IMPERIAL

Quantitative Risk Management Coursework 1

Imperial College London

Yve Meng (06060863)

Harry Qiao (02057393)

Yibo Wang (02199425)

Ganwen Zuo (02222591)

MSc Mathematics & Finance

November 20th 2025

Contents

1 Part A: Stylised Facts and GARCH Modelling

1.1 Introduction

This report analyses the daily log returns of the EURO STOXX 50 index (Bloomberg, 2025) with the aim of identifying key stylised facts, modelling log returns using GARCH processes, and evaluating the adequacy of different conditional mean and innovation specifications.

The dataset consists of daily closing values of the EURO STOXX 50 (ticker: SX5E) from 30 October 2013 to 27 October 2023. The EURO STOXX 50 is a benchmark index tracking the fifty largest and most liquid blue-chip companies in the Eurozone, providing a representative measure of regional economic conditions (Bloomberg, 2025).

With one observation per trading day, the dataset contains 2,512 data points. Our analysis uses logarithmic daily returns, which are more appropriate for time-series modelling and volatility analysis. Let S_t denote the closing value on day t; the log-return (in percentage) is computed as:

$$r_t := (\log(S_t) - \log(S_{t-1})) \times 100. \tag{1}$$

1.2 Stylised Facts

To characterise the distributional properties of the log returns, we compute their sample mean, standard deviation, skewness, and kurtosis.

Table 1:
Summary statistics of log returns (%)

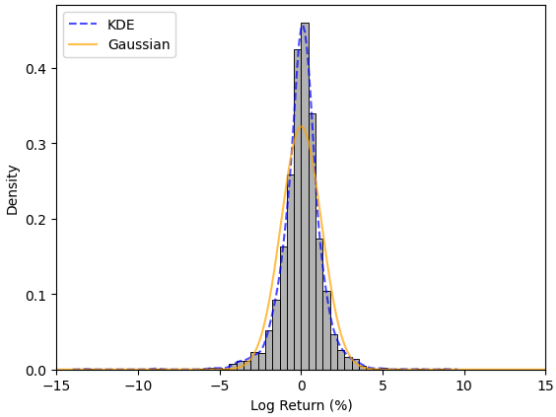
Mean	Standard Deviation	Skewness	Kurtosis
0.0118	1.24	-0.796	10.3

As shown in Table 1, the log returns (in percentage) have a small but positive mean, indicating modest average growth over the sample period. The standard deviation is large relative to the mean, highlighting substantial day-to-day variability. The return distribution displays clear departures from normality. The skewness of –0.796 indicates a longer, heavier left tail, while the kurtosis of 10.3 (excess kurtosis 7.3) points to a strongly leptokurtic distribution with a high peak and heavy tails.

Figure 1 further illustrates the deviation of log returns from normality. The kernel density estimate (KDE) in blue shows a sharper peak and heavier tails than the fitted normal density (orange) based on the sample mean and variance. This discrepancy is consistent with the summary statistics in Table 1.

To assess serial dependence in the log returns, we compute the empirical autocorrelation functions (ACFs) of the log returns, their absolute values, and their squares. These diagnostics capture linear dependence as well as volatility clustering and persistence. As shown in Figure 2(a), the ACF of the log returns exhibits no significant autocorrelation,

Figure 1:
Histogram and KDE of the log returns



indicating that log returns are serially uncorrelated. In contrast, Figures 2(b) and (c) show clear positive autocorrelations in the absolute and squared log returns, reflecting the volatility clustering effect in which large movements tend to be followed by further large movements.

1.3 GARCH(1,1) Model with Constant Mean and N(0,1) Innovations

To model the log returns, we begin by fitting a GARCH(1,1) model with a constant conditional mean and standard normal innovations. The model is specified as follows:

$$X_t = \mu + \sigma_t Z_t, \tag{2}$$

$$\sigma_t^2 = \omega + \alpha_1 (X_{t-1} - \mu)^2 + \beta_1 \sigma_{t-1}^2, \qquad t \in \mathbb{Z}. \tag{3}$$

The innovations Z_t are assumed to be i.i.d. N(0,1) random variables. The model is fitted using the `arch` package in Python. For the fitted model, the maximised log-likelihood is -3738.92, with AIC = 7485.84 and BIC = 7509.14.

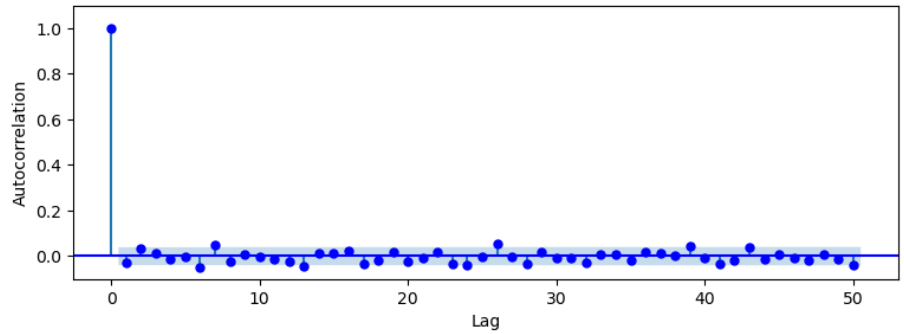
Table 2:
Coefficient outputs of the GARCH(1,1) model with constant mean and N(0,1) innovations

Variable	Estimate	Std. Error	T Value	$\mathbb{P} > \lvert t \rvert$
μ	0.0476	2.02×10^{-2}	2.36	1.83×10^{-2}
ω	0.0596	1.91×10^{-2}	3.13	1.77×10^{-3}
α_1	0.139	3.06×10^{-2}	4.55	5.35×10^{-6}
β_1	0.824	3.46×10^{-2}	23.8	2.36×10^{-125}

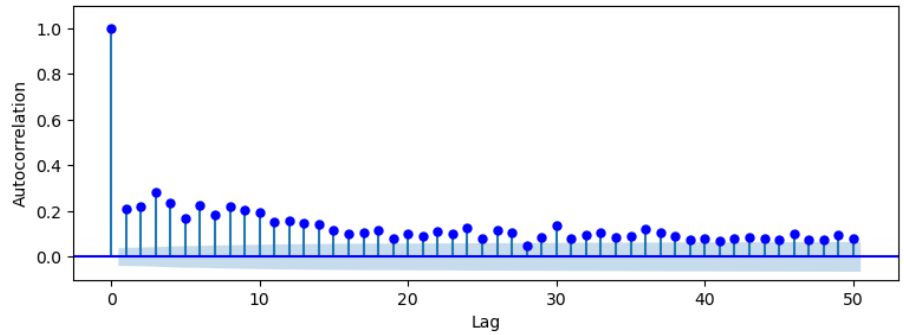
All coefficient estimates reported in Table 2 are statistically significant at the 5% level. The estimated parameters, $\alpha_1 = 0.139$ (95% CI: $[7.91 \times 10^{-2}, 0.199]$) and $\beta_1 = 0.824$ (95% CI: $[0.756, 0.892]$), sum to 0.963, which is close to but strictly below 1. This high degree of persistence is typical for equity index returns and reflects the slow decay of

Figure 2:
Empirical ACFs of the log returns, their absolute values, and their squares

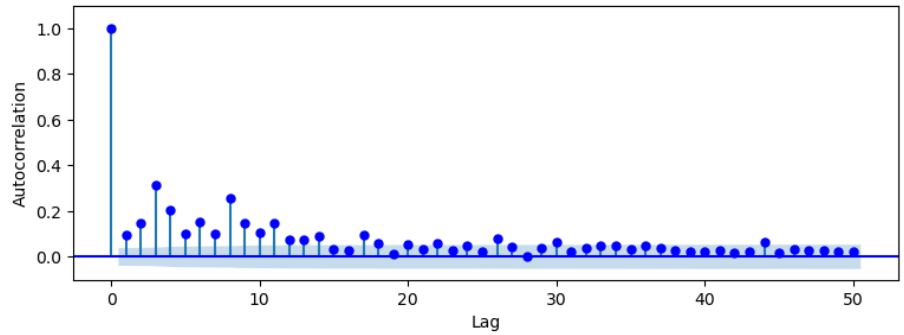
(a) *Empirical ACF of the log returns*



(b) *Empirical ACF of the absolute log returns*



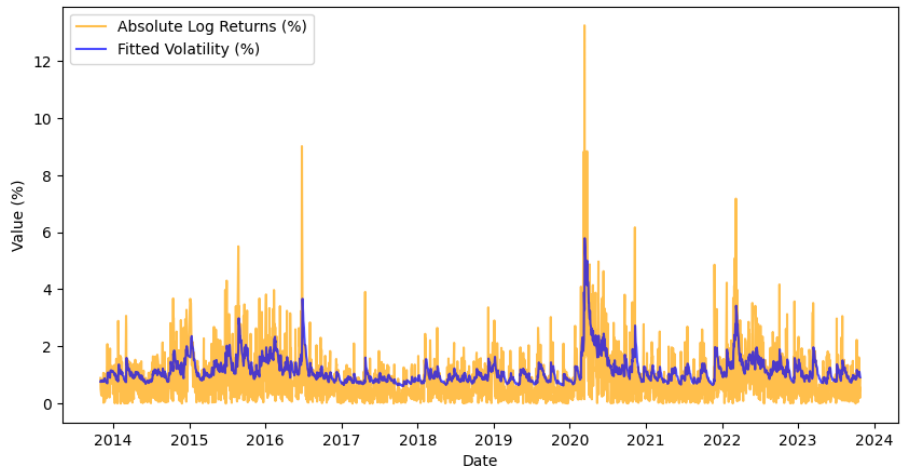
(c) *Empirical ACF of the squared log returns*



volatility shocks over time. The constant term in the variance equation $\omega = 0.0596$ (95% CI: $[2.23 \times 10^{-2}, 9.70 \times 10^{-2}]$) is small relative to the size of the volatility process. The conditional mean estimate, $\mu = 0.0476$ (95% CI: $[8.07 \times 10^{-3}, 8.72 \times 10^{-2}]$), corresponds to an average log return of 0.0476%, which is statistically significant but economically small.

Figure 3 compares the absolute log returns with the fitted volatility from the GARCH(1,1) model with constant conditional mean and standard normal innovations. The fitted volatility closely tracks periods of heightened market turbulence, rising sharply during times of

Figure 3:
Absolute log returns and fitted volatility of the GARCH(1,1) model with constant mean and N(0,1) innovations



large absolute returns (such as the volatility spikes in 2016 and early 2020). This shows that the model captures volatility clustering and the persistence of high-variance regimes. Although it smooths the most extreme movements, the fitted volatility still reflects the overall pattern of market variability over time.

To assess the model's goodness of fit, we examine whether the standardised residuals follow an i.i.d. sample from the assumed N(0,1) distribution. Moreover, a correctly specified model should yield residuals with no significant serial correlation. We evaluate this by analysing the ACFs of the standardised residuals, plotting their histogram and KDE, inspecting the normal QQ-plot, and performing the one-sample Anderson–Darling test (Nelson, 1998).

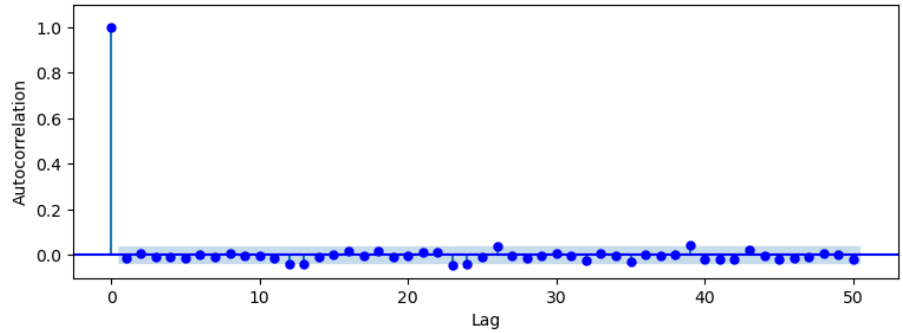
The empirical ACFs in Figure 4 shows that the standardised residuals display no meaningful serial correlation, which is consistent with the i.i.d. assumption required for a correctly specified model.

The histogram and KDE of the standardised residuals in Figure 5(a) indicate clear departures from the N(0,1) distribution as the empirical density exhibits higher peak heavier tails than that of the N(0,1). This observation is reinforced by the QQ-plot in Figure 5(b), where the lower and upper sample quantiles deviate substantially from the reference line $y = x$. The lower tail falls below the theoretical quantiles, while the upper tail lies above them, indicating heavier tails than N(0,1). These plots suggest that the N(0,1) innovation assumption is not adequate and that the model may not be correctly specified.

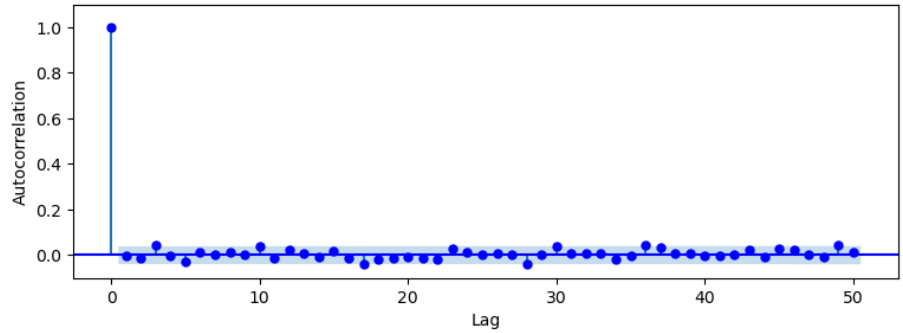
The one-sample Anderson–Darling test provides strong evidence against normality. The test statistic is 14.1, which exceeds the 1% critical value of 1.09. Consequently, we reject the null hypothesis that the standardised residuals follow a N(0,1) distribution.

Figure 4:
Empirical ACFs of the standardised residuals and their absolute values of the GARCH(1,1) model with constant mean and N(0,1) innovations

(a) *Empirical ACF of the standardised residuals*



(b) *Empirical ACF of the absolute standardised residuals*



1.4 ARMA(1,1)–GARCH(1,1) Model with Student-t Innovations

To allow for both serial dependence in the conditional mean and heavier-tailed innovations, we replace the constant conditional mean with an ARMA(1,1) process and assume Student-t distributed innovations. The model is specified as follows:

$$X_t = \mu_t + \sigma_t Z_t, \tag{4}$$

$$\mu_t = \mu + \phi_1(X_{t-1} - \mu) + \theta_1(X_{t-1} - \mu_{t-1}), \tag{5}$$

$$\sigma_t^2 = \alpha_0 + \alpha_1(X_{t-1} - \mu_{t-1})^2 + \beta_1\sigma_{t-1}^2, \qquad t \in \mathbb{Z}. \tag{6}$$

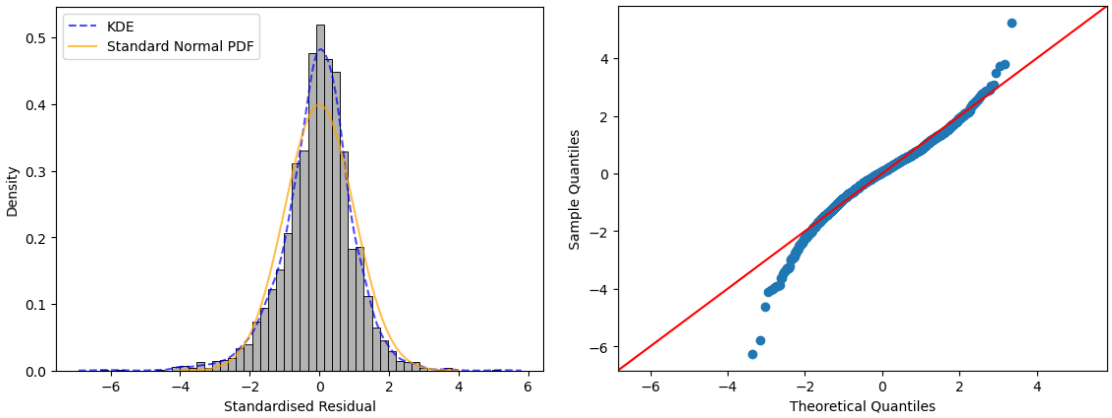
The innovations Z_t follow a normalised Student-t distribution with degree of freedom ν. Because the `arch` package in Python does not allow ARMA(1,1)–GARCH(1,1) to be fitted directly, the parameters are estimated using a constrained numerical optimiser that maximises the log-likelihood function:

$$\ell(\alpha_0, \alpha_1, \beta_1, \mu, \phi_1, \theta_1, \nu; X) = \sum_{t=1}^{T} \left[-\log \sigma_t + \log f_Z\left(\frac{X_t - \mu_t}{\sigma_t}\right)\right]. \tag{7}$$

To ensure stable convergence, the initial parameter values are set using a combination of empirical moments and estimates from an auxiliary ARMA(1,1) model. The GARCH

Figure 5:
Diagnostic plots of standardised residuals of the GARCH(1,1) model with constant mean and N(0,1) innovations

(a) *Histogram of the standardised residuals* ***(b)*** *QQ-plot of standardised residuals*



constant α_0 is initialised using the identity $\alpha_0 = \mathrm{Var}(X_t)(1 - \alpha_1 - \beta_1)$ with choices $\alpha_1 = 0.1$ and $\beta_1 = 0.85$. Starting values for the ARMA parameters ϕ_1 and θ_1 come from an ARMA(1,1) model fitted via `statsmodels`, while μ is set to the sample mean of the log returns. The degrees-of-freedom parameter ν is initialised at 10 to provide a suitably heavy-tailed starting point. Optimisation is performed using the SLSQP algorithm subject to parameter bounds and the stationarity constraint $\alpha_1 + \beta_1 < 1$.

Table 3:
Coefficient estimates of the ARMA(1,1)–GARCH(1,1) model with Student-t innovations

Variable	**Estimate**	**Std. Error**	**T Value**	$\mathbb{P} > \lvert t \rvert$
α_0	0.0475	1.28×10^{-2}	3.71	2.17×10^{-4}
α_1	0.148	2.42×10^{-2}	6.11	1.03×10^{-9}
β_1	0.834	2.44×10^{-2}	34.1	1.52×10^{-255}
μ	0.0699	1.35×10^{-2}	5.17	2.39×10^{-7}
ϕ_1	0.911	6.88×10^{-2}	13.2	5.13×10^{-40}
θ_1	-0.934	5.97×10^{-2}	-15.6	3.95×10^{-55}
ν	4.73	0.470	10.1	8.30×10^{-24}

The optimiser converges successfully, with parameter estimates reported in Table 3. The model exhibits strong persistence in volatility, with $\alpha_1 = 0.148$ (95% CI: $[0.100, 0.195]$), $\beta_1 = 0.834$ (95% CI: $[0.786, 0.881]$), and heavy-tailed innovations, with an estimated degrees-of-freedom parameter $\nu = 4.73$ (95% CI: $[3.81, 5.65]$). The ARMA(1,1) component captures short-term dependence in the conditional mean, with coefficient estimates $\phi_1 = 0.91$ (95% CI: $[0.777, 1.046]$) and $\theta_1 = -0.93$ (95% CI: $[-1.05, -0.817]$).

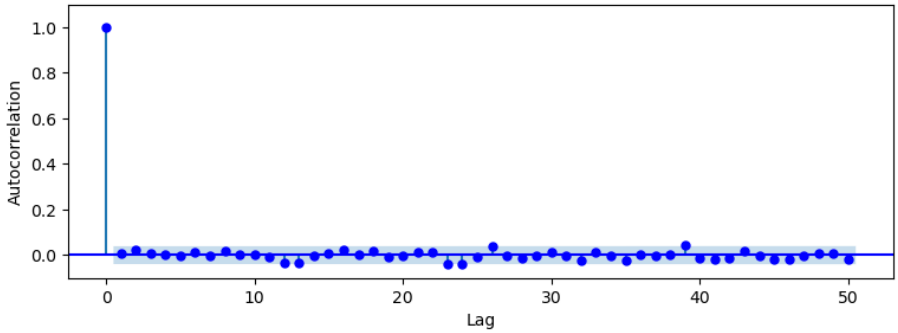
For the fitted ARMA(1,1)–GARCH(1,1) model with Student-t innovations, the maximised log-likelihood is -3635.09, with AIC = 7284.18 and BIC = 7324.97. Compared

with the GARCH(1,1) model with N(0,1) innovations, the ARMA(1,1)–GARCH(1,1) model achieves a higher log likelihood and lower information criteria despite its larger number of parameters. This indicates an improvement in overall model fit.

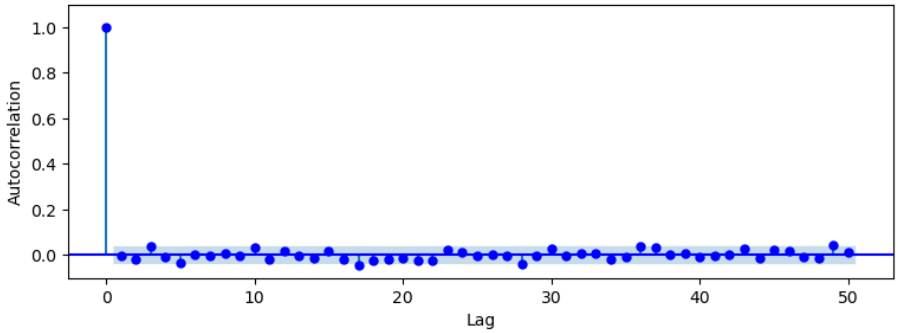
To assess the goodness of fit, we examine whether the standardised residuals behave like an i.i.d. sample from the fitted $t_{\hat{\nu}}$ distribution. The conditional mean and variance, and hence the standardised residuals, are computed iteratively using the sample mean and variance as initial values. A correctly specified model should also produce residuals with no meaningful serial correlation. We evaluate this by analysing the ACFs of the standardised residuals, plotting their histogram and KDE with the fitted t-density, inspecting the QQ-plot, and formally testing distributional adequacy using the k-sample Anderson–Darling test (Scholz & Stephens, 1987).

Figure 6:
Empirical ACFs of the standardised residuals and their absolute values of the ARMA(1,1)–GARCH(1,1) model with Student-t innovations

(a) *Empirical ACF of the standardised residuals*



(b) *Empirical ACF of the absolute standardised residuals*

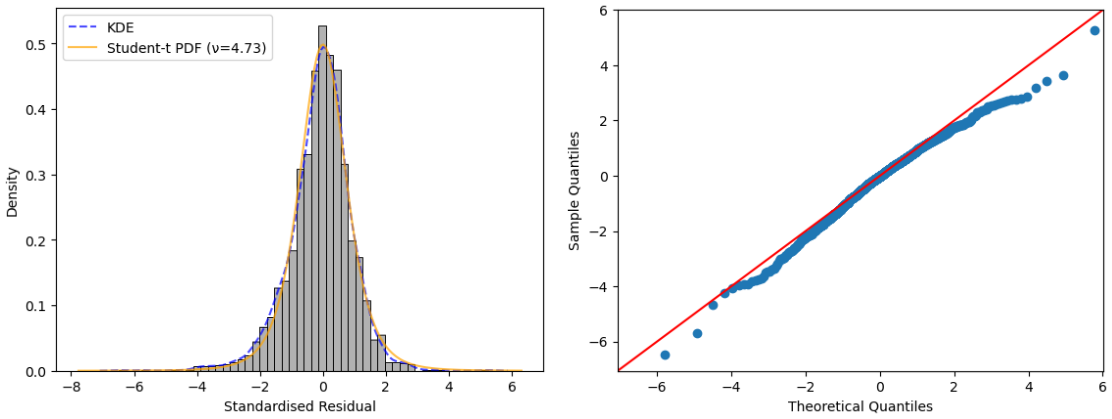


The empirical ACFs in Figure 6 shows that the standardised residuals display no meaningful serial correlation, which is consistent with the i.i.d. assumption required for a correctly specified model.

Figure 7(a) presents the histogram and KDE of the standardised residuals, overlaid with the fitted Student-t density with $\hat{\nu}$ degrees of freedom (scaled to have variance 1). The

Figure 7:
Diagnostic plots of standardised residuals of the ARMA(1,1)–GARCH(1,1) model with Student-t innovations

(a) *Histogram of the standardised residuals* ***(b)*** *QQ-plot of standardised residuals*



empirical distribution aligns closely with the fitted t-density, providing support for the heavy-tailed innovation assumption. Figure 7(b) shows some deviations from the Student-t distribution, though these are substantially smaller than under the normality assumption.

The k-sample Anderson–Darling test provides further evidence of remaining misspecification: the test statistic is 3.75 with an associated significance level of 0.01, leading to rejection of the null hypothesis that the standardised residuals follow a $t_{\hat{\nu}}$ distribution. While the Student-t specification substantially improves the fit relative to the Gaussian model, these diagnostics suggest that the innovation distribution is still not fully captured by the fitted model.

To assess whether the ARMA(1,1) component is necessary, we compare the ARMA(1,1)–GARCH(1,1) model with Student-t innovations to the GARCH(1,1) model with constant conditional mean and Student-t innovations. The model without the ARMA component achieves a maximised log-likelihood of -3639.90, only marginally below the value -3635.09 obtained under the ARMA(1,1)–GARCH(1,1) model. The resulting AIC and BIC values (AIC=7289.79, BIC=7318.92) are very close to those of the more complex model, indicating that the ARMA component offers little improvement.

The coefficient outputs and diagnostic checks for the GARCH(1,1) model with constant conditional mean and Student-t innovations are reported in the Appendix (Table 6, Figure 13 and 14). There are no evidence of serial correlation in the standardised residuals, and the distributional diagnostics (histogram, KDE, QQ-plot, and k-sample Anderson–Darling test) closely mirror those of the ARMA(1,1)–GARCH(1,1) model, with the k-sample Anderson–Darling test yielding a statistic of 2.30 and significance level 0.037. These results suggest that the ARMA(1,1) component is not necessary for modelling the conditional mean of the log returns, and that a constant-mean GARCH(1,1) model with Student-t innovations provides an adequate and parsimonious specification.

2 Part B: Risk Measures

2.1 Introduction

This section evaluates one-day ahead VaR and ES forecasts for the ProShares Short Dow30 (DOG) ETF (ProShares, 2025). The daily linearised loss relative to the investment value is defined as:

$$L_{t+1}^{\Delta} = \frac{L_{t+1}^{\Delta}}{V_t} = -r_{t+1} \tag{8}$$

where V_t is the investment value and $-r_{t+1}$ is the negative log return. We compare Historical Simulation (HS), Filtered Historical Simulation with EWMA (FHS-EWMA), and FHS-GARCH(1,1) at 95% and 99% confidence levels.

2.2 Historical Simulation (HS)

We utilize a sliding window of $N = 500$ days. For each forecasting date t, the dataset is $L = \{L_{t-499}, \ldots, L_t\}$. The estimated VaR at level α is the empirical quantile:

$$\widehat{\mathrm{VaR}}_\alpha(L_t) = q_\alpha(L) = \inf\left\{x \in \mathbb{R} : \mathbb{P}(L \le x) \ge \alpha\right\}. \tag{9}$$

The Expected Shortfall (ES) is calculated as the expected loss conditional on exceeding the VaR:

$$\widehat{\mathrm{ES}}_\alpha(L_t) = \frac{\mathbb{E}\Big[\, L\mathbf{1}_{\{L \ge \widehat{\mathrm{VaR}}_\alpha(L)\}}\Big]}{1-\alpha} = \frac{1}{\sum_{j=t-499}^{t} \mathbf{1}_{\{L_j \ge \widehat{\mathrm{VaR}}_\alpha(L)\}}} \sum_{j=t-499}^{t} L_j\, \mathbf{1}_{\{L_j \ge \widehat{\mathrm{VaR}}_\alpha(L)\}}. \tag{10}$$

The HS forecasts in Figure 8 exhibit a characteristic 'step-like' shape. Following the 2020 volatility spike, risk estimates remain artificially elevated for nearly two years, confirming that HS is strongly backward-looking. Extreme observations remain in the 500-day window long after market conditions stabilise, leading to risk overestimation in calm periods. Conversely, during the onset of the 2020 crash, the 95% VaR frequently fell below realised losses, indicating poor responsiveness to sudden shocks.

2.3 FHS with EWMA

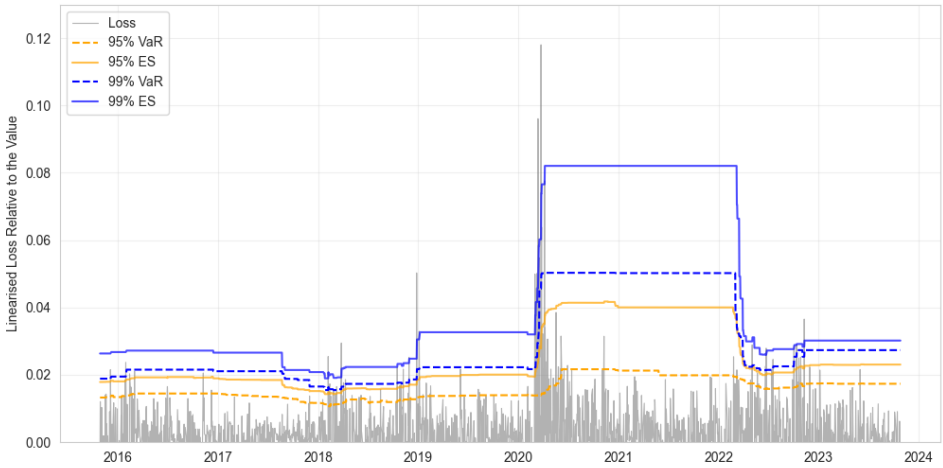
To address the lag in HS, we apply FHS using an EWMA volatility model. The volatility forecast is computed recursively:

$$\hat{\sigma}_{t+1}^2 = \alpha(L_t - \hat{\mu}_t)^2 + (1-\alpha)\hat{\sigma}_t^2. \tag{11}$$

Using the specified parameters $\alpha = 0.06$ and $\hat{\mu}_t = 0$, the update equation becomes:

$$\hat{\sigma}_{t+1}^2 = 0.06L_t^2 + 0.94\hat{\sigma}_t^2. \tag{12}$$

Figure 8:
VaR and ES using Historical Simulation (N=500)



The initial volatility $\hat{\sigma}_0$ is the variance of the first 500 observations:

$$\hat{\sigma}_0^2 = \frac{1}{n}\sum_{i=0}^{499}(L_i - \bar{L})^2. \tag{13}$$

VaR and ES are forecasted by scaling the quantile and ES of the standardised residuals ($Z_t = L_t/\hat{\sigma}_t$) by the current volatility:

$$\widehat{\text{VaR}}_\alpha(L_{t+1}) = \hat{\sigma}_{t+1} q_\alpha(Z_{t+1}), \quad \widehat{\text{ES}}_\alpha(L_{t+1}) = \hat{\sigma}_{t+1}\widehat{\text{ES}}_\alpha(Z_{t+1}). \tag{14}$$

Figure 9:
VaR and ES using FHS-EWMA

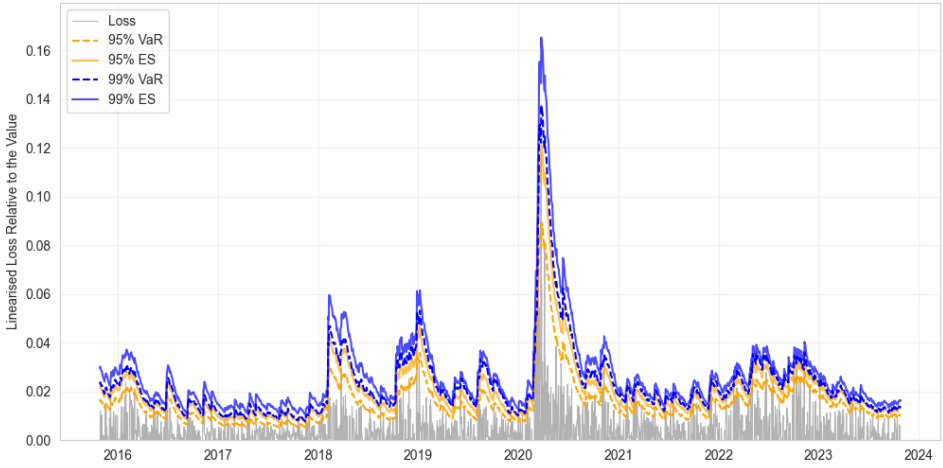
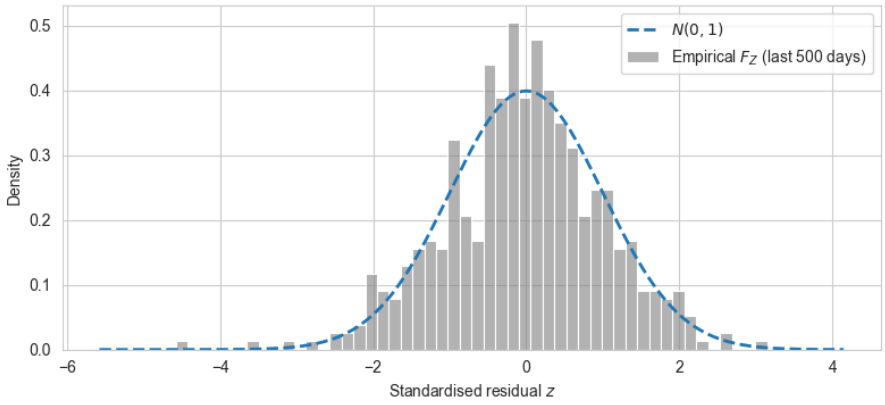


Figure 9 illustrates the results. Unlike HS, the FHS-EWMA forecasts track the realised volatility dynamically. The curves rise sharply during the 2020 shock and decay as mar-

kets stabilise, avoiding the 'ghost effects' of old data. The widening gap between 95% and 99% measures during stress periods reflects the heavy-tailed nature of the residuals.

Figure 10:
Empirical distribution of standardised residuals (FHS-EWMA)



The empirical distribution of the standardised residuals (Figure 10) is roughly symmetric and bell-shaped, validating the $\mu = 0$ assumption. However, the distribution shows higher concentration around the mean and slightly heavier tails than the standard normal, indicating excess kurtosis.

2.4 FHS with GARCH

We estimate VaR and ES using FHS-GARCH(1,1) with normal innovations. The process is defined as:

$$L_t = \sigma_t Z_t, \tag{15}$$
$$\sigma_t^2 = \alpha_0 + \alpha_1 L_{t-1}^2 + \beta_1 \sigma_{t-1}^2. \tag{16}$$

Parameters were re-estimated daily using a 500-day rolling window via conditional maximum likelihood:

$$\mathcal{L}(\alpha_0, \alpha_1, \beta_1; X, \sigma_0) = \prod_{t=1}^{T} \frac{1}{\sigma_t} f_Z\left(\frac{L_t}{\sigma_t}\right). \tag{17}$$

where f_Z is the standard normal density. The forecast formulas for VaR and ES remain the same as in section 3.2, utilizing the GARCH $\hat{\sigma}_{t+1}$.

As seen in Figure 11, the GARCH forecasts share the dynamic shape of EWMA but exhibit sharper, more frequent fluctuations. This sensitivity suggests that re-estimating GARCH parameters on a short window ($N = 500$) may introduce instability. While it reacts aggressively to clustering, the GARCH model produces less conservative risk estimates during quiet periods compared to EWMA. The residual distribution (Figure 12) confirms the model successfully filters volatility clustering, showing properties similar to the EWMA residuals.

Figure 11:
VaR and ES using FHS-GARCH

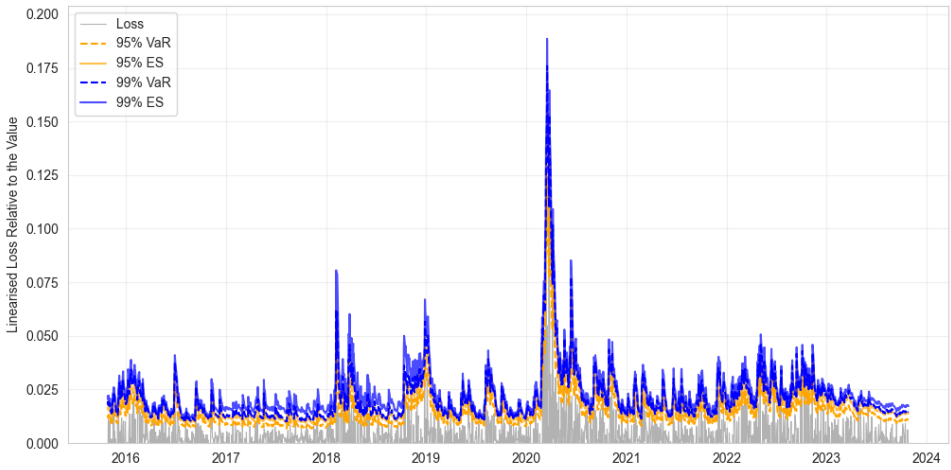
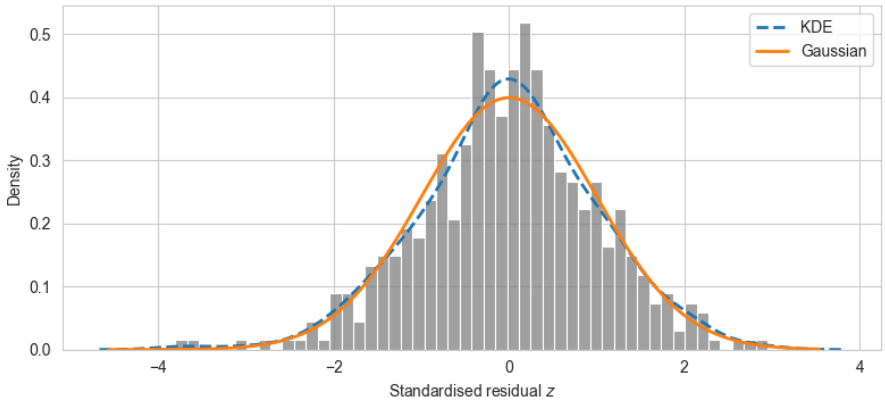


Figure 12:
Empirical distribution of standardised residuals (FHS-GARCH)



2.5 Model Backtest and Recommendation

We evaluate the models using Kupiec's tests for coverage and independence (Kupiec et al., 1995). We define the violation indicator I_t:

$$I_t := \begin{cases} 1, & L_t > \mathrm{VaR}_\alpha(L_t), \\ 0, & L_t \leq \mathrm{VaR}_\alpha(L_t). \end{cases} \tag{18}$$

Assuming $I_t \sim \text{Bernoulli}(\pi)$, the likelihood under the null hypothesis ($\pi = 1 - \alpha$) vs the alternative ($\pi = \hat{\pi}$) is tested using the Unconditional Coverage statistic:

$$\mathrm{LR}_{\mathrm{uc}} := -2 \log \left(\frac{L(1-\alpha; I)}{L(\hat{\pi}; I)} \right) \sim \chi^2(1). \tag{19}$$

For independence, we estimate transition probabilities π_{01} and π_{11} and calculate the Likelihood Ratio:

$$\mathrm{LR}_{\mathrm{ind}} := -2\log\left(\frac{L(\hat{\Pi}_0; I)}{L(\hat{\Pi}_1; I)}\right) \sim \chi^2(1). \tag{20}$$

The joint test is $\mathrm{LR}_{\mathrm{cc}} = \mathrm{LR}_{\mathrm{uc}} + \mathrm{LR}_{\mathrm{ind}} \sim \chi^2(2)$. Finally, ES is backtested using the test statistic Z on the residuals $\xi_t = (L_t - ES_\alpha)I_t$:

$$Z := \frac{\sum_{t=1}^{T} \xi_t}{\sqrt{\sum_{t=1}^{T} \xi_t^2}} \sim N(0,1). \tag{21}$$

Table 4:
VaR backtesting results

Model	**Level**	**Unconditional**		**Independence**		**Joint**	
		p_{uc}	**Result**	p_{ind}	**Result**	p_{cc}	**Result**
HS	99%	0.040	**Fail**	0.081	Pass	0.026	**Fail**
	95%	0.858	Pass	0.030	**Fail**	0.093	Pass
EWMA	99%	0.533	Pass	0.466	Pass	0.631	Pass
	95%	0.858	Pass	0.671	Pass	0.899	Pass
GARCH	99%	0.063	Pass	0.357	Pass	0.116	Pass
	95%	0.980	Pass	0.106	Pass	0.271	Pass

As shown in Table 4, HS fails the unconditional and joint tests at 99% and the independence test at 95%, confirming it is inadequate for tail risk estimation. Both FHS methods pass all tests, but EWMA shows significantly higher p-values, indicating a robust fit.

Table 5:
ES backtesting results

Model	**Level**	p**-value**	**Conclusion**
HS	99%	0.038	**Fail**
	95%	0.130	Pass
EWMA	99%	0.797	Pass
	95%	0.403	Pass
GARCH	99%	0.621	Pass
	95%	0.209	Pass

Table 5 reinforces that HS underestimates Expected Shortfall at extreme quantiles. Both FHS models pass, with EWMA again providing higher confidence.

Recommendation: We recommend the FHS-EWMA approach. It passes all backtests with the highest statistical confidence, accurately captures volatility clustering, and avoids the parameter instability observed in the rolling GARCH model. HS should be rejected due to its failure to adapt to market shocks and poor backtesting performance.

References

Bloomberg. (2025). About euro stoxx 50 price eur [Accessed: 2025-11-20]. https://www.bloomberg.com/quote/SX5E:IND

Kupiec, P. H., et al. (1995). *Techniques for verifying the accuracy of risk measurement models* (Vol. 95). Division of Research; Statistics, Division of Monetary Affairs, Federal ...

Nelson, L. S. (1998). The anderson-darling test for normality. *Journal of Quality Technology*, *30*(3), 298–299.

ProShares. (2025). Dog — short dow30 etf – proshares [Accessed: 2025-11-20]. https://www.proshares.com/our-etfs/leveraged-and-inverse/dog

Scholz, F. W., & Stephens, M. A. (1987). K-sample anderson–darling tests. *Journal of the American Statistical Association*, *82*(399), 918–924.

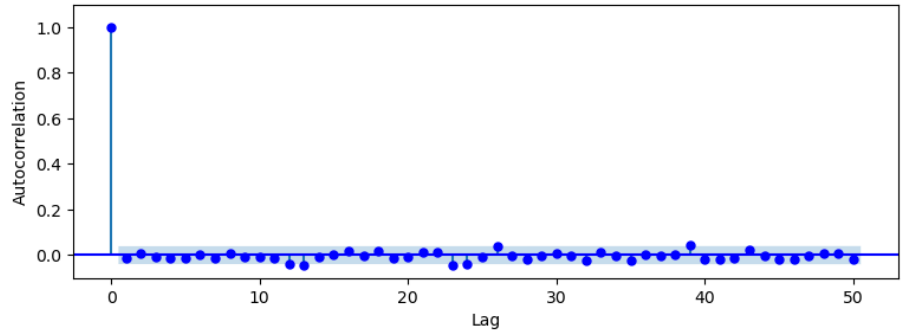
A Additional Tables and Figures

Table 6:
Coefficient estimates of the GARCH(1,1) model with constant mean and Student-t innovations

Variable	Estimate	Std. Error	T Value	$\mathbb{P} > \|t\|$
μ	0.0681	1.72×10^{-2}	3.97	7.22×10^{-5}
ω	0.0467	1.41×10^{-2}	3.31	9.24×10^{-4}
α_1	0.146	2.78×10^{-2}	5.24	1.61×10^{-7}
β_1	0.835	2.89×10^{-2}	28.9	3.12×10^{-183}
ν	4.85	4.65×10^{-1}	10.4	2.00×10^{-25}

Figure 13:
Empirical ACFs of the standardised residuals and their absolute values the GARCH(1,1) model with constant mean and Student-t innovations

(a) *Empirical ACF of the standardised residuals*



(b) *Empirical ACF of the absolute standardised residuals*

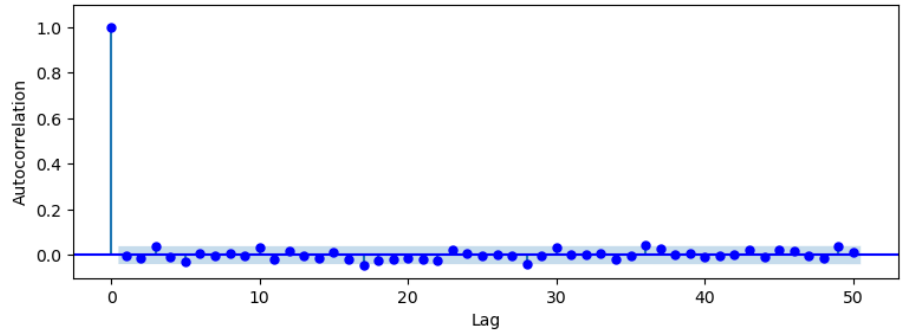
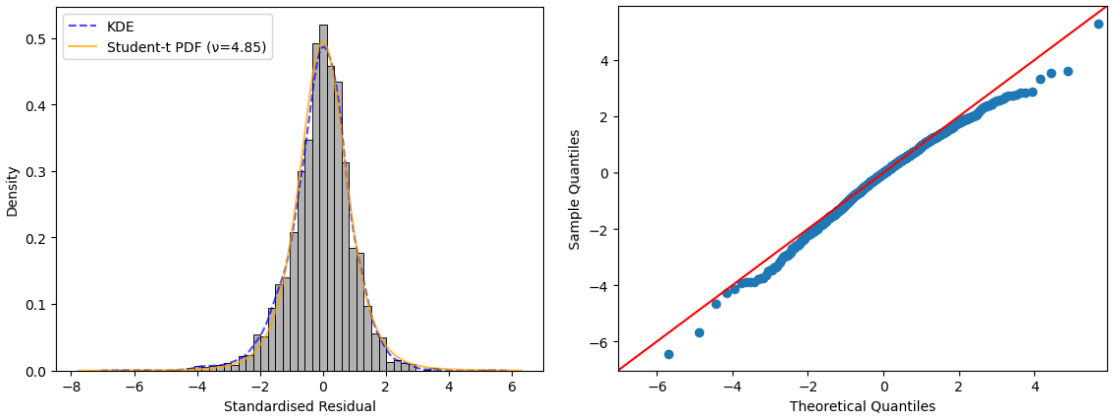


Figure 14:
Diagnostic plots of standardised residuals of the GARCH(1,1) model with constant mean and Student-t innovations

(a) *Histogram of the standardised residuals* ***(b)*** *QQ-plot of standardised residuals*



B Code

```python
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
from scipy.stats import norm, t
from scipy import stats
import statsmodels.api as sm
from arch.univariate import arch_model

# Part A
# Compute log returns (%)
df = pd.read_csv('../data/QRM-2025-cw1-data-a.csv')
df_nodelay = df['Adj Close']
df_delay = df['Adj Close'].shift(1)
df['log_return'] = (np.log(df_nodelay) - np.log(df_delay)) * 100

# Part (i):
mu = df['log_return'].mean()
std = df['log_return'].std()
s = df['log_return'].skew()
k = df['log_return'].kurtosis()
print(f'log_return_mean: {mu:.5f}')
print(f'log_return_std: {std:.5f}')
print(f'log_return_skewness: {s:.5f}')
print(f'log_return_kurtosis: {k:.5f}')
print()
print(df['log_return'].describe())

# Plot histogram of log returns, KDE, and the fitted Gaussian pdf
x = np.linspace(-15,15,1000)
ax = sns.histplot(data=df['log_return'], color="gray", alpha = 0.6, bins=50, stat="
    density")
sns.kdeplot(data=df['log_return'], color="blue", alpha = 0.7, linestyle="--", label = '
    KDE')
ax.plot(x,norm.pdf(x,mu,std),color = "orange", alpha = 0.7, label = 'Gaussian')
ax.set(xlabel='Log Return (%)', ylabel='Density')
ax.set_xlim(-15, 15)
ax.legend(loc = "upper left")
```

```
plt.show()

# Part (ii):
# Plot ACF of log returns, absolute log returns, and squared log returns
fig, ax = plt.subplots(figsize=(9, 3))
sm.graphics.tsa.plot_acf(
    df['log_return'].dropna(),
    lags=50,
    ax=ax,
    color="blue",
)
ax.set_title("")
ax.set_xlabel("Lag")
ax.set_ylabel("Autocorrelation")
ax.set_ylim(-0.1, 1.1)
plt.show()

fig, ax = plt.subplots(figsize=(9, 3))
sm.graphics.tsa.plot_acf(
    np.abs(df['log_return']).dropna(),
    lags=50,
    ax=ax,
    color="blue",
)
ax.set_title("")
ax.set_xlabel("Lag")
ax.set_ylabel("Autocorrelation")
ax.set_ylim(-0.1, 1.1)
plt.show()

fig, ax = plt.subplots(figsize=(9, 3))
sm.graphics.tsa.plot_acf(
    (df['log_return'] ** 2).dropna(),
    lags=50,
    ax=ax,
    color="blue",
)
ax.set_title("")
ax.set_xlabel("Lag")
ax.set_ylabel("Autocorrelation")
ax.set_ylim(-0.1, 1.1)
plt.show()

# Part (iii):
model = arch_model(df['log_return'].dropna(), mean='constant', p=1, q=1, dist='normal')
res = model.fit(update_freq = 3)
res.summary()

# Plot absolute log returns and fitted volatilities
df['Date'] = pd.to_datetime(df['Date'], dayfirst=True)
df2 = df.dropna(subset=['log_return']).copy()
abs_ret = df2['log_return'].abs()
vol = pd.Series(res.conditional_volatility, index=df2.index)
dates = df2['Date']

fig, ax = plt.subplots(figsize=(10,5))
ax.plot(dates, abs_ret, label='Absolute Log Returns (%)', alpha=0.7, color = 'orange')
ax.plot(dates, vol, label='Fitted Volatility (%)', linewidth=1.5, alpha=0.7, color = '
    blue')
ax.set_xlabel('Date')
ax.set_ylabel('Value (%)')
ax.legend(loc = "upper left")
plt.show()

# Part (iv):
std_resid = res.std_resid

```

```
# Plot ACFs of standardised residuals and their absolute values
fig, ax = plt.subplots(figsize=(9, 3))
sm.graphics.tsa.plot_acf(
    std_resid.dropna(),
    lags=50,
    ax=ax,
    color="blue",
)
ax.set_title("")
ax.set_xlabel("Lag")
ax.set_ylabel("Autocorrelation")
ax.set_ylim(-0.1, 1.1)
plt.show()

fig, ax = plt.subplots(figsize=(9, 3))
sm.graphics.tsa.plot_acf(
    np.abs(std_resid).dropna(),
    lags=50,
    ax=ax,
    color="blue",
)
ax.set_title("")
ax.set_xlabel("Lag")
ax.set_ylabel("Autocorrelation")
ax.set_ylim(-0.1, 1.1)

# Plot histogram of standardised residuals
x = np.linspace(-4, 4, 200)
ax = sns.histplot(std_resid, color="gray", alpha = 0.6, bins=50, stat="density")
sns.kdeplot(data=std_resid, color="blue", alpha = 0.7, linestyle="--", label = 'KDE')
ax.plot(x,stats.norm.pdf(x),color = "orange", alpha = 0.7, label = 'Standard Normal PDF'
    )
ax.set(xlabel='Standardised Residual', ylabel='Density')
plt.legend(loc = "upper left")
plt.show()

# Plot QQ-plot
sm.qqplot(std_resid, line='45', fit=True)
plt.show()

# A-D test
ad_res = stats.anderson(std_resid, dist = 'norm')
print(f'Anderson-Darling test:\n'
      f'statistic = {ad_res.statistic:.1f}\n'
      f'critical value with significance level 1% = {ad_res.critical_values[-1]:.2f}\n')

# Part (v)
df3 = df['log_return'].dropna()

def arma11_garch11_t_loglikelihood(alpha0, alpha1, beta1, arma_mu, phi1, theta1, nu, x):

    # Store time series of sigma_t^2 and mu_t
    var = np.zeros_like(x)
    mu = np.zeros_like(x)

    # Set up proxies for sigma_0^2 and \mu_0
    var[0] = np.var(x, ddof=1) # sample variance
    mu[0] = np.mean(x) # sample mean

    # Iterate over the ARMA(1,1)-GARCH(1,1) dynamics
    for i in range(1, len(x)):
        mu[i] = arma_mu + phi1 * (x.iloc[i-1] - arma_mu) + theta1 * (x.iloc[i-1] - mu[i
            -1])
        var[i] = alpha0 + alpha1 * (x.iloc[i-1] - mu[i-1])**2 + beta1 * var[i-1]

    return np.sum(-0.5*np.log(var[1:]) + np.log(t.pdf((x.iloc[1:]-mu[1:])/np.sqrt(var
        [1:]), nu, 0, np.sqrt((nu-2)/nu))))
```

```
167
168 from scipy.optimize import minimize
169 from statsmodels.tsa.arima.model import ARIMA
170
171 objfun_armagarch_t = lambda y : -1 * arma11_garch11_t_loglikelihood(y[0], y[1], y[2], y
        [3], y[4], y[5], y[6], df3)
172 constraint = lambda y :  1 - y[1] - y[2]
173
174 uncond_var = np.var(df3, ddof=1)
175 alpha1_0 = 0.1
176 beta1_0  = 0.85
177 alpha0_0 = uncond_var * (1 - alpha1_0 - beta1_0)
178
179 arma_0 = ARIMA(df3, order=(1, 0, 1)).fit()
180
181 arma_mu_0 = df3.mean()
182 phi1_0   = arma_0.params.get("ar.L1", 0.0)
183 theta1_0 = arma_0.params.get("ma.L1", 0.0)
184
185 y0 = [alpha0_0, alpha1_0, beta1_0, arma_mu_0, phi1_0, theta1_0, 10]
186 print("Initial objective function value: " + str(objfun_armagarch_t(y0)))
187
188 bnds = ((0,np.inf), (0,1), (0,1), (-np.inf, np.inf), (-np.inf, np.inf), (-np.inf, np.inf
        ), (2.01, np.inf))
189
190 con = {'type': 'ineq', 'fun': constraint}
191
192 sol = minimize(objfun_armagarch_t,y0,method='SLSQP',bounds=bnds,constraints=con, options
        ={'disp': True})
193
194 # Coefficient estimates
195 print("Estimate of alpha0: " + str(sol.x[0]))
196 print("Estimate of alpha1: " + str(sol.x[1]))
197 print("Estimate of beta1: " + str(sol.x[2]))
198 print("Estimate of mu: " + str(sol.x[3]))
199 print("Estimate of phi1: " + str(sol.x[4]))
200 print("Estimate of theta1: " + str(sol.x[5]))
201 print("Estimate of nu: " + str(sol.x[6]))
202
203 # AIC and BIC
204 k = len(sol.x)
205 n = len(df3)
206 ll = -objfun_armagarch_t(sol.x)
207 AIC = 2*k - 2*ll
208 BIC = k*np.log(n) - 2*ll
209 print(AIC, BIC)
210
211 # Standard Errors
212 from numdifftools import Hessian
213
214 # Evaluate Hessian of NEGATIVE loglikelihood (objective function)
215 H = Hessian(objfun_armagarch_t)(sol.x)
216 # Inverse Hessian = covariance matrix
217 cov = np.linalg.inv(H)
218 # Standard errors
219 std_err = np.sqrt(np.diag(cov))
220 print(std_err)
221
222 # T-values
223 t_values = sol.x / std_err
224 print(t_values)
225
226 # 95% CIs
227 ci_lower = sol.x - 1.96 * std_err
228 ci_upper = sol.x + 1.96 * std_err
229 print(ci_lower,
230       ci_upper)
```

```
231
232 # Calculate fitted values and residuals
233 alpha0_hat = sol.x[0]
234 alpha1_hat = sol.x[1]
235 beta1_hat  = sol.x[2]
236 arma_mu_hat = sol.x[3]
237 phi1_hat    = sol.x[4]
238 theta1_hat  = sol.x[5]
239 nu_hat      = sol.x[6]
240
241 def arma11_garch11_paths(params, x):
242     alpha0, alpha1, beta1, arma_mu, phi1, theta1, nu = params
243
244     x = x.to_numpy()  # work with ndarray for speed
245     n = len(x)
246
247     var = np.empty(n)
248     mu  = np.empty(n)
249
250     # Initial values
251     var[0] = np.var(x, ddof=1)
252     mu[0]  = np.mean(x)
253
254     for i in range(1, n):
255         # Mean recursion
256         mu[i] = arma_mu + phi1 * (x[i-1] - arma_mu) + theta1 * (x[i-1] - mu[i-1])
257         # Variance recursion
258         e_prev = x[i-1] - mu[i-1]
259         var[i] = alpha0 + alpha1 * e_prev * e_prev + beta1 * var[i-1]
260
261     # Standardized residuals
262     sigma = np.sqrt(var)
263     z = (x - mu) / sigma
264
265     # Wrap back into Series with same index as x
266     idx = df3.index  # or x_index saved separately if you prefer
267     mu_s    = pd.Series(mu,    index=idx, name="mu_hat")
268     var_s   = pd.Series(var,   index=idx, name="var_hat")
269     sigma_s = pd.Series(sigma, index=idx, name="sigma_hat")
270     z_s     = pd.Series(z,     index=idx, name="z_hat")
271
272     return mu_s, var_s, sigma_s, z_s
273
274 params_hat = (alpha0_hat, alpha1_hat, beta1_hat,
275               arma_mu_hat, phi1_hat, theta1_hat, nu_hat)
276 mu_hat, var_hat, sigma_hat, z_hat = arma11_garch11_paths(params_hat, df3)
277 z_std = z_hat.iloc[1:]
278 z = z_std.dropna()
279
280 # X-grid for plotting the fitted t-density
281 x = np.linspace(z.min() * 1.2, z.max() * 1.2, 200)
282 # Scale parameter to ensure Var = 1
283 scale_hat = np.sqrt((nu_hat - 2) / nu_hat)
284
285 # Histogram of standardised residuals
286 ax = sns.histplot(z, color="gray", alpha=0.6, bins=50, stat="density")
287 # KDE
288 sns.kdeplot(
289     data=z,
290     color="blue",
291     alpha=0.7,
292     linestyle="--",
293     label="KDE"
294 )
295 # Fitted Student-t PDF
296 ax.plot(
297     x,
```

```
    t.pdf(x, df=nu_hat, loc=0, scale=scale_hat),
    color="orange",
    alpha=0.7,
    label=f"Student-t PDF"
)
ax.set(xlabel="Standardised Residual", ylabel="Density")
plt.legend(loc="upper left")
plt.show()

# QQ-plot
sm.qqplot(
    z,
    dist=t,
    distargs=(nu_hat,),
    loc=0,
    scale=scale_hat,
    line='45'
)
plt.show()

# ACFs of standardised residuals
fig, ax = plt.subplots(figsize=(9, 3))
sm.graphics.tsa.plot_acf(
    z.dropna(),
    lags=50,
    ax=ax,
    color="blue",
)
ax.set_title("")
ax.set_xlabel("Lag")
ax.set_ylabel("Autocorrelation")
ax.set_ylim(-0.1, 1.1)

fig, ax = plt.subplots(figsize=(9, 3))
sm.graphics.tsa.plot_acf(
    np.abs(z).dropna(),
    lags=50,
    ax=ax,
    color="blue",
)
ax.set_title("")
ax.set_xlabel("Lag")
ax.set_ylabel("Autocorrelation")
ax.set_ylim(-0.1, 1.1)

from scipy.stats import anderson_ksamp
# Simulate reference t-distribution
t_sim = t.rvs(df=nu_hat, loc=0, scale=scale_hat, size=50000, random_state=42)
# A-D k-sample test
ad_stat, crit_vals, sig_level = anderson_ksamp([z, t_sim])

# Without ARMA(1,1)
model2 = arch_model(df['log_return'].dropna(), mean='constant', p=1, q=1, dist='t')
res2 = model2.fit(update_freq = 3)
res2.summary()

# Standardised residuals and diagnostic plots
std_resid2 = res2.std_resid
nu = res2.params["nu"]

# X-grid for the theoretical t density
x_vals = np.linspace(std_resid2.min() * 1.2, std_resid2.max() * 1.2, 500)
# Scale so that Var = 1 (same as in your likelihood)
scale2 = np.sqrt((nu - 2.0) / nu)

# Histogram
ax = sns.histplot(std_resid2, color="gray", alpha=0.6, bins=50, stat="density")
```

```
# KDE
sns.kdeplot(
    data=std_resid2,
    color="blue",
    alpha=0.7,
    linestyle="--",
    label="KDE"
)
# Fitted Student-t PDF
ax.plot(
    x,
    t.pdf(x, df=nu, loc=0, scale=scale2),
    color="orange",
    alpha=0.7,
    label=f"Student-t PDF"
)
ax.set(xlabel="Standardised Residual", ylabel="Density")
plt.legend(loc="upper left")
plt.show()

# QQ-plots
sm.qqplot(
    std_resid2,
    dist=t,
    distargs=(nu,),    # only df here
    loc=0,
    scale=scale2,
    line='45'
)
plt.show()

# ACFs of standardised residuals
fig, ax = plt.subplots(figsize=(9, 3))
sm.graphics.tsa.plot_acf(
    std_resid2.dropna(),
    lags=50,
    ax=ax,
    color="blue",
)
ax.set_title("")
ax.set_xlabel("Lag")
ax.set_ylabel("Autocorrelation")
ax.set_ylim(-0.1, 1.1)

fig, ax = plt.subplots(figsize=(9, 3))
sm.graphics.tsa.plot_acf(
    np.abs(std_resid2).dropna(),
    lags=50,
    ax=ax,
    color="blue",
)
ax.set_title("")
ax.set_xlabel("Lag")
ax.set_ylabel("Autocorrelation")
ax.set_ylim(-0.1, 1.1)

# Simulate reference t-distribution
t_sim2 = t.rvs(df=nu, loc=0, scale=scale2, size=50000, random_state=42)
# A-D ksample test
ad_stat2, crit_vals2, sig_level2 = anderson_ksamp([std_resid2, t_sim2])

# Part B
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from numpy.lib.stride_tricks import sliding_window_view
from arch import arch_model
```

```
from scipy.stats import chi2
from scipy.stats import norm
import seaborn as sns

# Initialisation
df = pd.read_csv('../data/QRM-2025-cw1-data-b.csv')
prices = df['Adj Close'].values
dates = pd.to_datetime(df['Date'], dayfirst=True).values[1:]
losses = -np.log(prices[1:] / prices[:-1])
T = len(losses)

# Common HS Setup
N = 500 # Window Size
targets = losses[N:]
target_dates = dates[N:]

# Part (i):
windows = sliding_window_view(losses[:-1], window_shape=N) # Technical reason to exclude
     the last date
var_95_i = np.quantile(windows, 0.95, axis=1)
es_95_i = np.array([w[w >= v].mean() for w, v in zip(windows, var_95_i)])

var_99_i = np.quantile(windows, 0.99, axis=1)
es_99_i = np.array([w[w >= v].mean() for w, v in zip(windows, var_99_i)])

plt.figure(figsize=(12, 6))
plt.plot(target_dates, targets, 'gray', alpha=0.6, lw=0.8, label='Loss')
plt.plot(target_dates, var_95_i, 'orange', linestyle='--', label='95% VaR')
plt.plot(target_dates, es_95_i, 'orange', alpha=0.7, label='95% ES')
plt.plot(target_dates, var_99_i, 'blue', linestyle='--', label='99% VaR')
plt.plot(target_dates, es_99_i, 'blue', alpha=0.7, label='99% ES')
plt.ylim(bottom=0) # Only include positive losses
plt.title(f'HS Forecasts (N={N})')
plt.ylabel('Linearised Loss Relative to the Value')
plt.legend(loc='upper left')
plt.grid(True, alpha=0.3)
plt.savefig('assets/hs.png')
plt.show()

# Part (ii)
sigma2 = np.zeros(T)
sigma2[0] = np.var(losses[:N]) # Take variance as initial condition (DUE TO DISCUSS)

# Run EWMA
for t in range(1, T):
    sigma2[t] = (1 - 0.06) * sigma2[t-1] + 0.06 * losses[t-1]**2
sigma = np.sqrt(sigma2)
plt.figure(figsize=(12, 6))
plt.plot(dates, sigma2, 'orange', alpha=0.6, lw=0.8, label='Volatility based on EWMA
    method')
plt.title('EWMA Volatility Forecasts')
plt.legend(loc='upper left')
plt.grid(True, alpha=0.3)
plt.savefig('assets/ewma_vol.png')
plt.show()

std_residuals = losses / sigma # Assume mean is zero (DUE TO DISCUSS)
z_window = std_residuals[-500:]

plt.figure(figsize=(8, 4))
ax = sns.histplot(data = z_window, bins=50, stat = "density", color = 'gray', alpha =
    0.6, label='Empirical $F_Z$ (last 500 days)')
xs = np.linspace(z_window.min() - 1, z_window.max() + 1, 200)
plt.plot(xs, norm.pdf(xs), linewidth=2, linestyle = '--',
         label='$N(0,1)$')
plt.title('Histogram of Standardised EWMA Residuals (Approx. $F_Z$)')
plt.xlabel('Standardised residual $z$')
```

```
plt.ylabel('Density')
plt.legend()
plt.tight_layout()
plt.savefig('assets/Distribution_Fz_EWMA.png')
plt.show()

windows = sliding_window_view(std_residuals[:-1], window_shape=N)
vols = sigma[N:]

var_95_z_ii = np.quantile(windows, 0.95, axis=1)
es_95_z_ii = np.array([w[w >= v].mean() for w, v in zip(windows, var_95_z_ii)])
var_95_ii = vols * var_95_z_ii
es_95_ii = vols * es_95_z_ii

var_99_z_ii = np.quantile(windows, 0.99, axis=1)
es_99_z_ii = np.array([w[w >= v].mean() for w, v in zip(windows, var_99_z_ii)])
var_99_ii = vols * var_99_z_ii
es_99_ii = vols * es_99_z_ii

plt.figure(figsize=(12, 6))
plt.plot(target_dates, targets, 'gray', alpha=0.6, lw=0.8, label='Loss')
plt.plot(target_dates, var_95_ii, 'orange', linestyle='--', label='95% VaR')
plt.plot(target_dates, es_95_ii, 'orange', alpha=0.7, label='95% ES')
plt.plot(target_dates, var_99_ii, 'blue', linestyle='--', label='99% VaR')
plt.plot(target_dates, es_99_ii, 'blue', alpha=0.7, label='99% ES')
plt.ylim(bottom=0) # Only include positive losses
plt.title(f'FHS-EWMA Forecasts (N={N})')
plt.ylabel('Linearised Loss Relative to the Value')
plt.legend(loc='upper left')
plt.grid(True, alpha=0.3)
plt.savefig('assets/fhs_ewma.png')
plt.show()

# Part (iii)
windows = sliding_window_view(losses[:-1], window_shape=N)
var_95_iii = np.zeros(T-N)
es_95_iii = np.zeros(T-N)
var_99_iii = np.zeros(T-N)
es_99_iii = np.zeros(T-N)
for i in range(T-N):
    am = arch_model(windows[i] * 100, mean='Constant', vol='Garch', p=1, q=1, dist='
        normal')
    res = am.fit(update_freq=0, disp='off')

    std_residuals = res.std_resid

    var_95_z_iii = np.quantile(std_residuals, 0.95)
    es_95_z_iii = std_residuals[std_residuals >= var_95_z_iii].mean()

    var_99_z_iii = np.quantile(std_residuals, 0.99)
    es_99_z_iii = std_residuals[std_residuals >= var_99_z_iii].mean()

    forecast = res.forecast(horizon=1)

    mu_forecast = forecast.mean.iloc[-1].values[0] / 100.0
    sigma_forecast = np.sqrt(forecast.variance.iloc[-1].values[0]) / 100.0

    var_95_iii[i] = mu_forecast + sigma_forecast * var_95_z_iii
    es_95_iii[i] = mu_forecast + sigma_forecast * es_95_z_iii

    var_99_iii[i] = mu_forecast + sigma_forecast * var_99_z_iii
    es_99_iii[i] = mu_forecast + sigma_forecast * es_99_z_iii

last_window = losses[-N:] * 100
am_last  = arch_model(last_window, mean='Constant', vol='Garch',
                      p=1, q=1, dist='normal')
res_last = am_last.fit(update_freq=0, disp='off')
```

```
z_last = res_last.std_resid

plt.figure(figsize=(8, 4))
sns.histplot(z_last, bins=50, stat="density",
             color="gray")
sns.kdeplot(z_last, color="#1f77b4", linestyle="--", linewidth=2,
            label="KDE")

x = np.linspace(z_last.min()*1.2, z_last.max()*1.2, 400)
plt.plot(x, norm.pdf(x), color="#ff7f0e", linewidth=2, label="Gaussian")

plt.title("Histogram of Standardised GARCH Residuals (Approx. $F_Z$)")
plt.xlabel("Standardised residual $z$")
plt.ylabel("Density")
plt.legend()
plt.tight_layout()
plt.savefig('assets/Distribution_Fz_GARCH.png')
plt.show()

plt.figure(figsize=(12, 6))
plt.plot(target_dates, targets, 'gray', alpha=0.6, lw=0.8, label='Loss')
plt.plot(target_dates, var_95_iii, 'orange', linestyle='--', label='95% VaR')
plt.plot(target_dates, es_95_iii, 'orange', alpha=0.7, label='95% ES')
plt.plot(target_dates, var_99_iii, 'blue', linestyle='--', label='99% VaR')
plt.plot(target_dates, es_99_iii, 'blue', alpha=0.7, label='99% ES')
plt.ylim(bottom=0) # Only include positive losses
plt.title(f'FHS-GARCH Forecasts (N={N})')
plt.ylabel('Linearised Loss Relative to the Value')
plt.legend(loc='upper left')
plt.grid(True, alpha=0.3)
plt.savefig('assets/fhs_garch.png')
plt.show()

# Backtesting
# Violations
v_95_i = targets > var_95_i
v_99_i = targets > var_99_i
v_95_ii = targets > var_95_ii
v_99_ii = targets > var_99_ii
v_95_iii = targets > var_95_iii
v_99_iii = targets > var_99_iii

def backtest(violations, alpha):

    T = len(violations)
    N1 = np.sum(violations)
    N0 = T - N1
    eps = 1e-15 # Prevent divide by zero

    # Unconditional Coverage
    pi = N1 / T

    logL_H0 = N0 * np.log(alpha) + N1 * np.log(1-alpha)
    logL_H1 = N0 * np.log(1 - pi) + N1 * np.log(pi)
    LR_uc = -2 * (logL_H0 - logL_H1)
    p_uc = 1 - chi2.cdf(LR_uc, 1)

    # Independence
    t00, t01, t10, t11 = 0, 0, 0, 0
    for i in range(1, T):
        if not violations[i-1] and not violations[i]: t00 += 1
        if not violations[i-1] and violations[i]: t01 += 1
        if violations[i-1] and not violations[i]: t10 += 1
        if violations[i-1] and violations[i]: t11 += 1

    pi01 = t01 / (t00 + t01)
    pi11 = t11 / (t10 + t11)
```

```
629
630     logL_H0 = (t00 + t10) * np.log(1 - pi + eps) + (t01 + t11) * np.log(pi + eps)
631     logL_H1 = t00 * np.log(1 - pi01 + eps) + t01 * np.log(pi01 + eps) + t10 * np.log(1 -
            pi11 + eps) + t11 * np.log(pi11 + eps)
632
633     LR_ind = -2 * (logL_H0 - logL_H1)
634     p_ind = 1 - chi2.cdf(LR_ind, 1)
635
636     # Joint
637     LR_cc = LR_uc + LR_ind
638     p_cc = 1.0 - chi2.cdf(LR_cc, 2)
639
640     return LR_uc, p_uc, LR_ind, p_ind, LR_cc, p_cc
641
642 models = [
643     ('HS', '95%', v_95_i, 0.95),
644     ('HS', '99%', v_99_i, 0.99),
645     ('FHS-EWMA', '95%', v_95_ii, 0.95),
646     ('FHS-EWMA', '99%', v_99_ii, 0.99),
647     ('FHS-GARCH', '95%', v_95_iii, 0.95),
648     ('FHS-GARCH', '99%', v_99_iii, 0.99),
649 ]
650 results_uc = []
651 results_ind = []
652 results_cc = []
653 for name, alpha_str, violations, alpha_val in models:
654     LR_uc, p_uc, LR_ind, p_ind, LR_cc, p_cc = backtest(violations, alpha_val)
655     results_uc.append({
656         'Model': name,
657         'alpha': alpha_str,
658         'LR_uc': LR_uc,
659         'p_uc': p_uc
660     })
661     results_ind.append({
662         'Model': name,
663         'alpha': alpha_str,
664         'LR_ind': LR_ind,
665         'p_ind': p_ind
666     })
667     results_cc.append({
668         'Model': name,
669         'alpha': alpha_str,
670         'LR_cc': LR_cc,
671         'p_cc': p_cc
672     })
673 df_results_uc = pd.DataFrame(results_uc)
674 df_results_ind = pd.DataFrame(results_ind)
675 df_results_cc = pd.DataFrame(results_cc)
676 print(df_results_uc.to_string(index=False))
677 print(df_results_ind.to_string(index=False))
678 print(df_results_cc.to_string(index=False))
679
680 # Function for backtest ES for different model
681 def es_backtest(loss, var, es):
682     #Indicator function for whether the loss exceeding the VaR
683     I = (loss > var).astype(float)
684     # value of loss compared to ES forecast
685     ex1 = (loss - es) * I
686     sum_ex1  = np.sum(ex1)
687     sum_ex2 = np.sum(ex1**2)
688     #Standard normal test statistics
689     Z = sum_ex1 / np.sqrt(sum_ex2)
690     #  p-value
691     p = 1 - norm.cdf(Z)
692     return {'Test Statistics': Z,'p-value': p}
693
694 # Create a data frame for backtest ES result
```

```
ES_backtest = pd.DataFrame({"HS-ES-95": es_backtest(losses[500:],var_95_i,es_95_i),
                        "HS-ES-99": es_backtest(losses[500:],var_99_i,es_99_i),
                        "FHS-EWMA_ES-95": es_backtest(losses[500:],var_95_ii,es_95_ii),
                        "FHS-EWMA_ES-99": es_backtest(losses[500:],var_99_ii,es_99_ii),
                        "FHS-GARCH_ES-95": es_backtest(losses[500:],var_95_iii,es_95_iii),
                        "FHS-GARCH_ES-99": es_backtest(losses[500:],var_99_iii,es_99_iii)
                            }).T
print(ES_backtest)
```