Data Analysis Report

Prediction of Market Volatility Using High Frequency Data

Luhuan Wu

2/25/2018

0 Introduction

Brief Outline
This project is an implementation of the paper, "Combining high frequency data with non-linear models for forecasting energy market volatility". The "paper" in the following text refers to this paper. The paper discusses 6 volatility measures and 5 models to evaluate and predict the volatility of energy market. Furthermore, it uses 2 statistical test methods to select the best volatility measures and models.

We replicated most of the work in the paper, with some improvements and discussions. Our main work included:

1. Implement 5 volatility measures: RV, RK, MedRV, BV and TSRV as described in paper.
2. Implement 4 prediction models: GARCH, HAR, ARFIMA, ANN and HAR-ANN as described in paper.
3. Implement a LSTM-RNN model – improvement
4. Implement 2 statistical tests: SPA and MCS, as described in paper.

Work Assignment
I am in charge of task 1 & 2, and my teammate is in charge of task 3 & 4.

About the Report
In the rest of the report, we will give description of the paper's implementation, our implementation and the analysis of the results. We will use an interactive mode of "R Markdown" which will allow us to see the codes and their direct outputs. The reader could download the attached R Markdown file (todo: a link to add) to run on their own.

The content of the report is divided into 4 parts:
1. Preprocessing
2. Estimation of Realized Volatility
3. Prediction Models
4. Conclusions and Reflections
5. Data Output

1 Preprocessing

1.1 Load the data

The data set in the paper consists of transaction prices for crude oil, heating oil, and natural gas traded on the New York Mercantile Exchange (NYME), which extends from January 5, 2004 to December 31, 2012. The time frequencies of the data set are 5-minutes and 1-second, for the calculation of different realized measures.

Due to the availability of data set, our data is the 1-min and 5-min price series of Maotai stock (600519.SH) from 2014-12-26 to 2018-02-23, with a total of 770 trading days, which is obtained from WIND database. The stock prices are split-adjusted share prices.

- For 5-min frequency data, there are 48 sample points in a day, from 9:35 to 15:00.
- For 1-min frequency data, there are 241 sample points in a day, from 9:00 to 15:00.

Now, we read the data to R and obtain an overview of it.

```r
library(readxl)
# Read the 5-min frequency data
data.5min <- read_xlsx("data/maotai_3data.xlsx", sheet = "pre_adjusted")
data.5min <- data.frame(data.5min)
l.data.5min <- nrow(data.5min)
data.5min <- data.5min[193:l.data.5min,]
l.data.5min <- nrow(data.5min)

#Read the 1-min frequency data
data.1min <- read_xlsx("data/maotai_1mfreq_data.xlsx", sheet = 1)
data.1min <- data.frame(data.1min)

#Data-preprocessing for alignments of the two datasets
#Delete 9:25 trading point
l.data.1min <- nrow(data.1min)
RowsToBeDeleted <- seq(242, l.data.1min, by=242) # 242, the first position of 9:25
data.1min <- data.1min[-RowsToBeDeleted,]
l.data.1min <- nrow(data.1min)
ndays = l.data.5min / 48

#Check the head of the datasets
head(data.5min)
```

```
##                    time     open     high      low    close
## 193 2014-12-26 09:35:00 159.4324 160.1301 159.0965 159.3893
## 194 2014-12-26 09:40:00 159.3893 160.0612 158.1231 159.7511
## 195 2014-12-26 09:45:00 159.4152 159.7856 158.5107 159.7080
## 196 2014-12-26 09:50:00 159.6133 160.0354 159.5530 160.0354
## 197 2014-12-26 09:55:00 160.0354 160.0440 159.3549 159.3549
## 198 2014-12-26 10:00:00 159.3549 159.7856 159.3549 159.3893
```

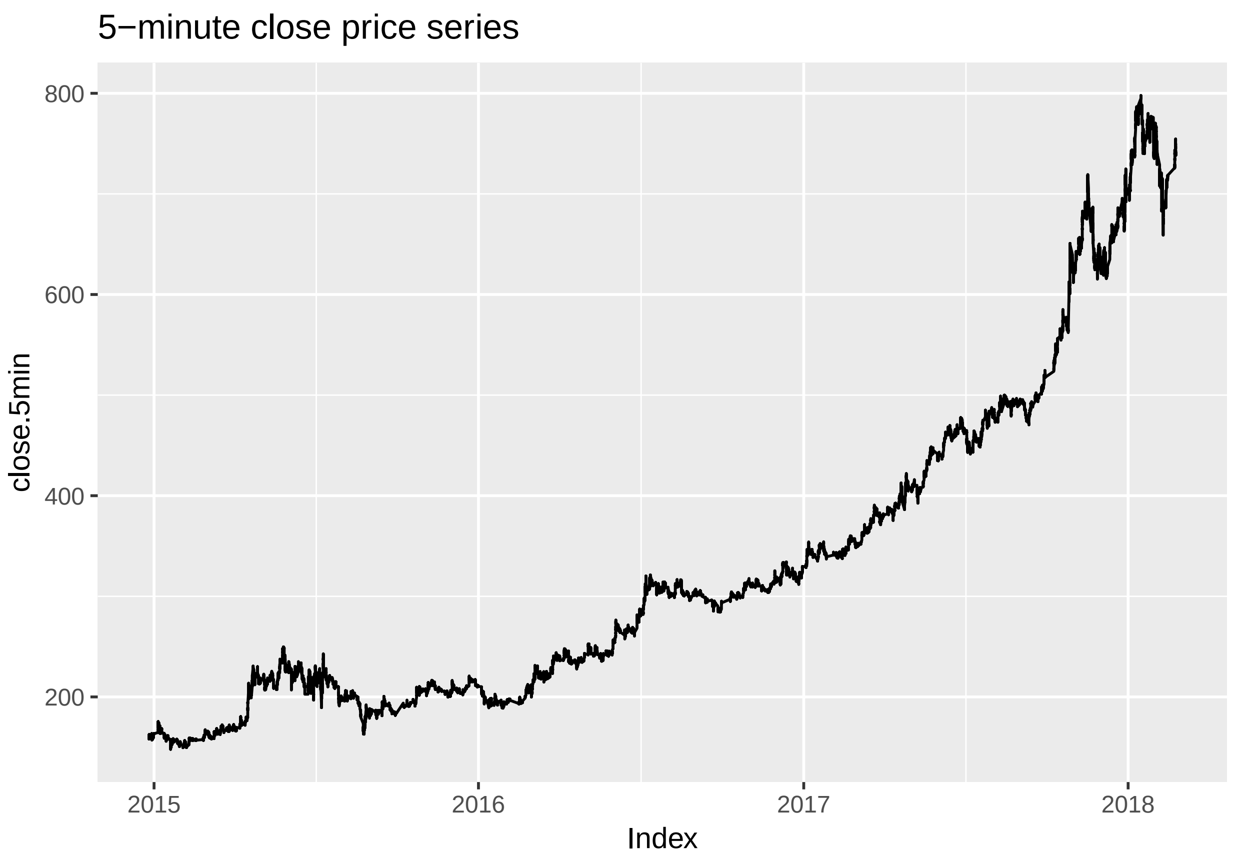
```
head(data.1min)
```

```
##                  time     open     high      low    close
## 1 2014-12-26 09:30:00 159.4324 159.4324 159.1223 159.1395
## 2 2014-12-26 09:31:00 159.1395 159.6133 159.1395 159.4324
## 3 2014-12-26 09:32:00 159.7770 160.1301 159.4324 160.1301
## 4 2014-12-26 09:33:00 159.8028 160.0612 159.3979 160.0612
## 5 2014-12-26 09:34:00 160.0612 160.0612 159.0965 159.3893
## 6 2014-12-26 09:35:00 159.3893 159.3893 158.4074 158.4074
```

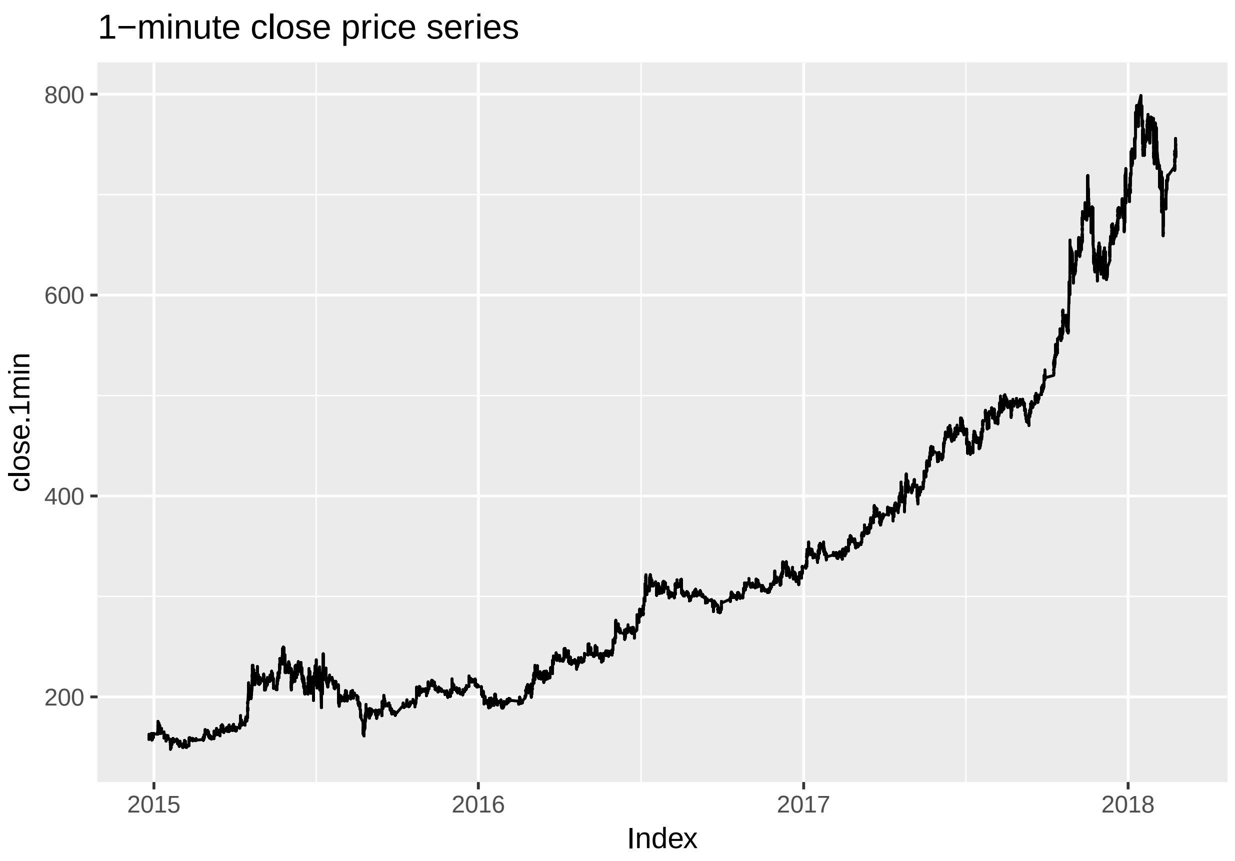
From the table above, we could see that there are 4 types of price available: open price, highest price, lowest price and close price. In the following, we will use the close price to perform the calculation and illustration.

Now, let's obtain an overview of data sets by plotting them.

```
library(ggplot2)
Sys.setenv(TZ = "UTC")
library(xts)
# Replace NA values
close.1min <- xts(data.1min$close, as.POSIXct(data.1min$time, tx="UTC", format='%Y/%m/%d %H:%M:%S'
close.1min <- na.locf(close.1min, fromLast = TRUE)
close.5min <- xts(data.5min$close, as.POSIXct(data.5min$time, tx="UTC", format='%Y/%m/%d %H:%M:%S'
autoplot(close.5min) + labs(title = "5-minute close price series")
```



```r
autoplot(close.1min) + labs(title = "1-minute close price series")
```



Obviously, the 1-min data is more verbose than the 5-min data. In general, the price of Maotai stock is rapidly increasing and there seems to be two periods of high volatility – around June 2015 and around Nov 2017 to Jan 2018.

2 Estimation of Realized Volatility

In this part, we will calculate the estimation of realized volatility of Maotai stock price series, using 5 different realized variance estimators.

2.1 Decomposition of the Stock price

In the analysis of the paper, the latent logarithmic commodity price follows a standard jump-diffusion process contaminated by micro structure noise, which could be modeled by

$$y_t = p_t + \epsilon_t.$$

The quadratic return variation over the interval $[t-h], t$ is usually decomposed into two parts: integrated variance and jump variation:

$$QV_{t,h} = \int_{t-h}^{t} \sigma_s^2 ds + \sum_{t-h \leq l \leq t} J_l^2 = IV_{t,h} + JV_{t,h}.$$

There are 6 estimators introduced in the paper to approximate QV or IV, which are RV, TSRV, RK, BV, MedRV and JWTSRV, where higher-frequency data are used to calculate TSRV and JWTSRV since there are two time scales involved (TS are short for two time scale). However, because the detailed information of JWTSRV is missed in the paper, we do not implement it in this project.

Before performing any calculations, we first define a helper function for plotting to compare different series.

```r
# A helper function to plot multiple xts objects using ggplot2
library(broom)
library(magrittr)
plt.multi.series <- function(mseries, names, title) {
    names(mseries) <- names
    index(mseries) <- as.Date(index(mseries))
    p <- tidy(mseries) %>% ggplot(aes(x = index, y = value, color = series)) +
        geom_line()
    p
}
```

We use 5-minute-frequency data to calculate MedRV, RK, RV and BV.

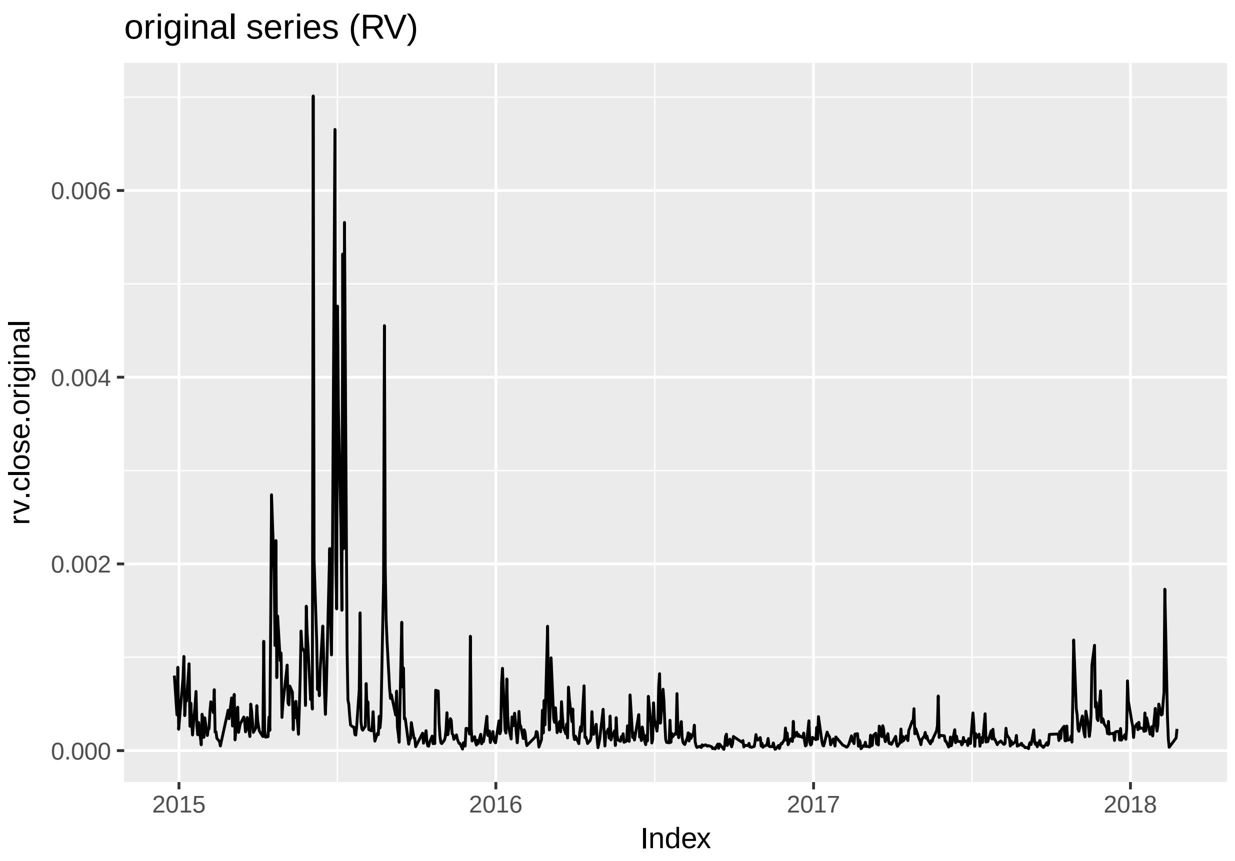
2.2 Calculation

2.2.1 RV (Realized Variance)

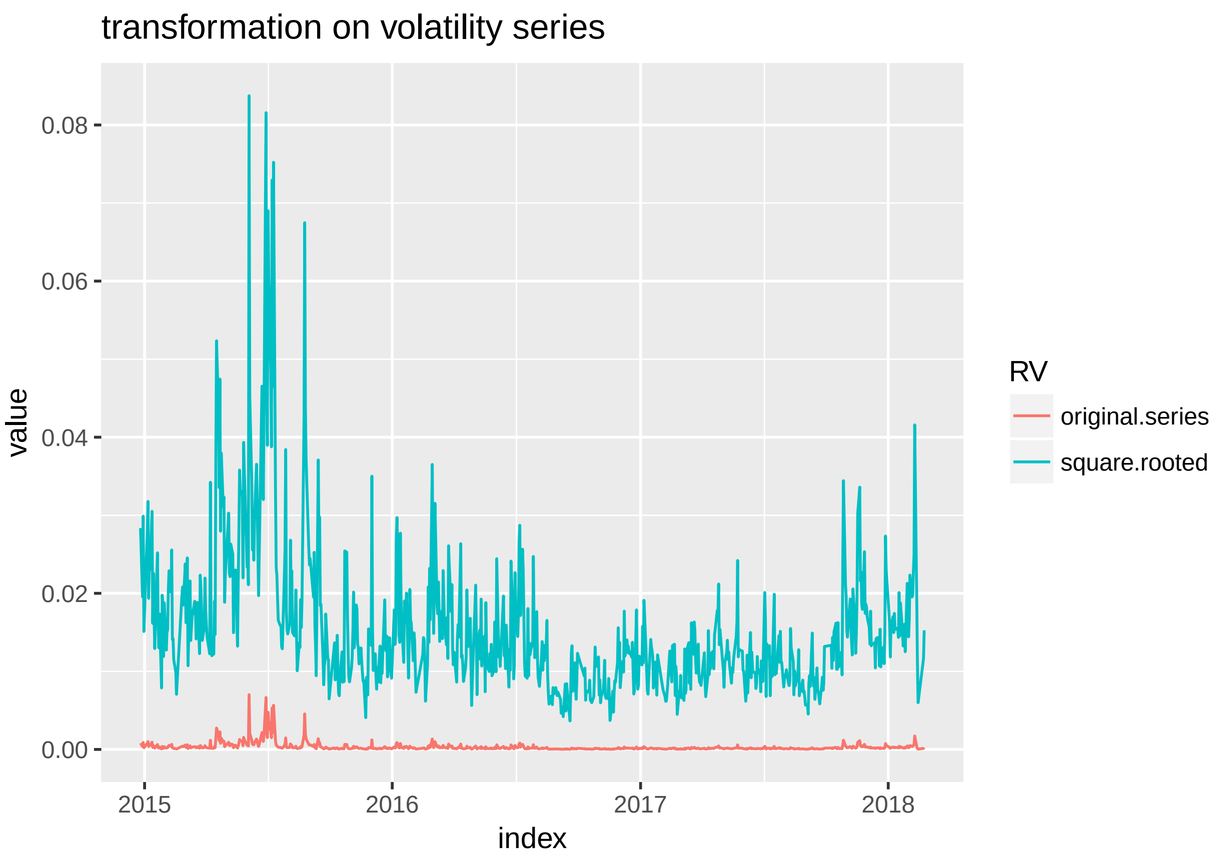
RV is an estimation of quadratic variance.

$$\widehat{QV}_{t,h}^{(RV)} = \sum_{k=1}^{N} (\Delta_k y_t)^2$$

```r
rv.close.original = highfrequency::rCov(rdata = close.5min, align.by = "minutes",
    align.period = 5, makeReturns = TRUE)
rv.close <- sqrt(rv.close.original)
autoplot(rv.close.original) + labs(title = "original series (RV)")
```



```
plt.multi.series(cbind(rv.close.original, rv.close), cbind("original series",
    "square-rooted")) + labs(title = "transformation on volatility series",
    colour = "RV")
```



We could see that there are very large spikes around June 2017, so we apply a square root function to smooth the time series for better prediction performance. We will perform the same transformation to other 4 realized volatility estimators.

2.2.2 RK (Realized Kernel)

A different approach to addressing noise developed by Barndorff-Nielsen et al. (2008) is realized kernels. The realized kernel variance estimator over $[t-h,t]$ is defined by

$$\widehat{QV}^{(RK)}_{t,h} = \gamma_0 + \sum_{\eta=1}^{H} K(\frac{\eta-1}{H})(\gamma_\eta + \gamma_{-\eta})$$

with $\gamma_\eta = \sum_{k=1}^{N} \Delta_k y_t \Delta_{k-\eta} y_t$ denoting the $\eta_t h$ realized autocovariance with $\eta = -H, \ldots, -1, 0, 1, \ldots, H$ and $K(.)$ denoting the kernel function.

There are some available kernels for the implementation of RK. We could check the available kernels using the code below:

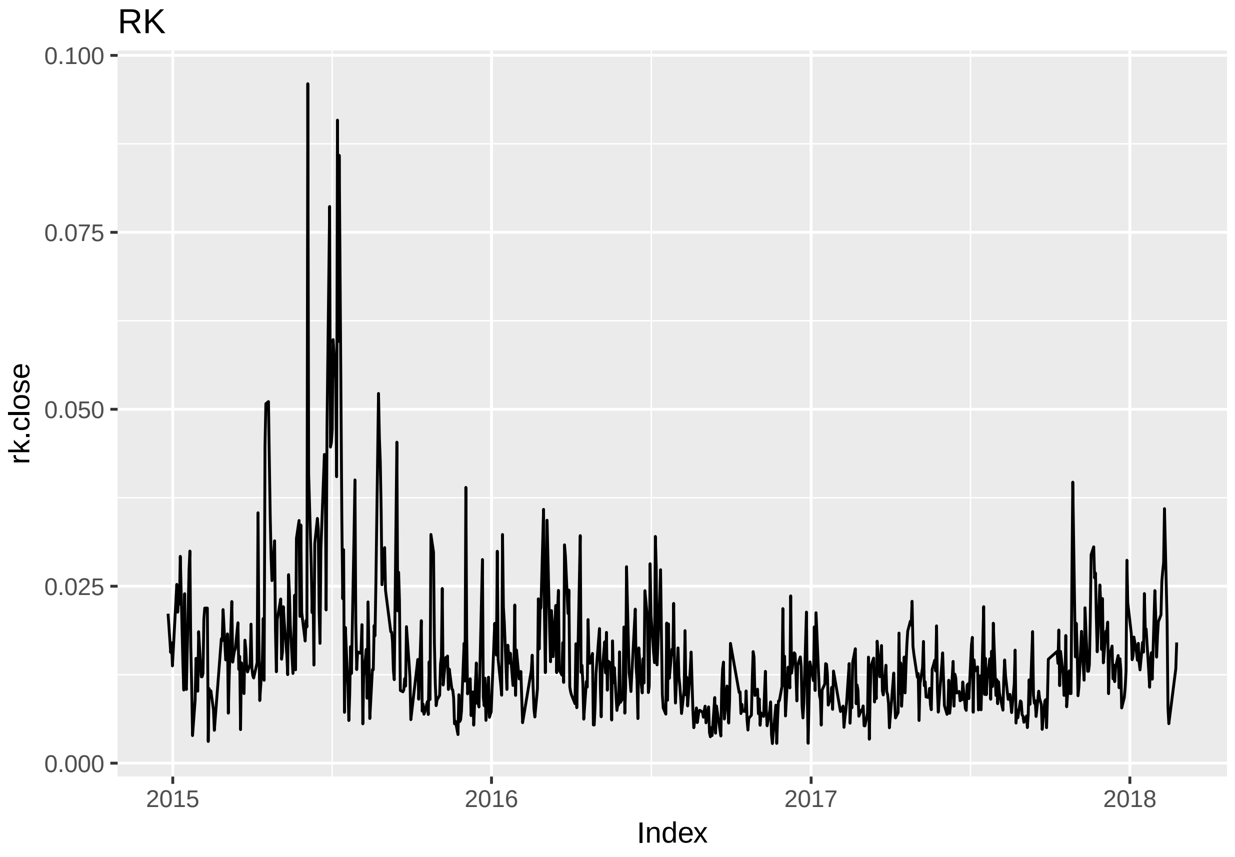
```
#check available kernels
highfrequency::rKernel.available()
```

```
##  [1] "Rectangular"          "Bartlett"             "Second"
##  [4] "Epanechnikov"         "Cubic"                "Fifth"
##  [7] "Sixth"                    "Seventh"                  "Eighth"
## [10] "Parzen"                   "TukeyHanning"             "ModifiedTukeyHanning"
```

We adopt Parzen() as out kernel function to compute RK.

```
rk.close.original <- highfrequency::rKernelCov( rdata = close.5min, kernel.type = "Parzen", period
rk.close <- sqrt(rk.close.original)
rk.close <- na.locf(rk.close, fromLast = TRUE)
```

```
autoplot(rk.close) + labs(title = "RK")
```



2.2.3 BV (Bipower Variation)

In the paper, Andersen, et al.'s adjustment is adopted. The bipower variation over $[t-h, t]$ is defined by

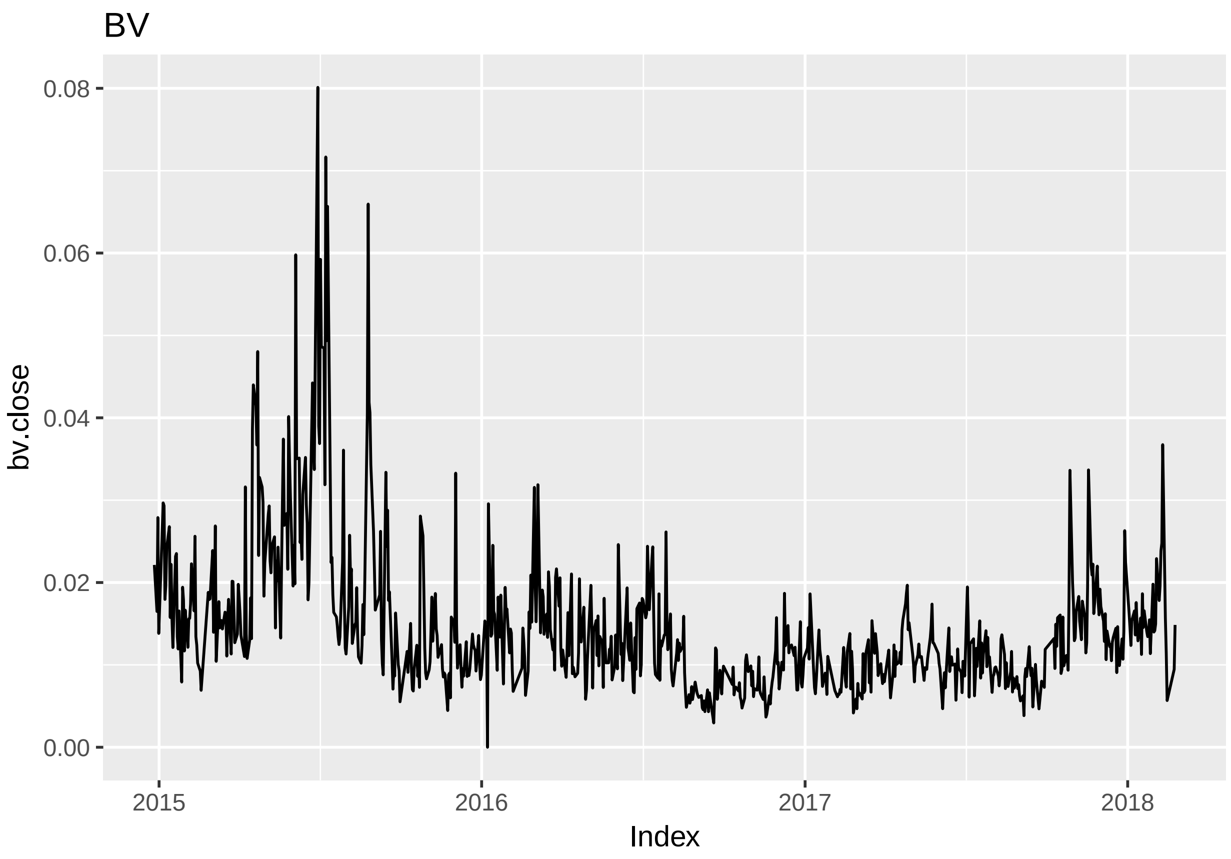
$$\widehat{IV}_{t,h}^{BV} = \mu^{-1} \frac{N}{N-2} \sum_{k=3}^{N} |\Delta_{k-2} y_t| |\Delta_k y_t|$$

where

$$\mu_\alpha = \pi/2 = E(|Z|^\alpha)$$

$$Z \sim N(0,1), \alpha > 0, \widehat{IV}_{t,h}^{(BV)} \rightarrow \int_{t-h}^{h} \sigma^2 ds$$

```
bv.close.original = highfrequency::rBPCov( rdata = close.5min, align.by ="minutes",
                    align.period =5, makeReturns=TRUE)
bv.close <- sqrt(bv.close.original)
autoplot(bv.close) + labs(title = "BV")
```

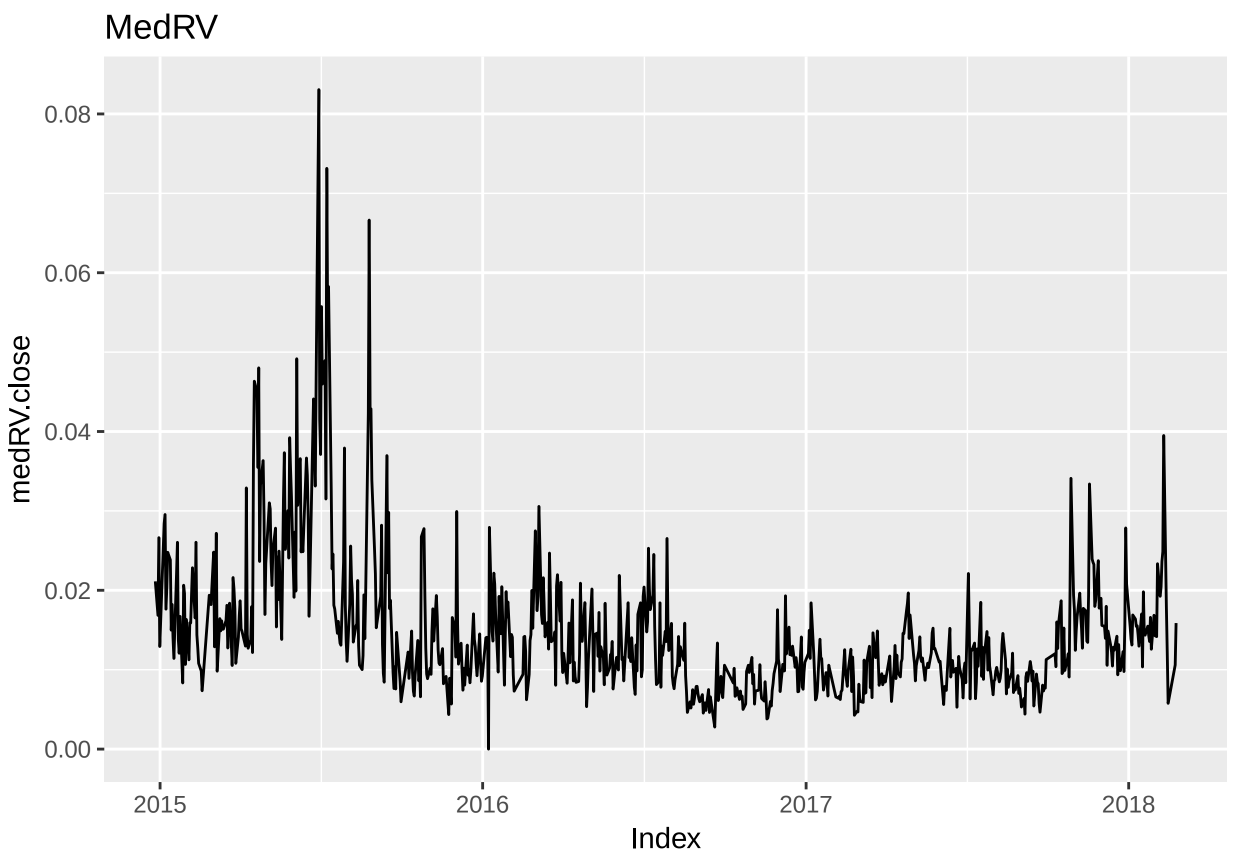


2.2.4 MedRV (Median Realized Volatility)

To estimate the integrated volatility in the presence of jumps, we employ an additional estimator, the median realized volatility.

$$\widehat{IV}_{t,h}^{(MedRV)} = \frac{\pi}{6-4\sqrt{3}+\pi}\left(\frac{N}{N-2}\right) \times \sum_{k=3}^{N} \text{med}(|\Delta_{k-2}y_t|, |\delta_{k-1}y_t|, |\delta_k y_t|)^2$$

```
library(highfrequency)
medRV.close.original <- medRV(rdata = close.5min, align.by="minutes", align.period = 5, makeReturn
#transform to volatility
medRV.close <- sqrt(medRV.close.original)
autoplot(medRV.close) + labs(title = "MedRV")
```



2.2.5 TSRV (two-time scale relalized varaince)

$$\widehat{QV}_{t,h}^{(TSRV)} = \widehat{QV}_{t,h}^{average} - \frac{\bar{N}}{N}\widehat{QV}_{t,h}^{(all)}$$

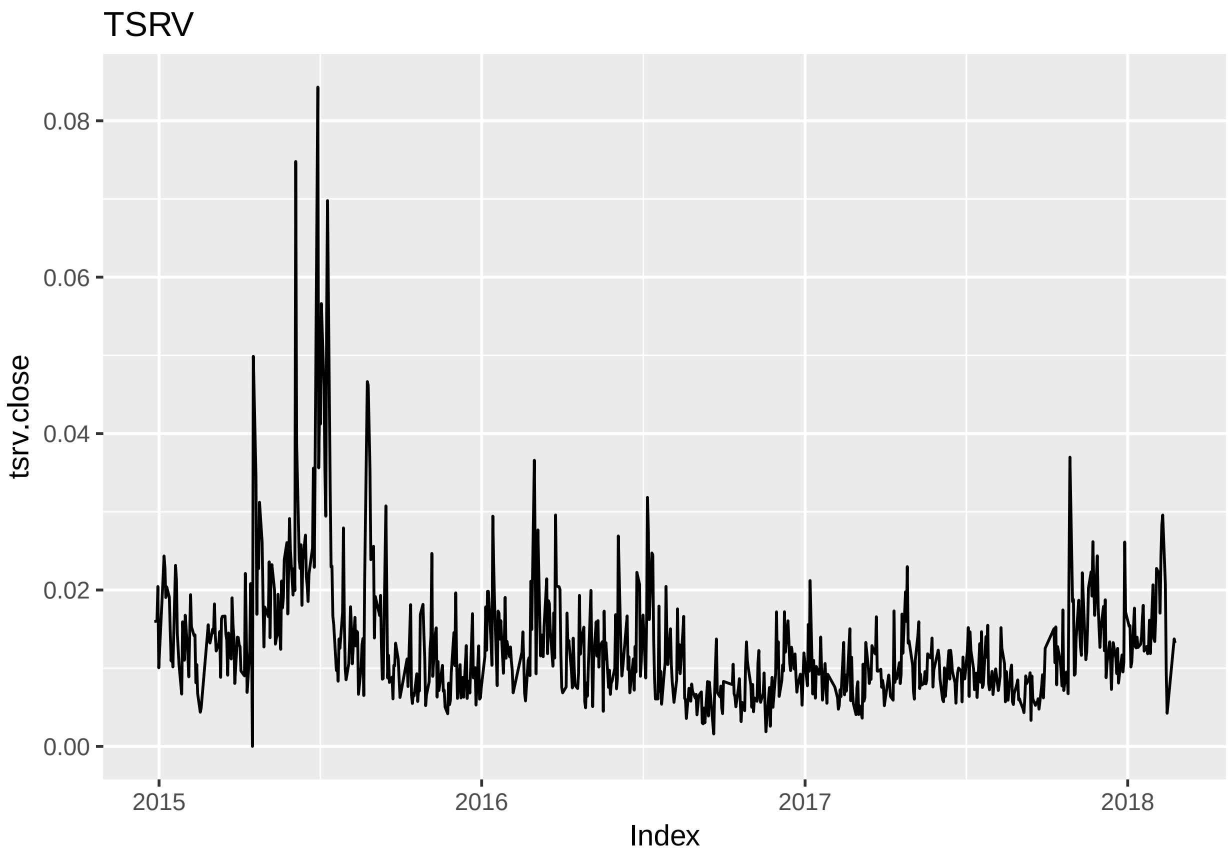
Because there are 2 time scales in the calculation of TSRV. We use 1-minute-frequency data to calculate it. In the original paper, 1-second frequency is used. However, 1-minute frequency is the highest frequency we could obtain from WIND database.

For 1-min close price time series, one day has 241 trading points. We choose the sub-sample frequency to be 14. We could write the function to compute TSRV by as follows:

```
rv <- c()
rv[1] <- 0
for(i in 1: ndays){
  start <- 1 +(i - 1) * 241
  end <- start + 240
  rv[i] <- rRTSCov( pdata = close.1min[start:end], K = 14)
}
tsrv.close.original<-xts(rv, time(rk.close))
```

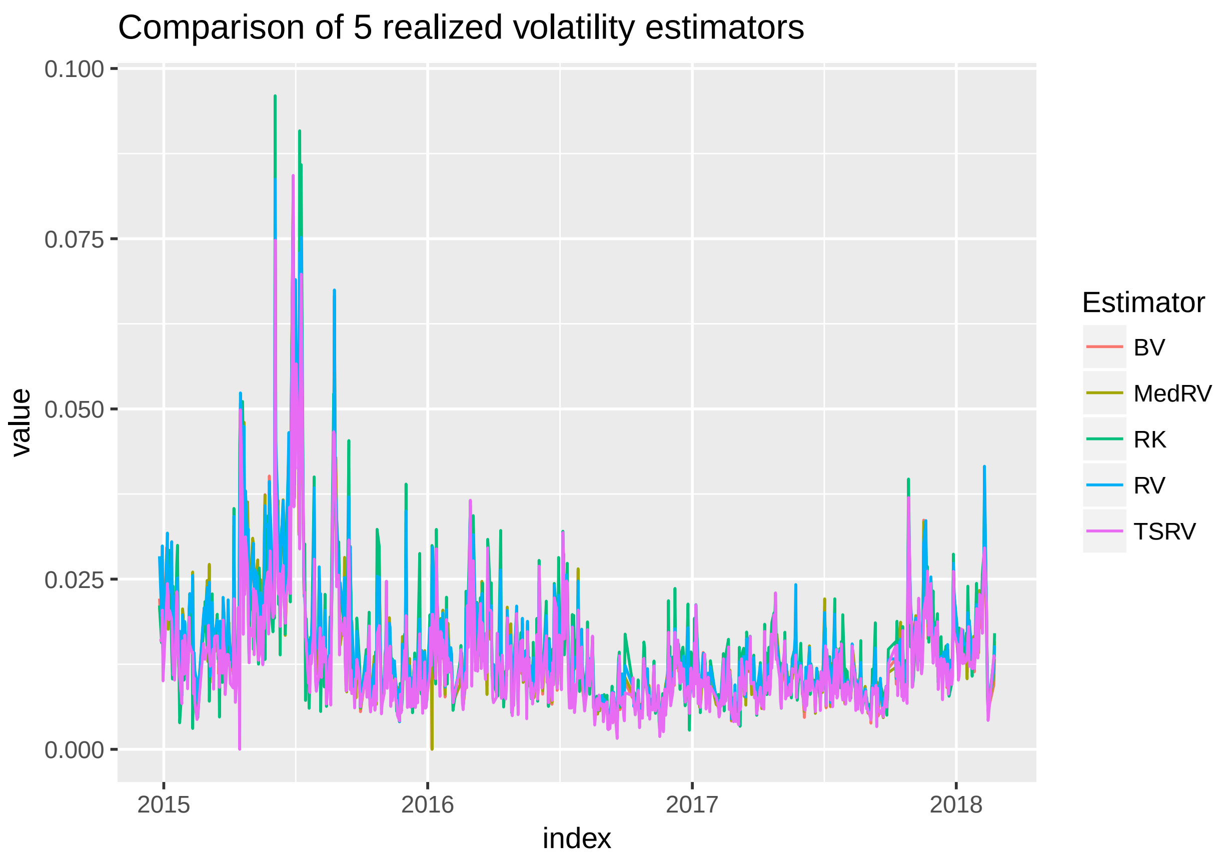
```r
tsrv.close <- sqrt(tsrv.close.original)
tsrv.close <- na.locf(tsrv.close, fromLast = TRUE)
```

```r
autoplot(tsrv.close) + labs(title="TSRV")
```



2.3 Comparison

```r
p<-plt.multi.series(cbind(rv.close, rk.close, medRV.close, bv.close, tsrv.close), cbind("RV", "RK"
p+labs(title="Comparison of 5 realized volatility estimators", colour="Estimator")
```



From the plot above, we could observe that the five volatilities are in accordance in general. However, the green line – RK seems to be most volatile and the purple line – TSRV seems to be the least volatile.

3. Prediction Models

Our data samples consist of estimated daily realized volatility (with 5 measures – RV, BV, RK, MedRV and TSRV) over the period from 2014-12-26 to 2018-02-23, with a length of 770.

In this part, we will fit the models and then make predictions to the 5 estimated series of realized volatility computed above.

3.1 Pre-treatments

3.1.1 Division of Dataset

We divide our sample as the following:

- For traditional time series models (GARCH, HAR, ARFIMA), we will use 80% for training set and 20% for test set.

- For deep learning models, we will use 72% for training set, 8% for cross-validation set and 20% for test set.

First, we calculate the log-return and division markers for the convenience of following treatments.

```
close.logreturn <- makeReturns(close.5min)
## division markers for daily-volatility sequence
total.rv <- length(rv.close)
first.rv <- round(total.rv * 0.72)
second.rv <- round(total.rv * 0.8)
```

3.1.2 Loss Functions

We use 4 loss functions to evaluate the forecasting performance, which are:

1. The root mean square error:

$$RMSE = \sqrt{\frac{1}{T}\sum_{t=1}^{T}(\hat{v}_{t+h} - v_{t+h})^2}$$

2. The mean absolute error:

$$MAE = \frac{1}{T}\sum_{t=1}^{T}|\hat{v}_{t+h} - v_{t+h}|$$

Two additional mean mixed error (MME), a mixture of positive and negative forecast errors with different weights that reveal the cases of over-prediction or under-prediction.

3. Mean Mixed Error (for over-prediction)

$$MME(O) = \frac{1}{T}(\sum_{t\in U}|\hat{v}_{t+h} - v_{t+h}| + \sqrt{\sum_{t\in O}|\hat{v}_{t+h} - v_{t+h}|})$$

4. Mean Mixed Error (for under-prediction)

$$MME(U) = \frac{1}{T}(\sum_{t\in O}|\hat{v}_{t+h} - v_{t+h}| + \sqrt{\sum_{t\in U}|\hat{v}_{t+h} - v_{t+h}|})$$

And then define the corresponding functions in R:

```
# Define the loss function
library(Metrics)
mmeo <- function(actual,predicted){
  diff = predicted - actual
  sum = 0
  for (i in (1:length(diff))){
    if (diff[i]>0) {sum = sum + sqrt(abs(diff[i])) }
    else {sum = sum + abs(diff[i])}
  }
  sum/length(diff)
}
mmeu <- function(actual,predicted){
  sum = 0
  diff = predicted - actual
  for (i in (1:length(diff))){
    if (diff[i]>0) {sum = sum +abs(diff[i])}
    else {sum = sum + sqrt(abs(diff[i]))}
  }
  sum/length(diff)
}
getErrors<-function(actual,predicted) {
  rmse.val <- rmse(predicted, actual)
  mae.val <- mae(predicted, actual)
  mmeo.val <- mmeo(actual, predicted)
  mmeu.val <- mmeu(actual, predicted)
  return(list("rmse"=rmse.val, "mae"=mae.val, "mmeo"=mmeo.val, "mmeu"=mmeu.val))
}
```

3.3.3 Research Design for the Prediction

We compute and evaluate 1-step-ahead and cumulative 5- and 10-step-ahead forecasts of price volatility. The cumulative h - step-ahead forecasts are obtained from the usual multi-step-ahead forecast by adding together

$$ {}^{2}_{t+h} = h^{-1} \sum_{j=1}^{h} v^{2}_{t+j} $$

.

3.2 Models

According to the paper, the strong temporal dependence of realized volatility suggest that the realized volatility should be modeled using an approach allowing for a slowly decaying auto correlation function and possibly long memory shew that volatility over long time intervals has a strong influence on volatility at shorter time intervals but that volatility over short time intervals has no effect on longer intervals. Therefore, in the paper, 5 models are introduced, which we will describe and implement in the following text.

3.2.1 GARCH – the benchmark model

We use GARCH(1,1) as indicated in the original paper.
First, define auxiliary functions for fitting and prediction:

```r
library(rugarch)
predict.GARCH <- function(garch.fit, h, roll) {
  # roll = total.rv - second.rv// pred: second.rv (T+1,..,T+h),..., total.rv(T+1,...,T+h)
  pred <- ugarchforecast(garch.fit, n.ahead = h, n.roll = roll)
  pred.data <- pred@forecast$seriesFor # a matrix, row: T+1, .., T+h; col:days
  pred.cul.data.matrix <-rollapply(pred.data, width=h,mean,by.column=TRUE) # matrix, one-row
  pred.cul.data <-pred.cul.data.matrix[1, 1:(roll-h+1)]
  pred.cul.time <- colnames((pred.cul.data.matrix))[(1+h):(roll+1)]
  pred.ts <- xts(unname(pred.cul.data), as.POSIXct(pred.cul.time))

  return(list("pred"=pred, "pred.ts"=pred.ts))
}

#for whole process
process.GARCH <- function(rv, h) {
  vol <- coredata(rv)
  total <- length(vol)
  second <- round(total*0.8)
  #fit
  spec <- ugarchspec(mean.model = list(armaOrder = c(1,1)))
  fit <- ugarchfit(spec,rv,out.sample = (total-second))
  #forecast
  pred <- predict.GARCH(fit, h = h, roll = (total-second))
  #errors
  errors <- getErrors(vol[(second+h):total], coredata(pred$pred.ts))
  #loss: element-wise squared error
  loss <- (vol[(second+h):total]- coredata(pred$pred.ts))**2
  loss <- xts(loss, index(rv[(second+h): total]))
  return(list("fit" = fit, "pred" = pred, "errors" = errors, "loss" = loss))
}
```

Then, we could fit the GARCH models and make 1-step, 5-step and 10-step predictions.

```r
garch.RV.1 <- process.GARCH(rv.close, 1)
garch.MedRV.1 <- process.GARCH(medRV.close, 1)
garch.RK.1 <- process.GARCH(rk.close, 1)
garch.BV.1 <- process.GARCH(bv.close, 1)
garch.TSRV.1 <- process.GARCH(tsrv.close, 1)

garch.RV.5 <- process.GARCH(rv.close, 5)
garch.MedRV.5 <- process.GARCH(medRV.close, 5)
garch.RK.5 <- process.GARCH(rk.close, 5)
garch.BV.5 <- process.GARCH(bv.close, 5)
garch.TSRV.5 <- process.GARCH(tsrv.close, 5)

garch.RV.10 <- process.GARCH(rv.close, 10)
garch.MedRV.10 <- process.GARCH(medRV.close, 10)
garch.RK.10 <- process.GARCH(rk.close, 10)
garch.BV.10 <- process.GARCH(bv.close, 10)
garch.TSRV.10 <- process.GARCH(tsrv.close, 10)
```

Checking performance:

- Fitting

  We use RV to illustrate it.

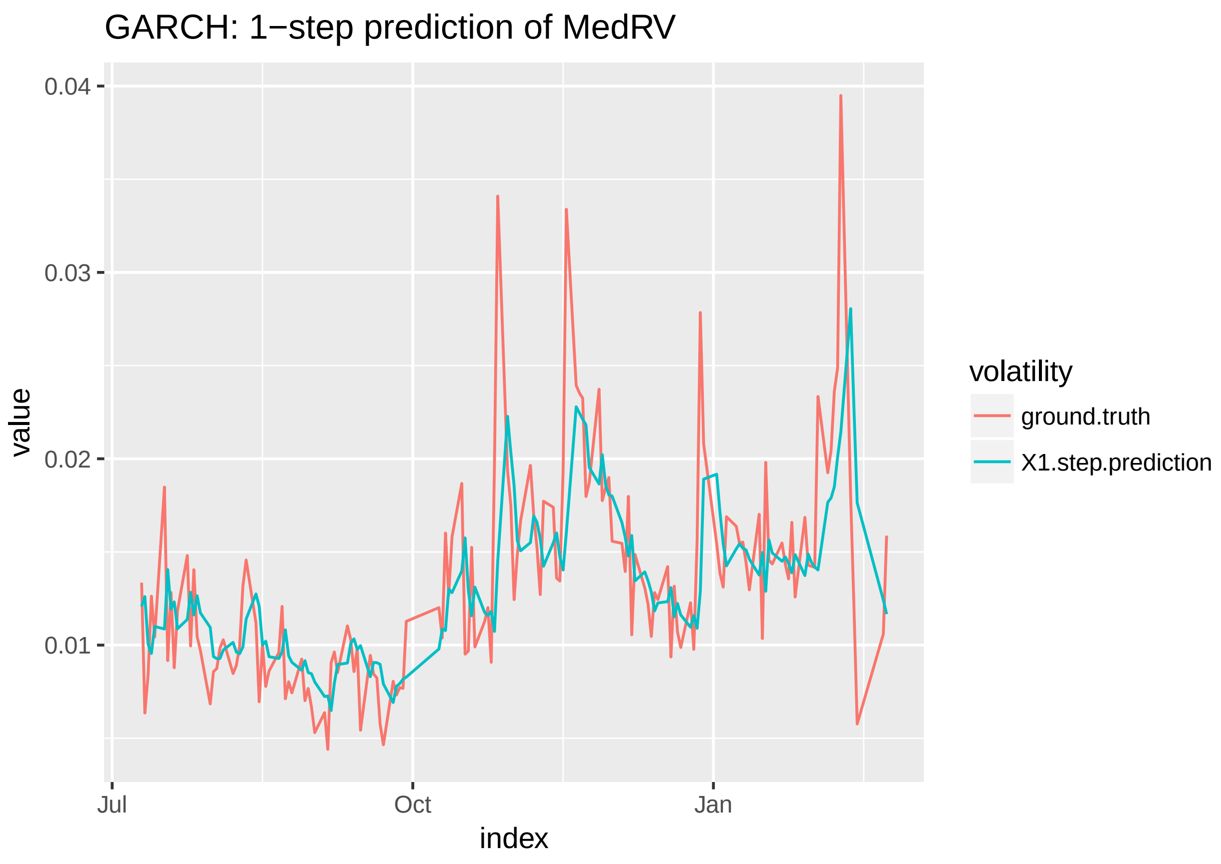
  Using the following code, we could examine the outputs of statistical tests of our fitted models. Considering the space limitation, we will now print the output here.

```r
garch.RV.1$fit
```

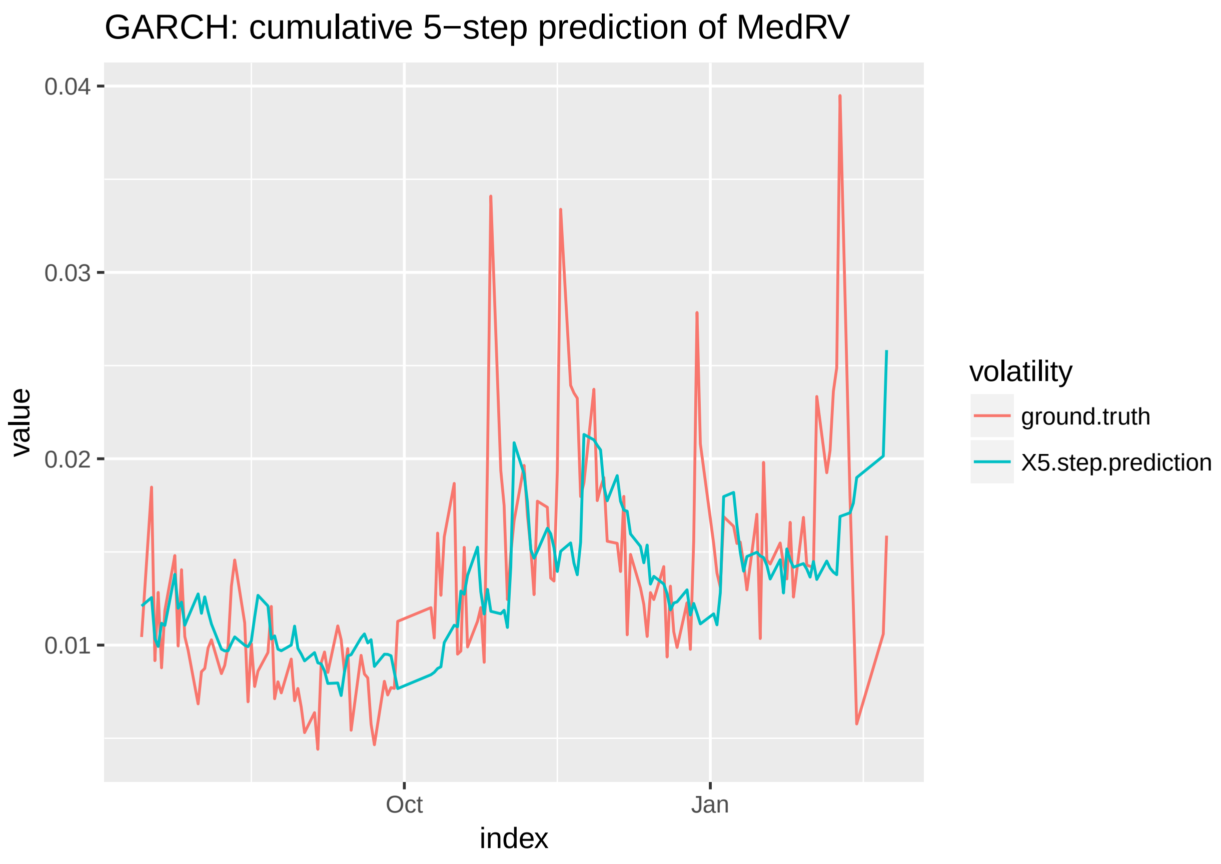
- Forecasting

We use medRV as an example for illustration.
First, plot the prediction series against the ground truth series for different step-ahead predictions.

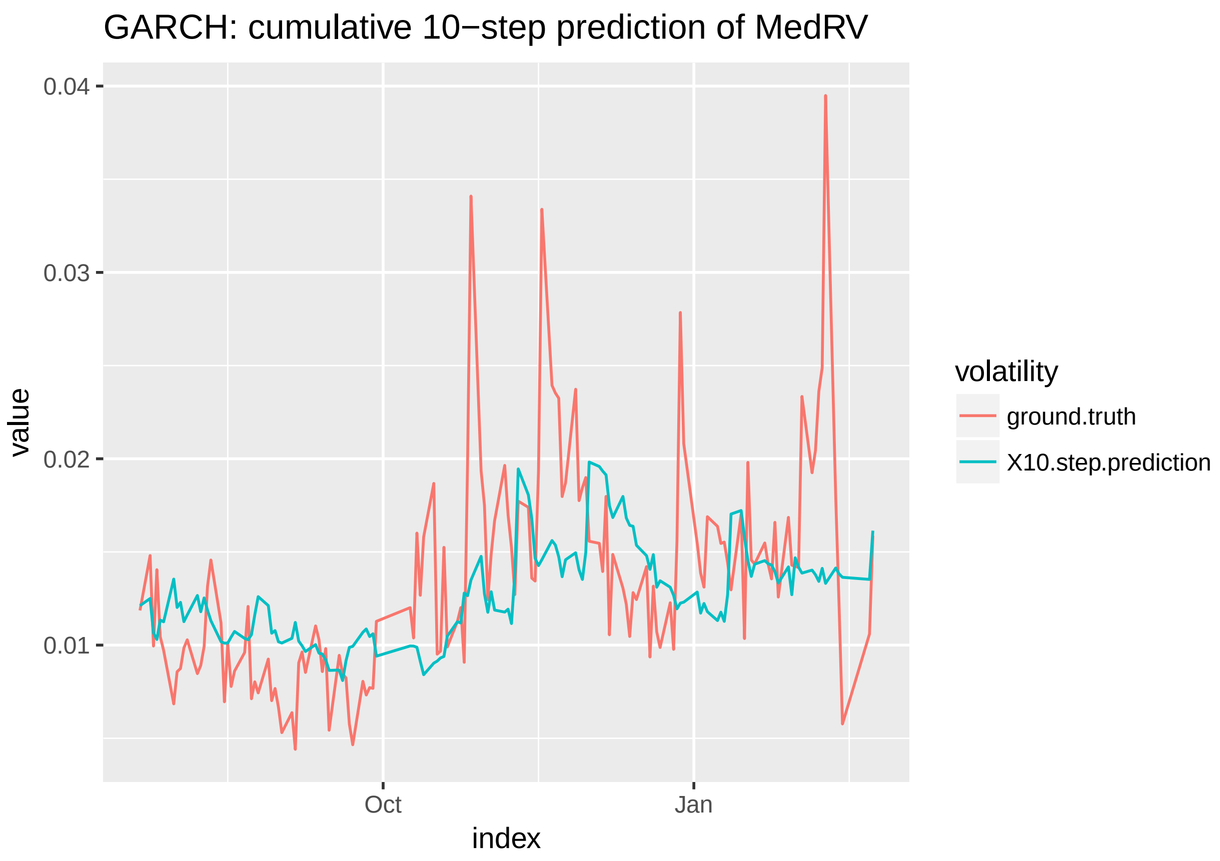
```r
plt.multi.series(cbind(medRV.close[(second.rv+1):total.rv], garch.MedRV.1$pred$pred.ts), cbind('gr
```



```
plt.multi.series(cbind(medRV.close[(second.rv+5):total.rv], garch.MedRV.5$pred$pred.ts), cbind('gr
```



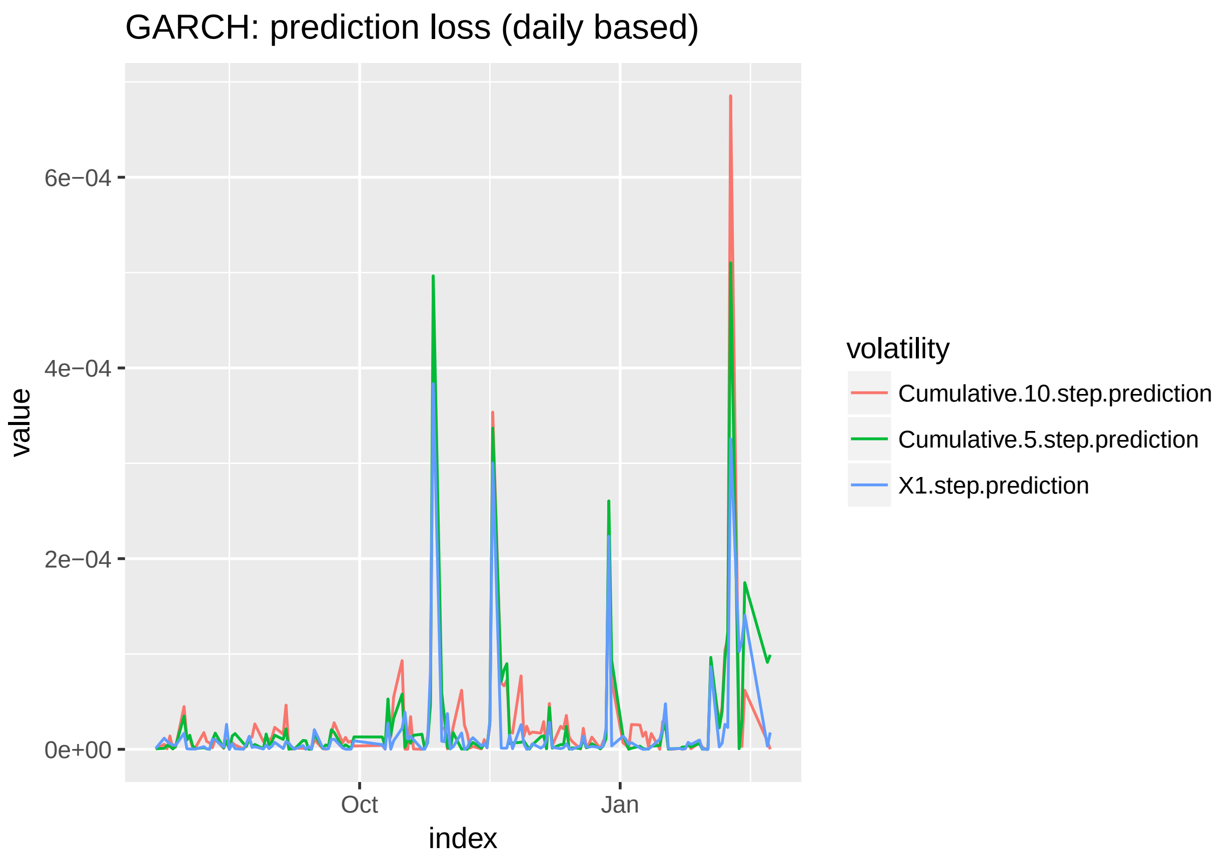
```
plt.multi.series(cbind(medRV.close[(second.rv+10):total.rv], garch.MedRV.10$pred$pred.ts), cbind('
```



Then, we could plot the daily prediction loss which is computed by the squared error:

$$loss = (v_t - \hat{v}_t)^2$$

```
l <- length(garch.MedRV.1$loss)
plt.multi.series(cbind(garch.MedRV.1$loss[10:l],garch.MedRV.5$loss[6:(l-4)], garch.MedRV.10$loss),
```



From the plots above, we could observe that as the number of the steps increases, the prediction accuracy decreases. In addition, cumulative prediction seems to be under-predicting and smoother, which is reasonable because the cumulative prediction is the average of several step-head predictions.

3.2.2 The linear heterogeneous autoregressive (HAR) model

HAR model is a simple and popular model for forecasting realized volatility. It is based on heterogeneous realized volatility components:

$$v_{t+1} = \alpha + \beta_D v_t + \beta_W v_{t,t-5} + \beta_M v_{t,t-5} + \epsilon_{t+1},$$

where $v_{t,t-k} = \frac{1}{k}\sum_{l=0}^{k-1} v_{t-j}$ is the average v_t over the past k days; where ${}_{t,h}$ is chosen from the estimated quadratic variation or its components, $\sqrt{\widehat{QV}^{(est)}_{t,h}}$, and $\sqrt{\widehat{IV}^{(est)}_{t,h}}$; and where (est) are the RV, RK, TRSV, CBV, MedRV, and JWTSRV measures.

First, define auxiliary functions for fitting and forecasting.

```
predict.HAR <- function(har.model, ts, start, end, h) {
  beta0 <- as.numeric(har.model$coefficients[1])
  beta1 <- as.numeric(har.model$coefficients[2])
  beta2 <- as.numeric(har.model$coefficients[3])
  beta3 <- as.numeric(har.model$coefficients[4])

  values <- coredata(ts)
  output <- c()
  for(i in start : (end - h)) {
    windows <- values[(i-21):i]
    # windows would record [v(i-21), ...,v(i), vhat(i+1), ..., vhat(i+h)]
    # windows[1,...,22| 22+1, ..., 22+h]
    #print(i)
    for(j in 23 : (22+h)) {
      RVt <- windows[(j-1)]
      RVt5 <- mean(windows[(j-5):(j-1)])
      RVt22 <- mean(windows[(j-22):(j-1)])
      temp <-beta0 + beta1*RVt + beta2*RVt5 + beta3*RVt22
      windows <- c(windows,temp)
    }
    temp <- mean(windows[23:(22+h)]) # obtain the cumulative h-step rv estimation
    output <- c(output, temp)
  }
  prediction <- xts(output, time(ts[(start+h):end]))
}

HAR <- function(rvdata) {
  x.rv = harModel(data=rvdata[1:second.rv] , periods = c(1,5,22), RVest = c("rCov"),  # after insp
    type="HARRV",h=1,transform=NULL);
return(x.rv)
}

#for whole process
process.HAR <- function(rv, h) {
  vol <- coredata(rv)
  total <- length(vol)
  second <- round(total*0.8)
  #fit
  fit <- HAR(rv)
  #forecast
  pred <- predict.HAR(fit, rv, start = second, end = total, h = h) # a xts object
  #errors
  errors <- getErrors(vol[(second+h):total], coredata(pred))
```

```r
  #loss
  loss <- (vol[(second+h):total]- coredata(pred))**2
  loss <- xts(loss, index(rv[(second+h): total]))
  return(list("fit" = fit, "pred" = pred, "errors" = errors, "loss" = loss))
}
```

We compute the 1-step prediction for illustrations.

```r
har.RV.1 <- process.HAR(rv.close, 1)
har.MedRV.1 <- process.HAR(medRV.close, 1)
har.RK.1 <- process.HAR(rk.close, 1)
har.BV.1 <- process.HAR(bv.close, 1)
har.TSRV.1 <- process.HAR(tsrv.close, 1)
```

Check the performance:

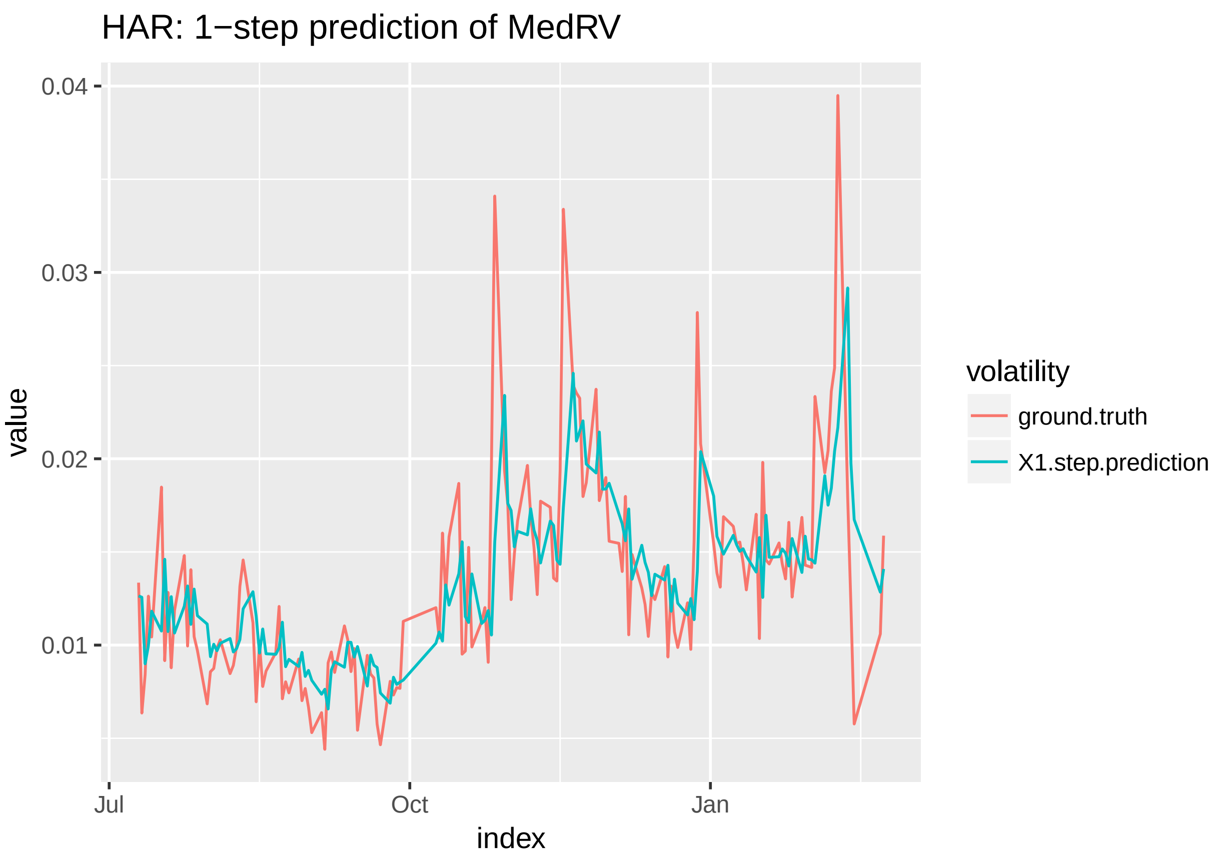
- Fitting

```r
har.RK.1$fit
```

```
##
## Model:
## RV1 = beta0  +  beta1 * RV1 +  beta2 * RV5 +  beta3 * RV22
##
## Coefficients:
##    beta0     beta1     beta2     beta3
## 0.002177  0.387876  0.307624  0.161123
##
##
##     r.squared  adj.r.squared
##        0.4573         0.4546
```

- Forecasting

```r
plt.multi.series(cbind(medRV.close[(second.rv+1):total.rv], har.MedRV.1$pred), cbind('ground-truth
```



3.2.3 Long-memory autoregressive fractionally integrated moving average (ARFIMA)

v_t is an ARFIMA(p, d, q) if it follows:

$$\alpha(L)(1-L)^d)(v_t-\mu)=\beta(L)v_t,$$

where $\alpha(z)=1-\alpha_1 z-\cdots-\alpha_p z^p$ and $\beta(z)=1+\beta_1 z-\cdots-\beta_q z^q$ are polynomials of order p and q , respectively, in the lag operator L , which is rooted strictly outside the unit circle, v_t is iid with zero mean and σ_v^2 variance, and $(1-L)^d$ is defined by its binomial expansion.

In the paper, a simple ARFIMA(1, d, 0) is estimated. And we will use this model in our implementation.

First, we define the auxiliary functions.

```
predict.ARFIMA <- function(arfima.fit, h, roll) {
    # roll = total.rv - second.rv// pred: second.rv (T+1,..,T+h),...,
    # total.rv(T+1,...,T+h)
    pred <- arfimaforecast(arfima.fit, n.ahead = h, n.roll = roll)
    pred.data <- pred@forecast$seriesFor  # a matrix, row: T+1, .., T+h; col:days
    pred.cul.data.matrix <- rollapply(pred.data, width = h, mean, by.column = TRUE)  # matrix, one
    pred.cul.data <- pred.cul.data.matrix[1, 1:(roll - h + 1)]
    pred.cul.time <- colnames((pred.cul.data.matrix))[(1 + h):(roll + 1)]
```

```r
    pred.ts <- xts(unname(pred.cul.data), as.POSIXct(pred.cul.time))

    return(list(pred = pred, pred.ts = pred.ts))
}

# for whole process
process.ARFIMA <- function(rv, h) {
    vol <- coredata(rv)
    total <- length(vol)
    second <- round(total * 0.8)
    # fit
    spec <- arfimaspec(mean.model = list(armaOrder = c(1, 0), arfima = TRUE))
    fit <- arfimafit(spec, rv, out.sample = (total - second), solver = "nlminb")
    # forecast
    pred <- predict.ARFIMA(fit, h = h, roll = (total - second))
    # errors
    errors <- getErrors(vol[(second + h):total], coredata(pred$pred.ts))
    # loss
    loss <- (vol[(second + h):total] - coredata(pred$pred.ts))^2
    loss <- xts(loss, index(rv[(second + h):total]))
    return(list(fit = fit, pred = pred, errors = errors, loss = loss))
}
```

```r
arfima.RV.1 <- process.ARFIMA(rv.close, 1)
arfima.MedRV.1 <- process.ARFIMA(medRV.close, 1)
arfima.RK.1 <- process.ARFIMA(rk.close, 1)
arfima.BV.1 <- process.ARFIMA(bv.close, 1)
arfima.TSRV.1 <- process.ARFIMA(tsrv.close, 1)
```

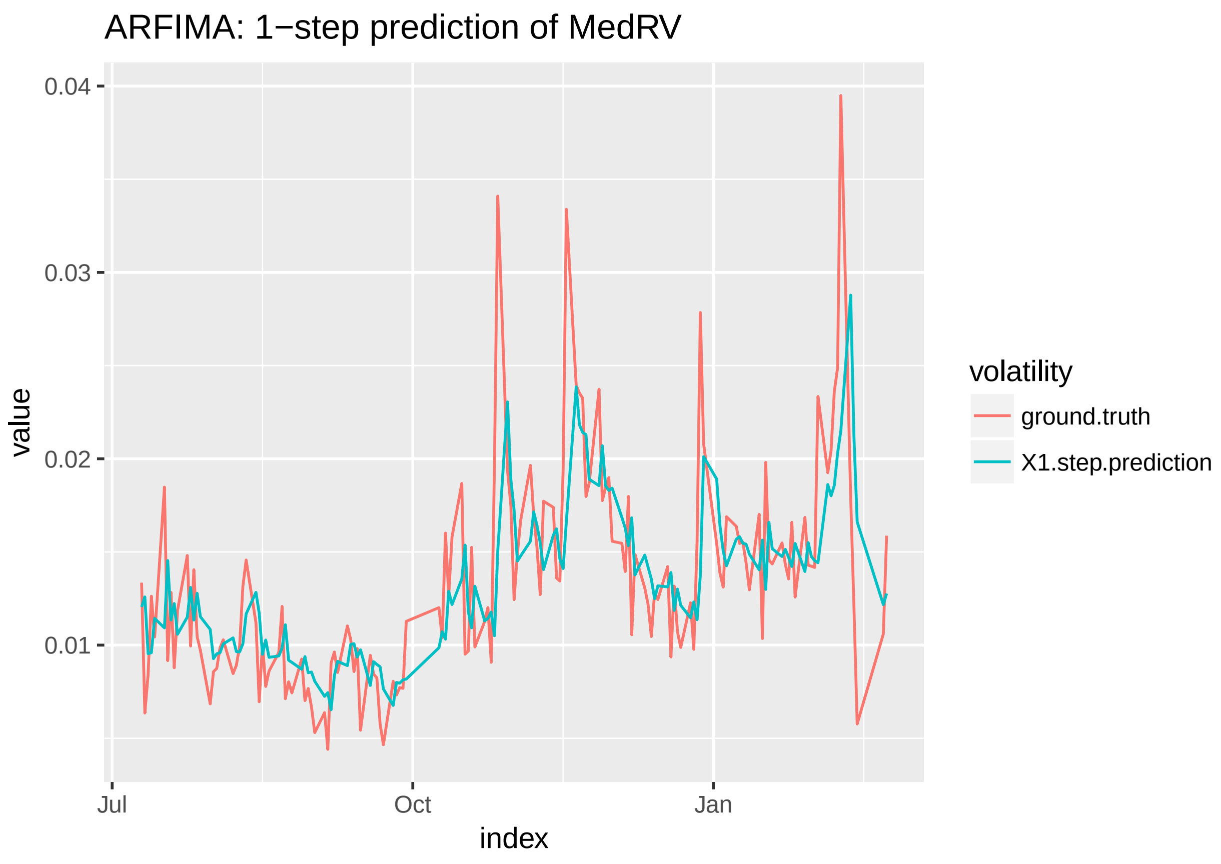
Check the performance:

- Fitting Use the code below to check detailed information of fitting like optimal parameters, robust standard errors, log likelihood, etc. We suppress the output here.

```r
arfima.RV.1$fit
```

2. Forecasting

```r
plt.multi.series(cbind(medRV.close[(second.rv+1):total.rv], arfima.MedRV.1$pred$pred.ts), cbind('g
```



3.2.4 NeuralNetwork

In the paper, the general feed-forward or multilayered perception (MLP) network is adopted, which is described by as follows:

$$n_{k,t} = \omega_{k,\alpha} + \sum_{i=0}^{21} \omega_{k,i} v_{t-i}$$

$$v_{t+h} = \gamma_0 + \sum_{k=1}^{k^\star} \gamma_k \Lambda(n_{k,t})$$

where n is the neuron, v is the volatility, ω and γ are weights and $\Lambda(.)$ is the activation function. In the paper, the logistic function:

$$\Lambda(x) = 1/(1 + e^{-x}),$$

is used for the activation function.

From the definition above, we could see that the paper's neural network is fed with volatilities within consecutive 21 days and predict the volatility on the 22rd day.

Moreover, the paper adopts resilient propagation algorithm to train the neural network. And there are only hyper-parameter to tune: either 7 or 15 hidden neurons (the paper's ANN model only have 1 hidden layer).

In this project, we will add more hyper-parameters and try deeper networks to enhance the performance, which we will described below.

First, define auxiliary functions for fitting and prediction. Our NN model will take in

- volatility time series
- the number of step-ahead for prediction
- lists of hidden units (e.g. [2,2,3] implies to 2 hidden units for the first hidden layer, 2 for the second and 3 for the last)
- CV – whether to include Cross Validation or not
- learning-rate
- activation function

```r
library(neuralnet)
library(plyr)
# predict from start -- start+h, start+h+1, ..., end
predict.ANN <- function(nn.fit, df, start, end, h) {
  i <- 1
  new.df <- df[(start-21):(end-22), 2:23] # 2 : 23 implies day1, .., day22
  for (i in 1:h) {
    pred.ann <- compute(nn.fit, new.df[,i:(21+i)])
    pred.result <- pred.ann$net.result[,1]
    new.df[paste("day",22+i,rep="")] <-  pred.result
  }
  l <- end - (start+h) + 1
  if (h == 1) {
    pred.data <- new.df[,23]
  } else{
    pred.data <- rowMeans(new.df[1:l,23:(22+h)])
  }
  pred.ts <- xts(pred.data, df[(start+h):end,"date"])
  return(list("ts" = pred.ts, "df"=new.df))
}

#for whole process
process.ANN <- function(rv, h, hidden, CV, learningrate, act.fct) {
  vol <- coredata(rv)
  total <- length(vol)
  first <- round(total*0.72)
  second <- round(total*0.8)
  df <- data.frame("date"=index(rv))
  # col1: "date"", col2: "day0", col3: "day1", .., col23: "day21"
  for(i in 1:23) {
    day.num <- paste("day", i, sep="")
    df[1:(total.rv-i+1), day.num] <- vol[i:total]
  }
  #fit
  net <- neuralnet(day23~day1+day2+day3+day4+day5+day6+day7+day8+day9+day10+day11+day12+day13+day1
  if(CV == TRUE) {
    #training set
    train <- predict.ANN(net, df, 22, first, h)
    train.errors <- getErrors(vol[(22+h):first], coredata(train$ts))
    #CV set
    cv <- predict.ANN(net, df, first, second, h)
    #cv.errors
    cv.errors <- getErrors(vol[(first+h):second], coredata(cv$ts))
  } else{
    #training set
    train <- predict.ANN(net, df, 22, second, h)
    train.errors <- getErrors(vol[(22+h):second], coredata(train$ts))
    cv.errors = NA
    cv = NULL
    }
  #forecast -- test set
  pred <- predict.ANN(net, df, second, total, h)
  #errors
  errors <- getErrors(vol[(second+h):total], coredata(pred$ts))
  #loss
  loss <- (vol[(second+h):total]- coredata(pred$ts))**2
  loss <- xts(loss, index(rv[(second+h): total]))
  return(list("fit" = net, "pred" = pred, "cv" = cv, "cv.errors" = cv.errors, "errors" = errors, "
}
```

- Cross-validation

We have three set of hyper-parameters to tune and below are the range of tuning: + learning-rate=[0.01, 0.001, 0.0001, 0.00001] + hidden units = [(7), (15), (7, 15), (7,7,15), (7,7,15,15), …] + activation function = [“tanh”, “logistic”“].

We will pick the desired combination according to the CV-set errors.

Here are some examples during the cross validation.

Different learning rates.

```r
ann.RV.model10 <- process.ANN(rv.close, 1, c(7), TRUE, 0.01, act.fct = "tanh")
ann.RV.model10$cv.errors$rmse
```

```
## [1] 0.00347169765
```

```r
ann.RV.model10 <- process.ANN(rv.close, 1, c(7), TRUE, 0.001, act.fct = "tanh")
ann.RV.model10$cv.errors$rmse
```

```
## [1] 0.003493745001
```

```r
ann.RV.model10 <- process.ANN(rv.close, 1, c(7), TRUE, 0.0001, act.fct = "tanh")
ann.RV.model10$cv.errors$rmse
```

```
## [1] 0.003543772807
```

```r
ann.RV.model10 <- process.ANN(rv.close, 1, c(7), TRUE, 0.00001, act.fct = "tanh")
ann.RV.model10$cv.errors$rmse
```

```
## [1] 0.003318384964
```

Different activation functions and hidden units.

```r
ann.RV.model1 <- process.ANN(rv.close, 1, c(7), TRUE, 0.0001, act.fct = "tanh")
ann.RV.model1$cv.errors$rmse
```

```
## [1] 0.003291937731
```

```r
ann.RV.model2 <- process.ANN(rv.close, 1, c(15), TRUE, 0.0001, act.fct = "tanh")
ann.RV.model2$cv.errors$rmse
```

```
## [1] 0.003368262773
```

```r
ann.RV.model3 <- process.ANN(rv.close, 1, c(7), TRUE, 0.0001, act.fct = "logistic")
ann.RV.model3$cv.errors$rmse
```

```
## [1] 0.003483661861
```

```r
ann.RV.model4 <- process.ANN(rv.close, 1, c(15), TRUE, 0.0001, act.fct = "logistic")
ann.RV.model4$cv.errors$rmse
```

```
## [1] 0.00349474626
```

After cross-validation, we choose this combination:
hidden unites=[7], learning rate=0.0001, and activation function = "logistic".

However, there are no obvious differences on their performance on CV set, especially considering the random errors.

Another thing that is worth noticing, is that the model seems to fail to "over fit". As is known to all, first step in neural network fitting is to make sure that the model could over fit.

```
print(cat("Training set error(rmse):", ann.RV.model1$train.errors$rmse))
```

```
## Training set error(rmse): 0.006680175445NULL
```

```
print(cat("CV set error(rmse):", ann.RV.model1$cv.errors$rmse))
```

```
## CV set error(rmse): 0.003291937731NULL
```

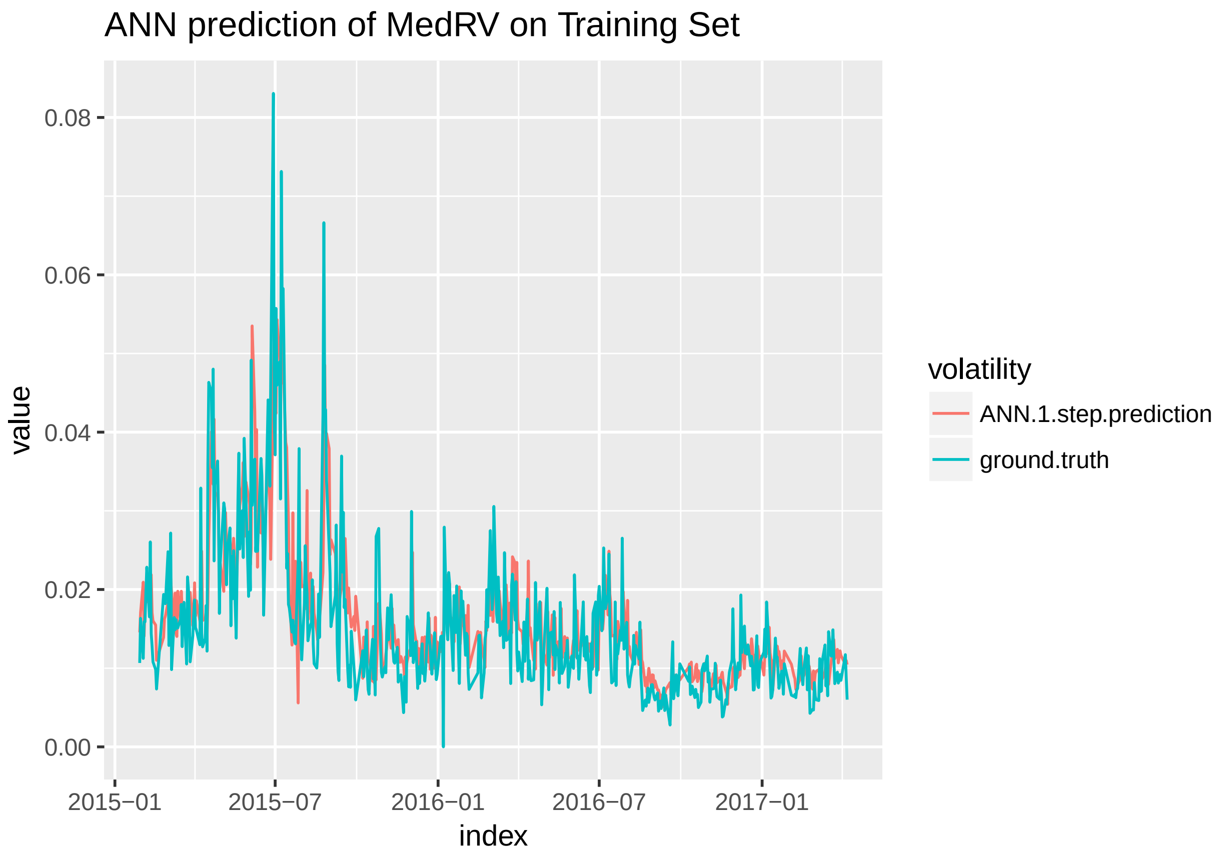
```
print(cat("Test set error(rmse):", ann.RV.model1$errors$rmse))
```

```
## Test set error(rmse): 0.004486361693NULL
```

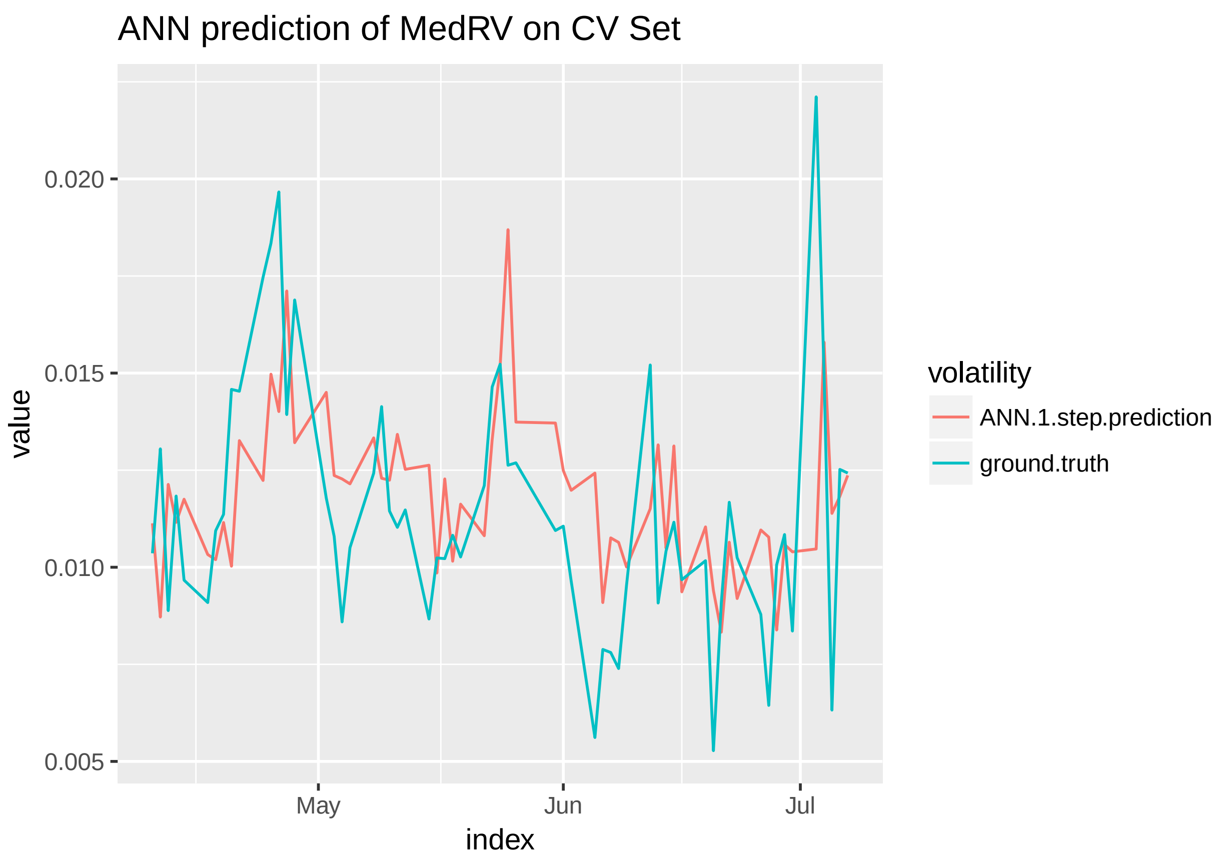
From above, we could see that the training set error is very large, even larger than CV set and test set

By plotting the ANN-prediction against the original time series, the possible reason might be that the volatilities in training set oscillates much stronger than the those in CV set and test set. Consequently, it would be harder for a simple, shallow neural network to capture the inner infrastructure to volatility series.

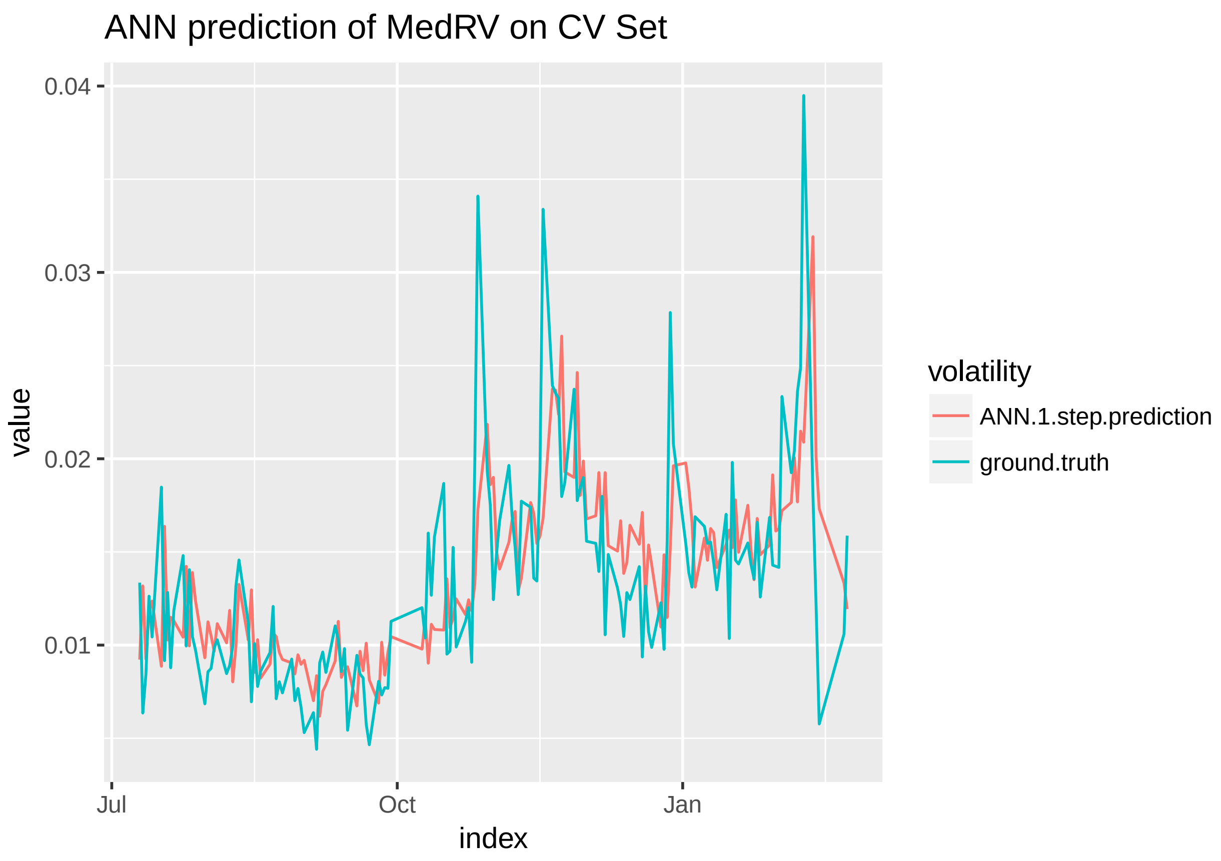
```
plt.multi.series(cbind(medRV.close[(22 + 1):first.rv], ann.RV.model1$train$ts),
    cbind("ground-truth", "ANN 1 step prediction")) + labs(title = "ANN prediction of MedRV on Tra
    colour = "volatility")
```



```
plt.multi.series(cbind(medRV.close[(first.rv + 1):second.rv], ann.RV.model1$cv$ts),
    cbind("ground-truth", "ANN 1 step prediction")) + labs(title = "ANN prediction of MedRV on CV
    colour = "volatility")
```



```
plt.multi.series(cbind(medRV.close[(second.rv + 1):total.rv], ann.RV.model1$pred$ts),
    cbind("ground-truth", "ANN 1 step prediction")) + labs(title = "ANN prediction of MedRV on CV
    colour = "volatility")
```

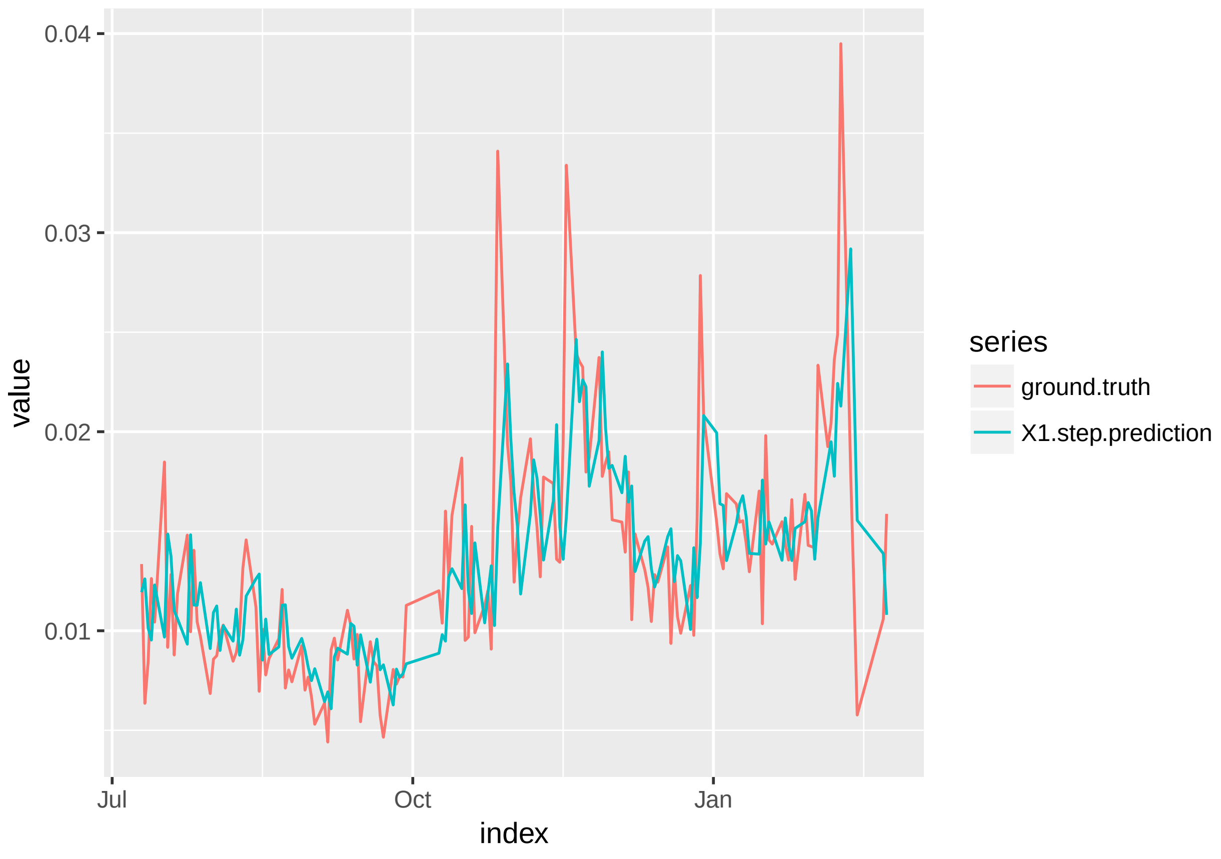


- 1-step head prediction

```
ann.RV.1 <- process.ANN(rv.close, 1, c(7), FALSE, 0.0001, act.fct = "logistic")
ann.MedRV.1 <- process.ANN(medRV.close, 1, 7, FALSE, 0.0001, act.fct = "logistic")
ann.RK.1 <- process.ANN(rk.close, 1, c(7), FALSE, 0.0001, act.fct = "logistic")
ann.BV.1 <- process.ANN(bv.close, 1, c(7), FALSE, 0.0001, act.fct = "logistic")
ann.TSRV.1 <- process.ANN(tsrv.close, 1, 7, FALSE, 0.0001, act.fct = "logistic")
```

- Plotting the results

```
plt.multi.series(cbind(medRV.close[(second.rv + 1):total.rv], ann.MedRV.1$pred$ts),
    cbind("ground-truth", "1 step prediction"))
```



3.2.5 HAR-ANN

An HAR-ANN model is simply averaging the results of HAR model output and ANN model output.

```
process.HAR.ANN <- function(rv, har.fit, ann.fit, h) {
  vol <- coredata(rv)
  total <- length(vol)
  first <- round(total*0.6)
  second <- round(total*0.8)

  pred.data <- (coredata(ann.fit$pred$ts)+ coredata(har.fit$pred))/2
  pred <- xts(pred.data, time(har.fit$pred))

  #errors
  errors <- getErrors(vol[(second+h):total], coredata(pred))
  #loss
  loss <- (vol[(second+h):total]- coredata(pred))**2
  loss <- xts(loss, index(rv[(second+h): total]))
  return(list("pred" = pred, "errors" = errors, "loss" = loss))
}
```

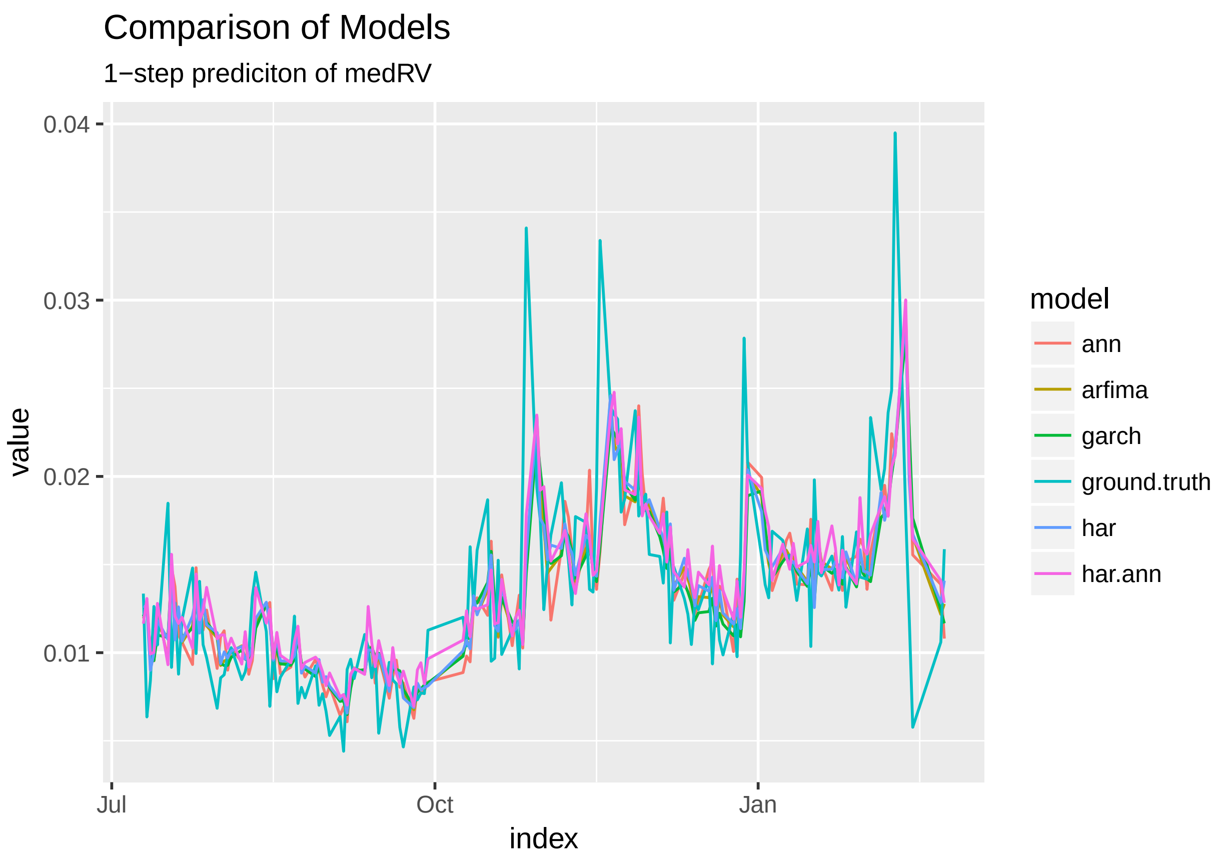
```
#1-step prediction
har.ann.RV.1 <- process.HAR.ANN(rv.close, har.RV.1, ann.RV.1, 1)
har.ann.MedRV.1 <- process.HAR.ANN(medRV.close, har.RV.1, ann.RV.1, 1)
har.ann.RK.1 <- process.HAR.ANN(rk.close, har.RV.1, ann.RV.1, 1)
har.ann.BV.1 <- process.HAR.ANN(bv.close, har.RV.1, ann.RV.1, 1)
har.ann.TSRV.1 <- process.HAR.ANN(tsrv.close, har.RV.1, ann.RV.1, 1)
```

3.3 Comparison of Models

In this part, we will make comparison between models.

First, we plot the predictions to obtain an overview.

```
plt.multi.series(cbind(medRV.close[(second.rv + 1):total.rv], garch.MedRV.1$pred$pred.ts,
    har.MedRV.1$pred, arfima.MedRV.1$pred$pred.ts, ann.MedRV.1$pred$ts, har.ann.MedRV.1$pred),
    cbind("ground-truth", "garch", "har", "arfima", "ann", "har-ann")) + labs(title = "Comparison
    subtitle = "1-step prediciton of medRV", colour = "model")
```



We can use mmeo error to gain the over-prediction error.

Using BV and MedRV as examples:

```r
garch.BV.1$errors$mmeo # 1st
```

```
## [1] 0.02692721586
```

```r
har.BV.1$errors$mmeo # 3rd
```

```
## [1] 0.02824104303
```

```r
arfima.BV.1$errors$mmeo # 2nd
```

```
## [1] 0.0273001912
```

```r
ann.BV.1$errors$mmeo # 4th
```

```
## [1] 0.02940807273
```

```r
har.ann.BV.1$errors$mmeo # 5th
```

```
## [1] 0.03436260181
```

```r
garch.MedRV.1$errors$mmeo # 1st
```

```
## [1] 0.02577848752
```

```r
har.MedRV.1$errors$mmeo # 3rd
```

```
## [1] 0.02692863677
```

```r
arfima.MedRV.1$errors$mmeo # 2nd
```

```
## [1] 0.0263925133
```

```r
ann.MedRV.1$errors$mmeo # 4th
```

```
## [1] 0.02887978069
```

```r
har.ann.MedRV.1$errors$mmeo # 5th
```

```
## [1] 0.03085617167
```

However, after checking on over all volatility estimators, there are no models that have consistent performance on the lowest over-prediction error (mmeo).

4 Conclusions and Reflections

- **Conclusions and Discussions**:

The paper asserts that the ANN model has the advantage of preventing over-prediction. However, according to the indicator mmao, we have not found a model that has consistent performance of preventing over-prediction.

Moreover, the 5 models make no big difference in prediction errors. Specifically, the ANN model did not outperform others. This is probably due to the following reasons. 1. Our data set consists of 770 observations, while the data set in the paper consists of 1631 observations. The relative short of data makes ANN model harder to train.
2. There are strong oscillations in the training set data, and rather smoother trend in the test set data, which makes ANN harder to generalize on training set, therefore leading to under-fitting. 3. The ANN model provided in the paper is very simple – only one hidden layer with 7 or 15 hidden neurons. However, even after adding more hyper-parameters, no optimal choice is obtained.

In addition, there seems to be a forward lag between the cumulative 5-step and cumulative 10-step prediction and the ground-truth data, which we assume to be explained by the averaging effect from the definition of cumulative h-step prediction. Despite of the lag and the relative large error, this cumulative prediction is more interesting in application because it provides a smoother and more stable prediction.

- **Reflections**:

1. Get more data.
2. Get a more complicated neural network (by reading literature) which could adjust to high oscillations.
3. The time dependency feature is crucial for predicting time series. However, the proposed ANN in the paper only utilizes this feature in the input, by feeding 21 consecutive price series. If the layers get deeper, it would be hard for the network to remember the feature or time-dependency, or consecutive input in this case. Maybe we could try other architectures like RNN. (My teammate has implemented LSTM-RNN.)

5 Data Output

Output the loss

```
df.loss <- data.frame(date = index(rv.close[(second.rv + 1):total.rv]), garch.RV.1$loss,
    garch.RK.1$loss, garch.MedRV.1$loss, garch.BV.1$loss, garch.TSRV.1$loss,
    har.RV.1$loss, har.RK.1$loss, har.MedRV.1$loss, har.BV.1$loss, har.TSRV.1$loss,
    arfima.RV.1$loss, arfima.RK.1$loss, arfima.MedRV.1$loss, arfima.BV.1$loss,
    arfima.TSRV.1$loss, ann.RV.1$loss, ann.RK.1$loss, ann.MedRV.1$loss, ann.BV.1$loss,
    ann.TSRV.1$loss, har.ann.RV.1$loss, har.ann.RK.1$loss, har.ann.MedRV.1$loss,
    har.ann.BV.1$loss, har.ann.TSRV.1$loss)
write.xlsx(df.loss, "loss.xlsx")
```

Output the computed volatilities

```
df.vols <- data.frame(RV = coredata(rv.close), RK = coredata(rk.close), MedRV = coredata(medRV.clo
    BV = coredata(bv.close), TSRV = coredata(tsrv.close))
write.xlsx(df.vols, "vols.xlsx")
```