Portfolio Backtesting

Daniel P. Palomar and Rui ZHOU
Hong Kong University of Science and Technology (HKUST)
2019-02-24

Contents

This vignette illustrates the usage of the package `portfolioBacktest` for automated portfolio backtesting. It can be used by a researcher/practitioner to check a set of different portfolios, as well as by a course instructor to evaluate the students in their portfolio design in a fully automated and convenient manner.

1 Installation

The package can currently be installed from GitHub:

```r
# install.packages("devtools")
devtools::install_github("dppalomar/portfolioBacktest")

# Getting help
library(portfolioBacktest)
help(package = "portfolioBacktest")
package?portfolioBacktest
?portfolioBacktest
```

2 Usage of the package

2.1 Loading data

We start by loading the package and some random sets of stock market data:

```r
library(PerformanceAnalytics)
library(portfolioBacktest)
data(dataset)
```

The dataset `dataset` is a list of data that contains the prices of random sets of stock market data from the S&P 500, over random periods of two years with a random selection of 50 stocks of each universe.

```r
length(dataset)
#> [1] 10
names(dataset[[1]])
#> [1] "open"     "high"     "low"      "close"    "volume"   "adjusted"
#> [7] "index"
str(dataset[[1]]$adjusted)
#> An 'xts' object on 2015-04-24/2017-04-24 containing:
#>   Data: num [1:504, 1:50] 22.1 22.1 22.7 22.5 22.3 ...
#>  - attr(*, "dimnames")=List of 2
#>   ..$ : NULL
#>   ..$ : chr [1:50] "MAS.Adjusted" "MGM.Adjusted" "CMI.Adjusted" "CSX.Adjusted" ...
#>   Indexed by objects of class: [Date] TZ: UTC
#>   xts Attributes:
#> List of 2
#>  $ src    : chr "yahoo"
#>  $ updated: POSIXct[1:1], format: "2018-12-29 16:24:41"

colnames(dataset[[1]]$adjusted)
#>  [1] "MAS.Adjusted"  "MGM.Adjusted"  "CMI.Adjusted"  "CSX.Adjusted"
#>  [5] "TGT.Adjusted"  "AWK.Adjusted"  "LNC.Adjusted"  "KO.Adjusted"
#>  [9] "CCI.Adjusted"  "RJF.Adjusted"  "ICE.Adjusted"  "SRE.Adjusted"
#> [13] "FOXA.Adjusted" "CERN.Adjusted" "ORLY.Adjusted" "EMR.Adjusted"
#> [17] "CME.Adjusted"  "AVB.Adjusted"  "AMT.Adjusted"  "TIF.Adjusted"
#> [21] "HAL.Adjusted"  "OMC.Adjusted"  "NTAP.Adjusted" "KORS.Adjusted"
#> [25] "AEP.Adjusted"  "A.Adjusted"    "KSS.Adjusted"  "BHGE.Adjusted"
#> [29] "BEN.Adjusted"  "HST.Adjusted"  "AMP.Adjusted"  "WY.Adjusted"
#> [33] "AGN.Adjusted"  "CPB.Adjusted"  "NWL.Adjusted"  "INTC.Adjusted"
#> [37] "XRAY.Adjusted" "VRSK.Adjusted" "MLM.Adjusted"  "CI.Adjusted"
#> [41] "PHM.Adjusted"  "MKC.Adjusted"  "OXY.Adjusted"  "GM.Adjusted"
#> [45] "CB.Adjusted"   "RHT.Adjusted"  "DOV.Adjusted"  "GLW.Adjusted"
#> [49] "FLIR.Adjusted" "GPC.Adjusted"
```

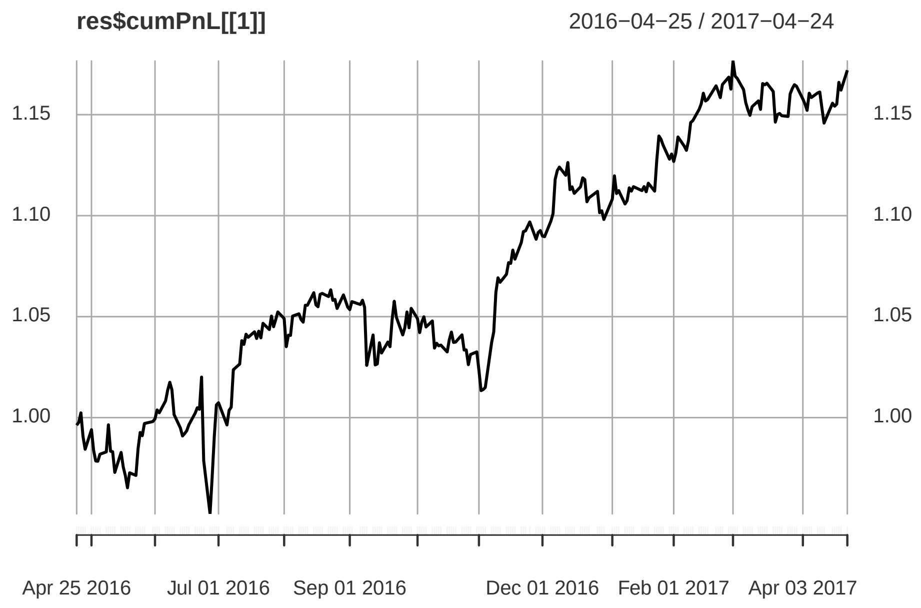
2.2 Backtesting a single portfolio

We start by defining a simple portfolio design in the form of a function that takes as input the prices and outputs the portfolio vector `w`:

```r
uniform_portfolio_fun <- function(data) {
  N <- ncol(data$adjusted)
  w <- rep(1/N, N)  # satisfies the constraints w>=0 amd sum(w)=1
  return(w)
}
```

Now we are ready to use the function `backtestPortfolio()` that will execute and evaluate the portfolio design function on a rolling-window basis, and the result can be easily handled with privided function `backtestSelector()`

```r
bt <- portfolioBacktest(uniform_portfolio_fun, dataset[1], shortselling = TRUE)
res <- backtestSelector(bt, portfolio_index = 1)
names(res)
#> [1] "performance"   "error"         "error_message" "cpu_time"
#> [5] "portfolio"     "return"        "cumPnL"
plot(res$cumPnL[[1]])
```



```
res$performance
#>       Sharpe ratio max drawdown annual return annual volatility
#> data1     1.467109   0.06642285     0.1720106         0.1172446
#>       Sterling ratio Omega ratio  ROT bps
#> data1        2.58963    1.280153 2871.655
```

Let's try with a slightly more sophisticated portfolio design, like the global minimum variance portfolio (GMVP):

```
GMVP_portfolio_fun <- function(data) {
  X <- diff(log(data$adjusted))[-1]  # compute log returns
  Sigma <- cov(X)  # compute SCM
  # design GMVP
  w <- solve(Sigma, rep(1, nrow(Sigma)))
  w <- w/sum(abs(w))  # it may not satisfy w>=0
  return(w)
}
bt <- portfolioBacktest(GMVP_portfolio_fun, dataset[1])
res <- backtestSelector(bt, portfolio_index = 1)
res$error
#> data1
#>  TRUE
res$error_message
#> $data1
#> [1] "No-shortselling constraint not satisfied."
```

Indeed, the GMVP does not satisfy the no-shortselling constraint. We can repeat the backtesting indicating that shortselling is allowed:

```
bt <- portfolioBacktest(GMVP_portfolio_fun, dataset[1], shortselling = TRUE)
res <- backtestSelector(bt, portfolio_index = 1)
res$error
#> data1
#> FALSE
res$error_message
#> $data1
#> [1] NA
res$cpu_time
#>       data1
#> 0.003076923
res$performance
#>       Sharpe ratio max drawdown annual return annual volatility
#> data1    0.4321617   0.02601129     0.0183104        0.04236933
#>       Sterling ratio Omega ratio  ROT bps
#> data1      0.7039406    1.075537 36.41359
```

We could be more sophisticated and design a Markowitz mean-variance portfolio satisfying the no-shortselling constraint:

```
Markowitz_portfolio_fun <- function(data) {
  library(CVXR) #install.packages("CVXR")
  X <- as.matrix(diff(log(data$adjusted))[-1])  # compute log returns
  mu <- colMeans(X)  # compute mean vector
  Sigma <- cov(X)  # compute the SCM
  # design mean-variance portfolio
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - 0.5*quad_form(w, Sigma)),
                  constraints = list(w >= 0, sum(w) == 1))
  result <- solve(prob)
  return(as.vector(result$getValue(w)))
}
```

We can now backtest it:

```
bt <- portfolioBacktest(Markowitz_portfolio_fun, dataset[1])
res <- backtestSelector(bt, portfolio_index = 1)
res$error
#> [1] FALSE
res$error_message
#> [[1]]
#> [1] NA
res$cpu_time
#> [1] 0.2315385
res$performance
#>      Sharpe ratio max drawdown annual return annual volatility
#> [1,]   -0.1049306    0.2096863   -0.02058176        0.1961464
#>      Sterling ratio Omega ratio  ROT bps
#> [1,]    -0.09815498   0.9985759 8.583926
```

Instead of backtesting a portfolio on a single `xts` dataset, it is more meaningful to backtest it on multiple datasets. This can be easily done simply by passing a list of `xts` objects:

```
mul_data_bt <- portfolioBacktest(Markowitz_portfolio_fun, dataset[1:5])
mul_data_res <- backtestSelector(mul_data_bt, portfolio_index = 1)
names(mul_data_res)
#> [1] "performance"   "error"         "error_message" "cpu_time"
#> [5] "portfolio"     "return"        "cumPnL"
mul_data_res$cpu_time
#>     data1     data2     data3     data4     data5
#> 0.2384615 0.2423077 0.2184615 0.2376923 0.2284615
mul_data_res$performance
#>       Sharpe ratio max drawdown annual return annual volatility
#> data1   -0.1049306    0.2096863   -0.02058176         0.1961464
#> data2    1.0281566    0.2334022    0.35935571         0.3495146
#> data3    1.1818257    0.1685567    0.29888243         0.2528989
#> data4    0.1904625    0.1512879    0.05060499         0.2656953
#> data5   -0.2957469    0.6467921   -0.21158243         0.7154171
#>       Sterling ratio Omega ratio    ROT bps
#> data1    -0.09815498   0.9985759   8.583926
#> data2     1.53964118   1.2101015 556.025649
#> data3     1.77318680   1.2211991 281.163758
#> data4     0.33449453   1.0537061  72.906528
#> data5    -0.32712589   1.0147774  40.755538
```

The results from backtesting on multiple datasets can be further summarized by function backtestSummary() based on user customized summary functions. For example, we can summarize results using median()

```
res_summary <- backtestSummary(mul_data_bt, summary_fun = median)
names(res_summary)
#> [1] "performance_summary" "failure_rate"        "cpu_time_summary"
#> [4] "error_message"
res_summary$performance_summary
#>                          fun1
#> Sharpe ratio       0.19046249
#> max drawdown       0.20968630
#> annual return      0.05060499
#> annual volatility  0.26569529
#> Sterling ratio     0.33449453
#> Omega ratio        1.05370607
#> ROT bps           72.90652820
```

2.3 Backtesting multiple portfolios

Backtesting multiple portfolios is equally simple. It suffices to pass a list of functions to the backtesting function portfolioBacktest():

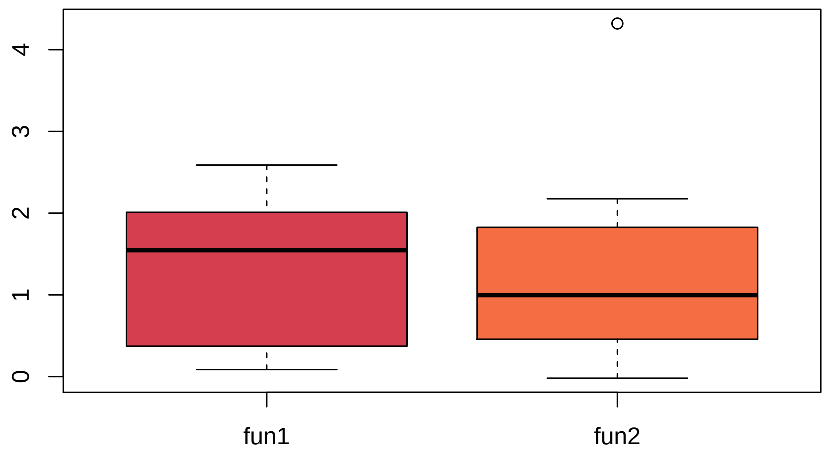
```
bt <- portfolioBacktest(list(uniform_portfolio_fun, GMVP_portfolio_fun),
                        dataset, shortselling = TRUE)
names(bt)
#> [1] "fun1" "fun2"
res_summary <- backtestSummary(bt, summary_fun = median)
res_summary$performance_summary
#>                           fun1         fun2
#> Sharpe ratio      1.546805e+00   0.99702912
#> max drawdown      8.946477e-02   0.03914431
#> annual return     1.651707e-01   0.04777475
#> annual volatility 1.215642e-01   0.04376135
#> Sterling ratio    2.219609e+00   0.97984607
#> Omega ratio       1.296275e+00   1.19370264
#> ROT bps           2.715873e+03 115.63512029
```

We also provide another function `backtestTable()` to extract the results into matrix form.

```
res_table <- backtestTable(bt)
names(res_table)
#>  [1] "Sharpe ratio"      "max drawdown"      "annual return"
#>  [4] "annual volatility" "Sterling ratio"    "Omega ratio"
#>  [7] "ROT bps"           "error"             "cpu_time"
#> [10] "error_message"
res_table$`Sharpe ratio`
#>              fun1        fun2
#> data1  1.46710897  0.43216171
#> data2  0.37214279  0.45708668
#> data3  2.45729621  1.41115502
#> data4  1.25775368 -0.01951969
#> data5  0.21333697  0.72256088
#> data6  1.84467935  2.17578547
#> data7  2.58831048  4.32024112
#> data8  0.08604527  1.13401883
#> data9  1.62650187  0.86003940
#> data10 2.01062146  1.82664160
```

The results from `backtestTable()` works well with the boxplot

```
library(RColorBrewer)
palette <- brewer.pal(9, "Spectral")
boxplot(res_table$`Sharpe ratio`, col=palette)
```



Note that the function names are given as `fun1` and `fun2` because only the function bodies are passed to `portfolioBacktest()`. To make the results more recognizable, we can manually pass the function names as follows:

```
bt <- portfolioBacktest(list('my_uniform' = uniform_portfolio_fun, 'my_GMVP' = GMVP_portfolio_fun),
                        dataset[1:5], shortselling = TRUE)
names(bt)
#> [1] "my_uniform" "my_GMVP"
res_summary <- backtestSummary(bt, summary_fun = median)
res_summary$performance_summary
#>                      my_uniform     my_GMVP
#> Sharpe ratio          1.2577537  0.45708668
#> max drawdown          0.1085410  0.04323931
#> annual return         0.1583309  0.02480846
#> annual volatility     0.1258838  0.05427518
#> Sterling ratio        1.4587189  0.67562935
#> Omega ratio           1.2505383  1.08372516
#> ROT bps            2368.6803778 60.43757407
```

2.4 Incoporate benchmarks

When perform the backtest of our designed portfolio functions, we may want to incorporate some benchmarks. Now the package suppport two benchmarks, which are `uniform portfolio` and `index` of the certain market. We can easily do that in any case by passing corresponding value to argument `benchmark`.

```
bt <- portfolioBacktest(list('my_uniform' = uniform_portfolio_fun, 'my_GMVP' = GMVP_portfolio_fun),
                        dataset[1:5], shortselling = TRUE, benchmark = c('uniform', 'index'))
names(bt)
#> [1] "my_uniform" "my_GMVP"    "uniform"    "index"
res_summary <- backtestSummary(bt, summary_fun = median)
res_summary$performance_summary
#>                      my_uniform     my_GMVP      uniform     index
#> Sharpe ratio          1.2577537  0.45708668    1.2577537 0.8683934
#> max drawdown          0.1085410  0.04323931    0.1085410 0.1055529
#> annual return         0.1583309  0.02480846    0.1583309 0.1083058
#> annual volatility     0.1258838  0.05427518    0.1258838 0.1247198
#> Sterling ratio        1.4587189  0.67562935    1.4587189 1.0260806
#> Omega ratio           1.2505383  1.08372516    1.2505383 1.1880570
#> ROT bps            2368.6803778 60.43757407 2368.6803778       Inf
```

2.5 Progress bar

In order to monitor the backtest progress, we add the progress bar display in all cases. Users can turn on the progress bar by setting argument `show_progress_bar` be `TRUE`.

```
mul_data_bt <- portfolioBacktest(uniform_portfolio_fun, dataset[1:5], show_progress_bar = TRUE)
#> Backtesting function fun1            (1/1)
#>
  |
  |=============                                                        |  20%
  |
  |==========================                                           |  40%
  |
  |=======================================                          |  60%
  |
  |====================================================             |  80%
  |
  |=================================================================| 100%
bt <- portfolioBacktest(list('my_uniform' = uniform_portfolio_fun, 'my_GMVP' = GMVP_portfolio_fun),
                        dataset[1:5], shortselling = TRUE, benchmark = c('uniform', 'index'),
                        show_progress_bar = TRUE)
#> Backtesting function my_uniform      (1/2)
#>
  |
  |=============                                                    |  20%
  |
  |==========================                                       |  40%
  |
  |=======================================                          |  60%
  |
  |====================================================             |  80%
  |
  |=================================================================| 100%Backtesting function my_GMVP
#>
  |
  |=============                                                    |  20%
  |
  |==========================                                       |  40%
  |
  |=======================================                          |  60%
  |
  |====================================================             |  80%
  |
  |=================================================================| 100%Evaluating benchmark-uniform
#>
  |
  |=============                                                    |  20%
  |
  |==========================                                       |  40%
  |
  |=======================================                          |  60%
  |
  |====================================================             |  80%
  |
  |=================================================================| 100%Evaluating benchmark-index
#>
  |
  |=============                                                    |  20%
  |
  |==========================                                       |  40%
  |
  |=======================================                          |  60%
  |
  |====================================================             |  80%
  |
  |=================================================================| 100%
```

2.6 Parallel mode

The backtest incurs very heavy computation load when numbers of portfolio functions or dataset go large. Therefore, we add support for parallel mode in this package. Users can choose if they want to parallel evaluate different portfolio functions or in a more fine-grained way, evaluating multiple datasets parallel for each function.

```
# parallel = 2 for functions
system.time(bt_nopar <- portfolioBacktest(list(Markowitz_portfolio_fun, Markowitz_portfolio_fun), datase
#>    user  system elapsed
#>   63.42    0.07   63.60
system.time(bt_parfuns <- portfolioBacktest(list(Markowitz_portfolio_fun, Markowitz_portfolio_fun), data
                                            par_portfolio = 2))
#>    user  system elapsed
#>    0.01    0.11   36.18

# parallel = 5 for datasets
system.time(bt_nopar <- portfolioBacktest(Markowitz_portfolio_fun, dataset))
#>    user  system elapsed
#>   31.61    0.00   31.67
system.time(bt_pardata <- portfolioBacktest(Markowitz_portfolio_fun, dataset, par_dataset = 5))
#>    user  system elapsed
#>    0.05    0.04   12.42
```

It is obvious that the evaluation time for backtesting has been significantly reduced. Note that the parallel evaluation time can not be exactly equal to the original time divided by parallel cores because starting new R sessions also takes extra time. For some technical reasons, the loaded packages information can not be automatically passed to parallel R sessions. Therefore we highly recommend users to cover the `library(XXX)` inside function body like

```
portfolio_fun <- function(x) {
  library(required_package_name)
  # here whatever code
}
```

2.7 Trace where execution error happens

The execution error might happen without any clue. While our function is robustly designed to not be stopped by any error from the user defined function. To help user trace where the execution error happens, we also report the call stack when a execution error happens. Such information is given as the attribution `error_stack` of returned `error_message`. For example, let's define a portfolio function which will throw a error:

```
library(CVXR)
sub_function2 <- function(x) {
  "a" + x # an error will happen here
}

sub_function1 <- function(x) {
  return(sub_function2(x))
}

wrong_portfolio_fun <- function(data) {
  N <- ncol(data$adjusted)
  uni_port <- rep(1/N, N)
  return(sub_function1(uni_port))
}
```

Then, we pass the above portfolio function into `portfolioBacktest()` and show how to check the error trace:

```
bt <- portfolioBacktest(wrong_portfolio_fun, dataset[1:5])
res <- backtestSelector(bt, portfolio_index = 1)

# information of 1st error
error1 <- res$error_message[[1]]
str(error1)
#>  chr "non-numeric argument to binary operator"
#>  - attr(*, "error_stack")=List of 2
#>   ..$ at   : chr "\"a\" + x"
#>   ..$ stack: chr "sub_function1(uni_port)\nsub_function2(x)"

# the exact location of error happening
cat(attr(error1, "error_stack")$at)
#> "a" + x

# the call stack of error happening
cat(attr(error1, "error_stack")$stack)
#> sub_function1(uni_port)
#> sub_function2(x)
```

3 Usage for grading students in a course

If an instructor wants to evaluate the students of a course in their portfolio design, it can also be done very easily. It suffices to ask each student to submit a .R script (necessary to be named uniquely like `STUDENTNUMBER-XXXX.R`) containing the portfolio function called exactly `portfolio_fun()` as well as any other auxiliary functions that it may require (needless to say that the required packages should be loaded in that script with `library()`). Then the instructor can put all those files in a folder and evaluate all of them at once.

```
bt_all_students <- portfolioBacktest(folder_path = "folder_path", dataset =  dataset[1:3])
res_all_students <- backtestSummary(bt_all_students, summary_fun = median)
res_all_students$performance_summary
#>                      0001-GMVP 0002-Markowitz     0003-max
#> Sharpe ratio        2.03200998      0.6102278   2.49279097
#> max drawdown        0.08183484      0.1353010   0.04821096
#> annual return       0.27119430      0.1369909   0.27568481
#> annual volatility   0.13346111      0.1710301   0.11059283
#> Sterling ratio      3.31392212      0.9723497   5.71830214
#> Omega ratio         1.40105559      1.1297211   1.48661227
#> ROT bps           373.18306365    315.9623959 346.58565659
res_all_students$cpu_time_average
#>      0001-GMVP 0002-Markowitz       0003-max
#>     0.06153846     0.23923077     0.06025641
res_all_students$failure_rate
#>      0001-GMVP 0002-Markowitz       0003-max
#>              0              0              0
```

Now we can rank the different portfolios/students based on a weighted combination of the rank percentiles (termed scores) of the performance measures:

```r
leaderboard <- portfolioLeaderboard(bt_all_students, weights = list('Sharpe ratio' = 7, 'max drawdown' =

# show leaderboard
library(gridExtra)
grid.table(leaderboard$leaderboard_scores)
```

	Sharpe ratio score	max drawdown score	annual return score	ROT bps score	final score
0003−max	100	100	100	50	95
0001−GMVP	50	50	50	100	55
0002−Markowitz	0	0	0	0	0

3.1 Example of a script file to be submitted by a student

Consider the student with id number 666. Then the script file should be named `666-XXX.R` and should contain the portfolio function called exactly `portfolio_fun()` as well as any other auxiliary functions that it may require (needless to say that the required packages should be loaded in that script with `library()`):

```r
library(CVXR)

auxiliary_function <- function(x) {
  # here whatever code
}

portfolio_fun <- function(prices) {
  X <- as.matrix(diff(log(data$adjusted))[-1])  # compute log returns
  mu <- colMeans(X)  # compute mean vector
  Sigma <- cov(X)  # compute the SCM
  # design mean-variance portfolio
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - 0.5*quad_form(w, Sigma)),
                  constraints = list(w >= 0, sum(w) == 1))
  result <- solve(prob)
  return(as.vector(result$getValue(w)))
}
```

4 Appendix

4.1 Performance criteria

The definition of performance criteria used in this package is listed as below

- **expetced return:** the annualized return
- **volatility:** the annualized standard deviation of returns
- **max drawdown:** the maximum loss from a peak to a trough of a portfolio, see also here

- **Sharpe ratio:** annualized Sharpe ratio, the ratio between `annualized return` and `annualized standard deviation`
- **Sterling ratio:** the return over average drawdown, see here for complete definition. In the package, we use
$$\text{Sterling ratio} = \frac{\text{annualized return}}{\text{max drawdown}}$$
- **Omega ratio:** the probability weighted ratio of gains over losses for some threshold return target, see here for complete definition. The ratio is calculated as:
$$\Omega(r) = \frac{\int_r^\infty (1 - F(x))dx}{\int_{-\infty}^r F(x)dx}$$
In the package, we use $\Omega(0)$, which is also known as Gain-Loss-Ratio.
- **Return over Turnover (ROT):** the sum of cummulative return over the sum of turnover.