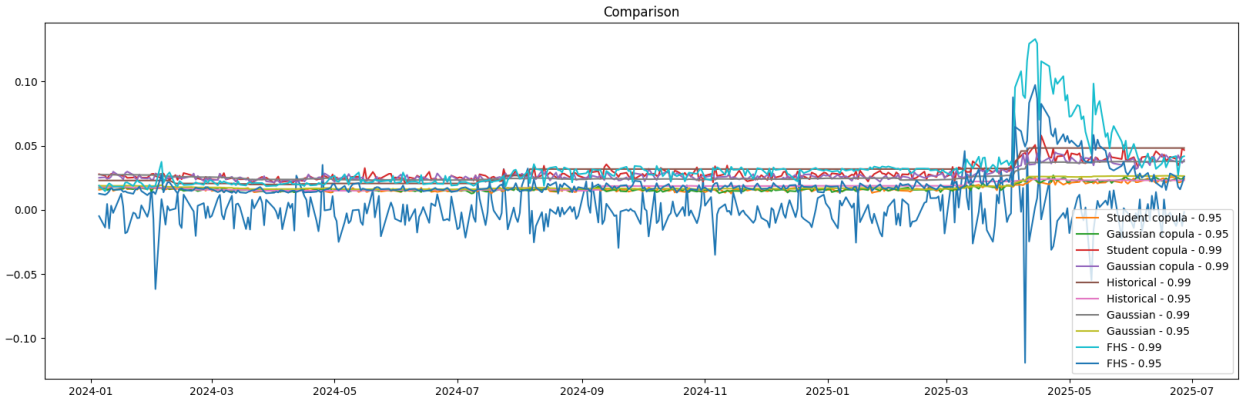

Figure 27: All methods VaR Backtest visualization

Putting all the evidence together, a naive "ranking" based purely on p–values would crown the historical model as the winner (with Student and FHS not far behind), especially for ES at $\alpha = 0.99$. This is intuitive: once the window contains several very large losses, historical simulation simply extrapolates this high–volatility regime, and the backtests—which are designed to penalize *under*estimation of tail risk—reward such conservative forecasts. However, this should not be over–interpreted as proof that the historical model is "true" or universally better: its good performance is largely driven by the fact that the

end of our sample is dominated by a single volatility episode, so a method that heavily weights the most recent data is naturally advantaged.

A more cautious interpretation is therefore the following:

- The backtests clearly reject simple Gaussian (and, to a lesser extent, Gaussian–GARCH) modelling of portfolio losses, especially for ES and at $\alpha = 0.99$.
- Models that incorporate heavy tails and/or volatility clustering—Historical, Student, FHS and the Student copula—are all broadly consistent with the data and substantially more reliable for tail risk.
- Within this group, historical and Student stand out in our finite sample, but their apparent dominance is partly mechanical (driven by the last volatility cluster). FHS and copula–based approaches offer a more structured way to model time–varying volatility and cross–asset dependence and would likely become more advantageous in longer samples with multiple volatility regimes.

In short, the combined VaR and ES backtests suggest that one should avoid purely Gaussian i.i.d. models and instead rely on specifications that allow for both heavy tails and time–varying dependence. In our data set, historical simulation provides the most conservative and backtest–friendly forecasts, with Student–t, FHS and copula–based models forming a close second tier that arguably offers a better balance between realism and robustness to regime changes.

A Python code

```python
# %% [markdown]
# # **Market Risk | VaR, ES, and Copulas**

# %% [markdown]
# **Names of all group members:**
# - Philipp Mayer (philipp.mayer@epfl.ch)
# - Rafael Barroso (rafael.barroso@epfl.ch)
# - Roberto Magno
# ---

# %% [markdown]
# ## **Part 0 - Setup**

# %%
import os
import numpy as np
import copulae
import math
import pandas as pd
import yfinance as yf

import matplotlib.pyplot as plt
from matplotlib import gridspec

from dataclasses import dataclass
from typing import Dict, Tuple
from pathlib import Path

from scipy.stats import jarque_bera
from scipy.stats import t as student_t
from scipy.stats import gaussian_kde
from scipy.stats import norm
from scipy.stats import t as t_stud
from scipy.stats import chi2
import scipy.stats as stats

# %%
# Locate the Project 1 directory to this notebook's working directory
PROJECT_DIR = Path.cwd()
DATA_DIR = PROJECT_DIR / 'data'      # Use Path division operator
OUT_DIR = PROJECT_DIR / 'output'     # Use Path division operator

for d in [DATA_DIR, OUT_DIR]:
    d.mkdir(parents=True, exist_ok=True)

# Parameters
TICKERS = ['AAPL', 'META', 'JPM']   # list of tickers
START = '2023-01-01'                # start date for data download
END = '2025-06-30'                  # end date for data download
WINDOW = 252                        # rolling/first-window length in periods (days)
ALPHAS = [0.95, 0.99]               # list of confidence levels
np.random.seed(0)

print('Project dir:', PROJECT_DIR)

print('Downloading data to', DATA_DIR)
for t in TICKERS:
    print(f'  -> {t}')
    df = yf.download(t, start=START, end=END, progress=False, auto_adjust=False)
    if df.empty:
        print(f'     Warning: no data for {t}')
        continue
    out = df.reset_index()
    out = out[['Date', 'Adj Close']]
    out.to_csv(DATA_DIR / f'{t}.csv', index=False)
print('Done.')

# %%
files = [f for f in os.listdir(DATA_DIR) if f.lower().endswith('.csv')]
frames = []
for f in files:
    p = os.path.join(DATA_DIR, f)
    df = pd.read_csv(p, parse_dates=['Date'])
    df = df[['Date', 'Adj Close']]
    # Coerce to numeric and drop malformed rows
    df['Adj Close'] = pd.to_numeric(df['Adj Close'], errors='coerce')
    df = df.dropna(subset=['Date', 'Adj Close'])
    df = df.rename(columns={'Adj Close': f.split('.')[0]})
    df = df.set_index('Date').sort_index()
    frames.append(df)
prices = pd.concat(frames, axis=1).dropna(how='all')
```

```python

print(prices.head())

# %%
# Visualize the prices over time

plt.figure(figsize=(12, 6))
for ticker in TICKERS:
    if ticker in prices.columns:
        plt.plot(prices.index, prices[ticker], label=ticker)
plt.title('Adjusted Close Prices Over Time')
plt.xlabel('Date')
plt.ylabel('Adjusted Close Price')
plt.legend()
plt.grid(True)
plt.tight_layout()
plt.show()

# %% [markdown]
# ## **Part 1 - Empirical Stylyzed facts**

# %% [markdown]
# ### 1.1 - Computing returns

# %%
# Compute log-returns
returns = np.log(prices / prices.shift(1)).dropna(how='all')
returns = returns.dropna()  # Drop rows with any NaN values
returns.columns = TICKERS
print(returns.head())

losses = - returns
print(losses.head())

# %%
# Plot the returns over time
fig, axes = plt.subplots(len(TICKERS), 1, figsize=(12, 8), sharex=True)

for i, ticker in enumerate(TICKERS):
    if ticker in returns.columns:
        axes[i].plot(returns.index, returns[ticker], label=ticker, color=f'C{i}')

        # Add horizontal zero line
        axes[i].axhline(0, color='black', linewidth=1.5, linestyle='--')

        # Shade regions where returns are negative
        axes[i].fill_between(
            returns.index,
            returns[ticker],
            0,
            where=(returns[ticker] < 0),
            color='red',
            alpha=1,
            interpolate=True
        )

        axes[i].set_title(f'{ticker} Log Returns')
        axes[i].set_ylabel('Log Return')
        axes[i].grid(True, linestyle=':', linewidth=0.7)
        axes[i].legend(loc='upper right')

axes[-1].set_xlabel('Date')
plt.tight_layout()
plt.show()

# %%
# Anual volatility and rolling volatility
ann_vol = returns.std() * np.sqrt(252) # Annualized volatility; 252 trading days in a year

rolling_vol = returns.rolling(window=21).std() * np.sqrt(252) # 21 trading days in a month
mean_rolling_vol = rolling_vol.mean()

vol_comparison = pd.DataFrame({
    'Annual Scaled Volatility': ann_vol,
    'Mean Rolling Volatility': mean_rolling_vol
})

print(vol_comparison)

fig, axes = plt.subplots(len(TICKERS), 1, figsize=(12, 8), sharex=True)

for i, ticker in enumerate(TICKERS):
    axes[i].plot(rolling_vol.index, rolling_vol[ticker], label='21-day Rolling Vol', color='orange')
```

```
    axes[i].axhline(ann_vol[ticker], color='blue', linestyle='--', label='Scaled (252)')
    axes[i].set_title(f'{ticker} Annualized Volatility Comparison')
    axes[i].set_ylabel('Volatility')
    axes[i].grid(True, linestyle=':', linewidth=0.7)
    axes[i].legend(loc='upper right')

axes[-1].set_xlabel('Date')
plt.tight_layout()
plt.show()

# %%
# Obtain statistical metrics for the returns
stats_vol = rolling_vol.describe().T
print(stats_vol)

# %% [markdown]
# ### 1.2 - Visualizing dependencies

# %%
# Computation of cross-correlations and auto-correlations
max_lag = 25

correlation_returns = {}
absolute_correlation_returns = {}

cols_now = [f"now_{c}" for c in returns.columns]
cols_lag = [f"lag_{c}" for c in returns.columns]

for lag in range(max_lag + 1):

    R_now = returns.add_prefix("now_")
    R_lag = returns.shift(lag).add_prefix("lag_")

    # Correlation matrix of [now | lag] and take the cross block
    M = pd.concat([R_now, R_lag], axis=1).corr()
    C = M.loc[cols_now, cols_lag]              # shape: (n_assets, n_assets)
    correlation_returns[lag] = C         # rows: now_i, cols: lag_j

    # Absolute version
    M_abs = pd.concat([R_now.abs(), R_lag.abs()], axis=1).corr()
    C_abs = M_abs.loc[cols_now, cols_lag]
    absolute_correlation_returns[lag] = C_abs



correlation_returns_df = pd.DataFrame({k: C.stack() for k, C in correlation_returns.items()})
correlation_returns_df = correlation_returns_df.T

absolute_correlation_returns_df = pd.DataFrame({k: C.stack() for k, C in absolute_correlation_returns.items()})
absolute_correlation_returns_df = absolute_correlation_returns_df.T

# %%
correlation_returns_df

# %%
# Ploting raw correlations
## Extract lag index
lags = correlation_returns_df.index

## Autocorr (diagonal)
auto = {
    "AAPL": correlation_returns_df[("now_AAPL", "lag_AAPL")],
    "META": correlation_returns_df[("now_META", "lag_META")],
    "JPM":  correlation_returns_df[("now_JPM",  "lag_JPM")],
}

## Cross-corr (average both directions per pair)
cross = {
    "AAPL{META": 0.5 * (correlation_returns_df[("now_AAPL", "lag_META")]
                        + correlation_returns_df[("now_META", "lag_AAPL")]),
    "AAPL{JPM":  0.5 * (correlation_returns_df[("now_AAPL", "lag_JPM")]
                        + correlation_returns_df[("now_JPM",  "lag_AAPL")]),
    "META{JPM":  0.5 * (correlation_returns_df[("now_META", "lag_JPM")]
                        + correlation_returns_df[("now_JPM",  "lag_META")]),
}

## Plot: 2 rows × 3 cols
fig, axes = plt.subplots(2, 3, figsize=(12, 6), sharex=True)

### Row 1: autocorrelations (one asset per subplot)
for ax, (label, series) in zip(axes[0], auto.items()):
    ax.plot(lags, series, marker="o")
    ax.set_title(f"{label} autocorr")
```

```
    ax.grid(alpha=0.3)
    ax.set_ylabel("Correlation")

### Row 2: cross-correlations (one pair per subplot)
for ax, (label, series) in zip(axes[1], cross.items()):
    ax.plot(lags, series, marker="o")
    ax.set_title(f"{label} cross-corr")
    ax.grid(alpha=0.3)
    ax.set_xlabel("Lag")
    ax.set_ylabel("Correlation")

fig.suptitle("Autocorrelations (row 1) and Cross-correlations (row 2)", y=0.98)
plt.tight_layout()
plt.show()

# %%
# Ploting absolute correlations
## Extract lag index
lags = absolute_correlation_returns_df.index

## Autocorr (diagonal)
auto = {
    "AAPL": absolute_correlation_returns_df[("now_AAPL", "lag_AAPL")],
    "META": absolute_correlation_returns_df[("now_META", "lag_META")],
    "JPM":  absolute_correlation_returns_df[("now_JPM",  "lag_JPM")],
}

## Cross-corr (average both directions per pair)
cross = {
    "AAPL{META": 0.5 * (absolute_correlation_returns_df[("now_AAPL", "lag_META")]
                        + absolute_correlation_returns_df[("now_META", "lag_AAPL")]),
    "AAPL{JPM":  0.5 * (absolute_correlation_returns_df[("now_AAPL", "lag_JPM")]
                        + absolute_correlation_returns_df[("now_JPM",  "lag_AAPL")]),
    "META{JPM":  0.5 * (absolute_correlation_returns_df[("now_META", "lag_JPM")]
                        + absolute_correlation_returns_df[("now_JPM",  "lag_META")]),
}

## Plot
fig, axes = plt.subplots(2, 3, figsize=(12, 6), sharex=True)

### Row 1: autocorrelations (one asset per subplot)
for ax, (label, series) in zip(axes[0], auto.items()):
    ax.plot(lags, series, marker="o")
    ax.set_title(f"{label} autocorr")
    ax.grid(alpha=0.3)
    ax.set_ylabel("|Correlation|")

### Row 2: cross-correlations (one pair per subplot)
for ax, (label, series) in zip(axes[1], cross.items()):
    ax.plot(lags, series, marker="o")
    ax.set_title(f"{label} cross-corr")
    ax.grid(alpha=0.3)
    ax.set_xlabel("Lag")
    ax.set_ylabel("|Correlation|")

fig.suptitle("Absolute autocorrelations (row 1) and Absolute cross-correlations (row 2)", y=0.98)
plt.tight_layout()
plt.show()

# %%
plt.figure(figsize=(10,5))
plt.plot(returns.index, returns['AAPL'], label='Raw returns (AAPL)', alpha=0.7)
plt.plot(returns.index, returns['AAPL'].abs(), label='Absolute returns', alpha=0.7)
plt.legend()
plt.title("Volatility clustering in AAPL: raw vs absolute returns")
plt.ylabel("Return / |Return|")
plt.grid(True, alpha=0.3)
plt.show()

# %%
window = 20
rolling_vol = returns['AAPL'].rolling(window).std()

plt.figure(figsize=(10,4))
plt.plot(returns.index, rolling_vol, color='tab:red')
plt.title(f"AAPL rolling {window}-day volatility")
plt.ylabel("Rolling $\sigma$")
plt.grid(alpha=0.3)
plt.show()


# %% [markdown]
# ### 1.3 - Testing dependencies
```

```python

# %%
fig, axes = plt.subplots(1, 3, figsize=(12, 4))

for ax, asset in zip(axes, TICKERS):
    stats.probplot(returns[asset], dist="norm", plot=ax)
    ax.set_title(f"{asset} Q{Q plot vs Normal")
    ax.get_lines()[1].set_color('red')  # line of best fit

plt.tight_layout()
plt.show()

# %%
for asset in TICKERS:
    jb_stat, p_value = jarque_bera(returns[asset])
    print(f"{asset}: JB = {jb_stat:.3f}, p-value = {p_value:.5f}")
    if p_value < 0.05:
        print(" -> Reject normality (non-normal distribution)\n")
    else:
        print(" -> Fail to reject normality (consistent with normal)\n")

# %% [markdown]
# ## **Part 2 - First window modelling**

# %% [markdown]
# ### Implementation of the models

# %%
W = 252  # approx. one trading year

def select_window_and_losses(returns: pd.DataFrame, tickers=('AAPL','META','JPM'), W: int = 252) -> pd.DataFrame:
    """
    Take the first W observations of log-returns for the given tickers and return a DataFrame of losses (L = -R).
    This function keeps column names unchanged (tickers) and aligns dates.
    """
    sub = returns.loc[:, list(tickers)].iloc[:W].copy()
    losses = -sub  # right-tail risk
    losses.columns = list(tickers)
    return losses

losses_W = select_window_and_losses(returns)
losses_W.head()

# %%
# Statistical fits

@dataclass
class FitResult:
    model: str                  # 'historical' | 'gaussian' | 'student_t' | 'AR_GARCH'
    params: Dict[str, float]    # e.g., {'mu': ..., 'sigma': ...} or {'nu': ..., 'mu': ..., 'sigma': ...}

def fit_historical(losses: pd.Series) -> FitResult:
    return FitResult(model='historical', params={})

def fit_gaussian(losses: pd.Series) -> FitResult:
    mu = float(losses.mean())
    sigma = float(losses.std(ddof=1))
    return FitResult(model='gaussian', params={'mu': mu, 'sigma': sigma})

def fit_student_t(losses: pd.Series) -> FitResult:
    # scipy's t.fit returns (nu, loc, scale) via MLE. We do not fix parameters, to keep it general.
    nu, loc, scale = student_t.fit(losses) # <-- changed that
    return FitResult(model='student_t', params={'nu': float(nu), 'mu': float(loc), 'sigma': float(scale)})

def fit_AR_GARCH(losses_W: pd.DataFrame, p=(1, 1, 1)) -> Dict:
    # Necessary imports
    try:
        from statsmodels.tsa.ar_model import AutoReg
        from arch import arch_model
    except ImportError as err:
        print(f"Import failed in 'forecast_AR_GARCH: {err}.")

    # Build return dictionnary
    res = {col_name: {'mu': 0.0, 'sigma': 0.0} for col_name in losses_W.columns}

    # Perform for evey ticker
    for idx, ticker in enumerate(losses_W.columns):
        series = losses_W[ticker].values

        # Fit AR model and get the residuals
        ar_fitted = AutoReg(series, lags=p[idx], old_names=False).fit()
        residuals = ar_fitted.resid

```

```python
            # Fit the HARCH(1,1) to the residuals
            garch_fitted = arch_model(residuals, vol='Garch', p=1, q=1, dist='normal', rescale=False).fit(disp='off')

            # Store results
            res[ticker]['mu'] = ar_fitted.forecast(steps=1)[0]
            res[ticker]['sigma'] = np.sqrt(garch_fitted.forecast(horizon=1).variance.values[-1, 0])

    return res

def fit_AR_GARCH(losses_W: pd.DataFrame, p: Tuple[int, ...] = (1,1,1)) -> Dict[str, Dict[str, float]]:
    """
    Fit AR(p) + GARCH(1,1) per ticker.
    Returns dict with one-step-ahead forecasts for mu and sigma.

    KEY FIX: Manually normalize residuals for GARCH stability, then rescale forecast back.
    This avoids the incorrect double-scaling that rescale=True introduces.
    """
    try:
        from statsmodels.tsa.ar_model import AutoReg
        from arch import arch_model
    except ImportError as err:
        raise ImportError(f"Required packages not found: {err}")

    res = {col: {'mu': 0.0, 'sigma': 0.0} for col in losses_W.columns}
    for idx, ticker in enumerate(losses_W.columns):
        series = losses_W[ticker].values

        # Fit AR model
        ar_fitted = AutoReg(series, lags=p[idx], old_names=False).fit()
        mu_ahead = float(ar_fitted.forecast(steps=1)[0])
        residuals = ar_fitted.resid

        # Manually normalize residuals for numerical stability
        resid_std = np.std(residuals, ddof=1)
        resid_normalized = residuals / resid_std

        # Fit GARCH(1,1) on normalized residuals with rescale=False
        garch_fitted = arch_model(
            resid_normalized,
            vol='Garch',
            p=1,
            q=1,
            dist='normal',
            rescale=False  # We handle scaling ourselves
        ).fit(disp='off')

        # Get one-step-ahead variance forecast (on normalized scale)
        variance_normalized = garch_fitted.forecast(horizon=1).variance.values[-1, 0]

        # Rescale back to original residual scale
        # If X_normalized = X / std(X), then Var(X) = Var(X_normalized) * std(X)^2
        sigma_ahead = np.sqrt(variance_normalized) * resid_std

        # Store results
        res[ticker]['mu'] = mu_ahead
        res[ticker]['sigma'] = sigma_ahead

    return res


# %%
# Workarounds

def one_ticker_var_es_gaussian(mu: float, sigma: float, alpha: float) -> Tuple[float, float]:
    """Closed-form VaR/ES for Normal(mu, sigma). VaR_alpha is the alpha-quantile (right tail is risky)."""
    z = norm.ppf(alpha)
    var = mu + sigma * z
    es = mu + sigma * (norm.pdf(z) / (1.0 - alpha))
    return float(var), float(es)

def one_ticker_var_es_student_t(nu: float, mu: float, sigma: float, alpha: float) -> Tuple[float, float]:
    """Closed-form VaR/ES for Student-t(nu) with location mu and scale sigma.
    VaR = mu + sigma * t_{nu}^{-1}(alpha).
    ES formula exists for nu > 1. We add guards for numerical stability.
    """
    if nu <= 1.0:
        raise ValueError("ES is not finite for nu <= 1. Please refit or constrain nu > 1.")
    q = student_t.ppf(alpha, df=nu)  # standard t quantile
    var = mu + sigma * q
    pdf_q = student_t.pdf(q, df=nu)
    es = mu + sigma * ( (nu + q**2) / ((nu - 1.0) * (1.0 - alpha)) ) * pdf_q
    return float(var), float(es)
```

```python
def one_ticker_var_es_historical(losses: pd.Series, alpha: float) -> Tuple[float, float]:
    """Empirical VaR/ES at level alpha for a loss series (right tail)."""
    # VaR: alpha-quantile of losses
    var = float(np.quantile(losses, alpha, method='higher'))
    # ES: mean of tail losses above VaR
    tail = losses[losses >= var]
    if len(tail) == 0:
        es = var  # fallback: degenerate tail
    else:
        es = float(tail.mean())
    return var, es


# %%
# Computes VaR and ES fit / predicitons in every cases a-e for losses_W

def var_es_historical(losses_W: Dict, alpha: float=0.95) -> Tuple[Dict, Dict]:

    res = {col_name: {"VaR": 0.0, "ES": 0.0} for col_name in losses_W.columns}
    params = {}
    for ticker in losses_W.columns:
        one_ticker_losses = losses_W[ticker].values
        params[ticker] = FitResult(model='historical', params={"values": one_ticker_losses})
        VaR, ES = one_ticker_var_es_historical(losses=one_ticker_losses, alpha=alpha)

        res[ticker]["VaR"], res[ticker]["ES"] = VaR, ES

    return res, params

def var_es_gaussian(losses_W: Dict, alpha: float=0.95) -> Tuple[Dict, Dict]:

    res = {col_name: {"VaR": 0.0, "ES": 0.0} for col_name in losses_W.columns}
    params = {}
    for ticker in losses_W.columns:
        one_ticker_losses = losses_W[ticker].values

        one_ticker_fit = fit_gaussian(losses=one_ticker_losses)
        params[ticker] = one_ticker_fit
        VaR, ES = one_ticker_var_es_gaussian(mu=one_ticker_fit.params["mu"], sigma=one_ticker_fit.params["sigma"],
        ↪  alpha=alpha)

        res[ticker]["VaR"], res[ticker]["ES"] = VaR, ES

    return res, params

def var_es_student_t(losses_W: Dict, alpha: float=0.95) -> Tuple[Dict, Dict]:

    res = {col_name: {"VaR": 0.0, "ES": 0.0} for col_name in losses_W.columns}
    params = {}
    for ticker in losses_W.columns:
        one_ticker_losses = losses_W[ticker].values

        one_ticker_fit = fit_student_t(losses=one_ticker_losses)
        params[ticker] = one_ticker_fit
        VaR, ES = one_ticker_var_es_student_t(nu=one_ticker_fit.params["nu"], mu=one_ticker_fit.params["mu"],
        ↪  sigma=one_ticker_fit.params["sigma"], alpha=alpha)

        res[ticker]["VaR"], res[ticker]["ES"] = VaR, ES

    return res, params

def forecast_var_es_AR_GARCH(
    losses_W: pd.DataFrame,
    p: Tuple[int, ...] = (1,1,1),
    alpha: float = 0.95
) -> Tuple[Dict, Dict]:
    """
    One-step-ahead VaR and ES per ticker using AR-GARCH with Gaussian innovations.

    VaR and ES are computed on the standardized residuals scale and then
    adjusted back to the original loss scale via mu and sigma.
    """
    if not 0 < alpha < 1:
        raise ValueError(f"Alpha must be between 0 and 1. Got {alpha}")

    step_ahead_forecasts = fit_AR_GARCH(losses_W, p)
    ret = {}
    param = {}

    # Precompute quantile and density for efficiency
    z_alpha = norm.ppf(alpha)
    pdf_z = norm.pdf(z_alpha)

```

```
    for ticker, fcast in step_ahead_forecasts.items():
        mu_ahead = fcast['mu']
        sigma_ahead = fcast['sigma']

        # Store parameters
        param[ticker] = FitResult(
            model="AR_GARCH",
            params={"mu": mu_ahead, "sigma": sigma_ahead}
        )

        # Right-tail VaR and ES (losses = -returns, so we work with loss distribution)
        # VaR_alpha = mu + sigma * z_alpha
        # ES_alpha = mu + sigma * phi(z_alpha) / (1 - alpha)
        ret[ticker] = {
            'VaR': mu_ahead + sigma_ahead * z_alpha,
            'ES': mu_ahead + sigma_ahead * pdf_z / (1 - alpha)
        }

    return ret, param

def forecast_var_es_fhs(losses_W: pd.DataFrame,
                        p=(1,1,1),
                        n_boot_samples: int=1000,
                        alpha: float=0.95,
                        seed: int|None=None) -> tuple[Dict, Dict]:
    """
    Filtered Historical Simulation (FHS):
    1. Fit AR-GARCH to extract standardized residuals
    2. Bootstrap from standardized residuals
    3. Apply tomorrow's forecasted mu and sigma to generate scenarios
    """
    from statsmodels.tsa.ar_model import AutoReg
    from arch import arch_model

    res = {col_name: {"VaR": 0.0, "ES": 0.0} for col_name in losses_W.columns}
    param = {}

    for (idx, ticker) in enumerate(losses_W.columns):
        series = losses_W[ticker].values

        # Fit AR(p)
        ar_fitted = AutoReg(series, lags=p[idx], old_names=False).fit()
        residuals = ar_fitted.resid

        # Fit GARCH(1,1) with rescale=True for stability
        garch_fitted = arch_model(
            residuals,
            vol='Garch',
            p=1,
            q=1,
            dist='normal',
            rescale=True
        ).fit(disp='off')

        # Extract conditional volatility on original scale
        scale = garch_fitted.scale
        cond_vola_original = garch_fitted.conditional_volatility * scale

        # Standardized residuals
        std_residuals = residuals / cond_vola_original

        # Forecast conditional mean and volatility (one-step-ahead)
        mu_hat = ar_fitted.forecast(steps=1)[0]

        variance_rescaled = garch_fitted.forecast(horizon=1).variance.values[-1, 0]
        sigma_hat = np.sqrt(variance_rescaled) * scale

        # Bootstrap from standardized residuals
        if seed is None:
            rng = np.random.default_rng()
        else:
            rng = np.random.default_rng(seed=seed)

        draws = rng.choice(a=std_residuals, size=n_boot_samples, replace=True)

        # Simulated loss scenarios
        sim_loss = mu_hat + sigma_hat * draws

        param[ticker] = FitResult(model="fhs", params={"values": sim_loss})

        # VaR and ES forecasts
        var_forecast = np.quantile(a=sim_loss, q=alpha)
        res[ticker]["VaR"] = var_forecast
```

```
661             res[ticker]["ES"] = sim_loss[sim_loss >= var_forecast].mean()
662
663         return res, param
664
665
666     # %% [markdown]
667     # ### Visualisations of the models
668
669     # %%
670     def KDE_visualisations_hist_gauss_stud(alpha: float=0.95) -> None:
671
672         fig, ax = plt.subplots(2, 3, figsize=(20,6))
673         # Historical simulation
674         x = np.linspace(-0.07,0.07, 1000)
675         methods = ["Historical", "Gaussian", "Student-t"]
676
677
678         for (idx, method) in enumerate(methods):
679             for ticker in losses_W.columns:
680                 # title = ""
681                 if method == methods[0]:
682                     res, params = var_es_historical(losses_W, alpha)
683                     var, es = res[ticker]["VaR"], res[ticker]["ES"]
684                     title = "KDEs of PDFs estimated from Historical Simulation"
685                     kde = gaussian_kde(params[ticker].params["values"])
686
687                     ax[0, idx].plot(x, kde(x), label=ticker)
688
689                 elif method == methods[1]:
690                     res, param = var_es_gaussian(losses_W, alpha)
691                     title = "KDEs of PDFs estimated from Gaussian fit"
692                     loc = param[ticker].params["mu"]
693                     scale = param[ticker].params["sigma"]
694
695                     f = lambda x: norm.pdf(x, loc=loc, scale=scale)
696                     ax[0, idx].plot(x, f(x), label=ticker)
697
698                 elif method == methods[2]:
699                     res, params = var_es_student_t(losses_W, alpha)
700                     title = "KDEs of PDFs estimated from Student-t fit"
701                     loc = params[ticker].params["mu"]
702                     df = params[ticker].params["nu"]
703                     scale = params[ticker].params["sigma"]
704
705                     f = lambda x: student_t.pdf((x - loc)/scale, df=df)/scale
706                     ax[0, idx].plot(x, f(x), label=ticker)
707
708
709             ax[0, idx].set_title(title)
710             ax[0, idx].legend()
711
712         for (jdx, ticker) in enumerate(losses_W.columns):
713             for method in methods:
714                 if method == "Historical":
715                     res, params = var_es_historical(losses_W, alpha)
716                     kde = gaussian_kde(params[ticker].params["values"])
717                     f = lambda x: kde(x)
718                     label = "Historical"
719
720                 elif method == "Gaussian":
721                     res, param = var_es_gaussian(losses_W, alpha)
722                     loc = param[ticker].params["mu"]
723                     scale = param[ticker].params["sigma"]
724                     f = lambda x: norm.pdf(x, loc=loc, scale=scale)
725                     label = "Gaussian"
726
727                 elif method == "Student-t":
728                     res, params = var_es_student_t(losses_W, alpha)
729                     loc = params[ticker].params["mu"]
730                     df = params[ticker].params["nu"]
731                     scale = params[ticker].params["sigma"]
732                     f = lambda x: student_t.pdf((x - loc) / scale, df=df) / scale
733                     label = "Student-t"
734
735                 ax[1, jdx].plot(x, f(x), label=label)
736
737             ax[1, jdx].set_title(f"PDFs by method for {ticker}")
738             ax[1, jdx].legend()
739
740         fig.suptitle(f"PDF comparisons for Historical, Gaussian and Student-t for alpha={alpha}")
741         fig.tight_layout()
742
743     def plot_pdfs_AR_GARCH_vs_FHS(
```

```python
    losses_W: pd.DataFrame,
    p: tuple[int, ...] = (1, 1, 1),
    alpha: float = 0.95,
    n_boot_samples: int = 1_000,
    n_grid: int = 500,
    bins: int = 30,
) -> None:
    """
    Replicates the figure 'PDF Comparison: AR-GARCH vs FHS for alpha=0.95'
    using the functions:
        - forecast_var_es_fhs
        - forecast_var_es_AR_GARCH

    Parameters
    ----------
    losses_W : pd.DataFrame
        Window of losses (columns = tickers, rows = time).
    p : tuple[int, ...]
        AR orders per ticker for both AR-GARCH and FHS.
    alpha : float
        Confidence level used in the title (does not affect the pdf shapes).
    n_boot_samples : int
        Number of bootstrap draws for FHS.
    n_grid : int
        Number of grid points for evaluating pdfs.
    bins : int
        Number of histogram bins for the observed losses.
    """

    # --- 1. Fit models and obtain simulated losses / parameters ---
    fhs_res, fhs_param = forecast_var_es_fhs(
        losses_W=losses_W, p=p, n_boot_samples=n_boot_samples, alpha=alpha
    )
    arg_res, arg_param = forecast_var_es_AR_GARCH(
        losses_W=losses_W, p=p, alpha=alpha
    )

    tickers = list(losses_W.columns)
    colors_obs = ["tab:blue", "tab:orange", "tab:green"]  # for observed histograms
    colors_kde = ["C0", "C1", "C2"]                       # for FHS KDEs

    # --- 2. Prepare figure layout (2x3 grid, first axis spans two columns) ---
    fig = plt.figure(figsize=(14, 9))
    gs = gridspec.GridSpec(
        2, 3, width_ratios=[2.0, 1.0, 1.0], height_ratios=[1.0, 1.0]
    )

    # ----- TOP-LEFT: all assets, observed histos + FHS KDEs -----
    ax_all = fig.add_subplot(gs[0, 0:2])
    for ticker, c_hist, c_kde in zip(tickers, colors_obs, colors_kde):
        obs = losses_W[ticker].values
        fhs_sim = fhs_param[ticker].params["values"]

        # histogram of observed losses
        ax_all.hist(
            obs,
            bins=bins,
            density=True,
            alpha=0.35,
            color=c_hist,
            label=f"{ticker}, observed",
        )

        # KDE of FHS simulated losses
        kde = gaussian_kde(fhs_sim)
        x_grid = np.linspace(
            min(obs.min(), fhs_sim.min()),
            max(obs.max(), fhs_sim.max()),
            n_grid,
        )
        ax_all.plot(
            x_grid,
            kde(x_grid),
            color=c_kde,
            lw=2,
            label=f"{ticker} (FHS)",
        )

    ax_all.set_title("FHS Histograms + KDEs (All)")
    ax_all.set_xlabel("Loss")
    ax_all.set_ylabel("Density")
    ax_all.legend()

```

```
    # ----- helper: single-asset panel (AR-GARCH pdf vs FHS KDE) -----
    def _single_asset_ax(ax, ticker: str):
        obs = losses_W[ticker].values
        fhs_sim = fhs_param[ticker].params["values"]
        mu = arg_param[ticker].params["mu"]
        sigma = arg_param[ticker].params["sigma"]

        # Grid covering both observed and FHS ranges
        x_grid = np.linspace(
            min(obs.min(), fhs_sim.min()) * 1.1,
            max(obs.max(), fhs_sim.max()) * 1.1,
            n_grid,
        )

        # AR-GARCH pdf (Gaussian with mean mu and std sigma)
        pdf_arg = norm.pdf(x_grid, loc=mu, scale=sigma)

        # FHS KDE on simulated losses
        kde_fhs = gaussian_kde(fhs_sim)
        pdf_fhs = kde_fhs(x_grid)

        ax.plot(x_grid, pdf_arg, label="AR-GARCH", lw=2)
        ax.plot(x_grid, pdf_fhs, "--", label="FHS", lw=2)
        ax.set_xlabel("Loss")
        ax.set_ylabel("Density")
        ax.set_title(f"{ticker} - PDF Comparison")
        ax.legend()

    # ----- TOP-RIGHT: AAPL -----
    ax_aapl = fig.add_subplot(gs[0, 2])
    _single_asset_ax(ax_aapl, tickers[0])

    # ----- BOTTOM-LEFT: META -----
    ax_meta = fig.add_subplot(gs[1, 0])
    _single_asset_ax(ax_meta, tickers[1])

    # ----- BOTTOM-RIGHT: JPM -----
    ax_jpm = fig.add_subplot(gs[1, 2])
    _single_asset_ax(ax_jpm, tickers[2])

    # Global title
    fig.suptitle(
        r"PDF Comparison: AR-GARCH vs FHS for $\alpha=0.95$", fontsize=14
    )
    fig.tight_layout(rect=[0, 0.02, 1, 0.96])
    plt.show()

# %%
KDE_visualisations_hist_gauss_stud(alpha=0.95)

KDE_visualisations_hist_gauss_stud(alpha=0.99)

# %%
plot_pdfs_AR_GARCH_vs_FHS(
    losses_W=losses_W,
    p=(1,1,1),
    alpha=0.95,
    n_boot_samples=1000
)

plot_pdfs_AR_GARCH_vs_FHS(
    losses_W=losses_W,
    p=(1,1,1),
    alpha=0.95,
    n_boot_samples=1000
)

# %% [markdown]
# ## **Part 3 - Backtesting Value at Risk and Expected Shortfall**

# %% [markdown]
# ### Backtest functions

# %%

def backtest_computations(
    losses: pd.DataFrame,
    methods: list,
    alphas: list[float]
) -> tuple[pd.DataFrame, dict[float, pd.DataFrame]]:
    """
    Rolling-window VaR/ES forecasts for several methods.

```

```
    Returns
    -------
    df : DataFrame
        Columns: MultiIndex (Alpha, Ticker, Method, Metric) with
        Metric in {"VaR", "ES"}.
    var_df : dict[float, DataFrame]
        For each alpha, a DataFrame with MultiIndex columns (Ticker, Method)
        containing only the VaR time series.
    """
    tickers = list(losses.columns)   # <--- use actual columns (this is really important! if removed the code breaks for backtesting portfolio)

    # results[alpha][ticker][method][metric] = list of forecasts
    results = {
        alpha: {
            ticker: {method: {"VaR": [], "ES": []} for method in methods}
            for ticker in tickers
        }
        for alpha in alphas
    }

    T = losses.shape[0]

    for t in range(W, T):
        window = losses.iloc[t - W: t]

        for alpha in alphas:
            res_historical, _ = var_es_historical(losses_W=window, alpha=alpha)
            res_gaussian,  _  = var_es_gaussian(losses_W=window, alpha=alpha)
            res_student,  _   = var_es_student_t(losses_W=window, alpha=alpha)
            res_garch,    _   = forecast_var_es_AR_GARCH(losses_W=window, alpha=alpha)
            res_fhs,      _   = forecast_var_es_fhs(losses_W=window, alpha=alpha, p=(7, 7, 5))

            for ticker in tickers:
                # Historical
                results[alpha][ticker]["historical"]["VaR"].append(res_historical[ticker]["VaR"])
                results[alpha][ticker]["historical"]["ES"].append(res_historical[ticker]["ES"])

                # Gaussian
                results[alpha][ticker]["gaussian"]["VaR"].append(res_gaussian[ticker]["VaR"])
                results[alpha][ticker]["gaussian"]["ES"].append(res_gaussian[ticker]["ES"])

                # Student t
                results[alpha][ticker]["student"]["VaR"].append(res_student[ticker]["VaR"])
                results[alpha][ticker]["student"]["ES"].append(res_student[ticker]["ES"])

                # GARCH
                results[alpha][ticker]["garch"]["VaR"].append(res_garch[ticker]["VaR"])
                results[alpha][ticker]["garch"]["ES"].append(res_garch[ticker]["ES"])

                # FHS
                results[alpha][ticker]["fhs"]["VaR"].append(res_fhs[ticker]["VaR"])
                results[alpha][ticker]["fhs"]["ES"].append(res_fhs[ticker]["ES"])

    # Out-of-sample dates
    dates_oos = losses.index[W:]

    # --- Build var_df: per alpha, a (Ticker, Method) VaR DataFrame ---
    var_df: dict[float, pd.DataFrame] = {}

    for alpha in alphas:
        data_var = {}
        for ticker in tickers:
            for method in methods:
                data_var[(ticker, method)] = results[alpha][ticker][method]["VaR"]

        var_df_alpha = pd.DataFrame(data_var, index=dates_oos)
        var_df_alpha.columns = pd.MultiIndex.from_tuples(
            var_df_alpha.columns, names=["Ticker", "Method"]
        )
        var_df[alpha] = var_df_alpha

    # --- Build a \long" df with (Alpha, Ticker, Method, Metric) ---
    data = {}
    for alpha in alphas:
        for ticker in tickers:
            for method in methods:
                for metric in ("VaR", "ES"):
                    key = (alpha, ticker, method, metric)
                    data[key] = results[alpha][ticker][method][metric]

    df = pd.DataFrame(data, index=dates_oos)
    df.columns = pd.MultiIndex.from_tuples(
```

```
            df.columns, names=["Alpha", "Ticker", "Method", "Metric"]
        )

        return df, var_df


    # %%
    def backtest_var_plot(losses: pd.DataFrame, var_df: dict, alpha_plot: float, methods: list) -> None:
        """
        Plot realized losses vs VaR forecasts.

        Parameters:
        - losses: DataFrame with shape (dates, tickers) containing realized losses
        - var_df: Nested dict with structure {alpha: {ticker: {method: {'VaR': [values]}}}}
        - alpha_plot: Confidence level to plot (e.g., 0.95, 0.99)
        - methods: List of VaR methods to plot
        """
        dates_oos = losses.index[W:]  # Assuming W is defined globally
        tickers = list(losses.columns)

        for ticker in tickers:
            plt.figure(figsize=(14, 5))

            # Realized losses
            plt.plot(dates_oos, losses[ticker].loc[dates_oos], label="Realized loss", linewidth=2)

            # VaR forecasts per method
            for method in methods:
                var_values = var_df[alpha_plot][ticker][method]
                plt.plot(
                    dates_oos,
                    var_values,
                    label=f"VaR {int(alpha_plot*100)}% - {method}"
                )

            plt.title(f"{ticker} { VaR {int(alpha_plot*100)}% vs realized losses")
            plt.xlabel("Date")
            plt.ylabel("Loss")
            plt.legend()
            plt.tight_layout()
            plt.show()

    # %% [markdown]
    # ### Value at Risk backtest

    # %%
    methods = ["historical", "gaussian", "student", "garch", "fhs"]

    df, var_df_095 = backtest_computations(losses=losses, methods=methods, alphas=[0.95])
    df, var_df_099 = backtest_computations(losses=losses, methods=methods, alphas=[0.99])

    # %%
    backtest_var_plot(losses=losses, var_df=var_df_095, alpha_plot=0.95, methods=methods)
    print("="*80)
    backtest_var_plot(losses=losses, var_df=var_df_099, alpha_plot=0.99, methods=methods)

    # %% [markdown]
    # #### Kupiec POF test

    # %%
    def compute_exceptions(loss_series: pd.Series, var_series: pd.Series) -> pd.Series:
        """
        Both series should be aligned on the same index.
        Exception = 1 if loss_t > VaR_t (VaR is violated).
        """
        L, V = loss_series.align(var_series, join="inner")
        I = (L > V).astype(int)
        return I

    def kupiec_pof_test(exceptions: pd.Series, alpha: float) -> Dict[str, float]:
        """
        Kupiec Proportion-of-Failures test (unconditional coverage).
        """
        p0 = 1 - alpha
        I = exceptions.values
        T = len(I)
        N = I.sum()

        if T == 0:
            raise ValueError("No observations for POF test.")

        p_hat = N / T
```

```python
    # Guard against log(0)
    if p_hat in (0.0, 1.0):
        # Extreme case: LR becomes very large
        LR_pof = 2 * T
    else:
        logL0 = (T - N) * np.log(1 - p0) + N * np.log(p0)
        logL1 = (T - N) * np.log(1 - p_hat) + N * np.log(p_hat)
        LR_pof = -2 * (logL0 - logL1)

    p_value = 1 - chi2.cdf(LR_pof, df=1)

    return {
        "N": int(N),
        "T": int(T),
        "hit_rate": p_hat,
        "LR_pof": LR_pof,
        "p_pof": p_value
    }

def christoffersen_ind_cc_tests(exceptions: pd.Series, lr_pof: float) -> Dict[str, float]:
    """
    Christoffersen independence and conditional coverage tests.
    """
    I = exceptions.values.astype(int)
    if len(I) < 2:
        raise ValueError("Need at least 2 observations for independence test.")

    I_prev = I[:-1]
    I_curr = I[1:]

    N00 = np.sum((I_prev == 0) & (I_curr == 0))
    N01 = np.sum((I_prev == 0) & (I_curr == 1))
    N10 = np.sum((I_prev == 1) & (I_curr == 0))
    N11 = np.sum((I_prev == 1) & (I_curr == 1))

    N0dot = N00 + N01
    N1dot = N10 + N11
    Ndot0 = N00 + N10
    Ndot1 = N01 + N11

    if N0dot == 0:
        pi0_hat = 0.0
    else:
        pi0_hat = N01 / N0dot

    if N1dot == 0:
        pi1_hat = 0.0
    else:
        pi1_hat = N11 / N1dot

    p_hat = (N01 + N11) / (N00 + N01 + N10 + N11)

    def safe_log(x):
        return np.log(np.clip(x, 1e-12, 1 - 1e-12))

    # Likelihood under Markov alternative (H1)
    logL1 = (
        N00 * safe_log(1 - pi0_hat) +
        N01 * safe_log(pi0_hat) +
        N10 * safe_log(1 - pi1_hat) +
        N11 * safe_log(pi1_hat)
    )

    # Likelihood under iid Bernoulli(p_hat) (H0)
    logL0 = (
        Ndot0 * safe_log(1 - p_hat) +
        Ndot1 * safe_log(p_hat)
    )

    LR_ind = -2 * (logL0 - logL1)
    p_ind = 1 - chi2.cdf(LR_ind, df=1)

    LR_cc = lr_pof + LR_ind
    p_cc = 1 - chi2.cdf(LR_cc, df=2)

    return {
        "N00": int(N00),
        "N01": int(N01),
        "N10": int(N10),
        "N11": int(N11),
        "LR_ind": LR_ind,
        "p_ind": p_ind,
        "LR_cc": LR_cc,
```

```
            "p_cc": p_cc
        }


    # %%
    def run_var_backtests(
        losses: pd.DataFrame,
        var_df: Dict[float, dict],
        alphas: list[float],
        methods: list[str]
    ) -> pd.DataFrame:
        """
        Run VaR backtests with Kupiec POF and Christoffersen tests.

        Parameters:
        - losses: DataFrame with shape (dates, tickers) containing realized losses
        - var_df: Nested dict with structure {alpha: {ticker: {method: {'VaR': [values]}}}}
        - alphas: List of confidence levels to test
        - methods: List of VaR methods to test
        """

        rows = []
        tickers = list(losses.columns)

        for alpha in alphas:
            var_dict = var_df[alpha]  # Get the dict for this alpha

            for ticker in tickers:
                for method in methods:
                    # Extract VaR values for this combination
                    var_values = var_dict[ticker][method]

                    # Align losses with VaR forecast dates
                    # Trim losses to match length of VaR values
                    L = losses[ticker].iloc[-len(var_values):]
                    V = pd.Series(var_values, index=L.index)

                    # Exceptions (1 if loss exceeds VaR, 0 otherwise)
                    I = compute_exceptions(L, V)

                    # Kupiec POF test
                    kup = kupiec_pof_test(I, alpha=alpha)

                    # Christoffersen independence & conditional coverage tests
                    ch = christoffersen_ind_cc_tests(I, lr_pof=kup["LR_pof"])

                    row = {
                        "Ticker": ticker,
                        "Method": method,
                        "Alpha": alpha,
                        "N_exceptions": kup["N"],
                        "T": kup["T"],
                        "Hit_rate": kup["hit_rate"],
                        "LR_POF": kup["LR_pof"],
                        "p_POF": kup["p_pof"],
                        "LR_ind": ch["LR_ind"],
                        "p_ind": ch["p_ind"],
                        "LR_CC": ch["LR_cc"],
                        "p_CC": ch["p_cc"]
                    }
                    rows.append(row)

        return pd.DataFrame(rows)

    def run_es_backtest(losses: pd.DataFrame,
                        models: list[str],
                        alphas: list[float],
                        W: int = 252,
                        M: int = 1000):

        T = losses.shape[0]
        # print("T: ",T)
        # print("W:", W)
        # print("T-W+1: ", T-W+1)
        tickers = list(losses.columns)

        # Store VaR / ES / params
        res = {model: {str(alpha): {} for alpha in alphas} for model in models}

        # Store simulations: alpha -> model -> time -> {"L", "I"}
        simulations = {str(alpha): {model: {ticker: {} for ticker in tickers} for model in models} for alpha in alphas}

        Z_1 = {str(alpha): {model: {ticker: 0.0 for ticker in tickers} for model in models} for alpha in alphas}
```

```
    N = {str(alpha): {model: {ticker: 0 for ticker in tickers} for model in models} for alpha in alphas}

    for t in range(T - W + 1, T):
        print(t, end='\r')

        window_t = losses.iloc[t - W : t, :]

        for alpha in alphas:

            # --- Compute VaR/ES ---
            est_hist, par_hist = var_es_historical(losses_W=window_t, alpha=alpha)
            est_gauss, par_gauss = var_es_gaussian(losses_W=window_t, alpha=alpha)
            est_stud, par_stud = var_es_student_t(losses_W=window_t, alpha=alpha)
            est_garch, par_garch = forecast_var_es_AR_GARCH(losses_W=window_t, p=(7,7,5), alpha=alpha)
            est_fhs, par_fhs = forecast_var_es_fhs(losses_W=window_t, p=(7,7,5), n_boot_samples=M, alpha=alpha)

            model_results = {
                "historical": (est_hist, par_hist),
                "gaussian": (est_gauss, par_gauss),
                "student": (est_stud, par_stud),
                "garch": (est_garch, par_garch),
                "fhs": (est_fhs, par_fhs)
            }

            # --- Store VaR / ES / params ---
            for model_name, (est, par) in model_results.items():
                for ticker in tickers:
                    res[model_name][str(alpha)][ticker] = {
                        "VaR": est[ticker]["VaR"],
                        "ES": est[ticker]["ES"],
                        "params": par.get(ticker, par)
                    }

            # For now: run per ticker independently
            for ticker in tickers:

                VaR_hist = res["historical"][str(alpha)][ticker]["VaR"]
                VaR_gauss = res["gaussian"][str(alpha)][ticker]["VaR"]
                VaR_stud = res["student"][str(alpha)][ticker]["VaR"]
                VaR_garch = res["garch"][str(alpha)][ticker]["VaR"]
                VaR_fhs = res["fhs"][str(alpha)][ticker]["VaR"]

                # --- Historical simulation ---
                L_hist = np.random.choice(window_t[ticker], size=M, replace=True)
                I_hist = (L_hist > VaR_hist).astype(int)

                # --- Gaussian ---
                gauss_param = res["gaussian"][str(alpha)][ticker]["params"]
                L_gauss = norm.rvs(
                    loc=gauss_param.params["mu"],
                    scale=gauss_param.params["sigma"],
                    size=M
                )
                I_gauss = (L_gauss > VaR_gauss).astype(int)

                # --- Student t ---
                stud_param = res["student"][str(alpha)][ticker]["params"]
                L_stud = t_stud.rvs(
                    df=stud_param.params["nu"],
                    loc=stud_param.params["mu"],
                    scale=stud_param.params["sigma"],
                    size=M
                )
                I_stud = (L_stud > VaR_stud).astype(int)

                # --- GARCH ---
                garch_param = res["garch"][str(alpha)][ticker]["params"]
                L_garch = norm.rvs(
                    loc=garch_param.params["mu"],
                    scale=garch_param.params["sigma"],
                    size=M
                )
                print("mu_t: ", garch_param.params["mu"], " | sigma_t: ", garch_param.params["sigma"], end="\r")
                I_garch = (L_garch > VaR_garch).astype(int)

                # --- FHS ---
                fhs_param = res["fhs"][str(alpha)][ticker]["params"]
                L_fhs = fhs_param.params["values"][:M]
                I_fhs = (L_fhs > VaR_fhs).astype(int)

                # --- Store simulations ---
                simulations[str(alpha)]["historical"][ticker][t] = {"L": L_hist, "I": I_hist, "ES":
                ↪  res["historical"][str(alpha)][ticker]["ES"]}
```

```
                    simulations[str(alpha)]["gaussian"][ticker][t] = {"L": L_gauss, "I": I_gauss, "ES":
                    ↪  res["gaussian"][str(alpha)][ticker]["ES"]}
                    simulations[str(alpha)]["student"][ticker][t] = {"L": L_stud, "I": I_stud, "ES":
                    ↪  res["student"][str(alpha)][ticker]["ES"]}
                    simulations[str(alpha)]["garch"][ticker][t] = {"L": L_garch, "I": I_garch, "ES":
                    ↪  res["garch"][str(alpha)][ticker]["ES"]}
                    simulations[str(alpha)]["fhs"][ticker][t] = {"L": L_fhs, "I": I_fhs, "ES":
                    ↪  res["fhs"][str(alpha)][ticker]["ES"]}

                    # Compute "benchmark" Z1
                    for model in models:
                        I_t = int(losses[ticker].iloc[t] > res[model][str(alpha)][ticker]["VaR"])
                        L_t = losses[ticker].iloc[t]

                        Z_1[str(alpha)][model][ticker] += I_t * L_t / res[model][str(alpha)][ticker]["ES"]
                        N[str(alpha)][model][ticker]   += I_t


        N_m = {str(alpha): {model: {ticker: np.sum([simulations[str(alpha)][model][ticker][t]["I"] for t in range(T-W+1, T)],
        ↪  axis=0) for ticker in tickers} for model in models} for alpha in alphas}
        Z_1_m ={str(alpha): {model: {ticker: np.sum([simulations[str(alpha)][model][ticker][t]["I"] *
        ↪  simulations[str(alpha)][model][ticker][t]["L"] / simulations[str(alpha)][model][ticker][t]["ES"] for t in
        ↪  range(T-W+1, T)], axis=0) for ticker in tickers} for model in models} for alpha in alphas}

        for alpha in alphas:
            for model in models:
                for ticker in tickers:
                    Z_1[str(alpha)][model][ticker] /= N[str(alpha)][model][ticker]
                    # Z_1[str(alpha)][model][ticker] += -1 this does not affect the comparison since it's a constant shift

                    for m in range(M):
                        Z_1_m[str(alpha)][model][ticker][m] = Z_1_m[str(alpha)][model][ticker][m] /
                        ↪  N_m[str(alpha)][model][ticker][m] if N_m[str(alpha)][model][ticker][m] != 0 else -np.inf




        p = {str(alpha): {model: {ticker: {(1 / (M - np.sum(np.array([Z_1_m[str(alpha)][model][ticker] ==
        ↪  -np.inf]).astype(int)))) * np.sum(np.array([Z_1_m[str(alpha)][model][ticker][m] >= Z_1[str(alpha)][model][ticker] for
        ↪  m in range(M)]).astype(int))} for ticker in tickers} for model in models} for alpha in alphas}

        # print("Nm: ", N_m)
        # print("Z_1_m: ", Z_1_m)

        # Compute "benchmark" Z1
        # print("Z_1: ", Z_1)
        # print("N: ", N)


        return p, N_m, Z_1_m, Z_1, simulations, res

    # %%
    def plot_es_p_values(p: pd.DataFrame) -> None:
        rows = []
        for alpha, models in p.items():
            for model, tickers in models.items():
                for ticker_name, val_set in tickers.items():
                    # val_set is a set with a single float
                    value = list(val_set)[0]  # extract the float
                    rows.append({'alpha': alpha, 'model': model, 'ticker': ticker_name, 'value': value})

        df = pd.DataFrame(rows)

        # Pivot for plotting
        pivot_df = df.pivot_table(index='ticker', columns=['alpha', 'model'], values='value')

        # Plot as grouped bar chart
        pivot_df.plot(kind='bar', figsize=(10,6))
        plt.ylabel('Value')
        plt.title('p values by Model and Ticker')
        plt.xticks(rotation=0)
        plt.legend(title='Alpha, Model', bbox_to_anchor=(1.05,1), loc='upper left')
        plt.tight_layout()
        plt.show()

    # %%
    var_backtests_095 = run_var_backtests(losses, var_df_095, alphas=[0.95], methods=methods)
    var_backtests_095.sort_values(["Ticker", "Alpha", "Method"])

    # %%
    var_backtests_099 = run_var_backtests(losses, var_df_099, alphas=[0.99], methods=methods)
    var_backtests_099.sort_values(["Ticker", "Alpha", "Method"])

```

```
# %% [markdown]
# ### Expected Shortfall backtest

# %%
p, _, _, _, _, _ = run_es_backtest(losses, alphas=[0.95, 0.99], models=["historical", "gaussian", "student", "garch", "fhs"])

# %%
plot_es_p_values(p=p)

# %%
rows = []
for alpha, models in p.items():
    for model, tickers in models.items():
        for ticker_name, val_set in tickers.items():
            # val_set is a set with a single float
            value = list(val_set)[0]  # extract the float
            rows.append({'alpha': alpha, 'model': model, 'ticker': ticker_name, 'value': value})

df = pd.DataFrame(rows)

# Pivot for plotting
pivot_df = df.pivot_table(index='ticker', columns=['alpha', 'model'], values='value')

# Plot as grouped bar chart
pivot_df.plot(kind='bar', figsize=(10,6))
plt.ylabel('Value')
plt.title('p values by Model and Ticker')
plt.xticks(rotation=0)
plt.legend(title='Alpha, Model', bbox_to_anchor=(1.05,1), loc='upper left')
plt.tight_layout()
plt.show()

# %% [markdown]
# ### Performance ranking

# %%
def ranking_old(p_es: pd.DataFrame, var_099: pd.DataFrame, var_095: pd.DataFrame, models: list=["historical", "garch",
↪  "student", "gaussian", "fhs"], alphas: list=[0.95, 0.99], tickers=["AAPL", "META", "JPM"]) -> None:
    # es_p_averages = ["alpha": {np.sum(p_es["alpha"], axis=0) for alpha in ["0.95", "0.99"]}]
    avg = {str(alpha): {model: {ticker: 0.0 for ticker in tickers} for model in models} for alpha in alphas}
    for alpha in alphas:
        for model in models:
            for ticker in tickers:
                if alpha == 0.95:

                    val_p_POF = var_095.loc[
                            (var_095["Ticker"] == "AAPL") &
                            (var_095["Alpha"] == 0.95) &
                            (var_095["Method"] == "garch"),
                            "p_POF"
                        ].iloc[0]
                    val_p_CC = var_095.loc[
                            (var_095["Ticker"] == "AAPL") &
                            (var_095["Alpha"] == 0.95) &
                            (var_095["Method"] == "garch"),
                            "p_CC"
                        ].iloc[0]
                    avg[str(alpha)][model][ticker] = (p_es[str(alpha)][model][ticker] + val_p_POF + val_p_CC) / 3

                elif alpha == 0.99:
                    val_p_POF = var_099.loc[
                            (var_099["Ticker"] == "AAPL") &
                            (var_099["Alpha"] == 0.99) &
                            (var_099["Method"] == "garch"),
                            "p_POF"
                        ].iloc[0]
                    val_p_CC = var_099.loc[
                            (var_099["Ticker"] == "AAPL") &
                            (var_099["Alpha"] == 0.99) &
                            (var_099["Method"] == "garch"),
                            "p_CC"
                        ].iloc[0]
                    avg[str(alpha)][model][ticker] = (p_es[str(alpha)][model][ticker] + val_p_POF + val_p_CC) / 3

    return avg

# %%
def ranking(p_es, var_099, var_095,
            models=["historical", "garch", "student", "gaussian", "fhs"],
            alphas=[0.95, 0.99],
            tickers=["AAPL", "META", "JPM"]):

    avg = {
```

```
        str(alpha): {
            model: {ticker: 0.0 for ticker in tickers}
            for model in models
        }
        for alpha in alphas
    }

    for alpha in alphas:
        for model in models:
            for ticker in tickers:

                # extract float from set
                p_es_val = next(iter(p_es[str(alpha)][model][ticker]))

                if alpha == 0.95:
                    df = var_095
                else:
                    df = var_099

                val_p_POF = df.loc[
                    (df["Ticker"] == ticker) &
                    (df["Alpha"] == alpha) &
                    (df["Method"] == model),
                    "p_POF"
                ].iloc[0]

                val_p_CC = df.loc[
                    (df["Ticker"] == ticker) &
                    (df["Alpha"] == alpha) &
                    (df["Method"] == model),
                    "p_CC"
                ].iloc[0]

                avg[str(alpha)][model][ticker] = (
                    p_es_val + val_p_POF + val_p_CC
                ) / 3

    return avg


# %%
r=ranking(p_es=p, var_095=var_backtests_095, var_099=var_backtests_099)
r

# %% [markdown]
# ## **Part 4 - Copula fitting**

# %% [markdown]
# ### Copula functions

# %%

def compute_U_t_i(losses_W: pd.DataFrame) -> pd.DataFrame:
    """
    Convert losses to uniform marginals using empirical CDF.

    Args:
        losses_W: DataFrame of losses with shape (W, n_assets)

    Returns:
        DataFrame of uniform marginals U_t_i in [0,1]
    """
    returns = -losses_W

    W = returns.shape[0]
    ranks = np.argsort(np.argsort(returns, axis=0), axis=0) + 1
    U_t_i = ranks / (W + 1)
    U_t_i = pd.DataFrame(U_t_i, columns=losses_W.columns, index=losses_W.index)
    return U_t_i, ranks

def fit_copula(U_data: pd.DataFrame, copula_type: str = "Gaussian"):
    """
    Fit a copula to uniform marginal data.

    Args:
        U_data: DataFrame of uniform marginals in [0,1]
        copula_type: Type of copula - "Gaussian", "Archimedean", or "Student"

    Returns:
        Tuple of (fitted copula object, parameters)
    """

    #if U_data.empty:
```

```
    #    raise ValueError("U_data cannot be empty")

    u = U_data.to_numpy()

    if copula_type == "Gaussian":
        copula = copulae.elliptical.GaussianCopula(dim=U_data.shape[1])
    elif copula_type == "Archimedean":
        from copulae.archimedean import ClaytonCopula
        copula = ClaytonCopula(dim=U_data.shape[1])
    elif copula_type == "Student":
        copula = copulae.elliptical.StudentCopula(dim=U_data.shape[1])
    else:
        raise ValueError(f"Unknown copula type: {copula_type}. Choose from 'Gaussian', 'Archimedean', or 'Student'")


    copula.fit(u)
    params = copula.params

    return copula, params

def generate_from_copula(U_data: pd.DataFrame, copula, number_of_samples: int = 1000) -> pd.DataFrame:
    """
    Generate random samples from a fitted copula.

    Args:
        U_data: Original DataFrame (used to get column names)
        copula: Fitted copula object
        number_of_samples: Number of samples to generate

    Returns:
        DataFrame of generated uniform samples with shape (number_of_samples, n_assets)
    """
    columns = U_data.columns
    samples = copula.random(number_of_samples)
    return pd.DataFrame(samples, columns=columns)

def inverse_transform(U_samples: pd.DataFrame, losses_W: pd.DataFrame) -> pd.DataFrame:
    """
    Convert uniform copula samples back to returns using the empirical distribution
    of the original losses (or returns).

    Args:
        U_samples: DataFrame of uniform samples from copula in [0,1]
        portfolio_losses: Original DataFrame of losses used to build empirical distribution

    Returns:
        DataFrame of recovered returns/losses with same columns as U_samples
    """
    returns_W = -losses_W
    recovered = pd.DataFrame(index=U_samples.index, columns=U_samples.columns)

    for col in U_samples.columns:
        sorted_vals = np.sort(returns_W[col].values)
        W = len(sorted_vals)

        # Map U to empirical quantile with linear interpolation
        # U=0 maps to smallest value, U=1 maps to largest value
        idx = U_samples[col].values * (W - 1)
        idx_lower = np.floor(idx).astype(int).clip(0, W - 1)
        idx_upper = np.minimum(idx_lower + 1, W - 1)
        weight = idx - idx_lower

        # Linear interpolation between adjacent empirical quantiles
        recovered[col] = (
            sorted_vals[idx_lower] * (1 - weight) +
            sorted_vals[idx_upper] * weight
        )

    return recovered

def plot_observed_copulas_results(losses_W: pd.DataFrame) -> None:
    U, _ = compute_U_t_i(losses_W)

    R = pd.DataFrame(-losses_W, columns=losses_W.columns)
    combinations = [("AAPL", "META"), ("AAPL", "JPM"), ("JPM", "META")]

    fig, ax = plt.subplots(3, 2, figsize=(20, 9))

    for (idx, comb) in enumerate(combinations):
        ticker1, ticker2 = comb
        ax[idx, 0].scatter(U[ticker1], U[ticker2], label=f"U[{ticker1}] vs U[{ticker2}]")
        ax[idx, 1].scatter(R[ticker1], R[ticker2], label=f"R[{ticker1}] vs R[{ticker2}]")
        ax[idx, 0].legend()
```

```
        ax[idx, 1].legend()

def plot_simulated_copulas_results() -> None:
    U, _ = compute_U_t_i(losses_W)
    cop_gaussian, par = fit_copula(U, "Gaussian")
    samples_gaussian = generate_from_copula(U, cop_gaussian, 252)

    cop_student, par = fit_copula(U, "Student")
    samples_student = generate_from_copula(U, cop_gaussian, 252)


    R_gaussian = inverse_transform(samples_gaussian, losses_W)
    R_student = inverse_transform(samples_student, losses_W)

    U_gaussian, _= compute_U_t_i(samples_gaussian)
    U_student, _ = compute_U_t_i(samples_student)

    combinations = [("AAPL", "META"), ("AAPL", "JPM"), ("JPM", "META")]
    R = pd.DataFrame(-losses_W, columns=losses_W.columns)

    fig, ax = plt.subplots(3, 2, figsize=(20, 9))
    for (idx, comb) in enumerate(combinations):
        ticker1, ticker2 = comb
        ax[idx, 0].scatter(U_gaussian[ticker1], U_gaussian[ticker2],
                           label=f"U[{ticker1}] vs U[{ticker2}] (Gaussian - generated)",
                           marker='o',
                           facecolors='none',
                           edgecolors='red')
        ax[idx, 1].scatter(R_gaussian[ticker1], R_gaussian[ticker2],
                           label=f"R[{ticker1}] vs R[{ticker2}] (Gaussian - generated)",
                           marker='o',
                           facecolors='none',
                           edgecolors='green')
        ax[idx, 0].scatter(U_student[ticker1], U_student[ticker2],
                           label=f"U[{ticker1}] vs U[{ticker2}] (Student - generated)",
                           marker='o',
                           facecolors='none',
                           edgecolors='purple')
        ax[idx, 1].scatter(R_student[ticker1], R_student[ticker2],
                           label=f"R[{ticker1}] vs R[{ticker2}] (Student - generated)",
                           marker='o',
                           facecolors='none',
                           edgecolors='orange')
        ax[idx, 1].scatter(-losses_W[ticker1], -losses_W[ticker2], label=f"R[{ticker1}] vs R[{ticker2}] (Observed)",
                           marker='+')

        ax[idx, 0].set_title(r"$(U_{t,i}, U_{t, j})_t$" + f" for i={ticker1} & j={ticker2}")
        ax[idx, 1].set_title(r"$(R_{t,i}, R_{t, j})_t$" + f" for i={ticker1} & j={ticker2}")
        ax[idx, 0].legend()
        ax[idx, 1].legend()
    fig.suptitle("Scatter plots, simulated and observed")
    fig.tight_layout()

    fig, ax = plt.subplots(3, 1, figsize=(20,9))
    x = np.arange(R.shape[0])
    for (idx, ticker) in enumerate(R.columns):
        ax[idx].plot(x, R_gaussian[ticker],
                     label=f"Simulated returns - {ticker} (Gaussian)")
        ax[idx].plot(x, R_student[ticker],
                     label=f"Simulated returns - {ticker} (Student)")
        ax[idx].plot(x, R[ticker],
                     label=f"Observed returns - {ticker}")

        ax[idx].set_title(f"Returns - {ticker}")
        ax[idx].legend()
    fig.suptitle("Returns, simulated and observed")
    fig.tight_layout()

    fig, ax = plt.subplots(1, 3, figsize=(20, 6))
    for (idx, ticker) in enumerate(R.columns):
        ax[idx].hist(R_gaussian[ticker],
                     label=f"Simulated returns - {ticker} (Gaussian)",
                     density=True,
                     bins='scott',
                     alpha=0.5)
        ax[idx].hist(R_student[ticker],
                     label=f"Simulated returns - {ticker} (Student)",
                     density=True,
                     bins='scott',
                     alpha=0.5)
        ax[idx].hist(R[ticker],
                     label=f"Observed returns - {ticker}",
                     density=True,
```

```
                    bins='scott',
                    alpha=0.5)

        ax[idx].set_title(f"Histograms of returns - {ticker}")
        ax[idx].legend()
    fig.suptitle("Histograms of returns, observed and simulated")
    fig.tight_layout()

# %% [markdown]
# ### Observed data

# %%
plot_observed_copulas_results(losses_W)

# %% [markdown]
# ### Gaussian and Student simulated copulas

# %%
plot_simulated_copulas_results()

# %% [markdown]
# ### Additional Diagnosis

# %%
U, ranks = compute_U_t_i(losses_W)
n_observations = U.shape[0]

# %%
def spearman_correlation(ranks: pd.array) -> pd.DataFrame:
    """
    Compute Spearman rank correlation matrix from ranks DataFrame.

    Args:
        ranks: DataFrame of ranks with shape (n_observations, n_assets)
    Returns:
        DataFrame of Spearman correlation matrix
    """
    # Convert to DataFrame only for convenience with labels
    if not isinstance(ranks, pd.DataFrame):
        ranks = pd.DataFrame(ranks)

    n, k = ranks.shape
    cols = ranks.columns

    corr = pd.DataFrame(np.eye(k), index=cols, columns=cols)
    values = ranks.to_numpy()

    for i in range(k):
        for j in range(i + 1, k):
            x = values[:, i]
            y = values[:, j]

            # Check for ties
            no_ties = (np.unique(x).size == n) and (np.unique(y).size == n)

            if no_ties:
                d = x - y
                rho = 1 - 6 * np.sum(d ** 2) / (n * (n**2 - 1))
            else:
                # rank transform and Pearson
                rx = pd.Series(x).rank().to_numpy()
                ry = pd.Series(y).rank().to_numpy()
                rho = np.corrcoef(rx, ry)[0, 1]

            corr.iat[i, j] = rho
            corr.iat[j, i] = rho

    return corr

def pvalues_spearman(corr: pd.DataFrame, n: int) -> pd.DataFrame:
    """
    Compute approximate p-values for Spearman correlation coefficients
    using a t-distribution approximation valid for large samples.
    """

    # avoid division by zero when corr = ±1
    corr_clipped = corr.clip(-0.999999, 0.999999)

    # t-statistic approximation
    t_stat = corr_clipped * np.sqrt((n - 2) / (1 - corr_clipped ** 2))

    # p-values (numpy array)
    p_array = 2 * (1 - student_t.cdf(np.abs(t_stat), df=n - 2))
```

```
    # convert to DataFrame
    p_vals = pd.DataFrame(p_array, index=corr.index, columns=corr.columns)

    # set diagonal to 0
    np.fill_diagonal(p_vals.values, 0.0)

    return p_vals

# %%
corr = spearman_correlation(ranks)
p_vals_spearman = pvalues_spearman(corr, n_observations)
corr, p_vals_spearman

# %%
def kendall_tau_b(x, y):
    """Kendall's tau-b with two-tailed p-value (supports ties)."""
    n = len(x)
    num_concord = num_discord = 0
    tie_x = tie_y = 0

    for i in range(n - 1):
        for j in range(i + 1, n):
            dx = x[i] - x[j]
            dy = y[i] - y[j]

            if dx == 0 and dy == 0:
                tie_x += 1
                tie_y += 1
            elif dx == 0:
                tie_x += 1
            elif dy == 0:
                tie_y += 1
            else:
                if dx * dy > 0:
                    num_concord += 1
                else:
                    num_discord += 1

    numerator = num_concord - num_discord
    denom_x = math.sqrt((n * (n - 1) / 2) - tie_x)
    denom_y = math.sqrt((n * (n - 1) / 2) - tie_y)
    tau_b = numerator / (denom_x * denom_y)

    var_tau = (4 * n + 10) / (9 * n * (n - 1))
    z = tau_b / math.sqrt(var_tau)

    p_value = 2 * (1 - normal_cdf(abs(z)))
    return tau_b, p_value

def normal_cdf(z):
    return 0.5 * (1 + math.erf(z / math.sqrt(2)))

def kendall_matrix(arr):
    """Compute tau-b and p-values for all column pairs of a 2D array."""
    arr = np.asarray(arr)
    k = arr.shape[1]

    tau_mat = np.zeros((k, k))
    p_mat = np.zeros((k, k))

    for i in range(k):
        for j in range(k):
            if i == j:
                tau_mat[i, j] = 1.0
                p_mat[i, j] = 0.0
            else:
                tau, p = kendall_tau_b(arr[:, i], arr[:, j])
                tau_mat[i, j] = tau
                p_mat[i, j] = p

    return tau_mat, p_mat


# %%
tau_matrix, p_matrix = kendall_matrix(ranks)

print("Kendall Tau-b matrix:\n", tau_matrix)
print("\nP-value matrix:\n", p_matrix)

# %% [markdown]
# ## **Part 5 - Backtesting Value at Risk and Expected Shortfall for portfolio**
```

```python
# %% [markdown]
# ### Portfolio functions

# %%
portfolio_losses = 1/3*losses["AAPL"] + 1/3*losses["JPM"] + 1/3*losses["META"]


# If losses has a Date index
# portfolio_df = pd.DataFrame({
#     "Date": losses.index,
#     "portfolio": portfolio_losses.values
# })
portfolio_df = pd.DataFrame({
    "portfolio": portfolio_losses.values
}, index=losses.index)

print(portfolio_df)

# %%
def var_es_copula(losses_W: Dict, alpha: float=0.95, copula_type="Gaussian") -> Tuple[Dict, Dict]:

    res = {col_name: {"VaR": 0.0, "ES": 0.0} for col_name in losses_W.columns}

    U_data, _ = compute_U_t_i(losses_W=losses_W)
    copula, _ = fit_copula(U_data=U_data, copula_type=copula_type)

    generated_U = generate_from_copula(U_data=U_data, copula=copula, number_of_samples=1000)

    generated_returns = inverse_transform(U_samples=generated_U, losses_W=losses_W)

    portfolio_losses = -(1/3*generated_returns["AAPL"] + 1/3*generated_returns["META"] + 1/3*generated_returns["JPM"])

    # print(portfolio_losses)
    tmp_losses = pd.DataFrame({
    "portfolio": portfolio_losses.values
    })


    res, param = var_es_historical(losses_W=tmp_losses, alpha=alpha)

    return res, param

def portfolio_backtest(portfolio_df: pd.DataFrame, losses: pd.DataFrame, alpha: float, methods: pd.DataFrame=["historical",
↪  "gaussian", "student", "garch", "fhs", "gaussian_copula", "student_copula"]):
    # methods = ["historical", "gaussian", "student", "garch", "fhs", "gaussian_copula", "student_copula"]
    # alpha = 0.99
    #TICKERS = ["portfolio"]
    results = {"portfolio": { method: {"VaR": [], "ES": []} # results[ticker][method][metric] is a list to append.
                        for method in methods
                        }
                }

    for t in range(W, losses.shape[0]):

        window = portfolio_df.iloc[t - W: t][["portfolio"]]
        # print(window.columns)
        window_all_tickers = losses.iloc[t - W: t]


        res_historical, _ = var_es_historical(losses_W=window, alpha=alpha)
        res_gaussian, _ = var_es_gaussian(losses_W=window, alpha=alpha)
        res_student, _ = var_es_student_t(losses_W=window, alpha=alpha)
        res_garch, _ = forecast_var_es_AR_GARCH(losses_W=window, alpha=alpha, p=(7,7, 5))
        res_fhs, _ = forecast_var_es_fhs(losses_W=window, alpha=alpha, p=(7, 7, 5))

        res_g_copula, _ = var_es_copula(losses_W=window_all_tickers, alpha=alpha, copula_type="Gaussian");
        res_s_copula, _ = var_es_copula(losses_W=window_all_tickers, alpha=alpha, copula_type="Student");


        results["portfolio"]["historical"]["VaR"].append(res_historical["portfolio"]["VaR"])
        results["portfolio"]["historical"]["ES"].append(res_historical["portfolio"]["ES"])

        results["portfolio"]["gaussian"]["VaR"].append(res_gaussian["portfolio"]["VaR"])
        results["portfolio"]["gaussian"]["ES"].append(res_gaussian["portfolio"]["ES"])

        results["portfolio"]["student"]["VaR"].append(res_student["portfolio"]["VaR"])
        results["portfolio"]["student"]["ES"].append(res_student["portfolio"]["ES"])

        results["portfolio"]["garch"]["VaR"].append(res_garch["portfolio"]["VaR"])
        results["portfolio"]["garch"]["ES"].append(res_garch["portfolio"]["ES"])

        results["portfolio"]["fhs"]["VaR"].append(res_fhs["portfolio"]["VaR"])
        results["portfolio"]["fhs"]["ES"].append(res_fhs["portfolio"]["ES"])
```

```
        results["portfolio"]["gaussian_copula"]["VaR"].append(res_g_copula["portfolio"]["VaR"])
        results["portfolio"]["gaussian_copula"]["ES"].append(res_g_copula["portfolio"]["ES"])

        results["portfolio"]["student_copula"]["VaR"].append(res_s_copula["portfolio"]["VaR"])
        results["portfolio"]["student_copula"]["ES"].append(res_s_copula["portfolio"]["ES"])

    return results

def plot_var_visualisation(results: Dict, alpha: float=0.95) -> None:
    data = {}

    for ticker, ticker_data in results.items():
        for method, method_data in ticker_data.items():
            for metric, values in method_data.items():
                key = (ticker, method, metric)
                data[key] = values

    df = pd.DataFrame(data)

    # nice multi-level columns
    df.columns = pd.MultiIndex.from_tuples(
        df.columns,
        names=["Ticker", "Method", "Metric"]
    )

    plt.figure(figsize=(20, 6))

    y = portfolio_df["portfolio"][W:]
    x = np.arange(len(y))

    plt.plot(x, losses["META"][W:])

    plt.plot(x, df["portfolio"]["student"]["VaR"], label="Student")
    plt.plot(x, df["portfolio"]["gaussian"]["VaR"], label="gaussian")
    plt.plot(x, df["portfolio"]["historical"]["VaR"], label="historical")
    plt.plot(x, df["portfolio"]["fhs"]["VaR"], label="fhs")
    plt.plot(x, df["portfolio"]["garch"]["VaR"], label="garch")
    plt.plot(x, df["portfolio"]["gaussian_copula"]["VaR"], label="Gaussian copula")
    plt.plot(x, df["portfolio"]["student_copula"]["VaR"], label="Student copula")
    plt.title(f"Portfolio VaR forecasts for alpha={alpha}")
    plt.legend()

# %%
res_099 = portfolio_backtest(portfolio_df=portfolio_df, losses=losses, alpha=0.99);

res_095 = portfolio_backtest(portfolio_df=portfolio_df, losses=losses, alpha=0.95);

# %% [markdown]
# ### Univariate models - backtesting

# %%
plot_var_visualisation(res_095, 0.95)

plot_var_visualisation(res_099, 0.99)

# %% [markdown]
# ##### Value at Risk backtest

# %%
methods = ["historical", "gaussian", "student", "garch", "fhs"]

df, var_df_095 = backtest_computations(losses=portfolio_df, methods=methods, alphas=[0.95])
df, var_df_099 = backtest_computations(losses=portfolio_df, methods=methods, alphas=[0.99])

# %%
var_backtests_095 = run_var_backtests(portfolio_df, var_df_095, alphas=[0.95], methods=methods)
var_backtests_095.sort_values(["Ticker", "Alpha", "Method"])

# %%
var_backtests_099 = run_var_backtests(portfolio_df, var_df_099, alphas=[0.99], methods=methods)
var_backtests_099.sort_values(["Ticker", "Alpha", "Method"])

# %% [markdown]
# ##### Expected Shortfall backtest

# %%
p_es_portfolio, _, _, _, _, _ = run_es_backtest(portfolio_df, models=methods, alphas=[0.95, 0.99])
p_es_portfolio

# %%
plot_es_p_values(p=p_es_portfolio)
```

```
# %%
r_portfolio = ranking(p_es_portfolio, var_backtests_099, var_backtests_095, tickers=["portfolio"])
r_portfolio

# %% [markdown]
# ### Copulas approach

# %%
plt.figure(figsize=(20,6))
plt.plot(portfolio_losses[W:])
x = portfolio_losses.index[W:]

plt.plot(x, res_095["portfolio"]["student_copula"]["VaR"], label="Student copula - 0.95")
plt.plot(x, res_095["portfolio"]["gaussian_copula"]["VaR"], label="Gaussian copula - 0.95")

plt.plot(x, res_099["portfolio"]["student_copula"]["VaR"], label="Student copula - 0.99")
plt.plot(x, res_099["portfolio"]["gaussian_copula"]["VaR"], label="Gaussian copula - 0.99")

plt.title("Copulas")
plt.legend()

# %% [markdown]
# ### Backtesting comparisons

# %%
plt.figure(figsize=(20,6))
plt.plot(portfolio_losses[W:])
x = portfolio_losses.index[W:]

plt.plot(x, res_095["portfolio"]["student_copula"]["VaR"], label="Student copula - 0.95")
plt.plot(x, res_095["portfolio"]["gaussian_copula"]["VaR"], label="Gaussian copula - 0.95")

plt.plot(x, res_099["portfolio"]["student_copula"]["VaR"], label="Student copula - 0.99")
plt.plot(x, res_099["portfolio"]["gaussian_copula"]["VaR"], label="Gaussian copula - 0.99")

plt.plot(x, res_099["portfolio"]["historical"]["VaR"], label="Historical - 0.99")
plt.plot(x, res_095["portfolio"]["historical"]["VaR"], label="Historical - 0.95")

plt.plot(x, res_099["portfolio"]["gaussian"]["VaR"], label="Gaussian - 0.99")
plt.plot(x, res_095["portfolio"]["gaussian"]["VaR"], label="Gaussian - 0.95")

plt.plot(x, res_099["portfolio"]["fhs"]["VaR"], label="FHS - 0.99")
plt.plot(x, res_095["portfolio"]["fhs"]["VaR"], label="FHS - 0.95")
plt.title("Comparison")
plt.legend()
```