Model comparison to forecast stocks volatility

Federico Argilli

2023-06-28

Introduction

Time series analysis plays a crucial role in various fields, ranging from finance and economics to engineering. Understanding and accurately modeling the behavior of time-dependent data is essential for making informed decisions, predicting future outcomes, and managing risks.
One of the key components in time series analysis is the estimation of volatility or variance, which quantifies the degree of fluctuation and uncertainty present in the data.

Volatility serves as a proxy for risk. Higher volatility implies greater price fluctuations, indicating a higher level of uncertainty and potential for larger gains or losses. It also help in determining the correlation between different stocks or assets in a portfolio. By including assets with low or negatively correlated volatility, investors can reduce the overall risk of their portfolio.

In this project are compared two widely used models: **ARCH** (*Autoregressive Conditional Heteroscedasticity*) and **GARCH** (*Generalized Autoregressive Conditional Heteroscedasticity*).
Both models provide powerful tools for capturing the volatility dynamics and incorporating past information to predict future variance. After comparing their suitability for modeling and forecasting volatility in read-world data, the analysis focuses on the simulation control.

Data

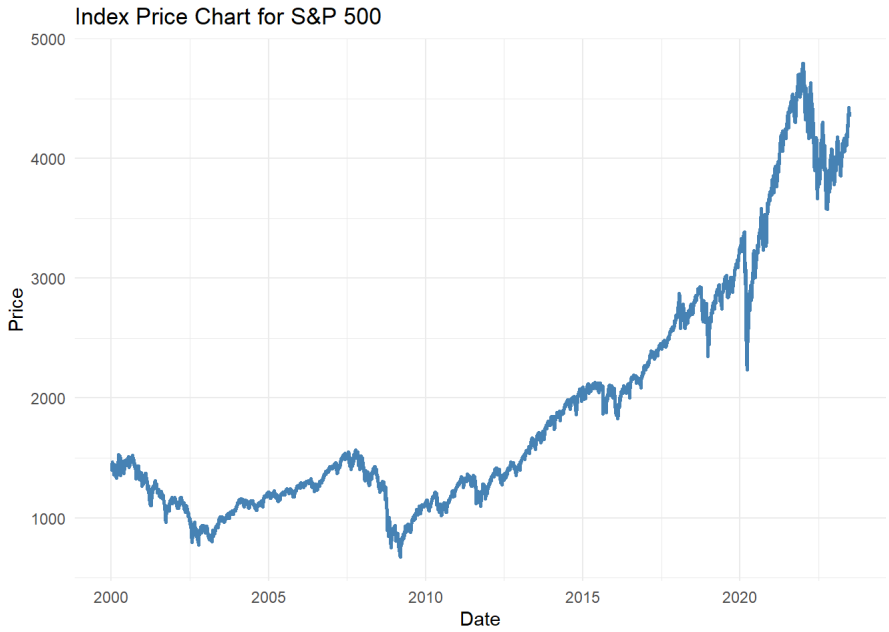
Data consists of the *daily (closing) price* of the S&P 500 from 1th January 2000 to 24th June 2023.
The S&P 500 Index, or Standard & Poor's 500 Index, is a market-capitalization-weighted index of 500 leading publicly traded companies in the U.S.

The data used in this project is provided by yahoo finance (https://finance.yahoo.com/quote/%5EGSPC/history/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAA-b-lAPv__it2eTy6bL4K_4BfpRfLd7rmsEnesA_DAZzaeZQUsbH_F9HDkWL7BWAHaibHanzrxZNGaUU_UoRkUmQfmOHENk28GTq7cJCfbvQY7z5qJEdVeUGoji8jyS_vk_vg4OyPC2SXX314VTfoKoXut_v5LXc) and it is publically available.

```
#devtools::install_github("rsquaredacademy/yahoofinancer")
library(yahoofinancer)

sp500 <- Index$new('^GSPC')
price_data <- sp500$get_history(start = '2000-01-01', end = '2023-06-24',interval = '1d')
```

date	volume	high	low	open	close	adj_close
2000-01-03 14:30:00	931800000	1478.00	1438.36	1469.25	1455.22	1455.22
2000-01-04 14:30:00	1009000000	1455.22	1397.43	1455.22	1399.42	1399.42
2000-01-05 14:30:00	1085500000	1413.27	1377.68	1399.42	1402.11	1402.11
2000-01-06 14:30:00	1092300000	1411.90	1392.10	1402.11	1403.45	1403.45
2000-01-07 14:30:00	1225200000	1441.47	1400.73	1403.45	1441.47	1441.47
2000-01-10 14:30:00	1064800000	1464.36	1441.47	1441.47	1457.60	1457.60

The high price is the highest price at which a stock was traded , the low price is the lowest price at which it was traded and so on. In this analysis the *closing price* (price at which trading ended for the day) is considered the index price for each day.



However, we are not interested in the price values themselves. From the price data we can extract the **daily returns** at day *t* defined as:

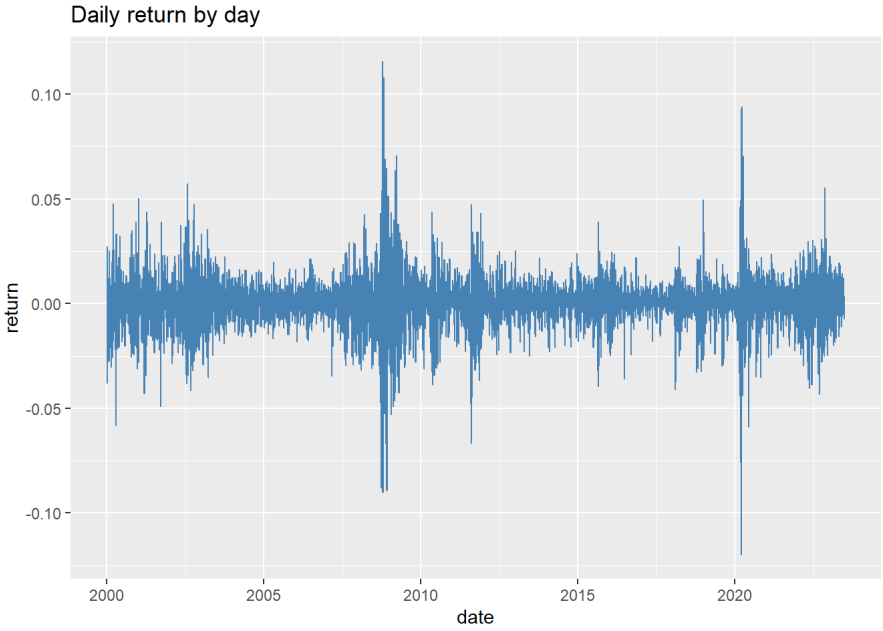
$$R_t = \frac{P_t - P_{t-1}}{P_{t-1}}$$

```
calculateReturns <- function(prices) {
  n <- length(prices)
  returns <- numeric(n - 1)
  for (i in 2:n) {
    returns[i - 1] <- (prices[i] - prices[i - 1]) / prices[i - 1]
  }

  returns
}

returns <- calculateReturns(price_data$close)
```

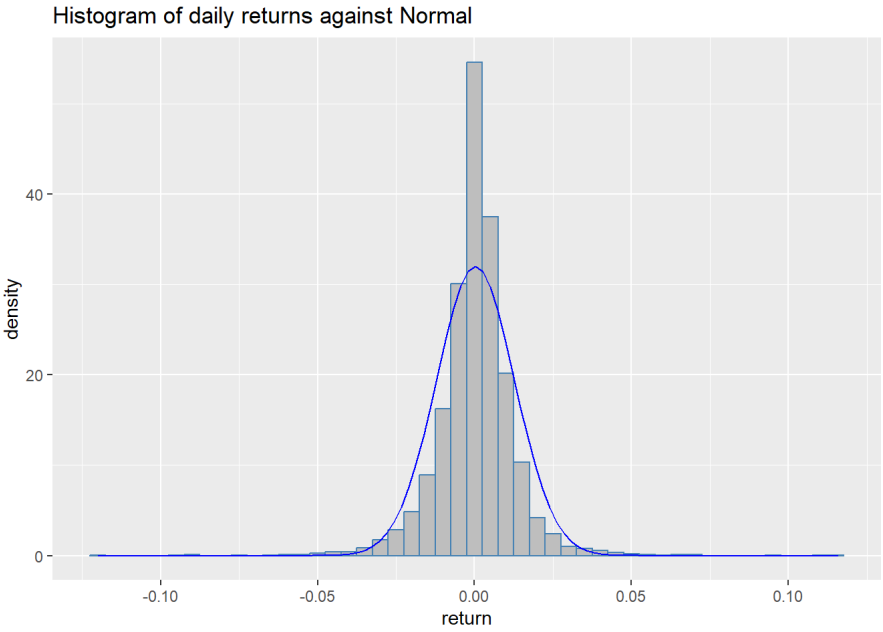
We can also have a quick look at the return distribution:

```
mean(returns)
```

```
## [1] 0.0002630196
```

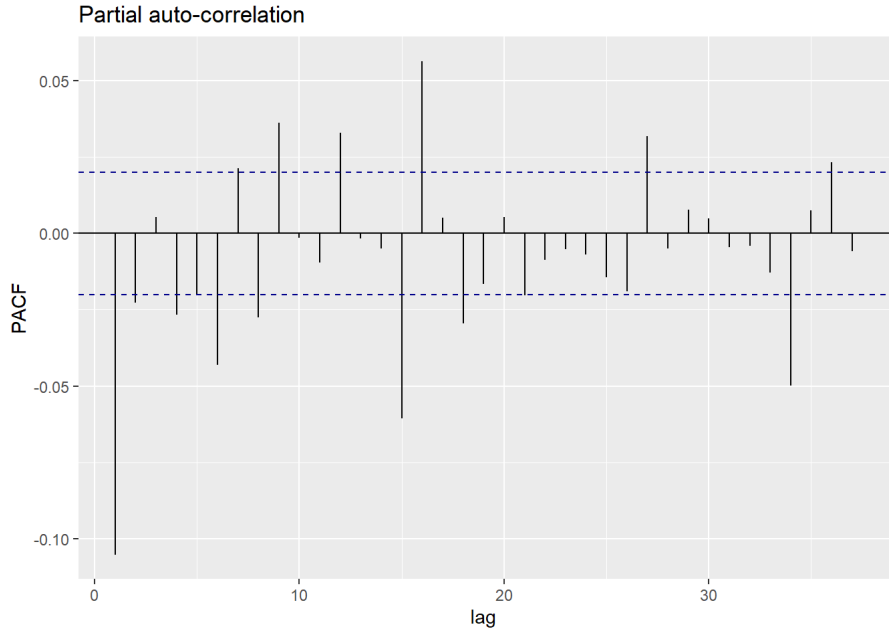


Approximately the mean is 0, but the volatility seems to be random.In the following, I will plot the histogram of the return and compare it with normal distribution with observed mean an standard deviation.



As we can see from the plot the return is not symmetrically distributed, rather it is positive skewed. A positive skewness indicating on average it gives us a positive (small) return. Furthermore, there are higher probabilities of extremely large and small returns respect to a Normal distribution.

It is also worth looking into the partial autocorrelation of the data as it representsa heuristic for choosing the q parameter. This because the partial autocorrelation of the underlying AR(q) (Autoregressive process) equals zero at lags larger than q[1].The plot has a single spike at lag 1 suggesting and ARCH(1) model.



Models

Both ARCH and GARCH are a particular type of Autoregressive (AR) model for describing the error variance.

ARCH(q): *Autoregressive Conditional Heteroscedasticity*[2].
This model was proposed in 1982 by Robert Engle to describes the variance of the current error term or innovation as a function of the actual sizes of the previous time periods' error terms.
In practice, only rather rich ARCH parametrizations are able to fit financial series adequately.

$$y_t = c + \epsilon_t = c + \sqrt{\sigma_t^2}\, z_t, \quad z_t \sim D(0,1)$$
$$\sigma_t^2 = \omega + \alpha_1\, \epsilon_{t-1}^2 + \ldots + \alpha_q\, \epsilon_{t-q}^2$$

In this analysis D is a Standard Normal as it represent the base case in the literature and $q = 1$ for the reasons discussed before regarding partial autocorrelation. The resulting model is:

$$y_t \sim N(c,\, \sigma_t^2)$$
$$\sigma_t^2 = \omega + \alpha\, \epsilon_{t-1}^2$$

Thus corresponding code:

```
model
{
  # Likelihood
  for (t in 1:T) {
    y[t] ~ dnorm(c, tau[t])
    tau[t] <- 1/pow(sigma[t], 2)
  }
  sigma[1] <- 1
  for(t in 2:T) {
    sigma[t] <- sqrt( omega + alpha * pow(y[t-1] - c, 2) )
  }

  # Priors
  c ~ dnorm(0.0, 0.1)
  omega ~ dunif(0, 10)
  alpha ~ dunif(0, 1)
}
```

GARCH(q,p) (*Generalized Autoregressive Conditional Heteroscedasticity*).
It was proposed later by Bollerslev (1986) as a generalized version of the ARCH model that also takes into account the lagged conditional variances.

$$y_t = c + \epsilon_t = c + \sqrt{\sigma_t^2}\, z_t, \quad z_t \sim D(0,1)$$
$$\sigma_t^2 = \omega + \alpha_1\, \epsilon_{t-1}^2 + \ldots + \alpha_q\, \epsilon_{t-q}^2 + \beta_1 \sigma_{t-1}^2 + \ldots + \beta_p \sigma_{t-p}^2$$

In this case I'm going to consider a GARCH(1,1) model with a Standard Normal as D.

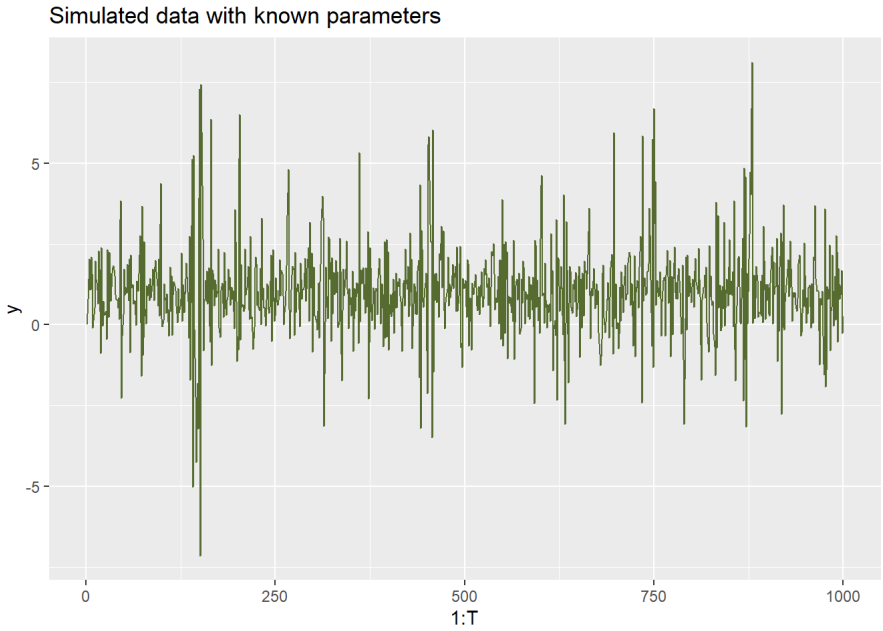
$$y_t \sim N(c,\, \sigma_t^2)$$
$$\sigma_t = \omega + \alpha\, \epsilon_{t-1}^2 + \beta\, \sigma_{t-1}^2$$

```
model
{
  # Likelihood
  for (t in 1:T) {
    y[t] ~ dnorm(c, tau[t])
    tau[t] <- 1/pow(sigma[t], 2)
  }
  sigma[1] <- 1
  for(t in 2:T) {
    sigma[t] <- sqrt( omega + alpha * pow(y[t-1] - c, 2) + beta * pow(sigma[t-1], 2) )
  }

  # Priors
  c ~ dnorm(0.0, 0.01)
  omega ~ dunif(0, 10)
  alpha ~ dunif(0, 1)
  beta ~ dunif(0, 1)
}
```

Simulate data from the model

What follows is a quick and easy check that the models formulated up to this point is capable of returning the correct (known) parameters of simulated data through Gibbs Sampling.
Fot time purposes it only uses the ARCH(1) model but the same result could be obtained with the GARCH one.

```
set.seed(123)
T <- 1000
c <- 1
alpha <- 1
omega <- 0.4
sigma <- y <- rep(1, length = T)
y[1] <- 0

for (t in 2:T) {
  sigma[t] <- sqrt(omega + alpha * (y[t - 1] - c)^2)
  y[t] <- rnorm(1, mean = c, sd = sigma[t])
}
```



Setting up a simple JAGS simulation with the following code:

```
model_data <- list(T = T, y = y)

fake_data_jags <- jags(data=mydata,
               parameters.to.save=parameters,
               model.file = textConnection(arch_model_code),
               n.chains=3,
               n.iter=10000,
               n.burnin = 2000
)
```

The obtained parameters are pretty close to the true parameters defined above.
We can compare them with the original ones by the means of HPD intervals resulting from the posterior distribution obtained by the model. As we can see they are all contained in the HPD interval at level 0.95.

```
#library(MCMCvis)
as.data.frame(MCMCsummary(fake_data_jags,
              params = c("c", "omega", "alpha"),
              HPD = TRUE,
              hpd_prob = 0.95,
              round = 2))
```

	mean	sd	95%_HPDL	95%_HPDU	Rhat	n.eff
c	1.02	0.02	0.97	1.06	1	3000
omega	0.41	0.03	0.35	0.47	1	3007
alpha	0.93	0.05	0.83	1.00	1	2981

```
# c <- 1
# alpha <- 1
# omega <- 0.4
```

JAGS Model choice

Now that we check the feasibility for the simulated data, it's time to compare the models on the real-world data taken into exam. In both cases the parameters for the simulation were the following:

```
mydata <- list(   T = T,
                  y = returns)

# arch_parameters <- c("c", "omega", "alpha")
# garch_parameters <- c("c", "omega", "alpha", "beta")

jags(data=mydata,
         parameters.to.save=parameters,
         model.file = textConnection(model_code),
         n.chains=3,
         n.iter=10000,
         n.burnin = 2000,
         n.thin = 8
)
```

In this analysis I took into account the DIC as the most effective and easy to implement way to compare the models. The deviance information criterion (DIC) was introduced by Spiegelhalter et al. as a measure of model comparison and adequacy[3] in the sense that smaller DIC values indicate a better-fitting model.

```
print(paste("ARCH DIC: ", archjags$BUGSoutput$DIC))
```

```
## [1] "ARCH DIC:  -35946.5927590358"
```

```
print(paste("GARCH DIC: ", garchjags$BUGSoutput$DIC))
```

```
## [1] "GARCH DIC:  -37970.5174905606"
```

After the two simulations, comparing the models on the DIC led to the conclusion that the **GARCH(1,1)** model fits better the data. It is important to note that this outcome does not imply a general superiority of the GARCH model.
Subsequently, the focus of the report centers on analyzing and exploring the specific performance of the GARCH model within the context of this dataset.

GARCH(1,1) deeper analysis

Now we can dive deeper into the simulation result on the GARCH model. The resulting simulation has the following features:

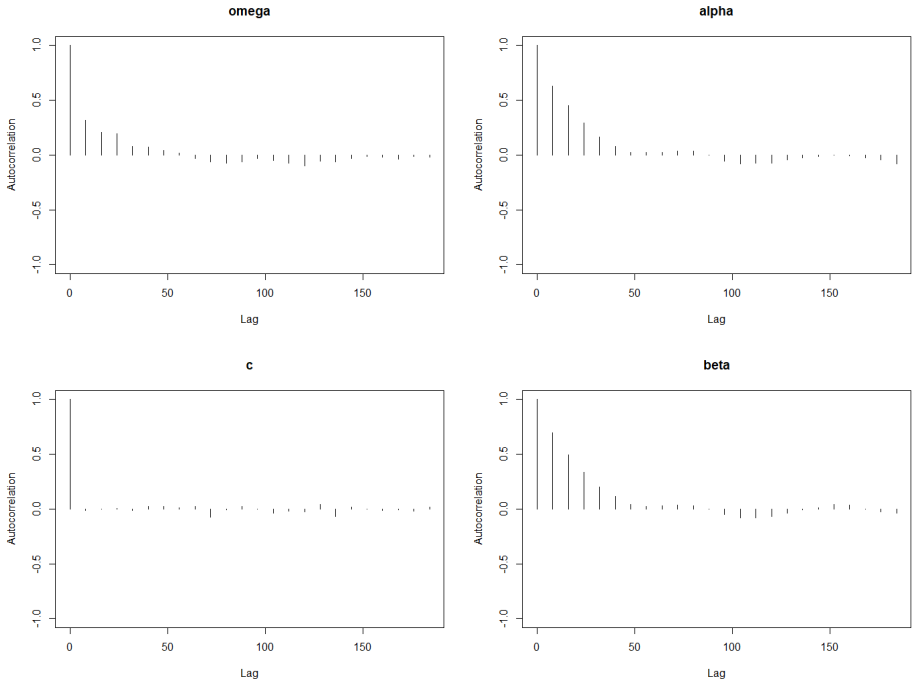
```
garchjags$BUGSoutput$n.sims
```

```
## [1] 3000
```

```
as.data.frame(garchjags$BUGSoutput$summary[c(1:3,5),])
```

	mean	sd	2.5%	25%	50%	75%	97.5%	Rhat	n.eff
alpha	0.1250656	0.0093919	0.1081548	0.1185703	0.1247375	0.1312841	0.1448345	1.006101	360
beta	0.8589047	0.0096657	0.8393951	0.8525019	0.8592679	0.8655539	0.8770361	1.007014	310
c	0.0006405	0.0001041	0.0004345	0.0005718	0.0006398	0.0007111	0.0008422	1.003200	740
omega	0.0000024	0.0000003	0.0000018	0.0000021	0.0000023	0.0000025	0.0000030	1.004161	550

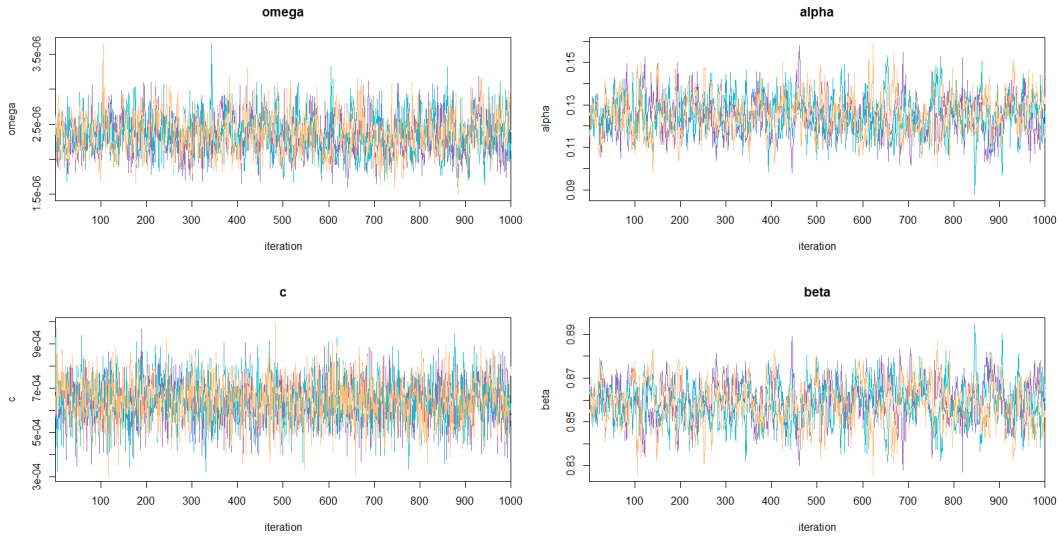
The observed autocorrelation appears relatively elevated for smaller lag values, indicating a potentially longer convergence time for the chain to reach the target distribution. Nevertheless, an analysis of the R hat values suggests that the simulation is at least in close proximity to convergence.



In addition, it is worth noting that the Monte Carlo Standard Error (MCSE) associated with these parameters is exceptionally low. The MCSE represents the standard deviation of the parameter estimates obtained from the Monte Carlo simulation.
The MCSE computed througt the `LaplacesDemon` library is:

	MCSE_error
c	0.0000031
omega	0.0000000
alpha	0.0008422
beta	0.0009176

Let's also have a look at the traceplots. The overlap indicates that the chains are exploring the parameter space similarly and are converging towards a common target distribution.



Comparison with the frequentist estimations

I wanted also to compare this approach based on Gibbs sampling with a frequentist approach based on ML estimators for the parameters of interest. The mean results obtained are the following:

```
#Library(rugarch)
spec <- ugarchspec() # base GARCH(1,1)
fit <-  ugarchfit(spec = spec, data = returns)

found_parameters <- fit@fit$coef[c("mu","omega","alpha1","beta1")]
as.data.frame(found_parameters)
```

	found_parameters
mu	0.0006551
omega	0.0000022
alpha1	0.1205117
beta1	0.8642471

The consistency in the estimated results, measured through HPD intervals, suggests that both frameworks provide similar and reliable estimates for the parameters of interest.
This convergence in the estimated intervals is reassuring as it strengthens the confidence in the accuracy and robustness of the results.

```
as.data.frame(MCMCsummary(garchjags,
              params = c("c", "omega", "alpha","beta"),
              HPD = TRUE,
              hpd_prob = 0.95,
              round = 5))
```

	mean	sd	95%_HPDL	95%_HPDU	Rhat	n.eff
c	0.00064	0.00010	0.00044	0.00085	1.00	3000
omega	0.00000	0.00000	0.00000	0.00000	1.00	974
alpha	0.12507	0.00939	0.10737	0.14374	1.01	693
beta	0.85890	0.00967	0.84084	0.87807	1.01	564

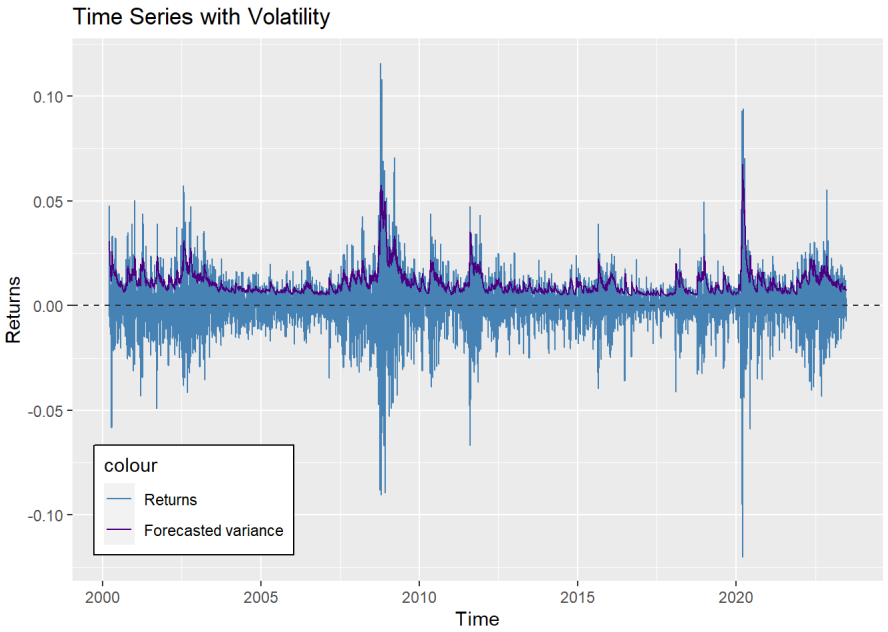
Forecasting

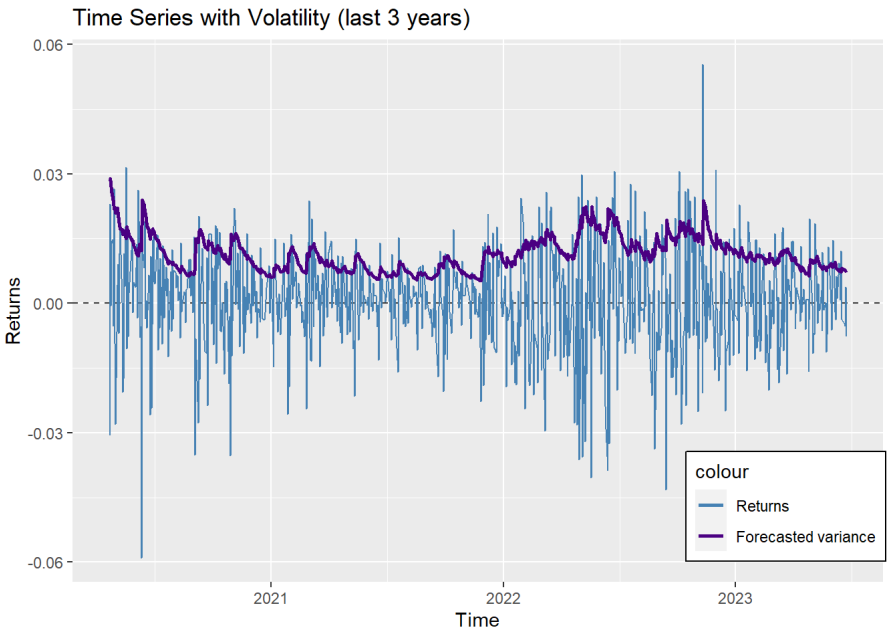
At last I wanted to use the GARCH model for it's original purpose: predicting variance.
The process involves starting with the last observed conditional error and variance and recursively updating it using the estimated GARCH model's equation. The resulting forecasted variances represent the expected volatility for each future period.

Plotting the forecasted variance against the actual returns provides a visual representation of its dynamics, allowing for the identification of patterns, trends, or changes in volatility over time.

It's visible by the plot that periods of high volatility in the returns, characterized by substantial oscillations, correspond to increases in the predicted variance.
This ability to forecast the variance helps make more informed decisions as provides valuable insights for risk assessment, portfolio management and financial planning.

```
c <- as.numeric(garchjags$BUGSoutput$median["c"])
omega <- as.numeric(garchjags$BUGSoutput$median["omega"])
alpha <- as.numeric(garchjags$BUGSoutput$median["alpha"])
beta <- as.numeric(garchjags$BUGSoutput$median["beta"])


sigma_g <- rep(1,length(returns))
for (t in 2:length(returns)) {
  sigma_g[t] <- sqrt(omega + alpha * (returns[t - 1] - c)^2 + beta * sigma_g[t - 1]^2)
}
```





1. https://en.wikipedia.org/wiki/Autoregressive_model (https://en.wikipedia.org/wiki/Autoregressive_model)↩
2. https://en.wikipedia.org/wiki/Autoregressive_conditional_heteroskedasticity (https://en.wikipedia.org/wiki/Autoregressive_conditional_heteroskedasticity)↩
3. Chapter 4, Ntzoufras (2010)↩