KF prototype

Cleaning the data

```r
library(data.table)
library(lubridate)
library(zoo)
library(ggplot2)

var_names <- c('timestamp',  'opening', 'high', 'low', 'close', 'volume')
dir <- './dataset/toyset/'

custom_merge <- function(x, y){
  return(merge(x, y, by = 'timestamp', all = TRUE))
}

create_timestamps <- function(start_date, end_date, opening_time, closing_time){
  date_seq <- seq.Date(start_date, end_date, by = 'day')
  temp <- list()
  for(i in 1:length(date_seq)){
    if(!is.na(opening_time[wday(date_seq[i])])){
      temp[[i]] <- data.table(
        timestamp = seq.POSIXt(
          ymd_hms(paste(date_seq[i], opening_time[wday(date_seq[i])])),
          ymd_hms(paste(date_seq[i], closing_time[wday(date_seq[i])])),
          by = 'min'
        )
      )
    }
  }
  return(rbindlist(temp))
}

get_data <- function(dir){
  filenames <- list.files(dir)
  dat <- list()
  for(i in 1:length(filenames)){
    temp_dat <- fread(paste0(dir, filenames[i]))
    setnames(temp_dat, var_names)
    temp_dat2 <- temp_dat[,.(
      timestamp = as.POSIXct(timestamp, format = "%Y%m%d %H%M%OS", tz = 'EST'),
      price = log(close)
      )]
    setnames(temp_dat2, c('timestamp', substr(filenames[i], 1, 6)))
    dat[[i]] <- temp_dat2
  }
  final_dat <- Reduce(custom_merge, dat)

  daily_times <- final_dat[
    ,.(opening_time = substr(timestamp, 12, 19),
       closing_time = substr(timestamp, 12, 19),
```

```r
      timestamp = as.Date(timestamp, tz = 'EST'))]
  daily_times <- daily_times[
    ,.(opening_time = head(sort(opening_time), 1),
      closing_time = tail(sort(closing_time), 1)),
    by = .(dow = wday(timestamp))
  ]
  daily_times <- daily_times[order(dow)]

  timestamp_list <- create_timestamps(
    start_date   = min(as.Date(final_dat$timestamp, tz = 'EST')),
    end_date     = max(as.Date(final_dat$timestamp, tz = 'EST')),
    opening_time = daily_times$opening_time[match(1:7, daily_times$dow)],
    closing_time = daily_times$closing_time[match(1:7, daily_times$dow)]
  )

  final_dat <- merge(timestamp_list, final_dat, by = 'timestamp', all = TRUE)
  final_dat <- final_dat[,lapply(.SD, na.locf, na.rm = FALSE, fromLast = TRUE)]
  return(final_dat)
}

dat <- get_data(dir)
dat[,time_gap := (as.numeric(timestamp) - as.numeric(shift(timestamp)))/60]
dat$time_gap[1] <- 1
head(dat)
```

```
##              timestamp  gbpjpy    gbpusd   usdjpy time_gap
## 1: 2019-05-01 00:00:00 4.97986 0.2656588 4.713935        1
## 2: 2019-05-01 00:01:00 4.97986 0.2656588 4.713935        1
## 3: 2019-05-01 00:02:00 4.97986 0.2656588 4.713935        1
## 4: 2019-05-01 00:03:00 4.97986 0.2656588 4.713935        1
## 5: 2019-05-01 00:04:00 4.97986 0.2656588 4.713935        1
## 6: 2019-05-01 00:05:00 4.97986 0.2656588 4.713935        1
```

Kalman Filter with known covariances

Here we assume independence in movements, just to get a simple prototype to work. proc_covar and meas_covar are the main settings. When proc_covar is relatively small when compared to meas_covar, then the filter will be smooth.

```r
proc_covar <- matrix(0, nrow = 3, ncol = 3)
diag(proc_covar) <- 0.0001^2

post_covar <- proc_covar

meas_covar <- matrix(0, nrow = 3, ncol = 3)
diag(meas_covar) <- 0.0003^2

pred_covar <- meas_covar

ident <- matrix(0, nrow = 3, ncol = 3)
diag(ident) <- 1

# assume the order of measurements is GBPJPY, GBPUSD, USDJPY
```

```r
# and order of latent variables is GBP, JPY, USD

state_old <- matrix(rep(1, 3), ncol = 1)
trans_mat <- matrix(c(
  1, -1, 0,
  1, 0, -1,
  0, -1, 1
), nrow = 3, ncol = 3, byrow = TRUE)

obs_mat <- as.matrix(dat[,!colnames(dat) %in% c('timestamp', 'time_gap'), with = FALSE])
time_gap <- dat$time_gap

latent_states <- list()
predicted_obs <- list()

for(i in 1:nrow(dat)){
  post_covar <- post_covar + time_gap[i] * proc_covar
  innovation <- obs_mat[i,] - trans_mat %*% state_old
  innovation_covar <- trans_mat %*% post_covar %*% t(trans_mat) + meas_covar
  kalman_gain <- post_covar %*% t(trans_mat) %*% solve(innovation_covar)
  state_new <- state_old + kalman_gain %*% innovation
  post_covar <- (ident - kalman_gain %*% trans_mat) %*% post_covar
  predicted_obs[[i]] <- t(trans_mat %*% state_new)
  latent_states[[i]] <- t(state_new)
  state_old <- state_new
}

predictions <- data.table(do.call(rbind, predicted_obs))
setnames(predictions, colnames(obs_mat))
predictions$timestamp <- shift(dat$timestamp, n = 1, type = 'lead')

latent_estimates <- data.table(do.call(rbind, latent_states))
setnames(latent_estimates, c('GBP', 'JPY', 'USD'))
latent_estimates$timestamp <- shift(dat$timestamp, n = 1, type = 'lead')
```

How good are the predictions?

```r
# baseline of one-step ahead forecast
baseline <- data.table(
  gbpjpy = dat$gbpjpy - shift(dat$gbpjpy, n = 1, type = 'lead'),
  gbpusd = dat$gbpusd - shift(dat$gbpusd, n = 1, type = 'lead'),
  usdjpy = dat$usdjpy - shift(dat$usdjpy, n = 1, type = 'lead')
)
baseline <- baseline[200:nrow(baseline)]
baseline[,lapply(.SD, function(x) sqrt(mean(x^2, na.rm = TRUE)))]
```

```
##           gbpjpy      gbpusd       usdjpy
## 1: 0.0001432096 0.000110966 9.673141e-05
```

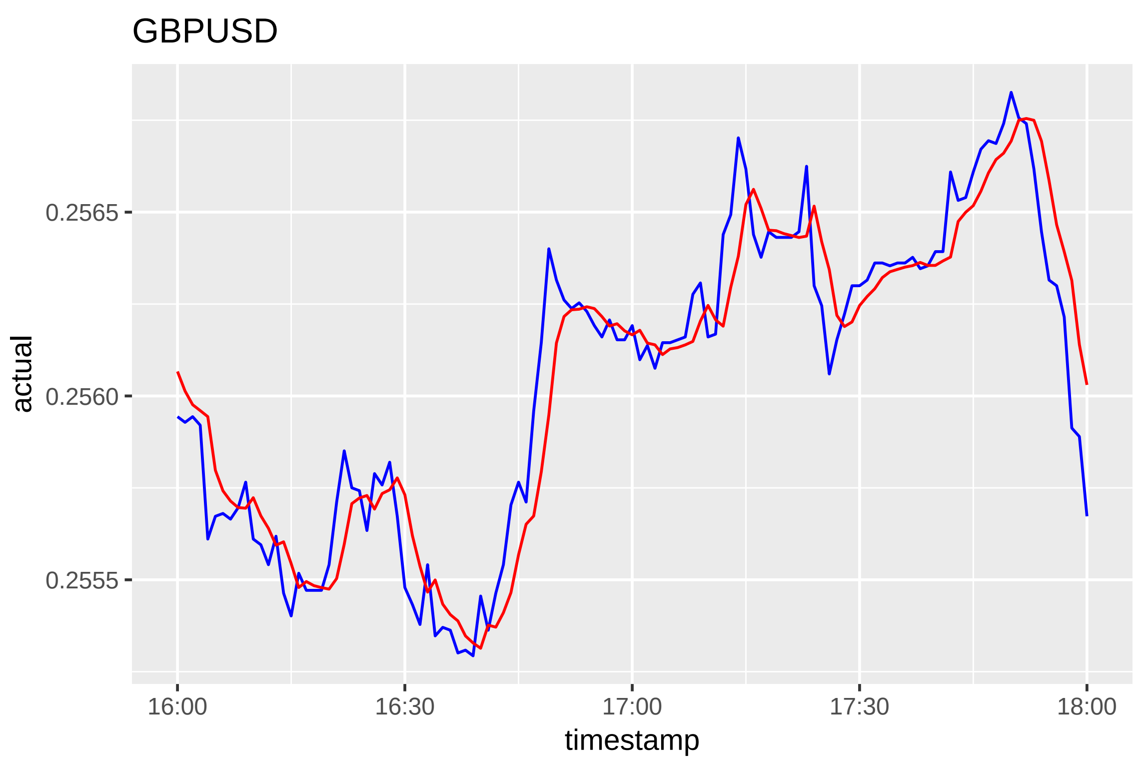
```r
predicted <- data.table(
  gbpjpy = predictions$gbpjpy - shift(dat$gbpjpy, n = 1, type = 'lead'),
  gbpusd = predictions$gbpusd - shift(dat$gbpusd, n = 1, type = 'lead'),
  usdjpy = predictions$usdjpy - shift(dat$usdjpy, n = 1, type = 'lead')
```

```
)
predicted <- predicted[200:nrow(predicted)]
predicted[,lapply(.SD, function(x) sqrt(mean(x^2, na.rm = TRUE)))]
```

```
##          gbpjpy       gbpusd       usdjpy
## 1: 0.0001791517 0.0001478329 0.0001318372
```

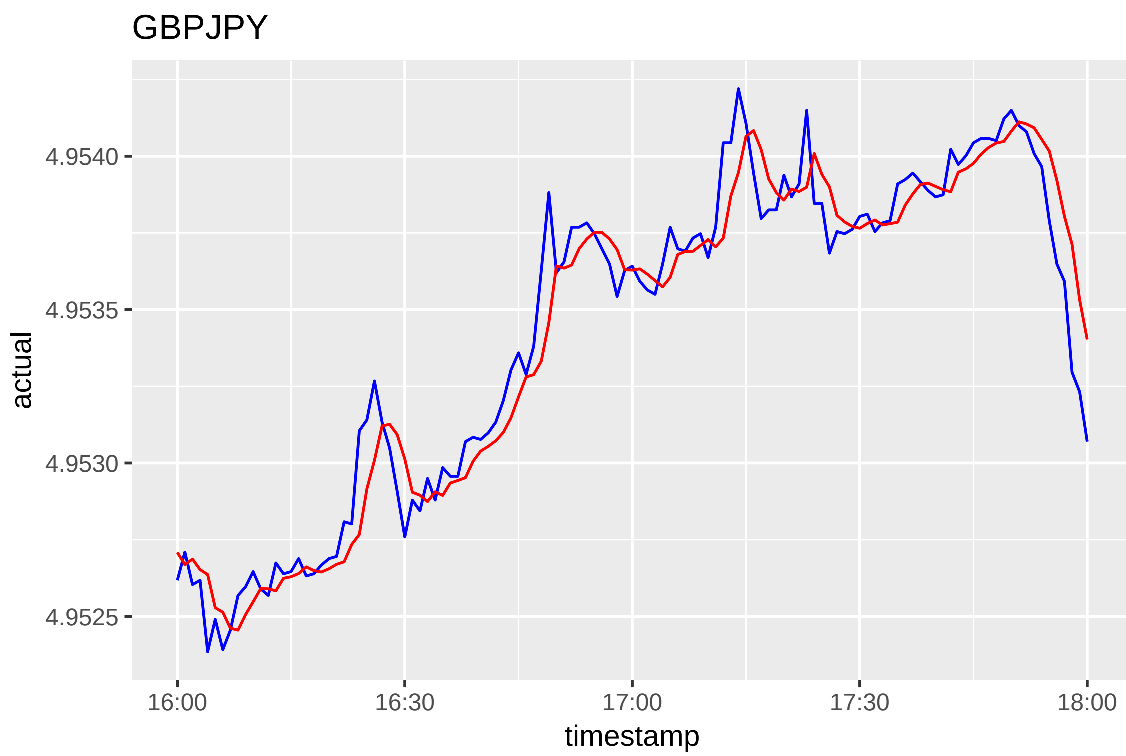
It's worse than the baseline. Let's see what's going on:

```
viz_dat <- merge(
  melt(dat[,.(timestamp, gbpjpy, gbpusd, usdjpy)],
       id.vars = 'timestamp',
       variable.name = 'currency',
       value.name = 'actual'),
  melt(predictions[,.(timestamp, gbpjpy, gbpusd, usdjpy)],
       id.vars = 'timestamp',
       variable.name = 'currency',
       value.name = 'predicted'),
  by = c('timestamp', 'currency')
)
ggplot(viz_dat[currency == 'gbpusd' &
                 timestamp >= '2019-05-14 12:00:00' &
                 timestamp <= '2019-05-14 14:00:00']) +
  geom_line(aes(x = timestamp, y = actual), color = 'blue') +
  geom_line(aes(x = timestamp, y = predicted), color = 'red') +
  ggtitle('GBPUSD')
```

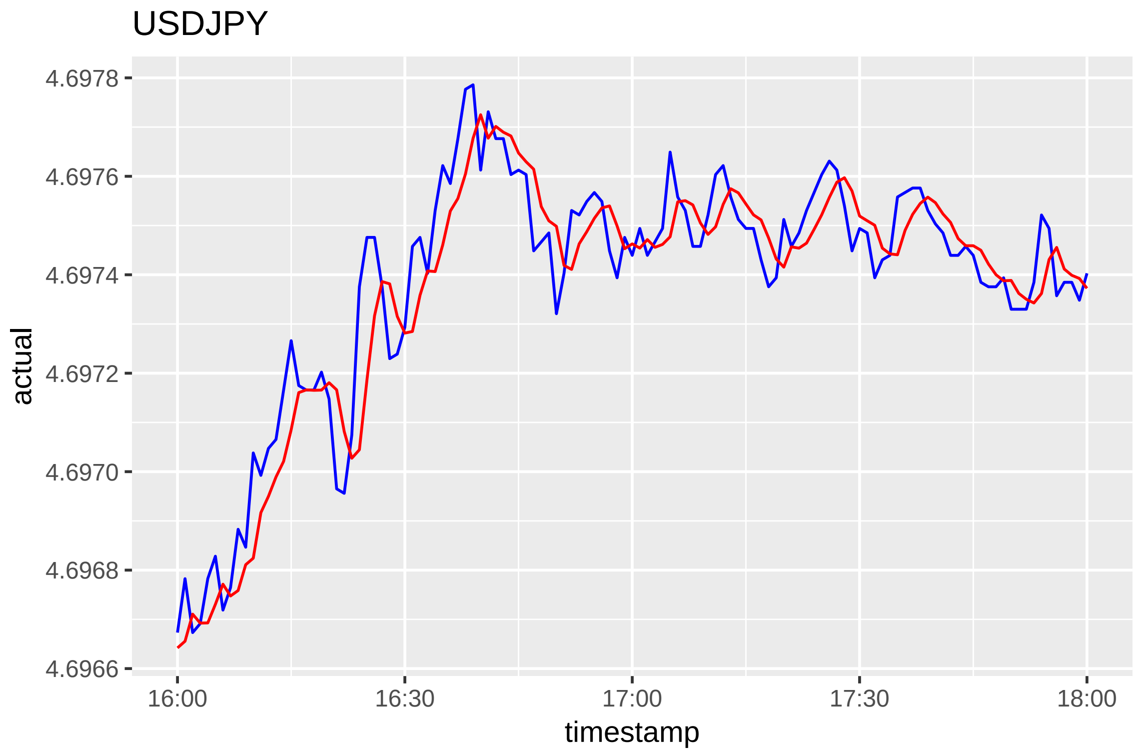



```
ggplot(viz_dat[currency == 'gbpjpy' &
                 timestamp >= '2019-05-14 12:00:00' &
                 timestamp <= '2019-05-14 14:00:00']) +
```

```
  geom_line(aes(x = timestamp, y = actual), color = 'blue') +
  geom_line(aes(x = timestamp, y = predicted), color = 'red') +
  ggtitle('GBPJPY')
```

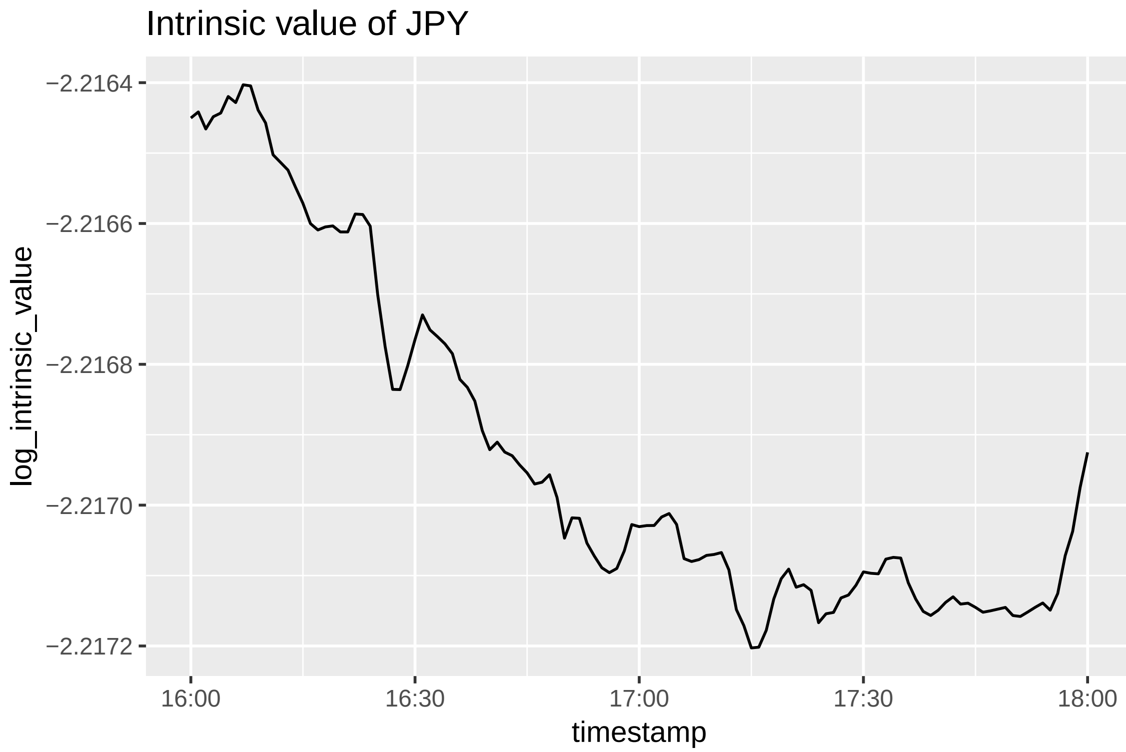



```
ggplot(viz_dat[currency == 'usdjpy' &
                 timestamp >= '2019-05-14 12:00:00' &
                 timestamp <= '2019-05-14 14:00:00']) +
  geom_line(aes(x = timestamp, y = actual), color = 'blue') +
  geom_line(aes(x = timestamp, y = predicted), color = 'red') +
  ggtitle('USDJPY')
```

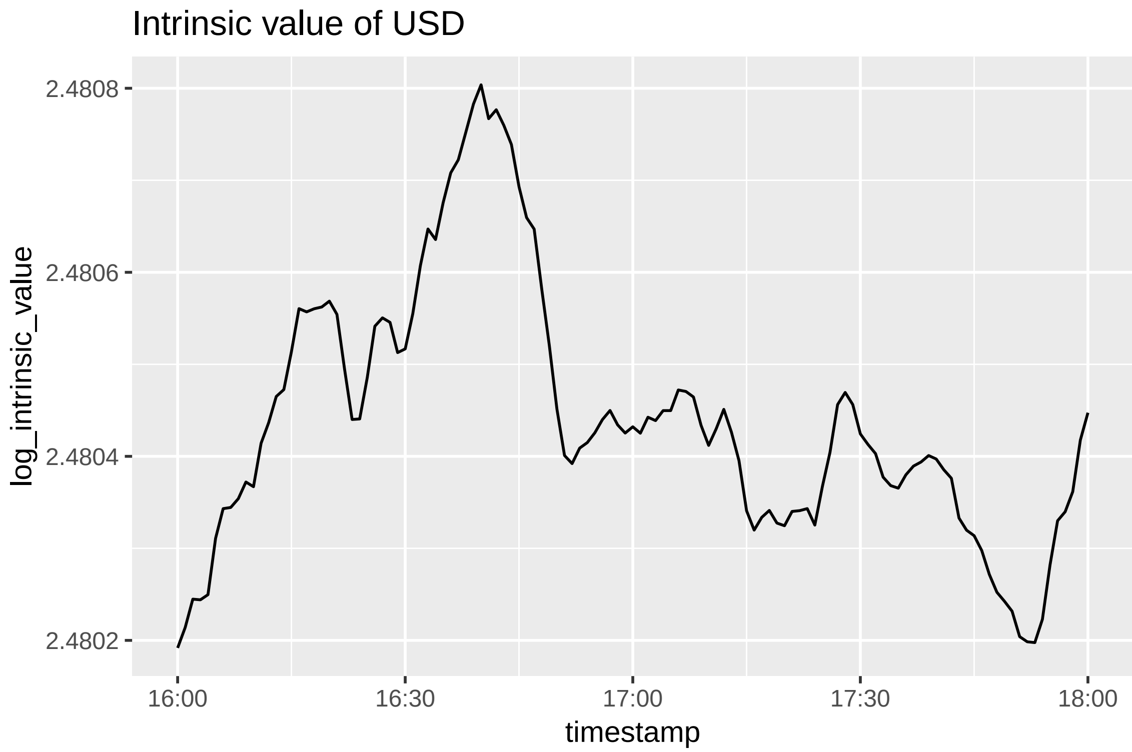


As we expected, currency exchange rates are extremely volatile, and our naive approach fails. Then again, the covariance matrices are not estimated in this case, and there may be a setting that works better. But on the bright side, the graph of currency intrinsic values is really neat:

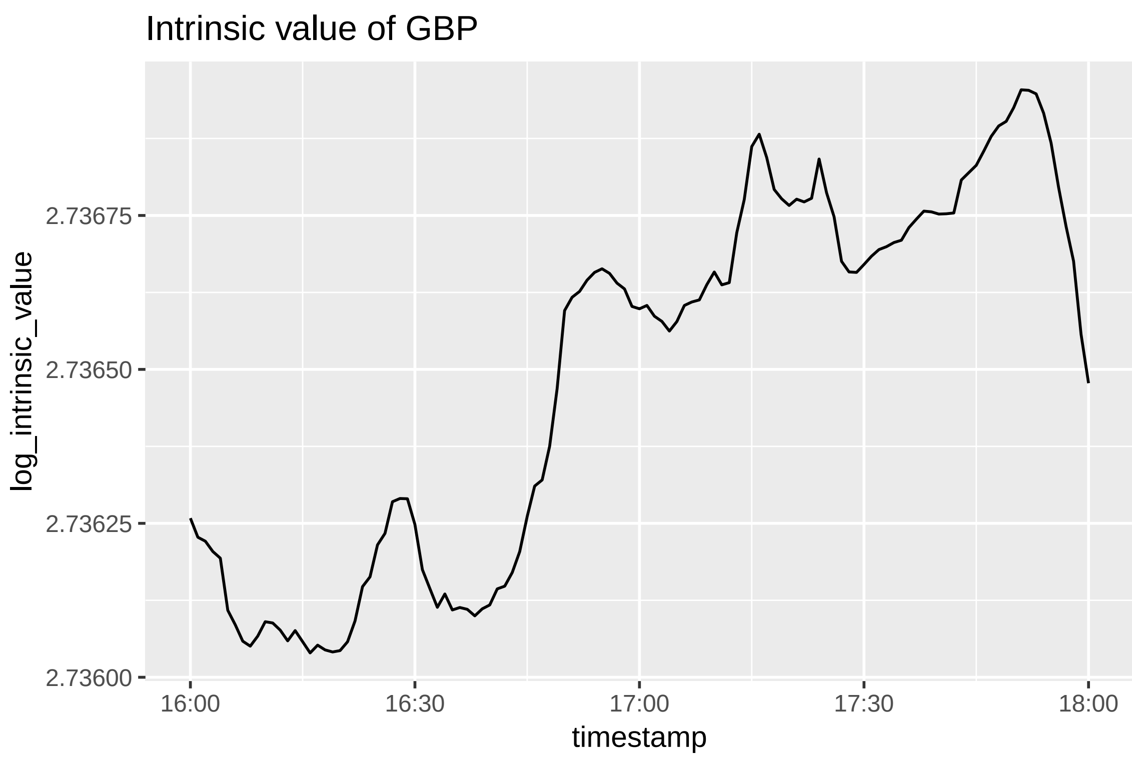
```
latent_viz <- melt(
  latent_estimates,
  id.vars = 'timestamp',
  variable.name = 'currency',
  value.name = 'log_intrinsic_value'
)
ggplot(latent_viz[currency == 'JPY' &
                  timestamp >= '2019-05-14 12:00:00' &
                    timestamp <= '2019-05-14 14:00:00']) +
  geom_line(aes(x = timestamp, y = log_intrinsic_value)) +
  ggtitle('Intrinsic value of JPY')
```




```r
ggplot(latent_viz[currency == 'USD' &
                  timestamp >= '2019-05-14 12:00:00' &
                    timestamp <= '2019-05-14 14:00:00']) +
  geom_line(aes(x = timestamp, y = log_intrinsic_value)) +
  ggtitle('Intrinsic value of USD')
```




```r
ggplot(latent_viz[currency == 'GBP' &
                   timestamp >= '2019-05-14 12:00:00' &
                     timestamp <= '2019-05-14 14:00:00']) +
  geom_line(aes(x = timestamp, y = log_intrinsic_value)) +
  ggtitle('Intrinsic value of GBP')
```


Intrinsic value of GBP
log_intrinsic_value
2.73675
2.73650
2.73625
2.73600
16:00
16:30
17:00
17:30
18:00
timestamp