Mean Returns in Time Series - Restarting after NA values - rstudio

Question Link

Has anyone encountered calculating historical mean log returns in time series datasets?

The dataset is ordered by individual security first and by time for each respective security. I am trying to form a historical mean log return, i.e. the mean log return for the security from its first appearance in the dataset to date, for each point in time for each security.

Luckily, the return time series contains NAs between returns for differing securities. My idea is to calculate a historical mean that restarts after each NA that appears.

A simple `cumsum()` probably will not do it, as the NAs will have to be dropped.

I thought about using `rollmean()`, if I only knew an efficient way to specify the 'width' parameter to the length of the vector of consecutive preceding non-NAs. The current approach I am taking, based on Count how many consecutive values are true, takes significantly too much time, given the size of the data set I am working with. For any x of the form x : [r(1) r(2) ... r(N)], where r(2) is the log return in period 2:

```
df <- data.frame(x, zcount = NA)
df[1,2] = 0 #df$x[1]=NA by construction of the data set
for(i in 2:nrow(df))
df$zcount[i] <- ifelse(!is.na(df$x[i]), df$zcount[i-1]+1, 0)
```

Any idea how to speed this up would be highly appreciated!

Answer

You will need to reshape the data.frame to apply the cumsum function over each security. Here's how:

First, I'll generate some data on 100 securities over 100 months which I think corresponds to your description of the data set

```
securities <- 100
months <- 100

time <- seq.Date(as.Date("2010/1/1"), by = "months", length.out = months)
ID <- rep(paste0("sec", 1:months), each = securities)
returns <- rnorm(securities * months, mean = 0.08, sd = 2)

df <- data.frame(time, ID, returns)

head(df)
```

```
##         time   ID    returns
## 1 2010-01-01 sec1  2.3657508
## 2 2010-02-01 sec1  1.6938412
## 3 2010-03-01 sec1  0.5450880
## 4 2010-04-01 sec1 -0.8374542
## 5 2010-05-01 sec1 -2.4396511
## 6 2010-06-01 sec1 -0.8035210
```

```
tail(df)
```

```
##             time     ID     returns
## 9995  2017-11-01 sec100  2.14161199
## 9996  2017-12-01 sec100  0.09974232
## 9997  2018-01-01 sec100  0.89294240
## 9998  2018-02-01 sec100  0.16453359
## 9999  2018-03-01 sec100 -2.66899889
## 10000 2018-04-01 sec100  0.46826481
```

Now, you must reshape your data so that each security column contains its returns, and each row represents the date.

```
library(tidyr)
df_wide <- spread(df, ID, returns)

head(df_wide[,1:6])
```

```
##         time       sec1      sec10     sec100      sec11      sec12
## 1 2010-01-01  2.3657508 -1.4901021  3.8628070  0.9110070  1.5926285
## 2 2010-02-01  1.6938412 -3.3758198 -0.2728564  0.3965130  2.7097608
## 3 2010-03-01  0.5450880  2.0977814  0.9703747 -0.9631422  1.1425272
## 4 2010-04-01 -0.8374542  2.6220169 -2.2886683 -2.2929446  0.5615811
## 5 2010-05-01 -2.4396511  0.6767944 -1.6392753 -1.0399276 -0.8568396
## 6 2010-06-01 -0.8035210  0.5167686  1.0828155  1.1376079 -1.0639955
```

Once this is done, you can use the `apply` function to `sum` every column which now represents each security. Or use the `cumsum` function. Notice the data object `df_wide[-1]`, which drops the time column. This is necessary to avoid the sum or cumsum functions throwing an error.

```
matrix_sum <- apply(df_wide[-1], 2, FUN = sum)

matrix_cumsum <- apply(df_wide[-1], 2, FUN = cumsum)
```

Now, add the `time` column back as a `data.frame` if you like:

```
df_final <- data.frame(time = df_wide[,1], matrix_cumsum)

head(df_final[,1:6])
```

```
##         time      sec1      sec10    sec100      sec11    sec12
## 1 2010-01-01 2.3657508 -1.4901021 3.8628070  0.9110070 1.592628
## 2 2010-02-01 4.0595920 -4.8659220 3.5899506  1.3075200 4.302389
## 3 2010-03-01 4.6046801 -2.7681406 4.5603254  0.3443777 5.444916
## 4 2010-04-01 3.7672259 -0.1461237 2.2716570 -1.9485668 6.006498
## 5 2010-05-01 1.3275748  0.5306707 0.6323817 -2.9884944 5.149658
## 6 2010-06-01 0.5240539  1.0474392 1.7151972 -1.8508865 4.085662
```

```
tail(df_final[,1:6])
```

```
##           time       sec1     sec10   sec100      sec11     sec12
## 95  2017-11-01  -3.956530 -17.01161 5.824611 -5.4687162 -4.656445
## 96  2017-12-01  -3.262235 -15.46555 5.924353 -5.9023424 -5.827747
## 97  2018-01-01  -8.696457 -17.00703 6.817296 -2.8361436 -6.148847
## 98  2018-02-01  -8.903971 -16.66781 6.981829 -3.7083231 -5.649179
## 99  2018-03-01 -12.097415 -11.88605 4.312830  0.5158508 -7.559351
## 100 2018-04-01 -10.019733 -10.67105 4.781095  2.7013424 -7.941380
```