Building a Machine Learning-based Volatility Prediction Model

Maya Le, Jason Yu

Winter 2024

Contents

1 Introduction

"Volatility is the backbone of finance in the sense that it not only provides an information signal to investors, but it also is an input to various financial models" (Karasan, n.d) 4.

Volatility can be seen as uncertainty or risk in the financial market. High-volatility stocks have prices that fluctuate greater and more frequently, and higher stock market volatility often translates to greater investment risks. As a result, modelling and forecasting volatility becomes an important strategy when navigating the stock market.

This project will explore the different models of volatility prediction, first, using GARCH(p,q) model, and then LSTM-GARCH hybrid models. This study therefore considers the volatility of the data_ml stock index returns.

Many of the codes and methods used in this report, especially data processing, was referred from the Machine Learning for Factor Investing textbook by Guillaume Coquerete and Tony Guida.

2 The Data Set

As mentioned in the book "Machine Learning for Factor Investing", this data set comprises information on 1,207 stocks listed in the US (possibly originating from Canada or Mexico). The time range starts in November 1998 and ends in March 2019. For each point in time, 93 characteristics describe the firms in the sample.

2.1 Data Pre-processing

First, we will perform data pre processing.

```
setwd("C:/Users/maian/OneDrive - University of Waterloo/Documents/UW/W24/AFM423/Homeworks")
load("data_ml.RData")
```

Data will be grouped by date, and arrange by stock id. We also wish to keep a particular period of time, from December 31, 1999 to January 01, 2019 for modelling purposes.

```
data_ml %>%
    group_by(date)
```

```
## # A tibble: 283,380 x 99
## # Groups:   date [245]
##    stock_id date       Advt_12M_Usd Advt_3M_Usd Advt_6M_Usd Asset_Turnover
##       <int> <date>            <dbl>       <dbl>       <dbl>          <dbl>
##  1       13 2006-12-31         0.25        0.33        0.27           0.22
##  2       13 2007-01-31         0.25        0.32        0.28           0.22
##  3       13 2007-02-28         0.26        0.3         0.3            0.22
##  4       17 2015-03-31         0.73        0.64        0.7            0.4
##  5       17 2015-04-30         0.72        0.62        0.66           0.4
##  6       17 2015-05-31         0.71        0.63        0.64           0.4
##  7       17 2015-06-30         0.7         0.62        0.63           0.38
##  8       17 2015-07-31         0.68        0.56        0.6            0.38
##  9       17 2015-08-31         0.67        0.47        0.57           0.38
## 10       17 2015-09-30         0.64        0.41        0.54           0.35
## # i 283,370 more rows
```

```
## # i 93 more variables: Bb_Yld <dbl>, Bv <dbl>, Capex_Ps_Cf <dbl>,
## #   Capex_Sales <dbl>, Cash_Div_Cf <dbl>, Cash_Per_Share <dbl>, Cf_Sales <dbl>,
## #   Debtequity <dbl>, Div_Yld <dbl>, Dps <dbl>, Ebit_Bv <dbl>, Ebit_Noa <dbl>,
## #   Ebit_Oa <dbl>, Ebit_Ta <dbl>, Ebitda_Margin <dbl>, Eps <dbl>,
## #   Eps_Basic <dbl>, Eps_Basic_Gr <dbl>, Eps_Contin_Oper <dbl>, Eps_Dil <dbl>,
## #   Ev <dbl>, Ev_Ebitda <dbl>, Fa_Ci <dbl>, Fcf <dbl>, Fcf_Bv <dbl>, ...
```

```
data_ml <- data_ml %>%
    filter(date > "1999-12-31",          # Keep the date with sufficient data points
           date < "2019-01-01") %>%
    arrange(stock_id, date)
```

Furthermore, for computation purpose, we will purge the data set of missing values, and keep only the stocks in which we have a complete set of observations.

```
stock_ids <- levels(as.factor(data_ml$stock_id)) # A list of all stock_ids
stock_days <- data_ml %>%                        # Compute the number of data points per stock
    group_by(stock_id) %>% summarize(nb = n())
stock_ids_short <- stock_ids[which(stock_days$nb == max(stock_days$nb))] # Stocks with full data
returns <- data_ml %>%                           # Compute returns, in matrix format, in 3 steps:
    filter(stock_id %in% stock_ids_short) %>%    # 1. Filtering the data
    dplyr::select(date, stock_id, R1M_Usd) %>%   # 2. Keep returns along with dates & firm names
    pivot_wider(names_from = "stock_id",
                values_from = "R1M_Usd")         # 3. Put in matrix shape
```

We also split the data set into training and testing set for later use

```
separation_date <- as.Date("2014-01-15")
training_sample <- filter(returns, date < separation_date)
testing_sample <- filter(returns, date >= separation_date)
```

For the purpose of modelling volatility, rather than using the whole set of explanatory variates, we will focus explicitly on R1M_Usd, which is the return forward 1 month.

2.2 Portfolio Returns

Let's take a look at the portfolio as a whole. Suppose we want to calculate the portfolio returns of N assets. The value of asset i, Vi, in the portfolio is defined as $Vi = \lambda_i * P_i$.

Here, λ_i is the number of shares of asset i, and Pi is the price of asset i.

The total portfolio value, V_p is defined as

$$V_p = \Sigma_{i=1}^{N} V_i$$

.

The weight of asset i, w_i, in the portfolio is then defined as

$$w_i = \frac{V_i}{V_p}$$

where V_i is the value of asset i and V_p is the total value of the portfolio.

The portfolio return at time t, R_t is defined as

$$R_t = \frac{V_{pi} - V_{pt-1}}{V_{pt-1}}$$

where V_{pt} is the portfolio value at time t.

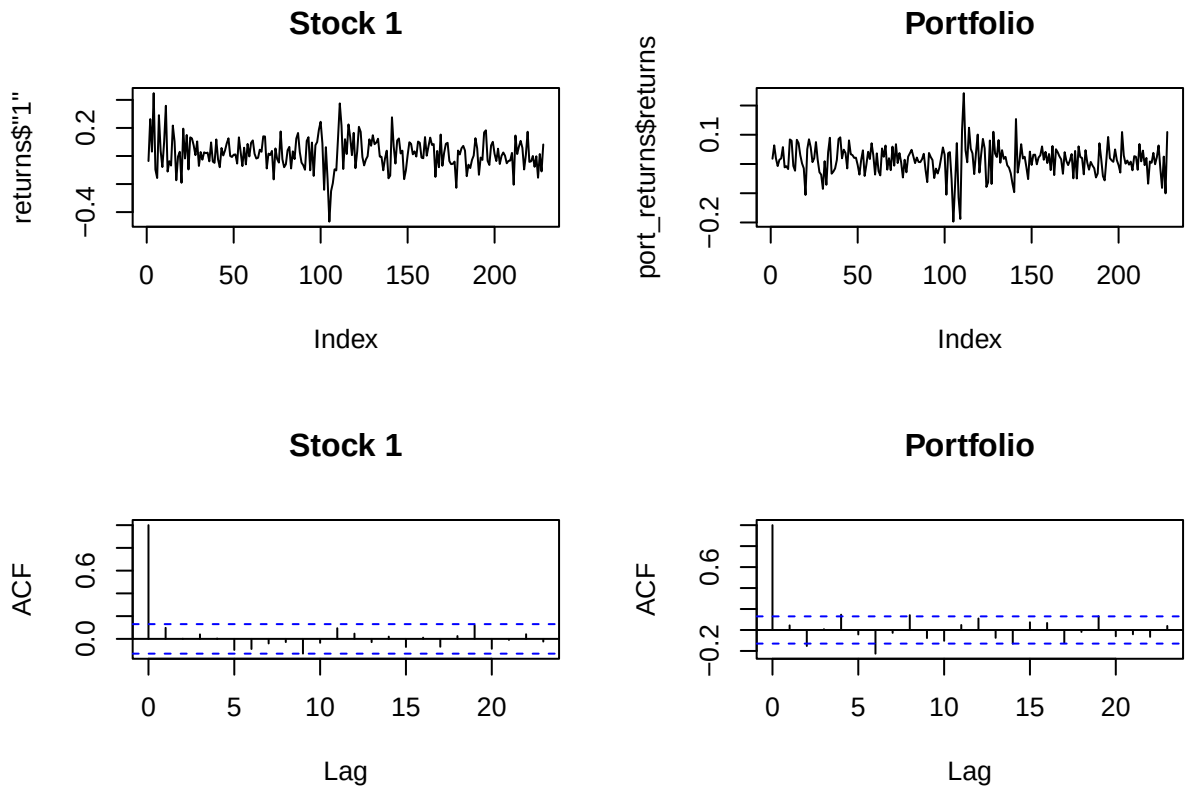
We will assume that we have an equal-weight portfolio, in which each assets has equal weights, such that weight for asset i is given as:

$$w_i = \frac{1}{N}$$

```
N <- ncol(returns)-1
weights <- 1/N
port_ret <- (rowSums(returns[,-1])*weights)
port_returns <- data.frame(date = returns$date, returns = port_ret)
```

Now, let's look at stock 1 returns and the portfolio average returns time series.

```
par(mfrow=c(2,2))
plot(returns$"1", type = "l", main = "Stock 1")
plot(port_returns$returns, type = "l", main = "Portfolio")
acf(returns$"1", main = "Stock 1")
acf(port_returns$returns, main = "Portfolio")
```



Overall, the acf indicates that time series are relatively stationary. However, the variance seems non-constant. Thus, this suggests a GARCH(p,q) model. First, we will try GARCH model to the average returns, then to the 793 stocks. We aim to find periods of high volatility or low volatility.

3 The GARCH(p,q) Approach

The financial market reacts greatly to stress sources, such as economic crises, political changes, etc, and prices of financial assets can fluctuate as a result. In statistic, we measure volatility using variance. Since we want to use the past history to forecast said variance,we are particularly interested in the conditional variance, denoted by

$$Var(rt|r_{t-1}, r_{t-2}, ...) = E(r_t^2|r_{t-1}, r_{t-2}, ...)$$

Intuitively, volatility becomes higher during stressful periods, which could take several periods for the market to become stable again. Therefore, high variance at time t can cause high variance at following times $t+1, t+2, ...$

3.1 ARCH(p)

An AutoRegressive Conditional Heteroscedasticity (ARCH(p)) model is defined hierarchically: first define $X_t = \sigma_t Z_t$ where $Z_t \sim^{i.i.d} N(0,1)$, but treat σ_t as being random such that

$$\sigma_t^2 = \alpha_0 + \alpha_1 X_{t-1}^2 + ... + \alpha_p X_{t-p}^2$$

Here, it can be seen that the variance is time dependent = a large value of X_t will result in period of high volatility.
As with all forecasting problems, we want to have a stationary time series. Sources of non-stationarity includes trend, seasonality, and non-constant variance. While SARIMA models trend and seasonality, ARCH/GARCH models sources of non-constant variance. For an ARCH(1) model, when $|\alpha_1| < 1$ there exists a unique causal stationary solution.

3.2 GARCH(p,q)

The GARCH(p,q) model is a generalized version of ARCH(p), which is defined by $X_t = \sigma_t Z_t$ where $Z_t \sim N(0,1)$ i.i.d and

$$\sigma_t^2 = \alpha_0 + \Sigma_{i=1}^p \alpha_i X_{t-i}^2 + \Sigma_{i=1}^q \beta_i \sigma_{t-i}^2$$

3.3 GARCH(1,1)

Suppose we want to fit a GARCH(1,1) process into the market average return.

```
Model <- ugarchspec(variance.model = list(model = "sGARCH",
garchOrder = c(1, 1) ),mean.model = list(armaOrder=c(0,0,0),
include.mean = FALSE) )

fit <- ugarchfit(data = port_returns$returns, spec = Model)

fit
```

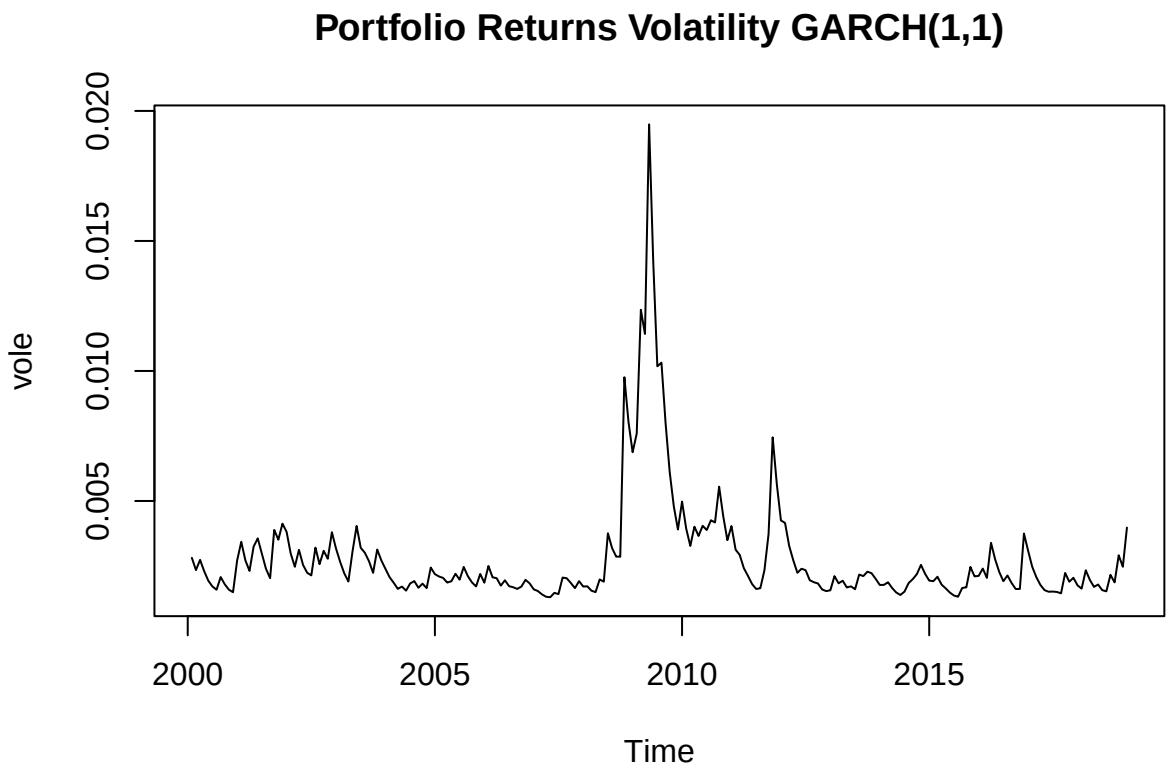
```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : sGARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : norm
##
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## omega   0.000336    0.000202   1.6641 0.096083
## alpha1  0.191631    0.076763   2.4964 0.012546
## beta1   0.691256    0.121464   5.6910 0.000000
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## omega   0.000336    0.000158   2.1284 0.033308
## alpha1  0.191631    0.082295   2.3286 0.019880
## beta1   0.691256    0.100919   6.8496 0.000000
##
## LogLikelihood : 363.9642
##
## Information Criteria
## ------------------------------------
##
## Akaike       -3.1664
## Bayes        -3.1212
## Shibata      -3.1667
## Hannan-Quinn -3.1481
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                  0.0007647  0.9779
## Lag[2*(p+q)+(p+q)-1][2] 0.9660692  0.5094
## Lag[4*(p+q)+(p+q)-1][5] 3.0994621  0.3892
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.2233  0.6365
## Lag[2*(p+q)+(p+q)-1][5]    4.1850  0.2319
## Lag[4*(p+q)+(p+q)-1][9]    8.5221  0.1016
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
```

```
## ARCH Lag[3]    0.7603 0.500 2.000 0.38322
## ARCH Lag[5]    7.5016 1.440 1.667 0.02643
## ARCH Lag[7]   10.2257 2.315 1.543 0.01631
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  0.4978
## Individual Statistics:
## omega  0.06742
## alpha1 0.09070
## beta1  0.08804
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          0.846 1.01 1.35
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## ------------------------------------
##                    t-value     prob sig
## Sign Bias           1.1479 0.252233
## Negative Sign Bias  1.3045 0.193411
## Positive Sign Bias  0.6217 0.534781
## Joint Effect       12.6560 0.005443 ***
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     37.79     0.006312
## 2    30     53.58     0.003616
## 3    40     66.04     0.004386
## 4    50     78.14     0.005098
##
##
## Elapsed time : 0.07373691
```

We can see that both alpha and beta has small p-values, which suggests that both of the GARCH terms are significant. Overall, GARCH seems to fit the time series very well.

Making a GARCH variance series,

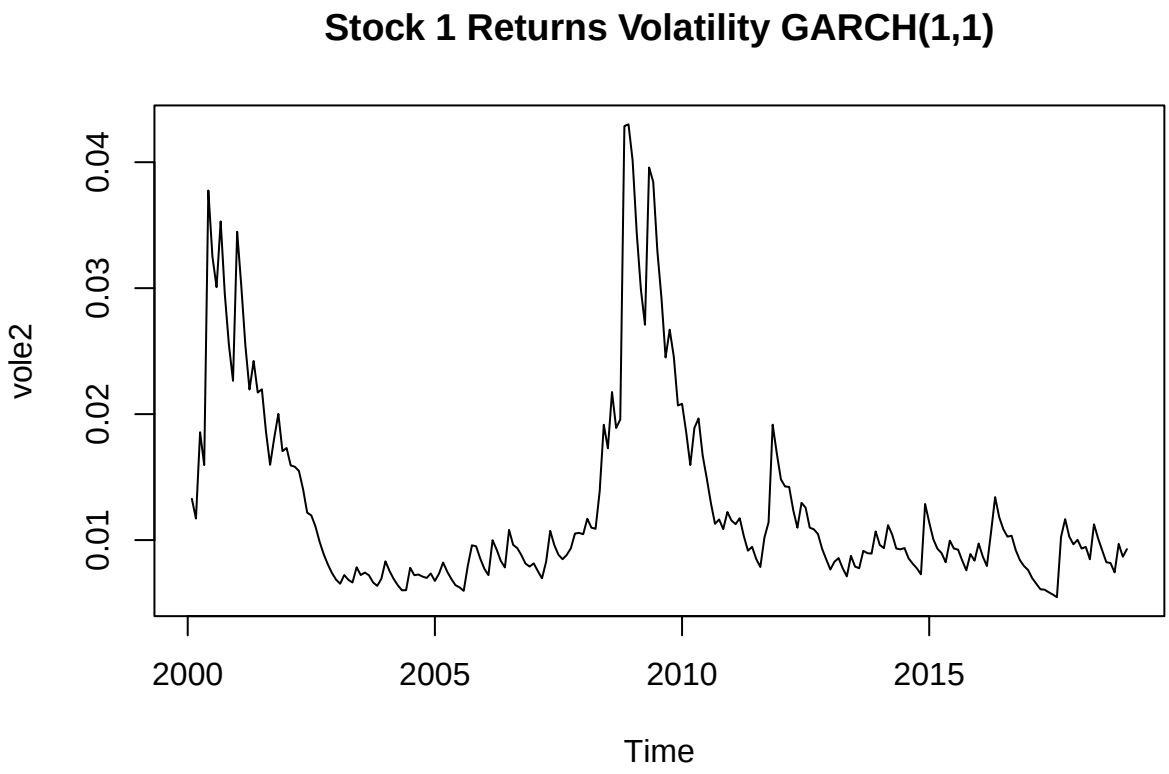
```
vole <- ts(fit@fit$sigma^2, start = 2000 +1/12, frequency = 12)

plot(vole, type = "l", main = "Portfolio Returns Volatility GARCH(1,1)")
```



We could that there is a big spike in volatility, but otherwise variance is relatively low and stable.

We can try the same model for Stock 1 just for illustration purpose:

```
fit2 <- ugarchfit(data = returns$"1", spec = Model)
vole2 <- ts(fit2@fit$sigma^2, start = 2000 +1/12,frequency = 12)
plot(vole2, type = "l", main = "Stock 1 Returns Volatility GARCH(1,1)")
```



Again, we can see big spikes indicating high volatility periods. Overall, individual stocks seems to be much more voltile than average portfolio returns.

3.4 Exponential GARCH model

We can also try fitting an exponential GARCH model - eGARCH:

```
ModelE <- ugarchspec(variance.model = list(model = "eGARCH",
garchOrder = c(1, 1) ),mean.model = list(armaOrder=c(0,0,0),
include.mean = FALSE) )

fit.e <- ugarchfit(data = port_returns$returns, spec = ModelE)

fit.e
```

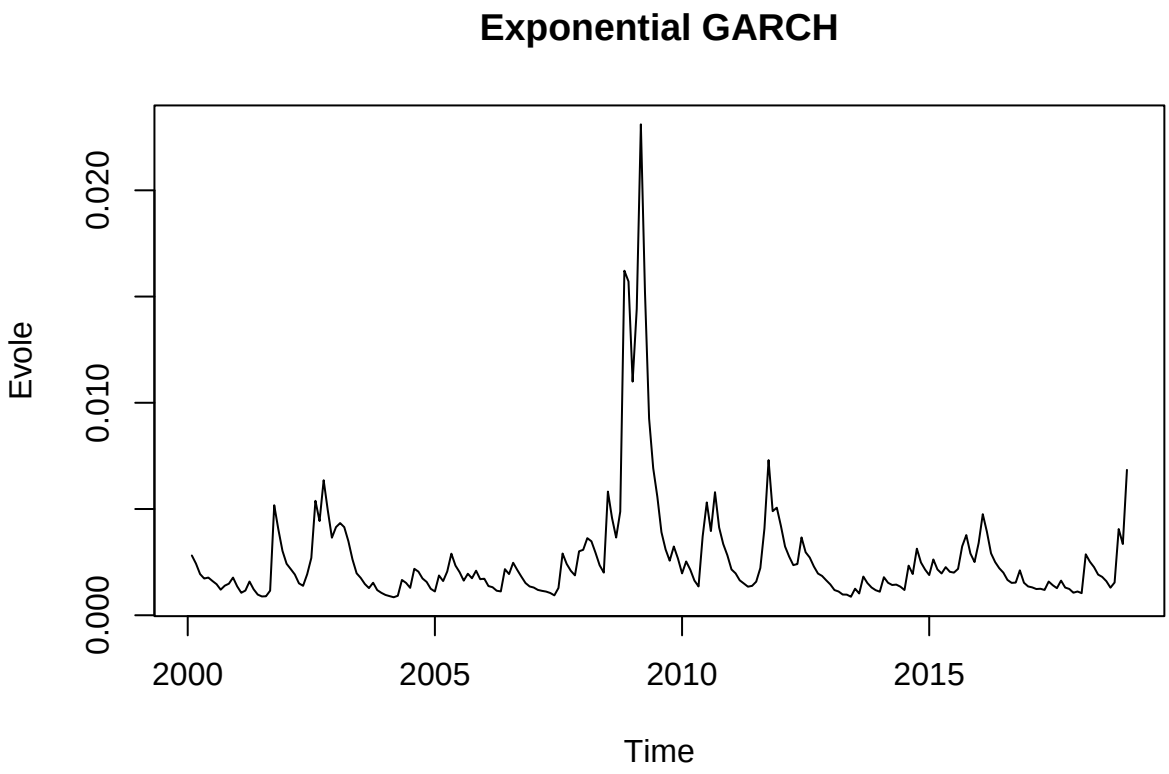
```
##
## *---------------------------------*
## *          GARCH Model Fit        *
## *---------------------------------*
##
## Conditional Variance Dynamics
## -----------------------------------
## GARCH Model  : eGARCH(1,1)
## Mean Model   : ARFIMA(0,0,0)
## Distribution : norm
##
```

```
## Optimal Parameters
## ------------------------------------
##         Estimate  Std. Error  t value Pr(>|t|)
## omega   -0.59902    0.031930 -18.7605 0.000000
## alpha1  -0.29432    0.050492  -5.8289 0.000000
## beta1    0.89076    0.006889 129.3009 0.000000
## gamma1   0.19156    0.061702   3.1046 0.001906
##
## Robust Standard Errors:
##         Estimate  Std. Error  t value Pr(>|t|)
## omega   -0.59902    0.039145 -15.3024        0
## alpha1  -0.29432    0.043082  -6.8315        0
## beta1    0.89076    0.005615 158.6460        0
## gamma1   0.19156    0.035067   5.4627        0
##
## LogLikelihood : 377.3231
##
## Information Criteria
## ------------------------------------
##
## Akaike       -3.2748
## Bayes        -3.2146
## Shibata      -3.2754
## Hannan-Quinn -3.2505
##
## Weighted Ljung-Box Test on Standardized Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.1766  0.6743
## Lag[2*(p+q)+(p+q)-1][2]    0.5592  0.6665
## Lag[4*(p+q)+(p+q)-1][5]    1.7494  0.6787
## d.o.f=0
## H0 : No serial correlation
##
## Weighted Ljung-Box Test on Standardized Squared Residuals
## ------------------------------------
##                         statistic p-value
## Lag[1]                     0.8602 0.35368
## Lag[2*(p+q)+(p+q)-1][5]    4.8086 0.16904
## Lag[4*(p+q)+(p+q)-1][9]    9.9790 0.05069
## d.o.f=2
##
## Weighted ARCH LM Tests
## ------------------------------------
##             Statistic Shape Scale P-Value
## ARCH Lag[3]    0.2047 0.500 2.000 0.65092
## ARCH Lag[5]    7.8869 1.440 1.667 0.02137
## ARCH Lag[7]   10.8354 2.315 1.543 0.01173
##
## Nyblom stability test
## ------------------------------------
## Joint Statistic:  0.5731
## Individual Statistics:
## omega  0.29984
```

```
## alpha1 0.21463
## beta1  0.30721
## gamma1 0.05517
##
## Asymptotic Critical Values (10% 5% 1%)
## Joint Statistic:          1.07 1.24 1.6
## Individual Statistic:     0.35 0.47 0.75
##
## Sign Bias Test
## -----------------------------------
##                     t-value   prob sig
## Sign Bias           1.3837 0.1678
## Negative Sign Bias  0.4961 0.6203
## Positive Sign Bias  0.1140 0.9093
## Joint Effect        3.2368 0.3565
##
##
## Adjusted Pearson Goodness-of-Fit Test:
## ------------------------------------
##   group statistic p-value(g-1)
## 1    20     32.00      0.03125
## 2    30     43.32      0.04256
## 3    40     60.42      0.01547
## 4    50     70.68      0.02293
##
##
## Elapsed time : 0.135937
```

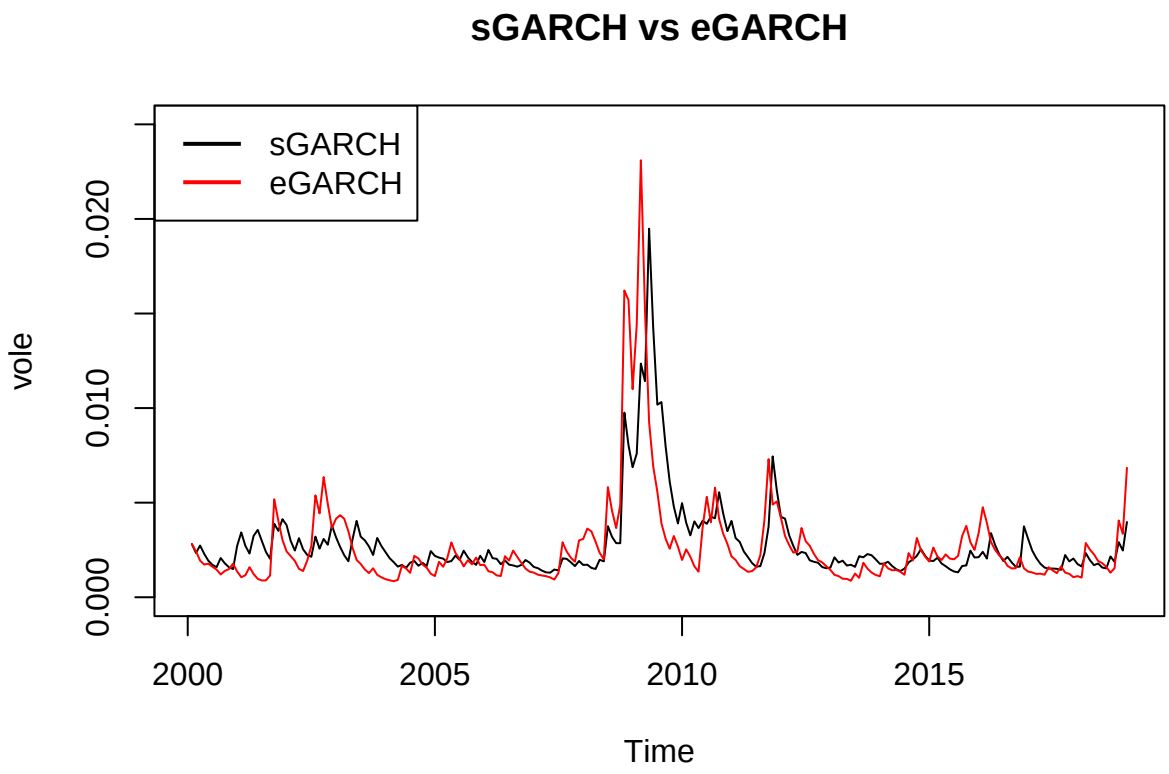
Looking at the output, alpha, beta, and gamma are all significant. We are definitely on the right track here. Let's try plotting this out.

```
Evole <- ts(fit.e@fit$sigma^2, start = 2000 +1/12, frequency = 12)

plot(Evole, type = "l", main = "Exponential GARCH")
```



Here, it seems that we have a relatively similar time series as the standard GARCH model. To further compare them, we can also plot them on the same plot.

```
plot(vole, type = "l", main = "sGARCH vs eGARCH", ylim = c(0, 0.025))
lines(Evole, type="l", col = "red")
legend("topleft", legend = c("sGARCH", "eGARCH"), col = c("black", "red"), lty = 1, lwd = 2)
```



Here, it is easy to see that exponential GARCH has a much higher extreme compared to standard GARCH.

```
a <- infocriteria(fit)
b <- infocriteria(fit.e)
c <- data.frame("sGARCH" = a, "eGARCH" =b)
colnames(c) = c("sGARCH", "eGARCH")
kable(c)
```

	sGARCH	eGARCH
Akaike	-3.166352	-3.274764
Bayes	-3.121230	-3.214600
Shibata	-3.166693	-3.275366
Hannan-Quinn	-3.148147	-3.250490

From this table, the standard GARCH seems to have a better fit, as it has a better score across all criteria.

3.5 Volatility and Risks

Let's circle back and talk about the purpose of this report. Overall, we have a very large data set of over 700 stocks, after purging the ones with missing values. Combining with over 200 observations, this means it is a challenging task to model volatility in a market with such large number of stocks. GARCH model is limited to a specific time series, and while fitting nearly 800 GARCH(1,1) model is technically not very computationally expensive, it's hard to interpret the outcomes.

Take a look at this fit_list, which holds 793 GARCH models for all stocks returns.

```
Model <- ugarchspec(variance.model = list(model = "sGARCH",
                                          garchOrder = c(1, 1) ),mean.model = list(armaOrder=c(0,0,0),
                                                                                   include.mean = FALSE)

fit_list <- list()
returns <- as.data.frame(returns)

ugarch <- function(x){
  ugarchfit(data = x, spec = Model)
}

ret <- returns[,-1]
fit_list <- apply(ret,2,ugarch)
```

This fit_list, which is already a large item itself, contains even more information within each fitted models. Here, I attempted to get the fit criteria for each stock.

```
remove(aic)
```

```
## Warning in remove(aic): object 'aic' not found
```

```
aic_df <- c
models <- c(names(fit_list))

for (model_name in names(fit_list)) {
  model <- fit_list[[model_name]]
  tryCatch({
    aic <- infocriteria(model)
    aic_df <- cbind(aic_df, aic)
  }, error = function(e) {
    cat("Error computing AIC for model", model_name, ":", conditionMessage(e), "\n")
  })
}
```

```
## Warning in log(log(nObs)): NaNs produced
```

```
## Error computing AIC for model 108 : replacement has length zero
```

```
models <- c("sGARCH", "eGARCH", models[models !="108"])
colnames(aic_df) <- models

min(aic_df[1,])
```

```
## [1] -3.290334
```

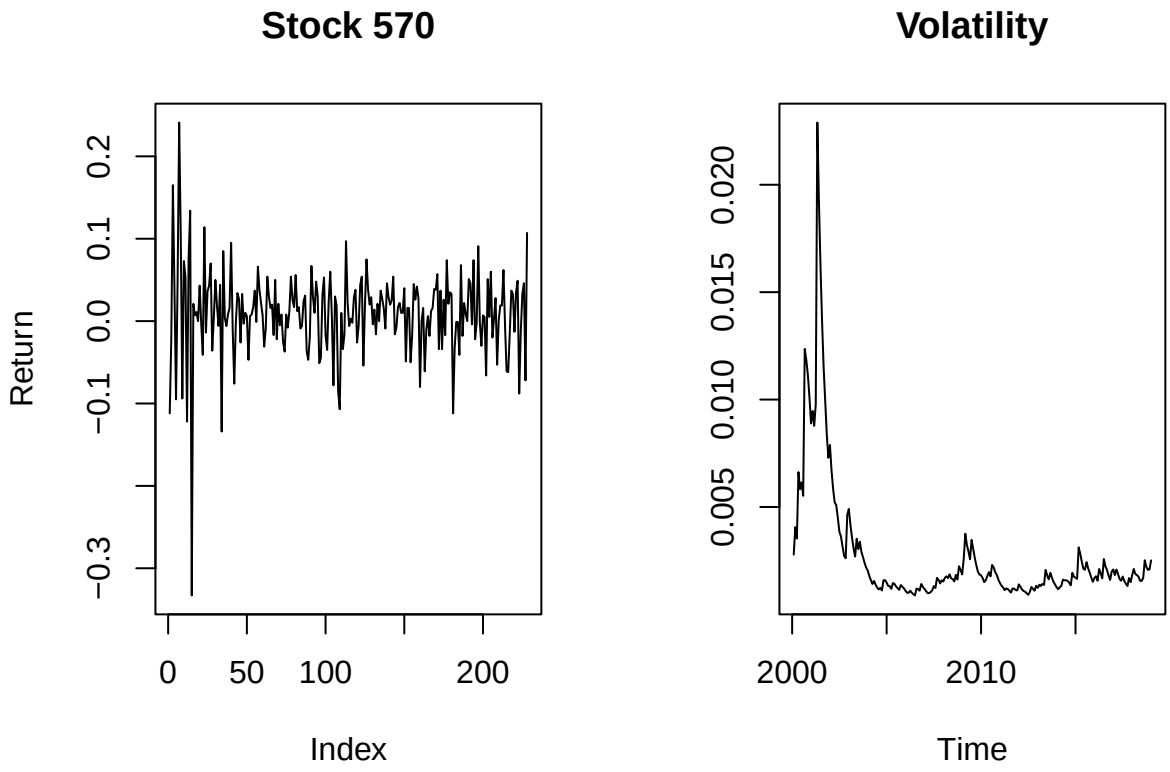
```
most_fit <- names(aic_df[which.min(aic_df[1,])])
most_fit
```

```
## [1] "570"
```

It seems that the GARCH(1,1) model for stock 570 has the best fit with AIC value of -3.290334.

Let's take a look at this stock.

```
mod_570 <- fit_list$"570"
ts_570 <- ts(mod_570@fit$sigma^2, start = 2000 +1/12, frequency = 12)

par(mfrow=c(1,2))
plot(returns$"570", type="l", main = "Stock 570", ylab = "Return")
plot(ts_570, type = "l", main = "Volatility", ylab = "")
```



It seems that GARCH models the volatility of this stock quite well. We can also perform forecasting using ugarchforcast

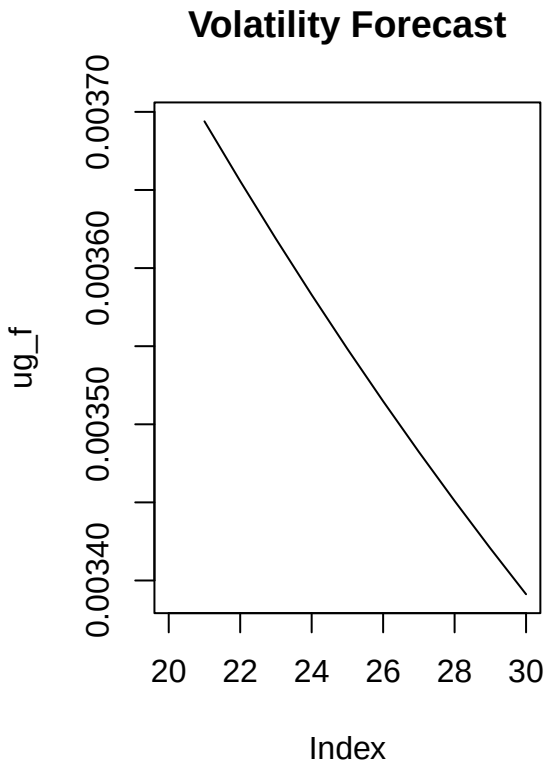
```
ugfore <- ugarchforecast(mod_570, n.ahead = 10)
ugfore
```

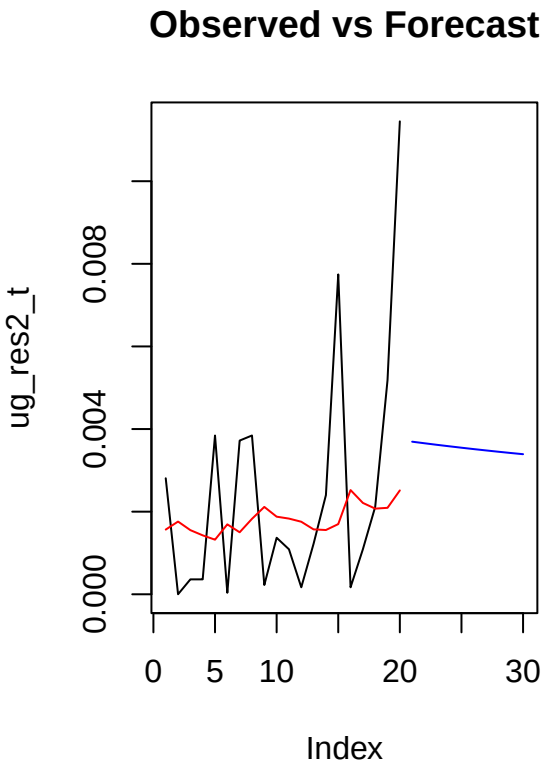
```
##
## *----------------------------------*
## *       GARCH Model Forecast       *
## *----------------------------------*
## Model: sGARCH
## Horizon: 10
## Roll Steps: 0
## Out of Sample: 0
##
```

```
## 0-roll forecast [T0=1970-08-17]:
##       Series   Sigma
## T+1        0 0.06078
## T+2        0 0.06046
## T+3        0 0.06016
## T+4        0 0.05986
## T+5        0 0.05957
## T+6        0 0.05929
## T+7        0 0.05901
## T+8        0 0.05874
## T+9        0 0.05849
## T+10       0 0.05823
```

Putting these forecast with the last 50 estimated observations:

```
ug_f <- ugfore@forecast$sigmaFor

ug_res2 <- mod_570@fit$residuals^2
ug_var <- mod_570@fit$var

ug_var_t <- c(tail(ug_var, 20), rep(NA, 10))
ug_res2_t <- c(tail(ug_res2,20),rep(NA,10))
ug_f <- c(rep(NA, 20), ug_f^2)

par(mfrow = c(1,2))
plot(ug_f, type = "l", main = "Volatility Forecast", xlim = c(20, 30))
plot(ug_res2_t, type = "l", main = "Observed vs Forecast")
lines(ug_f, col = "blue")
lines(ug_var_t, col = "red")
```






```
ug_var_t <- na.omit(ug_var_t)
ug_res2_t <- na.omit(ug_res2_t)
mse <- mean((ug_var_t - ug_res2_t)^2)
mse
```

```
## [1] 8.288269e-06
```

From this plot, it is clear that the forecast of the conditional variance picks up from the last estimated conditional variance. After a period of high volatility, it picked up the tren and moving up to the unconditional variance value.

Here is a peak of the model with the highest AIC:

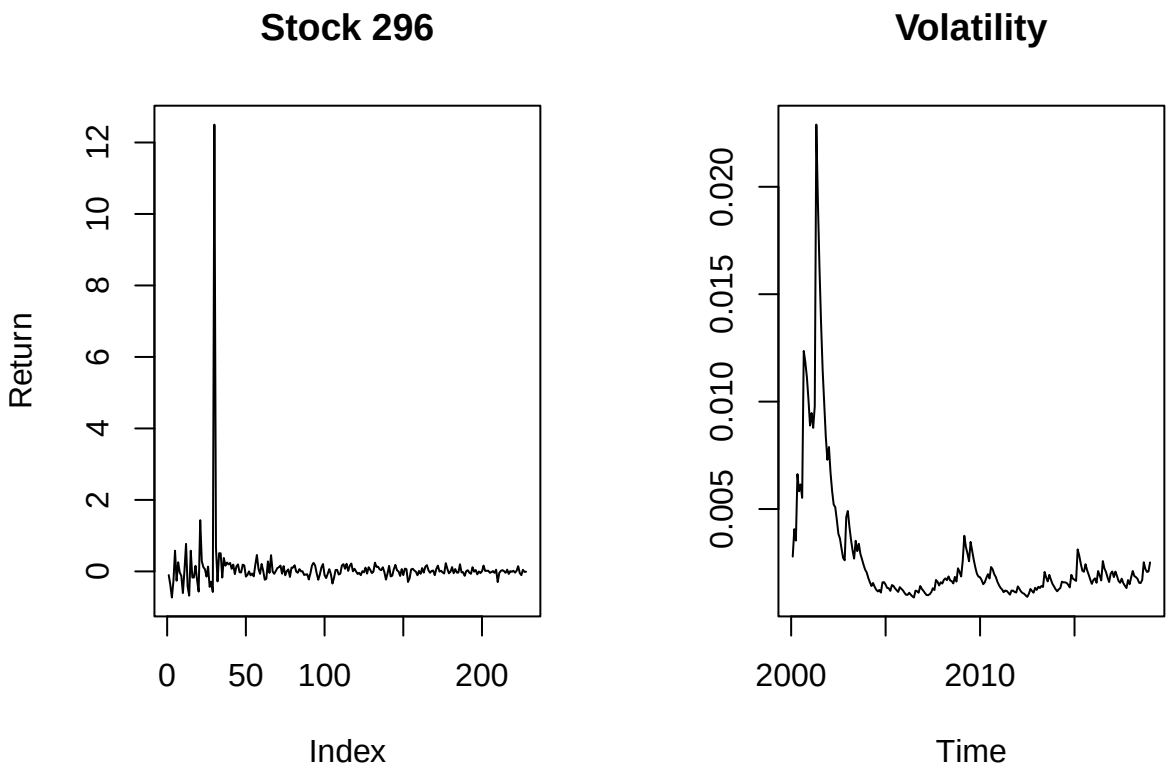
```
max(aic_df[1,])
```

```
## [1] 2.190439
```

```
least_fit <- names(aic_df[which.max(aic_df[1,])])
least_fit
```

```
## [1] "296"
```

```
mod_296 <- fit_list$"296"
ts_296 <- ts(mod_570@fit$sigma^2, start = 2000 +1/12, frequency = 12)

par(mfrow=c(1,2))
plot(returns$"296", type="l", main = "Stock 296", ylab = "Return")
plot(ts_570, type = "l", main = "Volatility", ylab = "")
```



3.6 Multivariate GARCH

So far, we have considered each stocks' volatility as independent of each other. Building onto the Univariate GARCH model, the multivariate GARCH can consider multiple time series. MGARCH allows us to consider the co-movements of numerous stocks. Here, we assume that we are using the same univariate volatility model specification for each stocks.

```
uspec.n = multispec(replicate(793,
                              ugarchspec(variance.model = list(model = "sGARCH",
                                         garchOrder = c(1, 1) ),
                                         mean.model = list(armaOrder=c(0,0,0),
                                         include.mean = FALSE) )))
```

Now we estimate these univariate models.

```
multf = multifit(uspec.n, ret)
```

```
## Warning in .sgarchfit(spec = spec, data = data, out.sample = out.sample, :
## ugarchfit-->warning: solver failer to converge.
```

For multivariates, DCC-GARCH allows the correlation matrix to depend of the time, which fits the purpose of volatility prediction. However, it should be noted that DCC can't model volatility spillovers. We will specify DCC model as followed:

```
spec1 <- dccspec(uspec = uspec.n, dccOrder = c(1,1), distribution = "mvnorm")
```

Finally, we can estimate the model. More specifically, we want to estimate the model as specified in spec1 using the return data of 793 stocks. fit.control ensures the estimation procedure produces standard errors for estimated parameters, and we use the already estimated univariate models of in multf.

```
fit1 <- dccfit(spec1, data = ret, fit.control = list(eval.se = TRUE), fit = multf)
```

Extracting time-variation in the correlation between the assets:

```
tryCatch(rcov(fit1), error = function(e) {
    cat("Unable to calculate rcov")
  })
```

```
## Unable to calculate rcov
```

```
tryCatch(rcor(fit1), error = function(e){
  cat("Unable to calculate rcor")
})
```

```
## Unable to calculate rcor
```

It seems that R was unable to produce a covariance/ correlation matrix. This error was probably due to the fact that this is such a large data set with 793 stocks/ models, thus calculating a correlation matrix is extremely computationally expensive, since we would get a 793x793 matrix. Therefore the question becomes: is there an efficient way to model Volatility while optimizing the amount of data we have?

Over the next section, we will go over the Machine Learning approach, which hopefully will improve upon the above limitations.

4 The LSTM Approach

In this section, we will attempt to train all returns data, while treating stock ids as a features. As stated above, even though GARCH(1,1) is extremely effective in modelling one time series, multivariate GARCH becomes computationally expensive as the number of assets increases. Thus, we turn to Machine Learning to overcome this limitation. Since we are working with time series, one approach is to use Recurrent Neural Network.

The benefit of an RNN is that it can accumulate a hidden state that encodes input over time. However, it is difficult to maintain information in its hidden state for a long time. As we work with historical data, we would want a tool that can learn long-term dependencies.

Improving upon RNN, Long Short Term Memory includes an additional hidden state that persists from step to step. It does not pass through an activation function or get multiplied by the connection weights at each time. Thus, we can avoid long-term dependencies problems.

4.1 A single stock

To begin, let's start with the first stock in our data set.

```
data <- returns$`1`  # stock numer 1 Full
create_sequences <- function(data, n_steps) {
  X <- array(dim = c(length(data) - n_steps, n_steps, 1))
  y <- vector(length = length(data) - n_steps)

  for (i in 1:(length(data) - n_steps)) {
    X[i,,1] <- data[i:(i + n_steps - 1)]
    y[i] <- data[i + n_steps]
  }

  return(list(X, y))
}
```

To train the model, we need to start with the data split.

```
n_steps <- 10
dataset <- create_sequences(data, n_steps)
X <- dataset[[1]]
y <- dataset[[2]]

set.seed(123) # For reproducibility
indices <- sample(1:nrow(X), size = 0.8 * nrow(X))  # Warning different test and train set this one is
X_train <- X[indices,,]
y_train <- y[indices]
X_test <- X[-indices,,]
y_test <- y[-indices]
```

we can now build a LSTM model using the keras_model_sequential function, and adding layers on top.

```
model <- keras_model_sequential() %>%
  layer_lstm(units = 60, return_sequences = TRUE, input_shape = c(n_steps, 1)) %>%
  layer_lstm(units = 50, return_sequences = FALSE) %>%
  layer_dropout(rate = 0.5) %>%
  layer_dense(units = 1)


model %>% compile(
  optimizer = 'adam',
  loss = 'mse'
)

history <- model %>% fit(
  X_train, y_train,
  epochs = 50,
  batch_size = 32,
  validation_split = 0.2
)
```

Thus, we have the predicted results as:

```
model %>% evaluate(X_test, y_test)
```

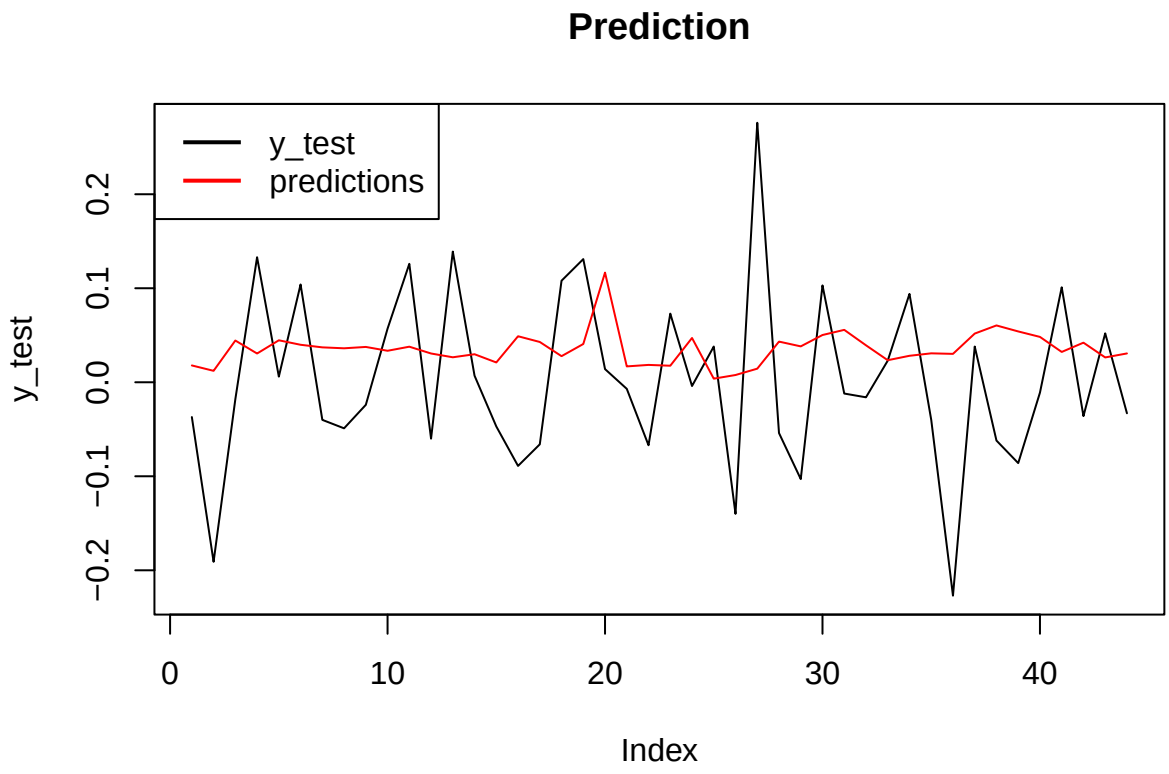
```
## 2/2 - 0s - loss: 0.0101 - 24ms/epoch - 12ms/step
```

```
##       loss
## 0.01012927
```

```
predictions <- model %>% predict(X_test)
```

```
## 2/2 - 1s - 516ms/epoch - 258ms/step
```

```
plot(y_test, type = 'l', ylim = range(c(y_test, predictions)), main = "Prediction")
lines(predictions, col = 'red')
legend("topleft", legend = c("y_test", "predictions"), col = c("black", "red"), lty = 1, lwd = 2)
```



As we can see, LSTM picked up the variance from the data set. However, it seems to fail to pick up larger spikes.
We have mse of 0.01023. However, comparing with the mse of GARCH model, which was 8.288e-06, it seems that GARCH out-performed LSTM.

```
mse <- mean((predictions-y_test)^2)
mse
```

```
## [1] 0.01012927
```

4.2 Full data set

Now, we try to fit all stock data at once. Following the same steps as above, we first split the testing and training data.

```
returns$date <- NULL
create_sequences <- function(data, n_steps) {
  n_obs <- nrow(data)
  n_cols <- ncol(data)

  # Pre-allocate the arrays for inputs (X) and outputs (y)
  X <- array(dim = c(n_obs - n_steps, n_steps, n_cols))
  y <- matrix(nrow = n_obs - n_steps, ncol = n_cols)

  # Populate the arrays
  col_names <- names(data)
  for (j in 1:n_cols){
    stock_data <- pull(data, !!sym(col_names[j]))  # Use pull to extract column as a vector
    for (i in 1:(n_obs - n_steps)) {
      X[i, , j] <- stock_data[i:(i + n_steps - 1)]
      y[i, j] <- stock_data[i + n_steps]
    }
  }

  list(X, y)
}

n_steps <- 20
dataset <- create_sequences(returns, n_steps)
X <- dataset[[1]]
y <- dataset[[2]]


indices <- sample(1:nrow(X), size = 0.8 * nrow(X))
X_train <- X[indices,,]
y_train <- y[indices,]
X_test <- X[-indices,,]
y_test <- y[-indices,]
```

Next, the activation function

```
model <- keras_model_sequential() %>%
  layer_lstm(units = 50, return_sequences = FALSE, input_shape = c(n_steps, ncol(returns))) %>%
  layer_dropout(rate = 0.2) %>%
  layer_dense(units = ncol(returns))

model %>% compile(
  optimizer = 'adam',
  loss = 'mse'
)
```

```
model <- keras_model_sequential() %>%
  layer_lstm(units = 20, return_sequences = TRUE, input_shape = c(n_steps, ncol(returns))) %>%
```

```
  layer_dropout(rate = 0.2) %>%
  layer_lstm(units = 50, kernel_regularizer = regularizer_l2(0.01)) %>%
  layer_dense(units = ncol(returns))
```

In this part, we will try different optimizer functions, and compare their performance.

For SGD with momentum and Nesterov accelerated gradient:

```
optimizer_sgd <- optimizer_sgd(learning_rate = 0.01, momentum = 0.9, nesterov = TRUE)
#0.06103797 0.07517219 #This one decays So-So
```

For RMSprop

```
optimizer_rmsprop <- optimizer_rmsprop(learning_rate = 0.001, rho = 0.9)
#0.01275908 0.07578062  So-So
```

For Adagrad

```
optimizer_adagrad <- optimizer_adagrad(learning_rate = 0.01)
#0.21223037 0.07525495 need way more ssteps
#0.02856970 0.07516009 with 500 look the best
```

For Adadelta

```
optimizer_adadelta <- optimizer_adadelta(learning_rate = 1.0, rho = 0.95)
#0.01268456 0.07554621 So-So
```

Overall, it seems that Adagrad (Adaptive Gradient Algorithm) has the best performance. Thus, we will choose this optimizer.

```
model %>% compile(
  optimizer = optimizer_adagrad,
  loss = 'mse',
  metrics = c('mae')
)

optimizer_adagrad <- optimizer_adagrad(learning_rate = 0.01)

early_stopping_callback <- callback_early_stopping(monitor = "val_loss", patience = 5)
```

```
model %>% evaluate(X_test, y_test)
```

```
## 2/2 - 0s - loss: 0.1587 - mae: 0.0716 - 26ms/epoch - 13ms/step
```

```
##       loss        mae
## 0.15869991 0.07156616
```

5 Conclusion

In conclusion, volatility stands as a fundamental pillar within the realm of finance, serving not only as an informative signal for investors but also as a crucial input for numerous financial models. By exploring various volatility prediction models, we aim to further understand the impacts of volatility in prediction stocks' returns.
Overall, it seems that even though LSTM solved the limitations of GARCH, it seems that it didn't pick up the volatility of the stocks as well as GARCH did. GARCH(1,1) did perform very well for individual stocks, as it could pick up spikes and high volatility periods relatively well. Multivariate GARCH models could also work if we have a smaller size portfolios.

5.1 Limitations

Some of limitations of the report includes the fact that modelling volatility of stocks is extremely complex when considering interactions between a large amount of assets. In this case, we have 793 assets, and while we could model each stock's volatility separately, using the whole data set could bring challenges.

5.2 Other Approaches

Based on the foundation of GARCH, there are also other Machine Learning approaches to model volatility. One approach is to forecast volatility with Support Vector Machine-based GARCH models, which might improve upon the stated methods.