Portfolio Optimization with R

MAFS6010R-
Portfolio Optimization with R MSc in Financial Mathematics Fall 2018-19,
HKUST,
Hong Kong

Prof. Daniel P. Palomar

Hong Kong University of Science and Technology (HKUST)

This R session will introduce the basics on portfolio design and evaluation with R. (For the evaluation the package **portfolioBacktest** (https://github.com/dppalomar/portfolioBacktest) is recommended.)

(Useful R links: Cookbook R (http://www.cookbook-r.com/), Quick-R (https://www.statmethods.net/), R documentation (https://www.rdocumentation.org/), CRAN (https://cran.r-project.org/), METACRAN (https://r-pkg.org/).)

Static portfolios

In this section, we will divide the stock market data into a training part (for the estimation of the expected return $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$, and subsequent portfolio design) and a test part (for the out-of-sample performance evaluation). A more sophisticated approach (considered in the next section) is based on a rolling window where the portfolio is updated with some frequency rather than kept fixed.

We start by loading some stock market data and dividing it into a training set and test set:

```
library(xts)
library(quantmod)
library(PerformanceAnalytics)

# set begin-end date and stock namelist
begin_date <- "2013-01-01"
end_date <- "2017-08-31"
stock_namelist <- c("AAPL", "AMD", "ADI",  "ABBV", "AET", "A",  "APD", "AA","CF")

# download data from YahooFinance
prices <- xts()
for (stock_index in 1:length(stock_namelist))
  prices <- cbind(prices, Ad(getSymbols(stock_namelist[stock_index],
                                        from = begin_date, to = end_date, auto.assign
= FALSE)))
colnames(prices) <- stock_namelist
indexClass(prices) <- "Date"
str(prices)
#> An 'xts' object on 2013-01-02/2017-08-30 containing:
#>   Data: num [1:1175, 1:9] 70.1 69.2 67.3 66.9 67.1 ...
#>  - attr(*, "dimnames")=List of 2
#>   ..$ : NULL
#>   ..$ : chr [1:9] "AAPL" "AMD" "ADI" "ABBV" ...
#>   Indexed by objects of class: [Date] TZ: UTC
#>   xts Attributes:
#>  NULL
head(prices)
#>                AAPL  AMD      ADI     ABBV      AET        A      APD
#> 2013-01-02 70.12886 2.53 37.90372 28.62027 43.32114 28.16434 67.74826
#> 2013-01-03 69.24367 2.49 37.29209 28.38393 42.40319 28.26521 67.51154
#> 2013-01-04 67.31493 2.59 36.62878 28.02536 42.59989 28.82338 68.41895
#> 2013-01-07 66.91895 2.67 36.74078 28.08241 43.23684 28.61492 68.35583
#> 2013-01-08 67.09905 2.67 36.36173 27.47122 41.75045 28.38627 68.48207
#> 2013-01-09 66.05039 2.63 36.26697 27.62606 42.31490 29.15291 69.40527
#>                  AA       CF
#> 2013-01-02 20.62187 29.72212
#> 2013-01-03 20.80537 29.58158
#> 2013-01-04 21.24121 30.24417
#> 2013-01-07 20.87419 30.13087
#> 2013-01-08 20.87419 29.68914
#> 2013-01-09 20.82831 30.72749
tail(prices)
#>                AAPL   AMD      ADI     ABBV      AET        A      APD
#> 2017-08-23 157.5971 12.48 77.08375 69.09010 154.8036 62.21310 140.5500
#> 2017-08-24 156.8977 12.50 77.27874 69.66007 154.3686 62.12389 140.6178
#> 2017-08-25 157.4789 12.43 76.98628 70.01749 154.3192 62.34195 141.3730
#> 2017-08-28 159.0649 12.23 77.44447 70.82896 154.7443 62.89698 141.1504
#> 2017-08-29 160.4835 12.15 77.55172 71.37959 154.6356 62.92671 140.7534
#> 2017-08-30 160.9169 12.67 81.61696 71.40857 155.1101 63.33307 140.8889
#>               AA       CF
#> 2017-08-23 41.06 28.08939
#> 2017-08-24 41.34 28.18649
#> 2017-08-25 41.21 28.13794
#> 2017-08-28 42.17 28.18649
#> 2017-08-29 43.00 27.99230
#> 2017-08-30 43.09 28.09910

# compute log-returns and linear returns
X_log <- diff(log(prices))[-1]
```

```
X_lin <- (prices/lag(prices) - 1)[-1]

# or alternatively...
X_log <- CalculateReturns(prices, "log")[-1]
X_lin <- CalculateReturns(prices)[-1]

N <- ncol(X_log)  # number of stocks
T <- nrow(X_log)  # number of days

# split data into training and test data
T_trn <- round(0.7*T)  # 70% of data
X_log_trn <- X_log[1:T_trn, ]
X_log_tst <- X_log[(T_trn+1):T, ]
X_lin_trn <- X_lin[1:T_trn, ]
X_lin_tst <- X_lin[(T_trn+1):T, ]
```

Modeling linear vs log returns

Now we are ready to obtain the sample estimates from the returns $\mathbf{x}_t$ (i.e., sample means and sample covariance matrix) as

$$\hat{\boldsymbol{\mu}} = \frac{1}{T}\sum_{t=1}^{T}\mathbf{x}_t$$
$$\hat{\boldsymbol{\Sigma}} = \frac{1}{T-1}\sum_{t=1}^{T}(\mathbf{x}_t - \hat{\boldsymbol{\mu}})(\mathbf{x}_t - \hat{\boldsymbol{\mu}})^T$$

However, it is not totally clear whether we should use linear returns or log returns to estimate $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$. Clearly, for the portfolio design we need the expected return $\boldsymbol{\mu}$ and covariance matrix $\boldsymbol{\Sigma}$ of the **linear returns**. There are three different philosophies in the estimation procedure that come to mind:

1. estimate them directly from the linear returns (even though linear returns are not supposed to be easily modeled): $\hat{\boldsymbol{\mu}} = \hat{\boldsymbol{\mu}}^{\text{lin}}$ and $\hat{\boldsymbol{\Sigma}} = \hat{\boldsymbol{\Sigma}}^{\text{lin}}$
2. estimate them from the log returns (and ignore the approximation error): $\hat{\boldsymbol{\mu}} = \hat{\boldsymbol{\mu}}^{\text{log}}$ and $\hat{\boldsymbol{\Sigma}} = \hat{\boldsymbol{\Sigma}}^{\text{log}}$
3. estimate them from the log returns but properly transforming them to linear:

$$\hat{\boldsymbol{\mu}} = \exp\left(\hat{\boldsymbol{\mu}}^{\text{log}} + \frac{1}{2}\text{diag}(\hat{\boldsymbol{\Sigma}}^{\text{log}})\right) - 1$$
$$\hat{\Sigma}_{ij} = \exp\left(\hat{\mu}_i^{\text{log}} + \hat{\mu}_j^{\text{log}} + \frac{1}{2}(\hat{\Sigma}_{ii}^{\text{log}} + \hat{\Sigma}_{jj}^{\text{log}})\right) \times (\exp(\hat{\Sigma}_{ij}^{\text{log}}) - 1)$$

In the following, we will explore these three different ways for different rebalancing frequencies (i.e., how often we update the portfolio held).

For the comparison we will use the simplest Global Minimum Variance Portfolio (GMVP):

$$\mathbf{w}_{\text{GMVP}} = \frac{1}{\mathbf{1}^T\boldsymbol{\Sigma}^{-1}\mathbf{1}}\boldsymbol{\Sigma}^{-1}\mathbf{1}$$

Daily rebalancing

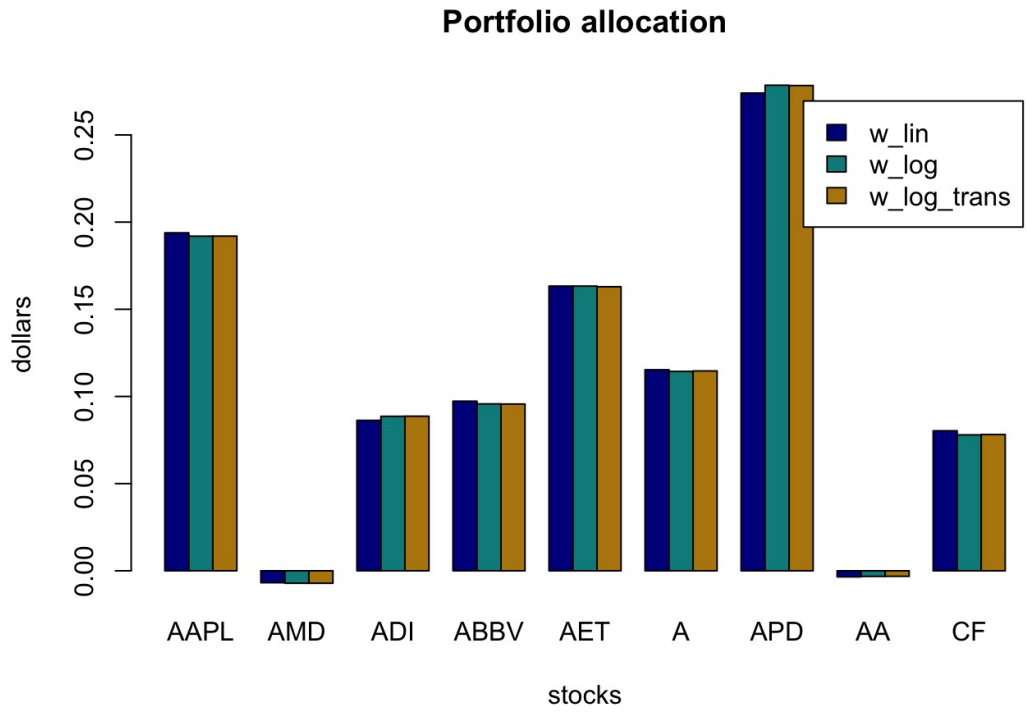
We will start with a daily rebalancing since we already have the daily returns readily available.

Let's estimate the three versions of $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ from the training set:

```
# Method 1: directly from linear returns
mu_lin <- colMeans(X_lin_trn)
X_ <- X_lin_trn - matrix(mu_lin, T_trn, N, byrow = TRUE)  #remove mean
Sigma_lin <- 1/(T_trn-1) * t(X_) %*% X_
# or more conveniently:
Sigma_lin_ <- cov(X_lin_trn)
norm(Sigma_lin - Sigma_lin_, "F")  # sanity check
#> [1] 7.872724e-19

# Method 2: directly from log returns
mu_log <- colMeans(X_log_trn)
Sigma_log <- cov(X_log_trn)

# Method 3: from log returns plus transformation
momentsReturnLog2Lin <- function(mu, Sigma) {
  K <- ncol(Sigma)
  mu_ <- exp(mu + 0.5*diag(Sigma)) - 1
  Sigma_ <- matrix(NA, nrow=K, ncol=K)
  for(ii in 1:K)
    for(jj in 1:K)
      Sigma_[ii,jj] <- exp( mu[ii] + mu[jj] + 0.5*(Sigma[ii,ii]+Sigma[jj,jj]) ) * (exp
(Sigma[ii,jj])-1)
  return( list(mu=mu_, Sigma=Sigma_) )
}

tmp <- momentsReturnLog2Lin(mu_log, Sigma_log)
mu_log_trans <- tmp$mu
Sigma_log_trans <- tmp$Sigma
```

Now let's compute the three corresponding GMV portfolios:

```
# create function for GMVP
portolioGMVP <- function(Sigma) {
  ones <- rep(1, nrow(Sigma))
  Sigma_inv_1 <- solve(Sigma, ones)  #same as: inv(Sigma) %*% ones
  w <- (1/as.numeric(ones %*% Sigma_inv_1)) * Sigma_inv_1
  return(w)
}

# compute the three versions of GMVP
w_lin <- portolioGMVP(Sigma_lin)
w_log <- portolioGMVP(Sigma_log)
w_log_trans <- portolioGMVP(Sigma_log_trans)
w_all <- cbind(w_lin, w_log, w_log_trans)
w_all
#>             w_lin        w_log  w_log_trans
#> AAPL  0.193860408  0.191932282  0.191965342
#> AMD  -0.006834041 -0.007147892 -0.007155189
#> ADI   0.086271848  0.088597918  0.088664947
#> ABBV  0.097268875  0.095721688  0.095626908
#> AET   0.163300168  0.163297212  0.162937913
#> A     0.115303919  0.114367209  0.114630099
#> APD   0.274031252  0.278514693  0.278330881
#> AA   -0.003493712 -0.003235348 -0.003177187
#> CF    0.080291282  0.077952238  0.078176286

# plot to compare the allocations
barplot(t(w_all), col = c("darkblue","darkcyan", "darkgoldenrod"), legend = colnames(w_
all),
        main = "Portfolio allocation", xlab = "stocks", ylab = "dollars", beside = TRU
E)
```



Finally, we can evaluate them (no need to use the test sample for this):

```
# compute returns of the three portfolios
ret_all_trn <- xts(X_lin_trn %*% w_all, index(X_lin_trn))
# compare them
StdDev.annualized(ret_all_trn)
#>                                    w_lin    w_log w_log_trans
#> Annualized Standard Deviation 0.1616268 0.161631   0.1616307
```

Clearly there is no apparent difference. This is not unexpected since the daily returns of the stocks are roughly within the interval [-0.04, 0.04] and the approximation $r_t = \log(1 + R_t) \approx R_t$ is very accurate:

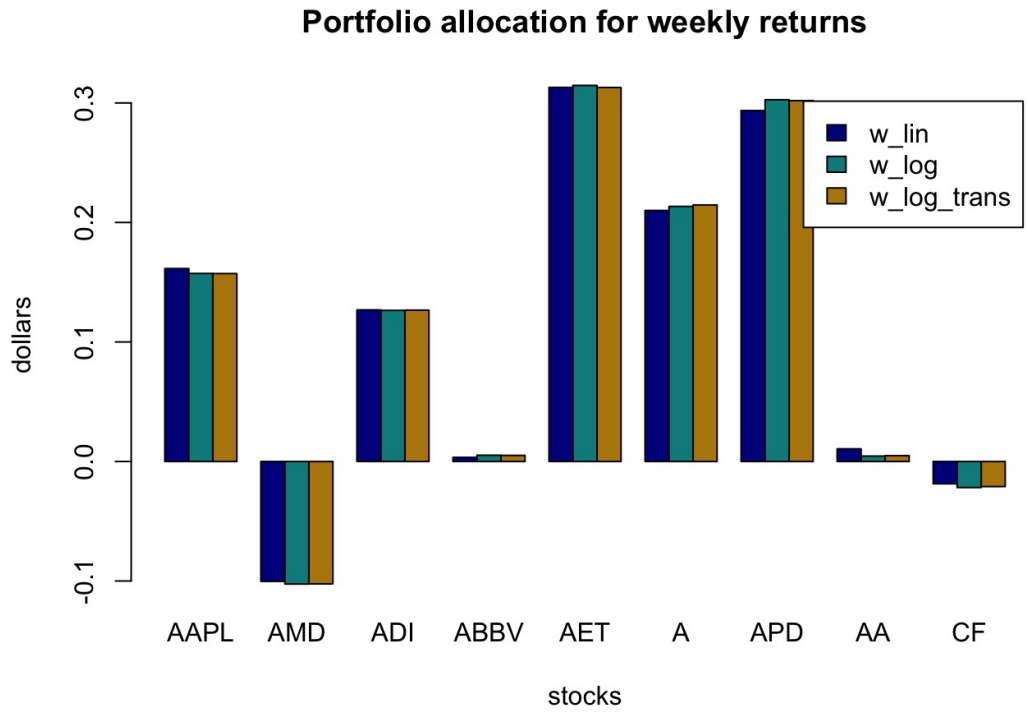
```
colMeans(X_lin)
#>         AAPL          AMD          ADI         ABBV          AET
#> 0.0008248217 0.0020802857 0.0007670934 0.0009108322 0.0011945419
#>            A          APD           AA           CF
#> 0.0007857280 0.0006985659 0.0008955944 0.0002155035
apply(X_lin, 2, sd)
#>       AAPL        AMD        ADI       ABBV        AET          A
#> 0.01527147 0.03832483 0.01510170 0.01621806 0.01466759 0.01379918
#>        APD         AA         CF
#> 0.01225525 0.02321209 0.02296528
```

The question is whether this approximation is still that accurate for weekly or monthly rebalancing.

Weekly rebalancing

Let's repeat the previous analysis but using from the start weekly returns:

```
periodicity(prices)
#> Daily periodicity from 2013-01-02 to 2017-08-30

# change periodicity of prices
prices_weekly <- xts()
for (i in 1:ncol(prices))
  prices_weekly <- cbind(prices_weekly, Cl(to.weekly(prices[, i])))
colnames(prices_weekly) <- colnames(prices)
indexClass(prices_weekly) <- "Date"
periodicity(prices_weekly)
#> Weekly periodicity from 2013-01-04 to 2017-08-30
head(prices_weekly)
#>                AAPL  AMD      ADI     ABBV      AET        A      APD
#> 2013-01-04 67.31493 2.59 36.62878 28.02536 42.59989 28.82338 68.41895
#> 2013-01-11 66.45911 2.67 36.40480 27.91370 42.83232 29.20671 69.72878
#> 2013-01-18 63.86615 2.46 37.09397 30.77517 44.14936 29.69091 69.98128
#> 2013-01-25 56.18689 2.85 37.73143 31.00606 46.81169 30.45757 69.72089
#> 2013-02-01 57.94192 2.60 38.48089 30.65972 45.60753 30.45757 70.01285
#> 2013-02-08 61.02388 2.59 39.36818 29.89281 47.60191 30.31634 69.71300
#>                  AA       CF
#> 2013-01-04 21.24121 30.24417
#> 2013-01-11 20.50717 31.28969
#> 2013-01-18 20.64481 31.41733
#> 2013-01-25 20.71362 33.02649
#> 2013-02-01 20.64481 32.98060
#> 2013-02-08 20.57622 32.43561

# recompute returns
X_weekly_log <- CalculateReturns(prices_weekly, "log")[-1]
X_weekly_lin <- CalculateReturns(prices_weekly)[-1]
T_weekly <- nrow(X_weekly_log)  # number of weeks

# split data into training and set data
T_weekly_trn <- round(0.7*T_weekly)  # 70% of data
X_weekly_log_trn <- X_weekly_log[1:T_weekly_trn, ]
X_weekly_log_tst <- X_weekly_log[(T_weekly_trn+1):T_weekly, ]
X_weekly_lin_trn <- X_weekly_lin[1:T_weekly_trn, ]
X_weekly_lin_tst <- X_weekly_lin[(T_weekly_trn+1):T_weekly, ]

# estimate mu and Sigma
mu_weekly_lin <- colMeans(X_weekly_lin_trn)
Sigma_weekly_lin <- cov(X_weekly_lin_trn)

mu_weekly_log <- colMeans(X_weekly_log_trn)
Sigma_weekly_log <- cov(X_weekly_log_trn)

tmp <- momentsReturnLog2Lin(mu_weekly_log, Sigma_weekly_log)
mu_weekly_log_trans <- tmp$mu
Sigma_weekly_log_trans <- tmp$Sigma

# compute GMVPs
w_weekly_lin <- portolioGMVP(Sigma_weekly_lin)
w_weekly_log <- portolioGMVP(Sigma_weekly_log)
w_weekly_log_trans <- portolioGMVP(Sigma_weekly_log_trans)
w_weekly_all <- cbind(w_weekly_lin, w_weekly_log, w_weekly_log_trans)
barplot(t(w_weekly_all), col = c("darkblue","darkcyan", "darkgoldenrod"), legend = coln
ames(w_all),
        main = "Portfolio allocation for weekly returns",
        xlab = "stocks", ylab = "dollars", beside = TRUE)
```

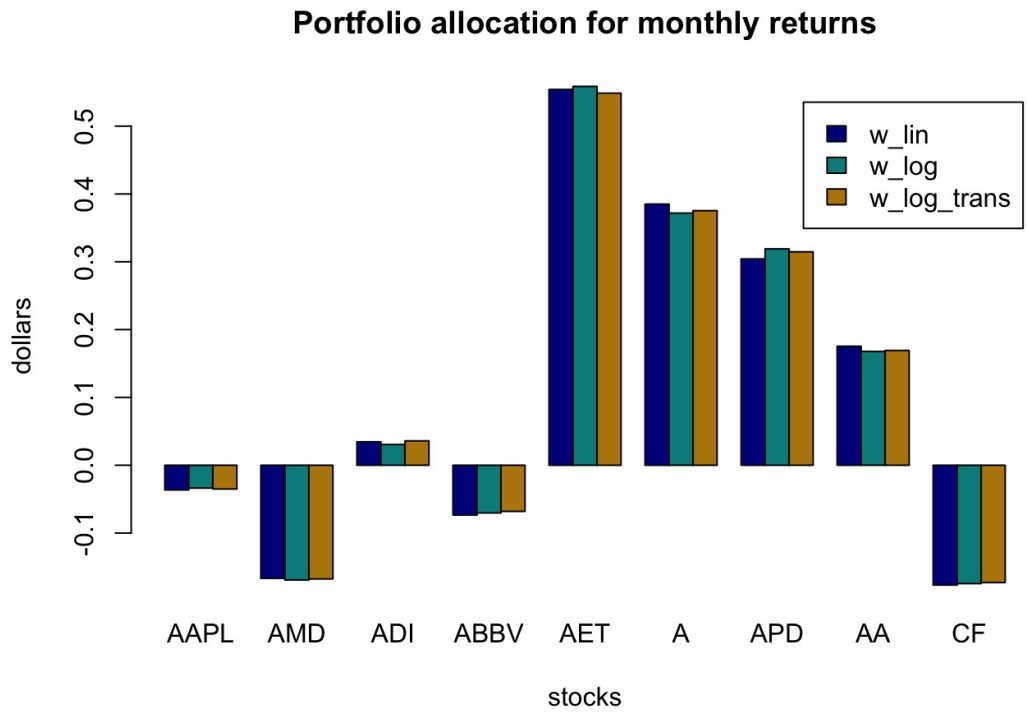



```
# compute weekly returns
ret_weekly_all_trn <- xts(X_weekly_lin_trn %*% w_weekly_all, index(X_weekly_lin_trn))
StdDev.annualized(ret_weekly_all_trn)
#>                              w_weekly_lin w_weekly_log w_weekly_log_trans
#> Annualized Standard Deviation     0.140611    0.1406392          0.1406347
```

We can see that the portfolios are still almost identical with identical performance (no benefit seems to come from the log-to-lin transformation).

Monthly rebalancing

If we now repeat the same analysis for monthly returns, we get the following results:




```
# compute monthly returns
ret_monthly_all_trn <- xts(X_monthly_lin_trn %*% w_monthly_all, index(X_monthly_lin_tr
n))
StdDev.annualized(ret_monthly_all_trn)
#>                              w_monthly_lin w_monthly_log
#> Annualized Standard Deviation     0.1094179     0.1095091
#>                              w_monthly_log_trans
#> Annualized Standard Deviation           0.1094726
```

In the monthly case the portfolios are slightly different with an almost identical performance (again no benefit from the log-to-lin transformation).

Thus, from now on we will use the log-returns for modeling and the linear returns for the computation of the portfolio return (we will not use the log-to-lin transformation for $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$).

Portfolio designs

We are now ready to consider several portfolio designs and compare their performance (the underlying constraint is simply $\mathbf{w} \geq \mathbf{0}$ and $\mathbf{1}^T\mathbf{w} = 1$). We will estimate $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ from the log-returns:

```
mu <- colMeans(X_log_trn)
Sigma <- cov(X_log_trn)
```

Buy & Hold (B&H)

Buy & Hold simply means that we allocate the whole budget to one stock and we stick to it. Since we have $N = 9$ stocks in our universe, we can define $N = 9$ different B&H portfolios, which we will store as column vectors.

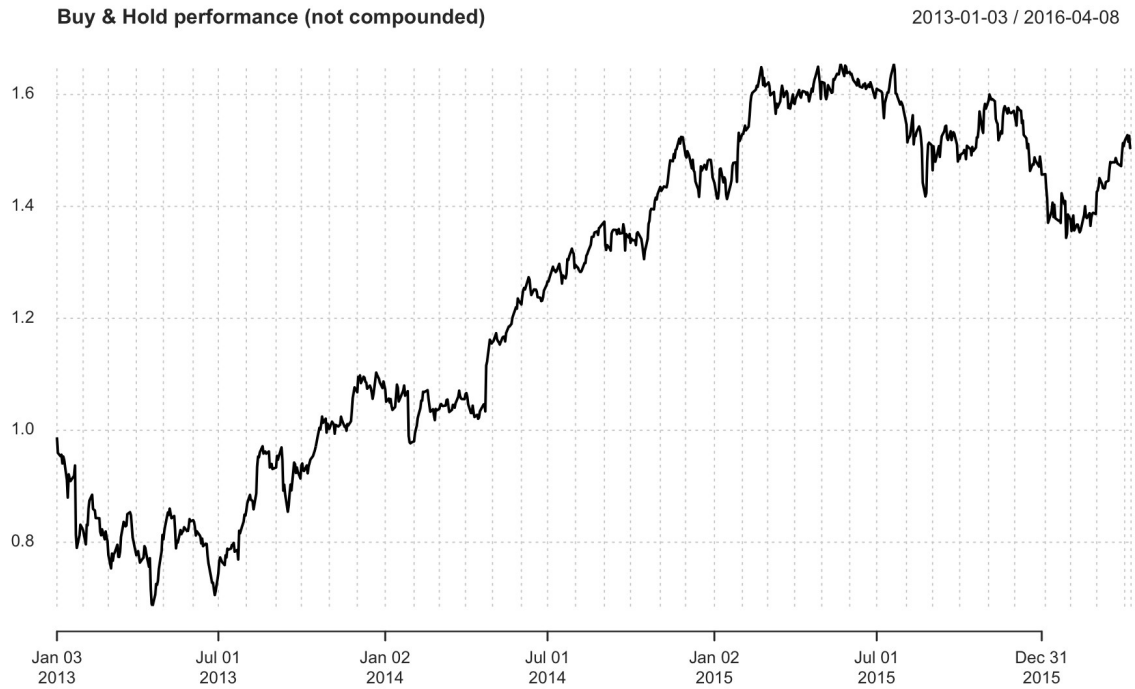
```
# a B&H portfolio is trivially the zero vector with a one on the stock held
w_BnH <- diag(N)
rownames(w_BnH) <- colnames(X_lin)
colnames(w_BnH) <- paste0("BnH-", colnames(X_lin))
w_BnH
#>      BnH-AAPL BnH-AMD BnH-ADI BnH-ABBV BnH-AET BnH-A BnH-APD BnH-AA BnH-CF
#> AAPL        1       0       0        0       0     0       0      0      0
#> AMD         0       1       0        0       0     0       0      0      0
#> ADI         0       0       1        0       0     0       0      0      0
#> ABBV        0       0       0        1       0     0       0      0      0
#> AET         0       0       0        0       1     0       0      0      0
#> A           0       0       0        0       0     1       0      0      0
#> APD         0       0       0        0       0     0       1      0      0
#> AA          0       0       0        0       0     0       0      1      0
#> CF          0       0       0        0       0     0       0      0      1
```

We can now compute the performance of those $N = 9$ portfolios in the training data:

```
# compute returns of all B&H portfolios
ret_BnH <- xts(X_lin %*% w_BnH, index(X_lin))
ret_BnH_trn <- ret_BnH[1:T_trn, ]
ret_BnH_tst <- ret_BnH[-c(1:T_trn), ]
head(ret_BnH)
#>                BnH-AAPL      BnH-AMD      BnH-ADI     BnH-ABBV
#> 2013-01-03 -0.012622264 -0.015810277 -0.016136410 -0.008257575
#> 2013-01-04 -0.027854400  0.040160643 -0.017786801 -0.012632852
#> 2013-01-07 -0.005882513  0.030888031  0.003057486  0.002035692
#> 2013-01-08  0.002691300  0.000000000 -0.010316712 -0.021764333
#> 2013-01-09 -0.015628641 -0.014981273 -0.002606036  0.005636335
#> 2013-01-10  0.012395583 -0.003802281  0.012113804  0.002949788
#>                 BnH-AET        BnH-A       BnH-APD       BnH-AA
#> 2013-01-03 -0.021189491  0.003581479 -0.0034941709  0.008898612
#> 2013-01-04  0.004638825  0.019747811  0.0134409177  0.020948193
#> 2013-01-07  0.014951845 -0.007232634 -0.0009226391 -0.017278772
#> 2013-01-08 -0.034377837 -0.007990623  0.0018468652  0.000000000
#> 2013-01-09  0.013519591  0.027007604  0.0134808277 -0.002197690
#> 2013-01-10  0.017341032  0.007381870 -0.0009095275 -0.012114520
#>                  BnH-CF
#> 2013-01-03 -0.004728498
#> 2013-01-04  0.022398600
#> 2013-01-07 -0.003745979
#> 2013-01-08 -0.014660478
#> 2013-01-09  0.034974169
#> 2013-01-10  0.015028951
```

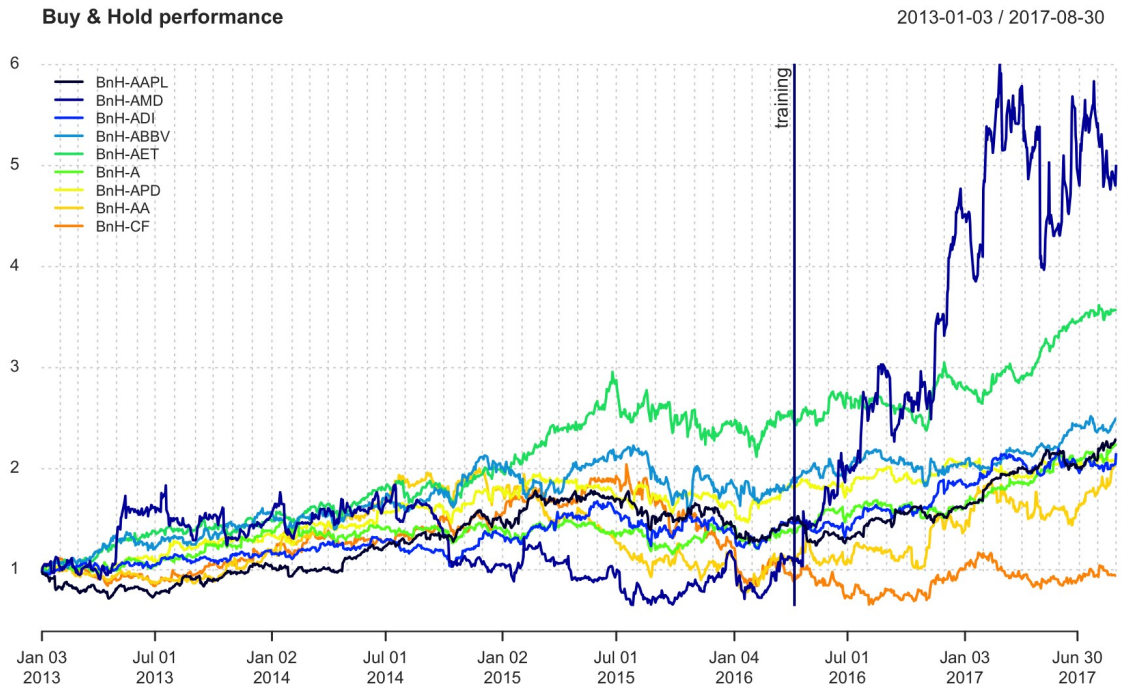
To compute the wealth or cumulative P&L, we have two options: one assumes the same quantity is repeateadly invested whereas the other assumes reinvesting (compounding):

```
# compute cumulative wealth
wealth_arith_BnH_trn <- 1 + cumsum(ret_BnH_trn)  # initial budget of 1$
wealth_geom_BnH_trn <- cumprod(1 + ret_BnH_trn)  # initial budget of 1$

# performance measures
table.AnnualizedReturns(ret_BnH_trn)
#>                           BnH-AAPL BnH-AMD BnH-ADI BnH-ABBV BnH-AET  BnH-A
#> Annualized Return           0.1284  0.0247  0.1246   0.2106  0.3154 0.1023
#> Annualized Std Dev          0.2605  0.5245  0.2467   0.2822  0.2503 0.2279
#> Annualized Sharpe (Rf=0%)   0.4927  0.0472  0.5052   0.7463  1.2599 0.4489
#>                           BnH-APD BnH-AA  BnH-CF
#> Annualized Return          0.2000 0.0242 -0.0312
#> Annualized Std Dev         0.2108 0.3521  0.3237
#> Annualized Sharpe (Rf=0%)  0.9488 0.0689 -0.0963
table.AnnualizedReturns(ret_BnH_tst)
#>                           BnH-AAPL BnH-AMD BnH-ADI BnH-ABBV BnH-AET  BnH-A
#> Annualized Return           0.3670  1.9929  0.3163   0.2315  0.3138 0.4230
#> Annualized Std Dev          0.1939  0.7677  0.2229   0.1876  0.1860 0.1969
#> Annualized Sharpe (Rf=0%)   1.8927  2.5959  1.4190   1.2340  1.6875 2.1485
#>                           BnH-APD BnH-AA BnH-CF
#> Annualized Return          0.1034 0.6026 0.0343
#> Annualized Std Dev         0.1502 0.4040 0.4462
#> Annualized Sharpe (Rf=0%)  0.6889 1.4916 0.0769
table.DownsideRisk(ret_BnH_trn)
#>                              BnH-AAPL BnH-AMD BnH-ADI BnH-ABBV BnH-AET
#> Semi Deviation                 0.0120  0.0231  0.0106   0.0129  0.0113
#> Gain Deviation                 0.0106  0.0248  0.0118   0.0114  0.0100
#> Loss Deviation                 0.0122  0.0240  0.0105   0.0127  0.0103
#> Downside Deviation (MAR=210%)  0.0164  0.0272  0.0152   0.0171  0.0156
#> Downside Deviation (Rf=0%)     0.0117  0.0228  0.0103   0.0124  0.0107
#> Downside Deviation (0%)        0.0117  0.0228  0.0103   0.0124  0.0107
#> Maximum Drawdown               0.2889  0.6524  0.2793   0.3161  0.2839
#> Historical VaR (95%)          -0.0249 -0.0469 -0.0236  -0.0278 -0.0245
#> Historical ES (95%)           -0.0372 -0.0772 -0.0333  -0.0402 -0.0339
#> Modified VaR (95%)            -0.0270 -0.0494 -0.0204  -0.0283 -0.0251
#> Modified ES (95%)             -0.0535 -0.0721 -0.0204  -0.0469 -0.0356
#>                               BnH-A BnH-APD  BnH-AA  BnH-CF
#> Semi Deviation               0.0102  0.0090  0.0149  0.0143
#> Gain Deviation               0.0092  0.0100  0.0174  0.0149
#> Loss Deviation               0.0094  0.0083  0.0141  0.0143
#> Downside Deviation (MAR=210%) 0.0150  0.0137  0.0196  0.0190
#> Downside Deviation (Rf=0%)    0.0099  0.0085  0.0147  0.0142
#> Downside Deviation (0%)       0.0099  0.0085  0.0147  0.0142
#> Maximum Drawdown              0.2296  0.2454  0.6131  0.5889
#> Historical VaR (95%)         -0.0232 -0.0197 -0.0331 -0.0331
#> Historical ES (95%)          -0.0312 -0.0274 -0.0485 -0.0468
#> Modified VaR (95%)           -0.0225 -0.0178 -0.0308 -0.0300
#> Modified ES (95%)            -0.0339 -0.0206 -0.0357 -0.0399

# plots
# same as:
#   plot(wealth_arith_BnH_trn[, 1], main = "Buy & Hold performance (not compounded)", y
lab = "wealth")
chart.CumReturns(ret_BnH_trn[, 1], main = "Buy & Hold performance (not compounded)",
                 geometric = FALSE, wealth.index = TRUE)
```

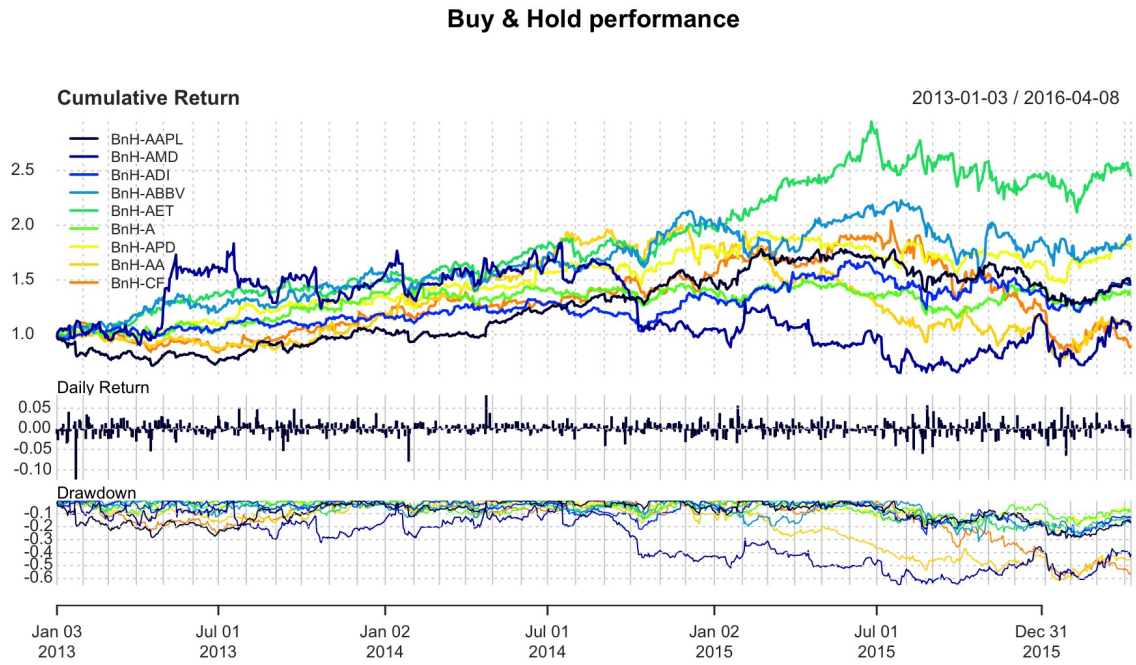



```
# same as:
#   plot(wealth_geom_BnH_trn[, 1], main = "Buy & Hold performance (compounded)", ylab
= "wealth")
chart.CumReturns(ret_BnH_trn[, 1], main = "Buy & Hold performance (compounded)",
                 geometric = TRUE, wealth.index = TRUE)
```

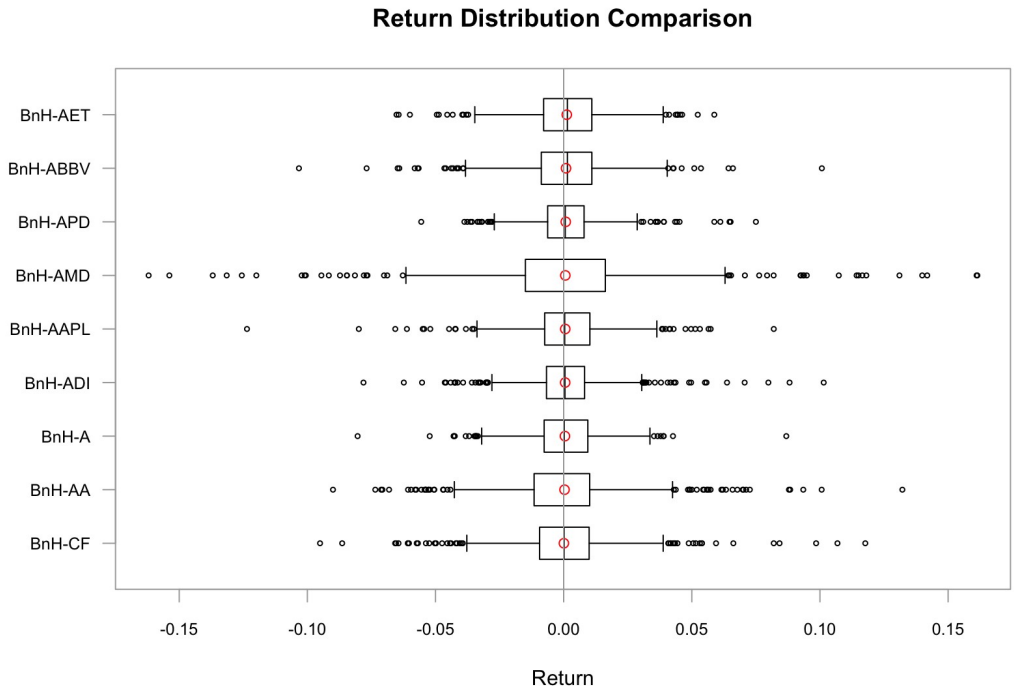

```
# more plots
{ chart.CumReturns(ret_BnH, main = "Buy & Hold performance",
                   wealth.index = TRUE, legend.loc = "topleft", colorset = rich10equal)
  addEventLines(xts("training", index(X_lin[T_trn])), srt=90, pos=2, lwd = 2, col = "da
rkblue") }
```



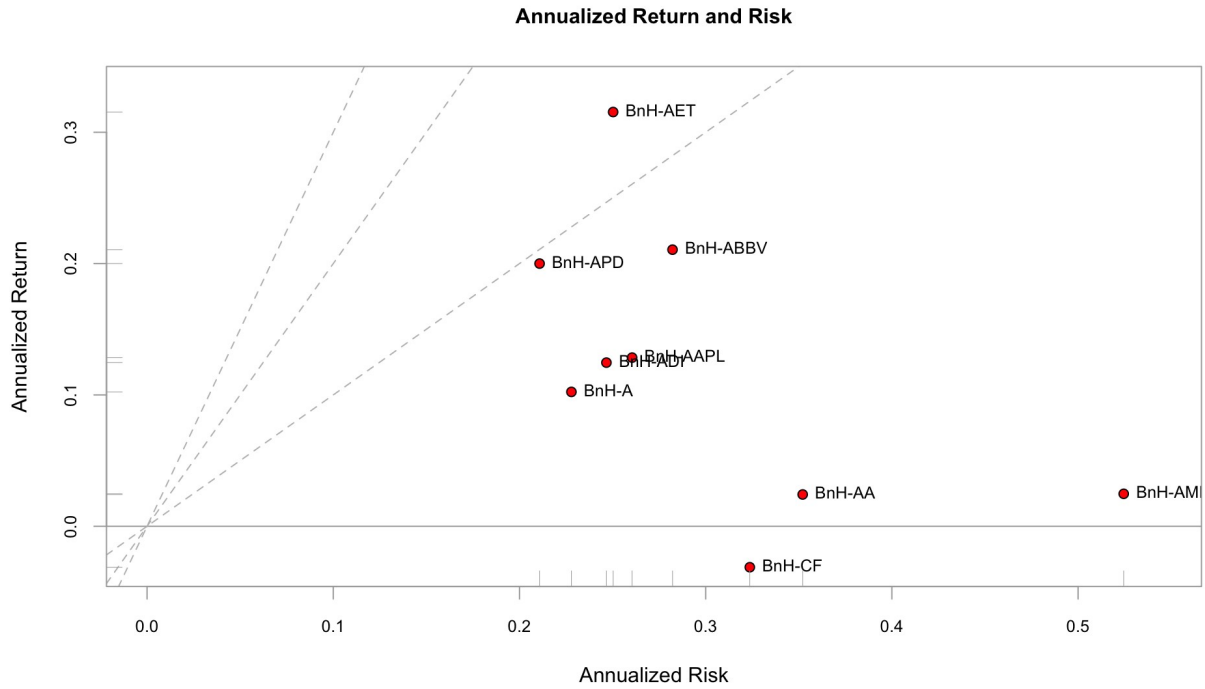

```
charts.PerformanceSummary(ret_BnH_trn, main = "Buy & Hold performance",
                          wealth.index = TRUE, colorset = rich10equal)
```




```
chart.Boxplot(ret_BnH_trn)
```




```
chart.RiskReturnScatter(ret_BnH_trn, symbolset = 21, bg = "red")
```



Uniform portfolio

The uniform portfolio allocates equal weight to each stock: $\mathbf{w} = \frac{1}{N}\mathbf{1}$.

```
w_unif <- rep(1/N, N)
```

GMVP

The Global Minimum Variance Portfolio (GMVP) is formulated as

$$\begin{array}{ll} \underset{\mathbf{w}}{\text{minimize}} & \mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w} \\ \text{subject to} & \mathbf{1}^T\mathbf{w} = 1 \\ & \mathbf{w} \geq \mathbf{0} \end{array}$$

We can compute it with a solver (the closed-form solution does not exist for $\mathbf{w} \geq \mathbf{0}$):

```
library(CVXR)

portolioGMVP <- function(Sigma) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Minimize(quad_form(w, Sigma)),
                  constraints = list(w >= 0, sum(w) == 1))
  result <- solve(prob)
  return(as.vector(result$getValue(w)))
}

w_GMVP <- portolioGMVP(Sigma)
```

Markowitz portfolio

The mean-variance Markowitz portfolio with no shorting is formulated as

$$\begin{array}{ll} \underset{\mathbf{w}}{\text{maximize}} & \boldsymbol{\mu}^T\mathbf{w} - \lambda\mathbf{w}^T\boldsymbol{\Sigma}\mathbf{w} \\ \text{subject to} & \mathbf{1}^T\mathbf{w} = 1 \\ & \mathbf{w} \geq \mathbf{0} \end{array}$$

```
portolioMarkowitz <- function(mu, Sigma, lmd = 0.5) {
  w <- Variable(nrow(Sigma))
  prob <- Problem(Maximize(t(mu) %*% w - lmd*quad_form(w, Sigma)),
                  constraints = list(w >= 0, sum(w) == 1))
  result <- solve(prob)
  return(as.vector(result$getValue(w)))
}

w_Markowitz <- portolioMarkowitz(mu, Sigma, lmd = 2)
```

Maximum Sharpe ratio portfolio

The maximum Sharpe ratio portfolio is a nonconvex problem but can be rewritten in convex form as

$$\begin{array}{ll} \underset{\mathbf{w}}{\text{minimize}} & \tilde{\mathbf{w}}^T\boldsymbol{\Sigma}\tilde{\mathbf{w}} \\ \text{subject to} & \tilde{\mathbf{w}}^T\boldsymbol{\mu} = 1 \\ & \tilde{\mathbf{w}} \geq \mathbf{0} \end{array}$$

and then $\mathbf{w} = \tilde{\mathbf{w}}/(\mathbf{1}^T\tilde{\mathbf{w}})$.

```
portolioMaxSharpeRatio <- function(mu, Sigma) {
  w_ <- Variable(nrow(Sigma))
  prob <- Problem(Minimize(quad_form(w_, Sigma)),
                  constraints = list(w_ >= 0, t(mu) %*% w_ == 1))
  result <- solve(prob)
  return(as.vector(result$getValue(w_)/sum(result$getValue(w_))))
}

w_maxSR <- portolioMaxSharpeRatio(mu, Sigma)
```

Long or Long-Short quintile portfolios

The Long (L) or Long-Short (LS) quintile portfolios are widely used by practitioners (as opposed to the mean-variance portfolio). The idea is to 1) rank the N stocks, 2) divide them into five parts, and 3) long the top part (and possibly short the bottom part). One can rank the stocks in a multitude of ways (typically based on expensive factors that investment funds buy at a premium price). For our experiments, we will consider three possible rankings according to:

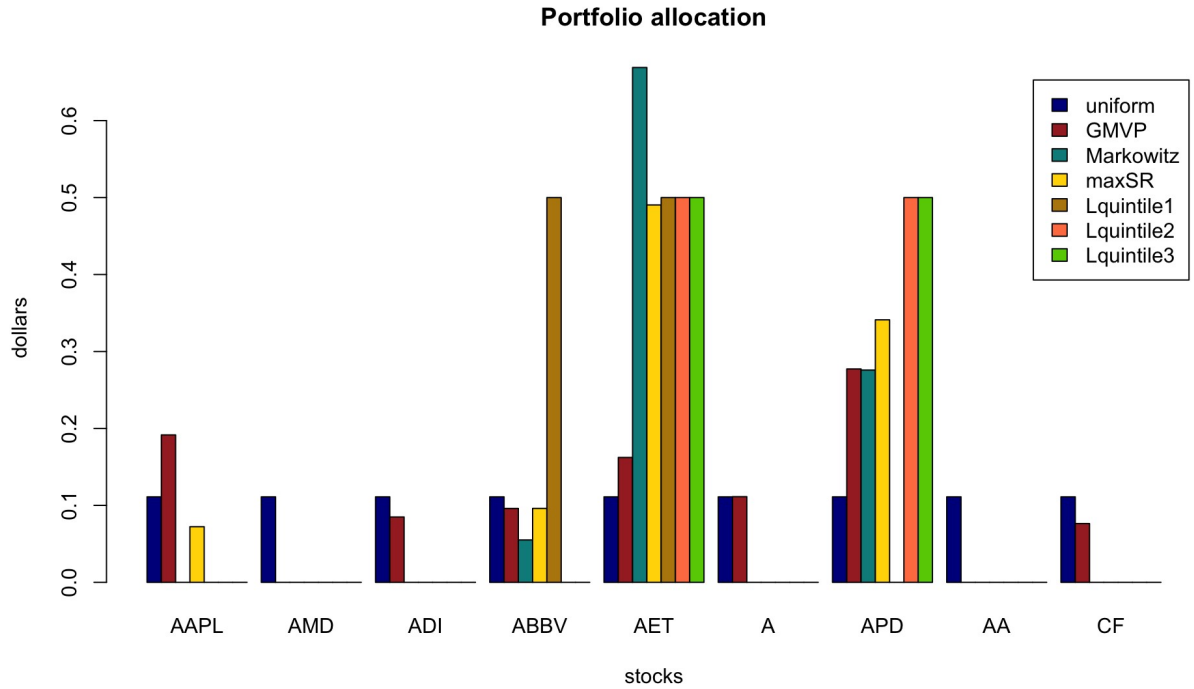
1. $\boldsymbol{\mu}$
2. $\boldsymbol{\mu}/\text{diag}(\boldsymbol{\Sigma})$
3. $\boldsymbol{\mu}/\sqrt{(\text{diag}(\boldsymbol{\Sigma}))}$

```
# find indices of sorted stocks
i1 <- sort(mu, decreasing = TRUE, index.return = TRUE)$ix
i2 <- sort(mu/diag(Sigma), decreasing = TRUE, index.return = TRUE)$ix
i3 <- sort(mu/sqrt(diag(Sigma)), decreasing = TRUE, index.return = TRUE)$ix

# create portfolios
w_Lquintile1 <- w_Lquintile2 <- w_Lquintile3 <- rep(0, N)
w_Lquintile1[i1[1:round(N/5)]] <- 1/round(N/5)
w_Lquintile2[i2[1:round(N/5)]] <- 1/round(N/5)
w_Lquintile3[i3[1:round(N/5)]] <- 1/round(N/5)
w_Lquintile <- cbind(w_Lquintile1, w_Lquintile2, w_Lquintile3)
rownames(w_Lquintile) <- colnames(X_lin)
colnames(w_Lquintile) <- c("Lquintile1", "Lquintile2", "Lquintile3")
w_Lquintile
#>      Lquintile1 Lquintile2 Lquintile3
#> AAPL        0.0        0.0        0.0
#> AMD         0.0        0.0        0.0
#> ADI         0.0        0.0        0.0
#> ABBV        0.5        0.0        0.0
#> AET         0.5        0.5        0.5
#> A           0.0        0.0        0.0
#> APD         0.0        0.5        0.5
#> AA          0.0        0.0        0.0
#> CF          0.0        0.0        0.0
```

Return-risk tradeoff for all portfolios

We can now compare the allocations of the portfolios:

```
# put together all portfolios
w_meanvar <- cbind(w_unif, w_GMVP, w_Markowitz, w_maxSR)
rownames(w_meanvar) <- colnames(X_lin)
colnames(w_meanvar) <- c("uniform", "GMVP", "Markowitz", "maxSR")
w_all <- cbind(w_BnH, w_meanvar, w_Lquintile)
round(w_all, digits = 2)
#>      BnH-AAPL BnH-AMD BnH-ADI BnH-ABBV BnH-AET BnH-A BnH-APD BnH-AA BnH-CF
#> AAPL        1       0       0        0       0     0       0      0      0
#> AMD         0       1       0        0       0     0       0      0      0
#> ADI         0       0       1        0       0     0       0      0      0
#> ABBV        0       0       0        1       0     0       0      0      0
#> AET         0       0       0        0       1     0       0      0      0
#> A           0       0       0        0       0     1       0      0      0
#> APD         0       0       0        0       0     0       1      0      0
#> AA          0       0       0        0       0     0       0      1      0
#> CF          0       0       0        0       0     0       0      0      1
#>      uniform GMVP Markowitz maxSR Lquintile1 Lquintile2 Lquintile3
#> AAPL    0.11 0.19      0.00  0.07        0.0        0.0        0.0
#> AMD     0.11 0.00      0.00  0.00        0.0        0.0        0.0
#> ADI     0.11 0.08      0.00  0.00        0.0        0.0        0.0
#> ABBV    0.11 0.10      0.06  0.10        0.5        0.0        0.0
#> AET     0.11 0.16      0.67  0.49        0.5        0.5        0.5
#> A       0.11 0.11      0.00  0.00        0.0        0.0        0.0
#> APD     0.11 0.28      0.28  0.34        0.0        0.5        0.5
#> AA      0.11 0.00      0.00  0.00        0.0        0.0        0.0
#> CF      0.11 0.08      0.00  0.00        0.0        0.0        0.0
barplot(t(cbind(w_meanvar, w_Lquintile)),
        main = "Portfolio allocation", xlab = "stocks", ylab = "dollars", beside = TRU
E,
        legend = c(colnames(w_meanvar), colnames(w_Lquintile)),
        col = c("darkblue", "brown", "darkcyan", "gold", "darkgoldenrod", "coral", "cha
rtreuse3"))
```

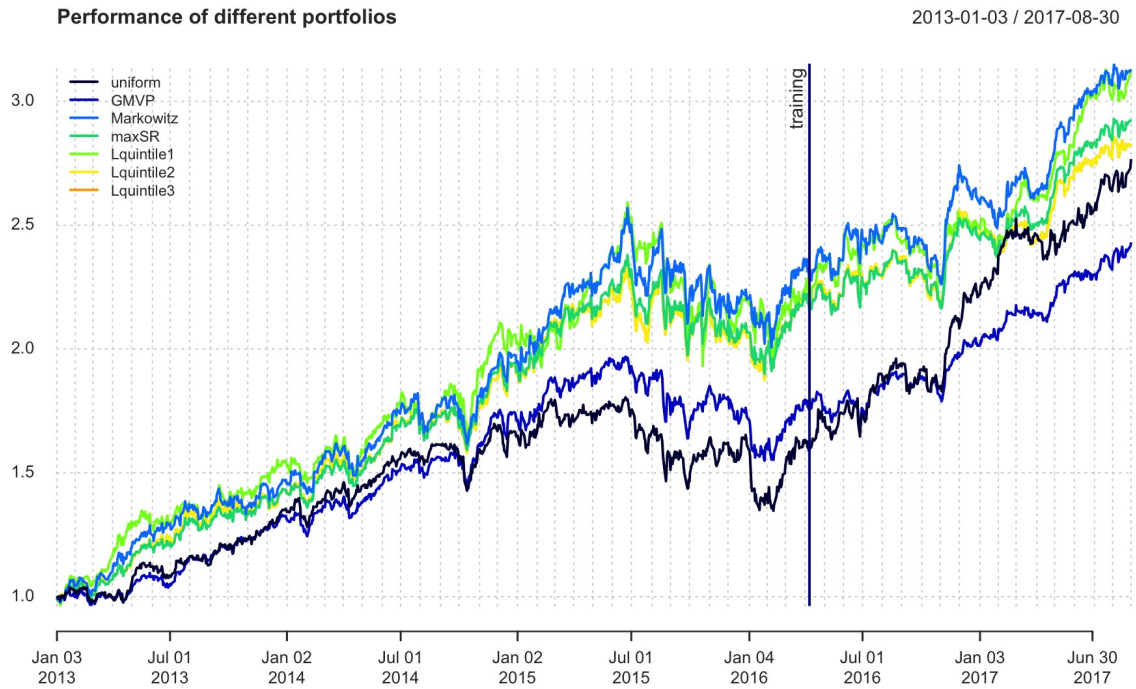


Then we can compare the performance (in sample vs out-of-sample):

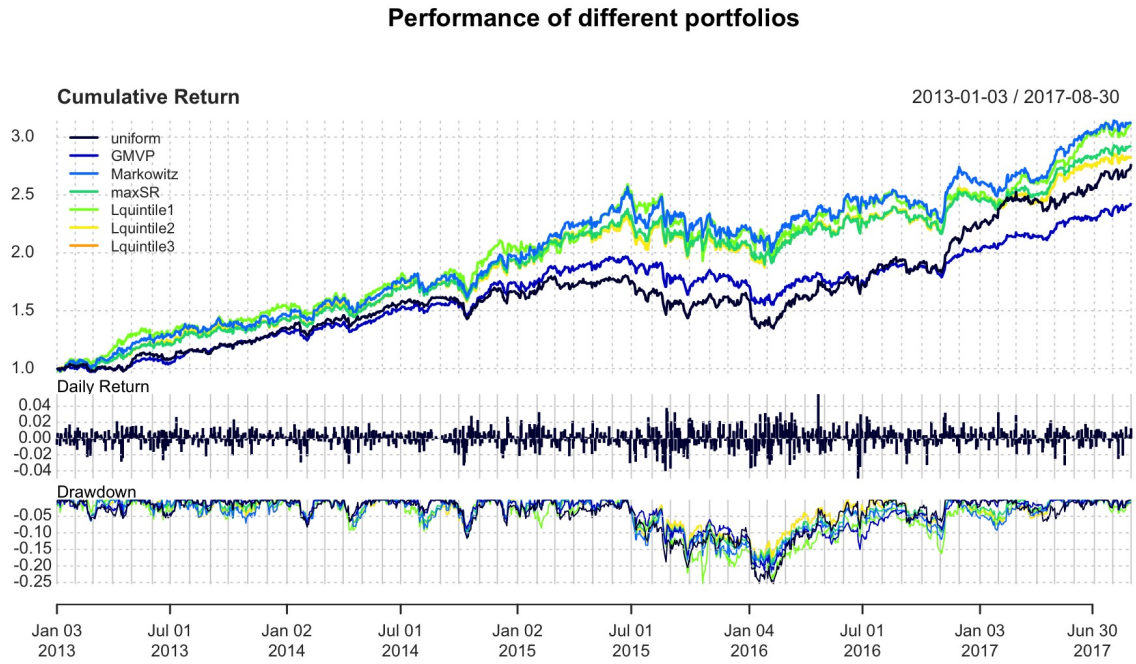
```
# compute returns of all portfolios
ret_all <- xts(X_lin %*% w_all, index(X_lin))
ret_all_trn <- ret_all[1:T_trn, ]
ret_all_tst <- ret_all[-c(1:T_trn), ]

# performance
table.AnnualizedReturns(ret_all_trn)
#>                           BnH-AAPL BnH-AMD BnH-ADI BnH-ABBV BnH-AET  BnH-A
#> Annualized Return           0.1284  0.0247  0.1246   0.2106  0.3154 0.1023
#> Annualized Std Dev          0.2605  0.5245  0.2467   0.2822  0.2503 0.2279
#> Annualized Sharpe (Rf=0%)   0.4927  0.0472  0.5052   0.7463  1.2599 0.4489
#>                           BnH-APD BnH-AA  BnH-CF uniform   GMVP Markowitz
#> Annualized Return          0.2000 0.0242 -0.0312  0.1539 0.1887    0.2877
#> Annualized Std Dev         0.2108 0.3521  0.3237  0.1805 0.1617    0.2030
#> Annualized Sharpe (Rf=0%)  0.9488 0.0689 -0.0963  0.8524 1.1675    1.4176
#>                            maxSR Lquintile1 Lquintile2 Lquintile3
#> Annualized Return         0.2666     0.2757     0.2673     0.2673
#> Annualized Std Dev        0.1824     0.2223     0.1902     0.1902
#> Annualized Sharpe (Rf=0%) 1.4611     1.2404     1.4050     1.4050
table.AnnualizedReturns(ret_all_tst)
#>                           BnH-AAPL BnH-AMD BnH-ADI BnH-ABBV BnH-AET  BnH-A
#> Annualized Return           0.3670  1.9929  0.3163   0.2315  0.3138 0.4230
#> Annualized Std Dev          0.1939  0.7677  0.2229   0.1876  0.1860 0.1969
#> Annualized Sharpe (Rf=0%)   1.8927  2.5959  1.4190   1.2340  1.6875 2.1485
#>                           BnH-APD BnH-AA BnH-CF uniform   GMVP Markowitz
#> Annualized Return          0.1034 0.6026 0.0343  0.4835 0.2613    0.2538
#> Annualized Std Dev         0.1502 0.4040 0.4462  0.1725 0.1234    0.1463
#> Annualized Sharpe (Rf=0%)  0.6889 1.4916 0.0769  2.8035 2.1181    1.7348
#>                            maxSR Lquintile1 Lquintile2 Lquintile3
#> Annualized Return         0.2424     0.2788     0.2103     0.2103
#> Annualized Std Dev        0.1293     0.1556     0.1347     0.1347
#> Annualized Sharpe (Rf=0%) 1.8746     1.7912     1.5620     1.5620
```

We can observe that the maximum Sharpe ratio portfolio indeed achieves the maximum Sharpe ratio among the portfolios in sample, but not necessarily out-of-sample since the statistics may have changed. The GMVP consistently achieves the minimum variance (equivalently, minimum standard deviation). It is always good to plot the wealth evolution over time:

```
{ chart.CumReturns(ret_all[, -c(1:9)], main = "Performance of different portfolios",
                   wealth.index = TRUE, legend.loc = "topleft", colorset = rich8equal)
  addEventLines(xts("training", index(X_lin[T_trn])), srt=90, pos=2, lwd = 2, col = "da
rkblue") }
```

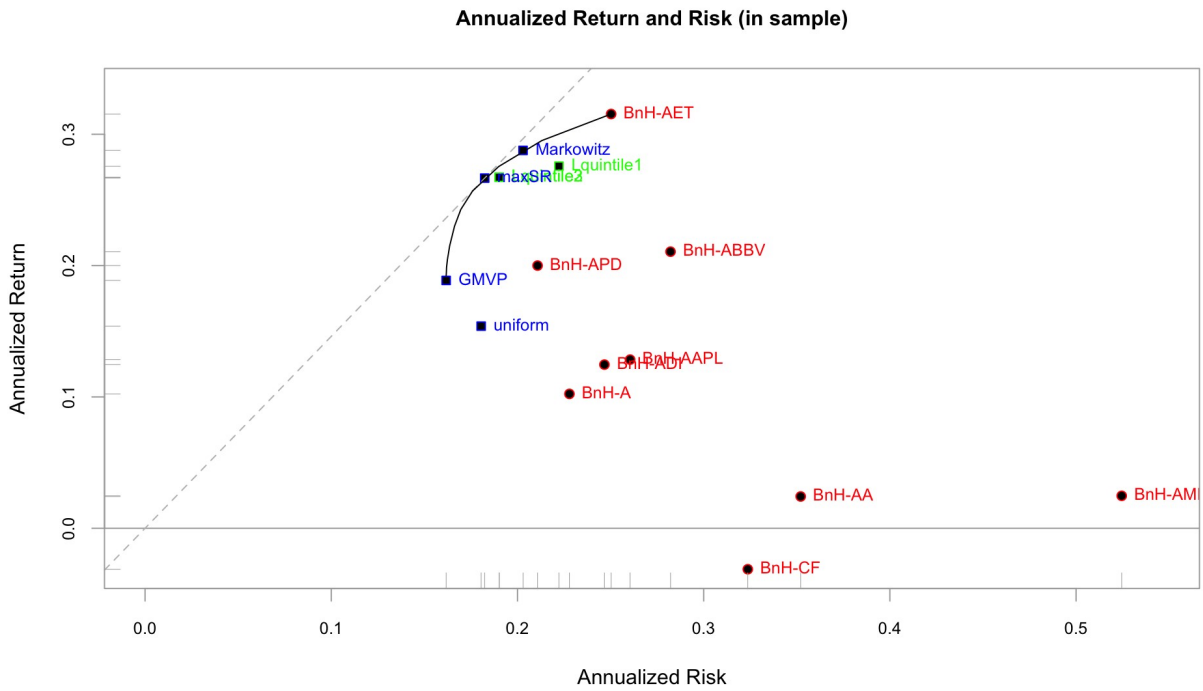



```
charts.PerformanceSummary(ret_all[, -c(1:9)], main = "Performance of different portfoli
os",
                          wealth.index = TRUE, colorset = rich8equal)
```



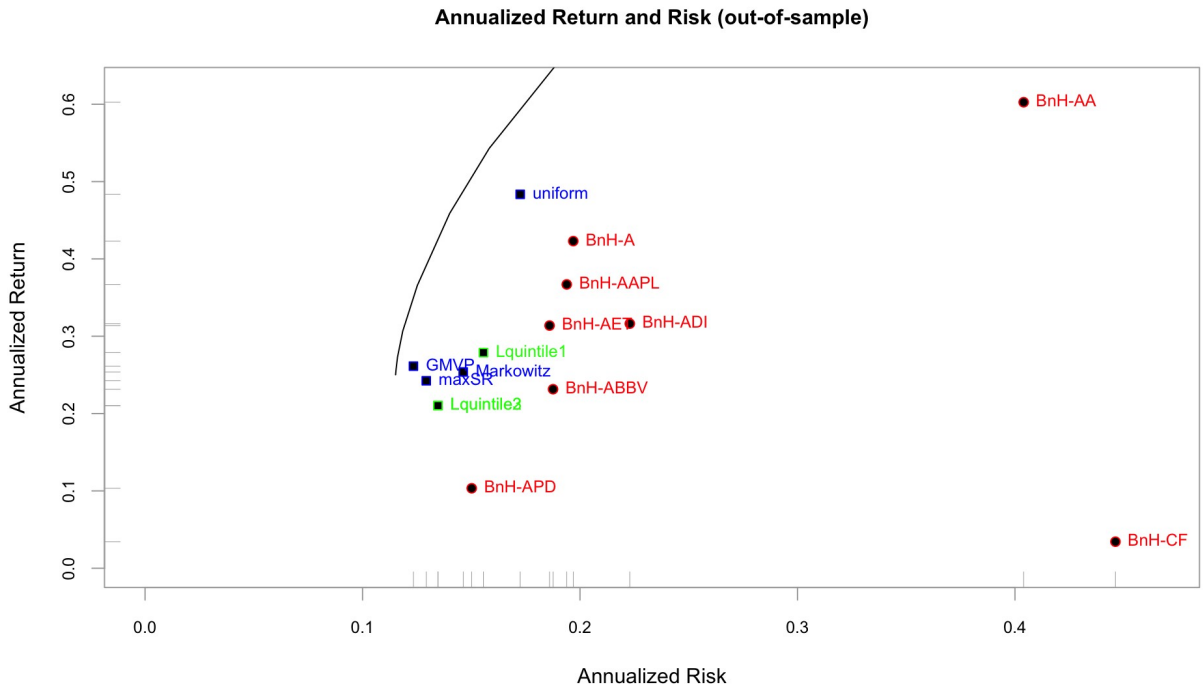
Finally, we can plot the expected return vs the standard deviation along with the efficient frontier:

```
# first, compute the efficient frontier
w_frontier_trn <- NULL
lmd_sweep <- exp(seq(-5, 5, by = 0.5))
for (lmd in lmd_sweep)
  w_frontier_trn <- cbind(w_frontier_trn, portolioMarkowitz(mu, Sigma, lmd))
ret_frontier_trn <- xts(X_lin_trn %*% w_frontier_trn, index(X_lin_trn))
mu_frontier_trn <- table.AnnualizedReturns(ret_frontier_trn)[1, ]
sd_frontier_trn <- table.AnnualizedReturns(ret_frontier_trn)[2, ]

# plot in-sample sd-mu scatter plot
maxSR <- table.AnnualizedReturns(ret_all_trn[, "maxSR"])[3, ]
chart.RiskReturnScatter(ret_all_trn,
                        main = "Annualized Return and Risk (in sample)",
                        symbolset = c(rep(21, 9), rep(22, 4+3)),
                        colorset = c(rep("red", 9), rep("blue", 4), rep("green", 3)),
                        bg = "black",
                        add.sharpe = maxSR)
lines(sd_frontier_trn, mu_frontier_trn)
```



Observe that this nice return-risk scatter plot totally deforms in an unpredictable way when we evaluate it in the out-of-sample set:

```
# compute the efficient frontier again but based on the test data
mu_tst <- colMeans(X_log_tst)
Sigma_tst <- cov(X_log_tst)

w_frontier_tst <- NULL
lmd_sweep <- exp(seq(-5, 5, by = 0.5))
for (lmd in lmd_sweep)
  w_frontier_tst <- cbind(w_frontier_tst, portolioMarkowitz(mu_tst, Sigma_tst, lmd))
ret_frontier_tst <- xts(X_lin_tst %*% w_frontier_tst, index(X_lin_tst))
mu_frontier_tst <- table.AnnualizedReturns(ret_frontier_tst)[1, ]
sd_frontier_tst <- table.AnnualizedReturns(ret_frontier_tst)[2, ]

# plot out-of-sample sd-mu scatter plot
chart.RiskReturnScatter(ret_all_tst[, -2],
                        main = "Annualized Return and Risk (out-of-sample)",
                        symbolset = c(rep(21, 9-1), rep(22, 4+3)),
                        colorset = c(rep("red", 9-1), rep("blue", 4), rep("green", 3)),
                        bg = "black",
                        add.sharpe = NA)
lines(sd_frontier_tst, mu_frontier_tst)
```

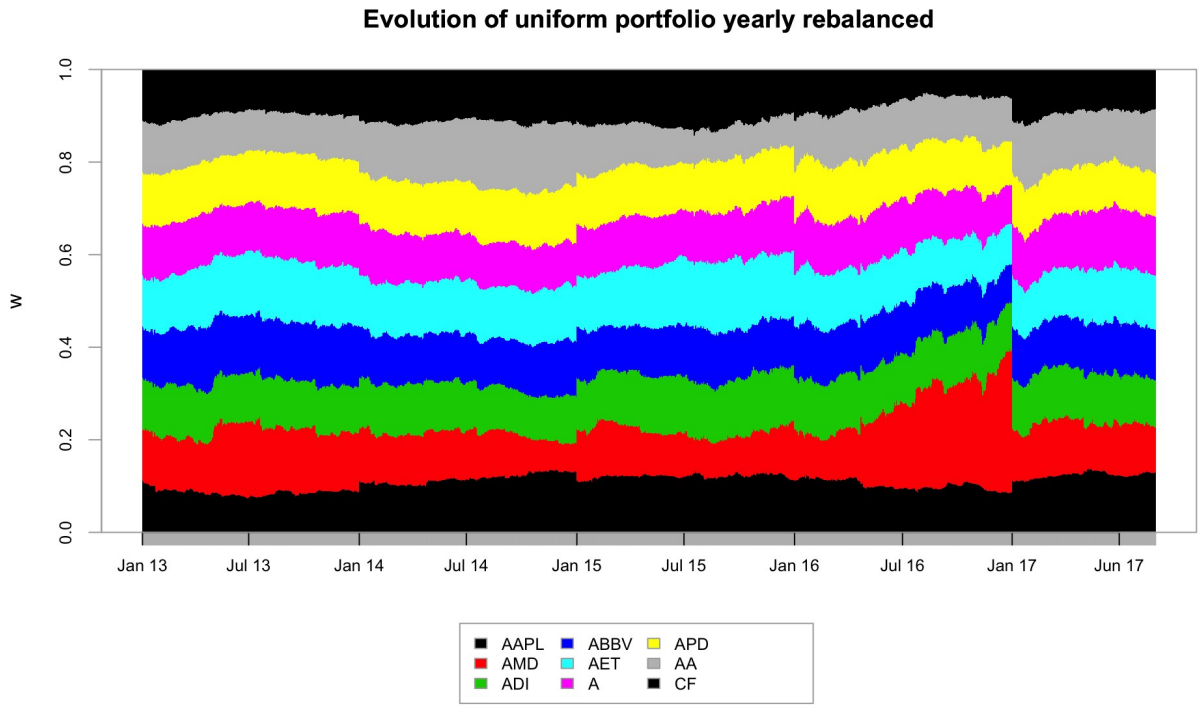


Performance with slower rebalancing frequencies

In the previous performance analysis of the different portfolios, we were implicitly assuming a daily rebalancing and that's why computing the return of the portfolio was as easy as `ret_portf <- X_lin %*% w`. In this section, we will consider a lower rebalancing frequency (weekly, monthly, quarterly, and yearly) for a given portfolio (in particular, we will just use the mean-variance portfolio `w_Markowitz`). Observe that on the day of the rebalancing, the portfolio held will be `w_Markowitz` but in the subsequent days without rebalancing such portfolio slowly deviates.

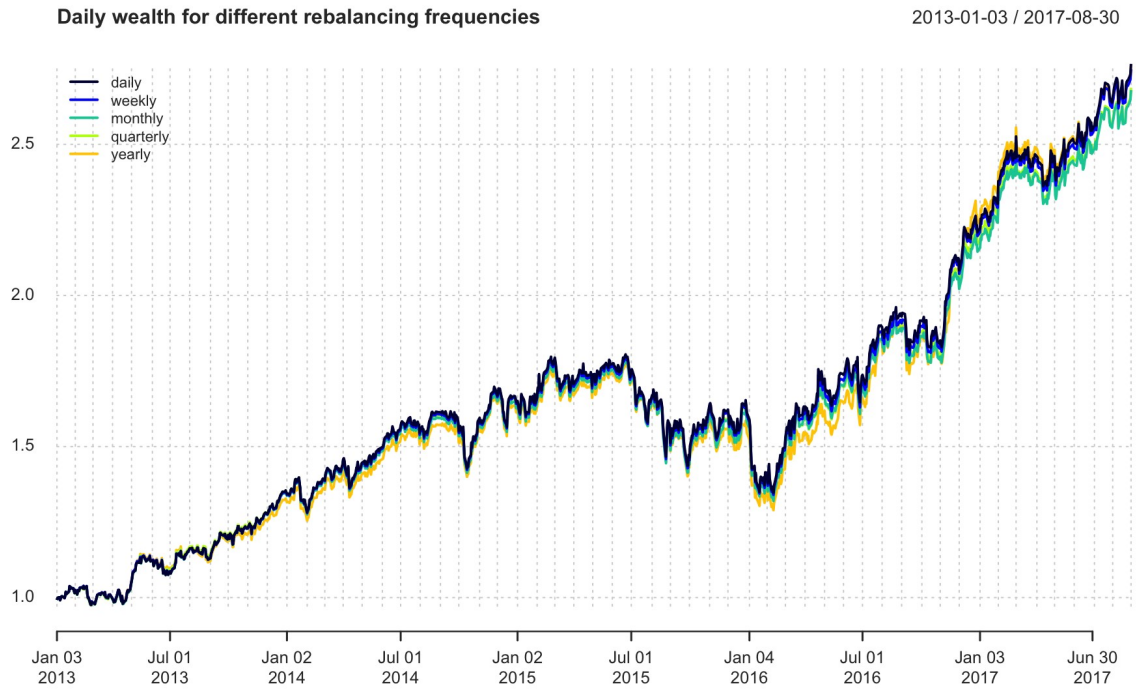
The package **PerformanceAnalytics** has the convenient function `Return.portfolio()` that allows us to compute the portfolio return with different rebalancing schemes:

```
# choose portfolio for the comparison
w <- w_unif

# recall the computation for the daily rebalancing (with daily returns)
ret_daily_rebal <- X_lin %*% w

# we can alternativaly use this function:
ret_daily_rebal_ <- Return.portfolio(X_lin, weights = w, rebalance_on = "days")
norm(ret_daily_rebal - ret_daily_rebal_)  #sanity check
#> [1] 9.342538e-14

# let's observe how the portfolio slowly deviates from its original design
tmp <- Return.portfolio(X_lin, weights = w, rebalance_on = "years", verbose = TRUE)
round(head(tmp$BOP.Weight, 15), digits = 4)
#>               AAPL    AMD    ADI   ABBV    AET      A    APD     AA     CF
#> 2013-01-03 0.1111 0.1111 0.1111 0.1111 0.1111 0.1111 0.1111 0.1111 0.1111
#> 2013-01-04 0.1106 0.1102 0.1102 0.1111 0.1096 0.1124 0.1116 0.1130 0.1114
#> 2013-01-07 0.1067 0.1138 0.1075 0.1089 0.1093 0.1138 0.1123 0.1145 0.1131
#> 2013-01-08 0.1059 0.1171 0.1076 0.1089 0.1108 0.1128 0.1120 0.1124 0.1125
#> 2013-01-09 0.1072 0.1182 0.1075 0.1075 0.1080 0.1129 0.1133 0.1134 0.1119
#> 2013-01-10 0.1048 0.1157 0.1065 0.1074 0.1087 0.1152 0.1140 0.1124 0.1151
#> 2013-01-11 0.1056 0.1146 0.1072 0.1072 0.1100 0.1154 0.1133 0.1105 0.1162
#> 2013-01-14 0.1048 0.1167 0.1062 0.1079 0.1094 0.1147 0.1138 0.1100 0.1164
#> 2013-01-15 0.1015 0.1159 0.1066 0.1091 0.1104 0.1154 0.1138 0.1102 0.1172
#> 2013-01-16 0.0985 0.1178 0.1062 0.1109 0.1112 0.1149 0.1136 0.1104 0.1165
#> 2013-01-17 0.1018 0.1188 0.1056 0.1131 0.1096 0.1135 0.1123 0.1091 0.1161
#> 2013-01-18 0.0998 0.1181 0.1067 0.1144 0.1117 0.1142 0.1123 0.1084 0.1144
#> 2013-01-22 0.1001 0.1068 0.1075 0.1181 0.1120 0.1158 0.1135 0.1100 0.1161
#> 2013-01-23 0.1002 0.1055 0.1069 0.1143 0.1144 0.1161 0.1135 0.1115 0.1176
#> 2013-01-24 0.1003 0.1156 0.1056 0.1167 0.1131 0.1134 0.1104 0.1091 0.1157
chart.StackedBar(tmp$BOP.Weight, main = "Evolution of uniform portfolio yearly rebalanc
ed",
                 ylab = "w", space=0, border = NA)
```



We can now compare the different rebalancing frequencies:

```
# now different rebalancing frequencies
ret_weekly_rebal <- Return.portfolio(X_lin, weights = w, rebalance_on = "weeks")
ret_monthly_rebal <- Return.portfolio(X_lin, weights = w, rebalance_on = "months")
ret_quarterly_rebal <- Return.portfolio(X_lin, weights = w, rebalance_on = "quarters")
ret_yearly_rebal <- Return.portfolio(X_lin, weights = w, rebalance_on = "years")
ret_allfreqs <- cbind(ret_daily_rebal, ret_weekly_rebal, ret_monthly_rebal, ret_quarter
ly_rebal, ret_yearly_rebal)
colnames(ret_allfreqs) <- c("daily", "weekly", "monthly", "quarterly", "yearly")
round(head(ret_allfreqs, 25), digits = 4)
#>              daily  weekly monthly quarterly  yearly
#> 2013-01-03 -0.0078 -0.0078 -0.0078   -0.0078 -0.0078
#> 2013-01-04  0.0070  0.0071  0.0071    0.0071  0.0071
#> 2013-01-07  0.0018  0.0018  0.0017    0.0017  0.0017
#> 2013-01-08 -0.0094 -0.0094 -0.0093   -0.0093 -0.0093
#> 2013-01-09  0.0066  0.0064  0.0066    0.0066  0.0066
#> 2013-01-10  0.0056  0.0056  0.0055    0.0055  0.0055
#> 2013-01-11  0.0008  0.0008  0.0009    0.0009  0.0009
#> 2013-01-14 -0.0040 -0.0040 -0.0039   -0.0039 -0.0039
#> 2013-01-15 -0.0021 -0.0020 -0.0018   -0.0018 -0.0018
#> 2013-01-16  0.0075  0.0073  0.0071    0.0071  0.0071
#> 2013-01-17  0.0131  0.0132  0.0131    0.0131  0.0131
#> 2013-01-18 -0.0072 -0.0072 -0.0077   -0.0077 -0.0077
#> 2013-01-22  0.0084  0.0084  0.0083    0.0083  0.0083
#> 2013-01-23  0.0175  0.0172  0.0167    0.0167  0.0167
#> 2013-01-24 -0.0089 -0.0087 -0.0074   -0.0074 -0.0074
#> 2013-01-25  0.0047  0.0053  0.0056    0.0056  0.0056
#> 2013-01-28 -0.0044 -0.0044 -0.0052   -0.0052 -0.0052
#> 2013-01-29  0.0012  0.0013  0.0006    0.0006  0.0006
#> 2013-01-30 -0.0088 -0.0087 -0.0090   -0.0090 -0.0090
#> 2013-01-31 -0.0060 -0.0059 -0.0061   -0.0061 -0.0061
#> 2013-02-01  0.0095  0.0096  0.0095    0.0097  0.0097
#> 2013-02-04 -0.0140 -0.0140 -0.0140   -0.0136 -0.0136
#> 2013-02-05  0.0137  0.0136  0.0135    0.0129  0.0129
#> 2013-02-06  0.0002  0.0002  0.0002    0.0001  0.0001
#> 2013-02-07 -0.0024 -0.0024 -0.0024   -0.0031 -0.0031

# performance
table.AnnualizedReturns(ret_allfreqs)
#>                            daily weekly monthly quarterly yearly
#> Annualized Return         0.2442 0.2424  0.2356    0.2364 0.2439
#> Annualized Std Dev        0.1782 0.1782  0.1781    0.1781 0.1816
#> Annualized Sharpe (Rf=0%) 1.3701 1.3605  1.3226    1.3270 1.3433
chart.CumReturns(ret_allfreqs, main = "Daily wealth for different rebalancing frequenci
es",
                 wealth.index = TRUE, legend.loc = "topleft", colorset = rich6equal)
```



Rolling window portfolios

Until now, we have considered static portfolios in the sense that they are first designed based on a training set and then they remain fixed and are used in the test set. In a more realistic setting, however, one wants to implement this procedure in a rolling-window basis. That is, with some frequency the portfolio is reoptimized (since the dynamics of the asset returns may change with time) and rebalanced.

Recall the procedure for the static portfolio where one estimates $\boldsymbol{\mu}$ and $\boldsymbol{\Sigma}$ from the training set, designs some portfolio (say, Markowitz mean-variance), and then applies it to the test set with some rebalancing frequency (say, monthly):

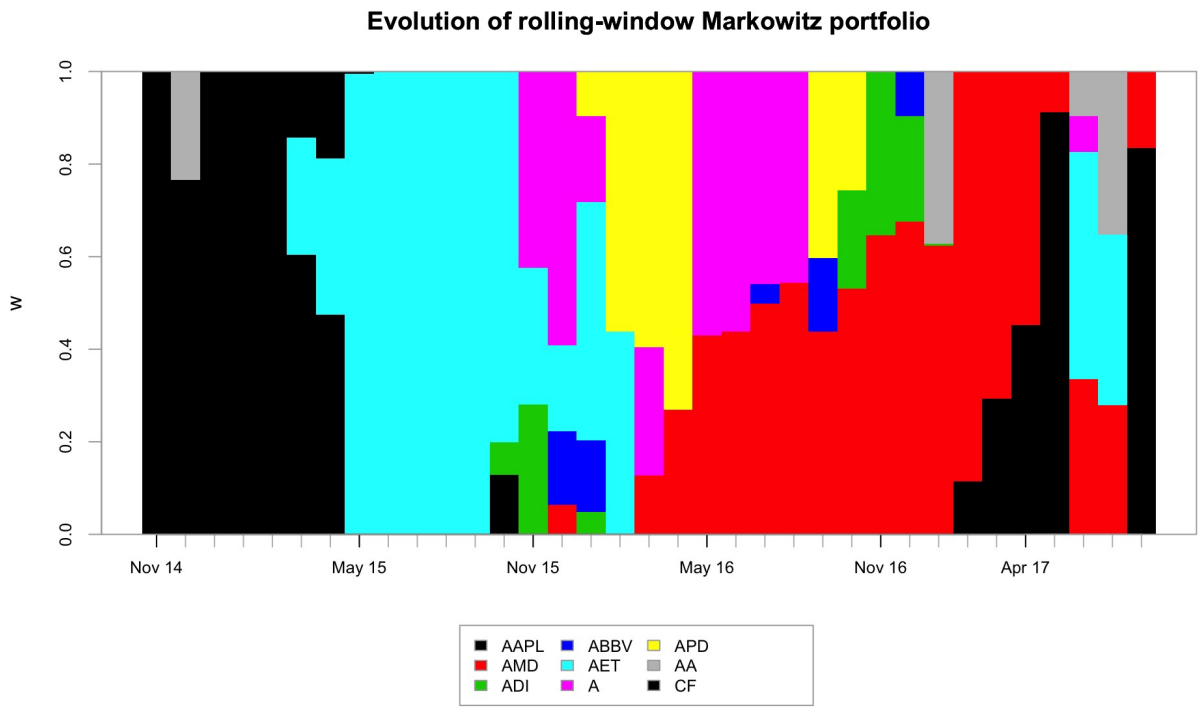
```
# recall the procedure for the static portfolio (Markowitz, for example):
T_trn <- round(0.4*T)
X_log_trn <- X_log[1:T_trn, ]
X_log_tst <- X_log[(T_trn+1):T, ]
X_lin_trn <- X_lin[1:T_trn, ]
X_lin_tst <- X_lin[(T_trn+1):T, ]
mu <- colMeans(X_log_trn)
Sigma <- cov(X_log_trn)
w_Markowitz_static <- portolioMarkowitz(mu, Sigma, lmd = 2)
ret_static <- Return.portfolio(X_lin_tst, weights = w_Markowitz_static, rebalance_on =
"months")
```

The package **PerformanceAnalytics** provides two functions to handle the two main rolling-window strategies: `apply.rolling()` is for a rolling window of fixed length, whereas `apply.fromstart()` is for an expanding rolling window. However, those only work independently on each column of the multivariate time series, which is not what we want. The package **xts** has the function `apply.rolling()` but it only works with nonoverlapping periods, which is not what we want. What we want is to choose a window of, say, 6 months (about 120 days) and shift it, say, every month. We can do it by hand with a simple loop:

```
# create empty portfolio matrix
w_Markowitz_rolling <- X_log
w_Markowitz_rolling[] <- NA
head(w_Markowitz_rolling)
#>            AAPL AMD ADI ABBV AET  A APD AA CF
#> 2013-01-03   NA  NA  NA   NA  NA NA  NA NA NA
#> 2013-01-04   NA  NA  NA   NA  NA NA  NA NA NA
#> 2013-01-07   NA  NA  NA   NA  NA NA  NA NA NA
#> 2013-01-08   NA  NA  NA   NA  NA NA  NA NA NA
#> 2013-01-09   NA  NA  NA   NA  NA NA  NA NA NA
#> 2013-01-10   NA  NA  NA   NA  NA NA  NA NA NA

# find rebalancing indices
rebal_indices <- T_trn + endpoints(X_log_tst, on = "months")
index(X_log)[rebal_indices]
#>  [1] "2014-11-12" "2014-11-28" "2014-12-31" "2015-01-30" "2015-02-27"
#>  [6] "2015-03-31" "2015-04-30" "2015-05-29" "2015-06-30" "2015-07-31"
#> [11] "2015-08-31" "2015-09-30" "2015-10-30" "2015-11-30" "2015-12-31"
#> [16] "2016-01-29" "2016-02-29" "2016-03-31" "2016-04-29" "2016-05-31"
#> [21] "2016-06-30" "2016-07-29" "2016-08-31" "2016-09-30" "2016-10-31"
#> [26] "2016-11-30" "2016-12-30" "2017-01-31" "2017-02-28" "2017-03-31"
#> [31] "2017-04-28" "2017-05-31" "2017-06-30" "2017-07-31" "2017-08-30"

# run the rolling window loop
lookback <- 10*20  # maximum value is: floor(T_trn/20)*20
for (i in 1:length(rebal_indices)) {
  # estimate moments
  X_ <- X_log[(rebal_indices[i]-lookback+1):rebal_indices[i], ]
  mu <- colMeans(X_)
  Sigma <- cov(X_)
  # design portfolio
  w_Markowitz_rolling[rebal_indices[i], ] <- portolioMarkowitz(mu, Sigma, lmd = 2)
}
w_Markowitz_rolling <- na.omit(w_Markowitz_rolling)
w_Markowitz_rolling
#>                     AAPL          AMD          ADI          ABBV
#> 2014-11-12 1.000000e+00 3.819348e-10 7.382914e-10 2.157220e-09
#> 2014-11-28 7.660026e-01 9.217035e-10 2.141786e-09 1.591763e-08
#> 2014-12-31 1.000000e+00 5.230457e-10 1.621244e-09 2.995989e-09
#> 2015-01-30 9.999999e-01 1.898148e-09 3.771865e-09 7.843722e-09
#> 2015-02-27 9.999998e-01 1.341519e-09 4.466925e-09 4.945676e-09
#> 2015-03-31 6.032743e-01 2.111278e-10 1.217692e-09 9.358022e-10
#> 2015-04-30 4.746979e-01 4.556487e-10 5.264664e-09 1.805345e-08
#> 2015-05-29 7.970214e-08 4.084947e-09 6.776097e-08 2.933007e-08
#> 2015-06-30 6.349515e-09 8.876010e-10 9.079647e-09 4.000520e-09
#> 2015-07-31 1.412364e-08 2.515863e-09 9.924369e-08 1.880446e-08
#> 2015-08-31 3.353580e-08 8.645514e-09 6.123227e-08 2.784719e-08
#> 2015-09-30 3.317958e-08 4.987550e-09 2.448373e-08 9.778022e-09
#> 2015-10-30 1.284144e-01 1.583535e-09 7.102665e-02 2.855170e-09
#> 2015-11-30 2.163691e-08 4.932557e-09 2.804220e-01 4.781639e-07
#> 2015-12-31 7.709435e-09 6.332092e-02 2.127699e-08 1.592680e-01
#> 2016-01-29 6.607589e-08 6.106122e-08 4.775099e-02 1.549149e-01
#> 2016-02-29 4.778836e-09 2.061693e-08 2.044994e-08 1.263952e-08
#> 2016-03-31 1.052528e-08 1.267888e-01 1.132483e-08 1.213295e-08
#> 2016-04-29 1.166045e-09 2.687081e-01 2.132233e-09 2.830671e-09
#> 2016-05-31 9.352918e-09 4.300170e-01 1.399663e-08 1.208735e-08
#> 2016-06-30 5.077005e-09 4.380619e-01 7.710671e-09 1.406399e-08
#> 2016-07-29 1.037846e-08 4.987723e-01 2.162388e-08 4.136552e-02
#> 2016-08-31 2.017859e-09 5.430357e-01 3.130117e-09 5.883096e-09
```

```
#> 2016-09-30 8.301652e-09 4.377906e-01 2.162225e-08 1.597104e-01
#> 2016-10-31 5.818678e-08 5.312668e-01 2.115042e-01 6.070904e-09
#> 2016-11-30 7.236076e-09 6.461812e-01 3.538187e-01 7.017603e-09
#> 2016-12-30 1.590645e-08 6.751820e-01 2.282747e-01 9.654304e-02
#> 2017-01-31 9.129977e-08 6.241726e-01 3.841161e-03 5.199044e-08
#> 2017-02-28 1.138073e-01 8.861926e-01 5.012663e-09 6.406461e-10
#> 2017-03-31 2.924855e-01 7.075145e-01 8.262744e-09 1.687241e-09
#> 2017-04-28 4.526496e-01 5.473503e-01 1.091751e-08 6.036032e-09
#> 2017-05-31 9.118022e-01 8.815514e-02 1.887434e-07 8.689972e-09
#> 2017-06-30 9.542875e-09 3.355354e-01 5.922701e-09 3.216986e-09
#> 2017-07-31 5.711802e-09 2.784536e-01 5.935666e-09 3.729444e-09
#> 2017-08-30 8.348357e-01 1.647554e-01 1.373168e-08 1.165728e-08
#>                      AET            A           APD           AA
#> 2014-11-12 1.366712e-09 6.382592e-10 1.702570e-09 1.999652e-09
#> 2014-11-28 7.121492e-09 1.623030e-09 4.254742e-09 2.339974e-01
#> 2014-12-31 2.594453e-09 1.159268e-09 2.705667e-09 3.960278e-09
#> 2015-01-30 9.138465e-09 3.697054e-09 9.237314e-09 6.456811e-09
#> 2015-02-27 1.822749e-08 2.641511e-09 1.856625e-08 2.822513e-09
#> 2015-03-31 2.545705e-01 5.733594e-10 1.310623e-08 3.856161e-10
#> 2015-04-30 3.379593e-01 1.961015e-09 3.463277e-09 9.657371e-10
#> 2015-05-29 9.955811e-01 1.296324e-08 1.652837e-08 6.416075e-09
#> 2015-06-30 9.999999e-01 1.827577e-09 3.060772e-09 1.051464e-09
#> 2015-07-31 9.999998e-01 7.376054e-09 1.055100e-08 2.423115e-09
#> 2015-08-31 9.999997e-01 1.689259e-08 3.622694e-08 7.227088e-09
#> 2015-09-30 9.999999e-01 1.056399e-08 1.595614e-08 5.321519e-09
#> 2015-10-30 8.005590e-01 3.731796e-09 4.308600e-09 9.515262e-10
#> 2015-11-30 2.947962e-01 4.247813e-01 1.669783e-08 3.325739e-09
#> 2015-12-31 1.861650e-01 5.912460e-01 1.116364e-08 4.669113e-09
#> 2016-01-29 5.159110e-01 1.849540e-01 9.646892e-02 1.369721e-08
#> 2016-02-29 4.385029e-01 3.956996e-08 5.614970e-01 1.912163e-09
#> 2016-03-31 2.133717e-08 2.777106e-01 5.955005e-01 4.235180e-09
#> 2016-04-29 5.549075e-09 1.639908e-08 7.312919e-01 2.998303e-09
#> 2016-05-31 1.240025e-08 5.699830e-01 1.939440e-08 8.948668e-09
#> 2016-06-30 1.079967e-08 5.619381e-01 1.628094e-08 5.015481e-09
#> 2016-07-29 1.332332e-08 4.598620e-01 2.269626e-08 9.081858e-09
#> 2016-08-31 8.760845e-09 4.569643e-01 1.467821e-08 3.774557e-09
#> 2016-09-30 1.189884e-08 2.849215e-08 4.024989e-01 4.160930e-09
#> 2016-10-31 6.424336e-09 1.421126e-08 2.572289e-01 1.042137e-08
#> 2016-11-30 9.028817e-09 7.314592e-09 8.551342e-09 2.678126e-08
#> 2016-12-30 3.775660e-08 5.216140e-08 8.330368e-08 1.886048e-08
#> 2017-01-31 6.193847e-08 1.711416e-07 4.722014e-08 3.719858e-01
#> 2017-02-28 1.289788e-09 1.028059e-09 7.725830e-10 1.823476e-08
#> 2017-03-31 1.548702e-09 1.966189e-09 1.358074e-09 8.579488e-09
#> 2017-04-28 8.072763e-09 8.565367e-09 6.295319e-09 8.319491e-09
#> 2017-05-31 2.856500e-08 3.553194e-08 1.092773e-08 4.232588e-05
#> 2017-06-30 4.904264e-01 7.762745e-02 2.265164e-09 9.641069e-02
#> 2017-07-31 3.697768e-01 1.015935e-08 2.794494e-09 3.517695e-01
#> 2017-08-30 3.092973e-08 5.104771e-08 6.603226e-09 4.087503e-04
#>                      CF
#> 2014-11-12 1.663224e-09
#> 2014-11-28 4.337510e-09
#> 2014-12-31 2.100266e-09
#> 2015-01-30 1.473420e-08
#> 2015-02-27 1.908806e-07
#> 2015-03-31 1.421552e-01
#> 2015-04-30 1.873428e-01
#> 2015-05-29 4.418719e-03
#> 2015-06-30 7.453695e-08
#> 2015-07-31 1.279126e-08
```

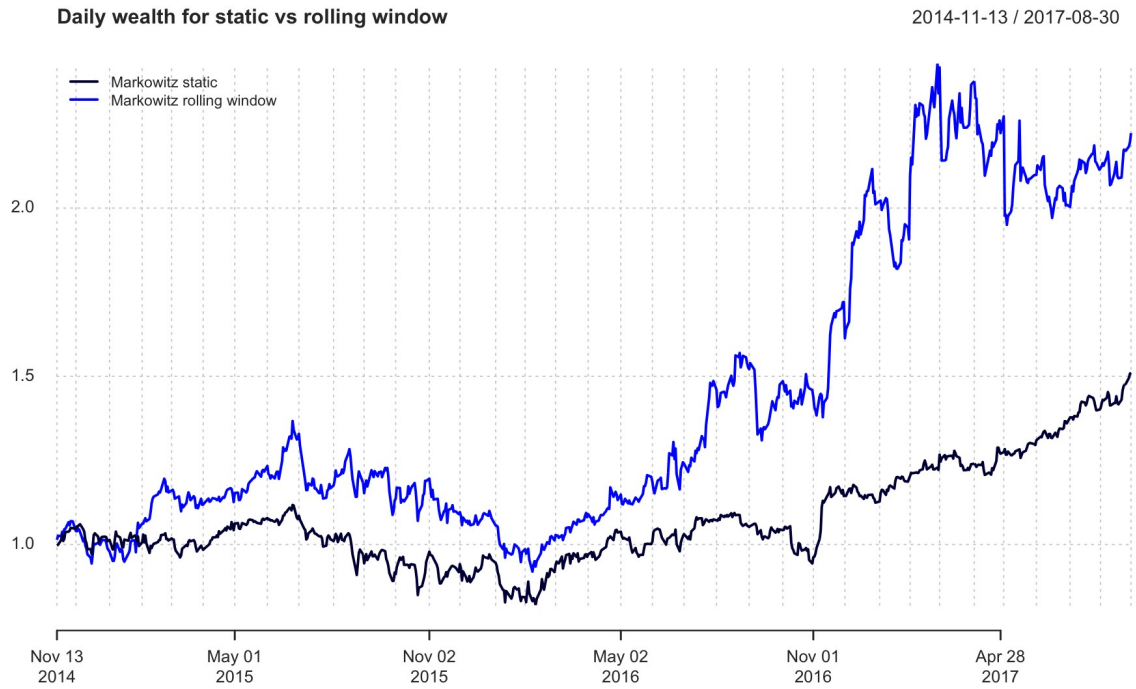
```
#> 2015-08-31 7.977780e-08
#> 2015-09-30 1.645905e-08
#> 2015-10-30 2.524911e-09
#> 2015-11-30 6.603669e-09
#> 2015-12-31 4.059056e-09
#> 2016-01-29 1.273989e-08
#> 2016-02-29 1.557250e-09
#> 2016-03-31 1.644383e-09
#> 2016-04-29 5.153657e-10
#> 2016-05-31 2.687818e-09
#> 2016-06-30 1.808361e-09
#> 2016-07-29 2.963381e-09
#> 2016-08-31 5.057611e-10
#> 2016-09-30 1.112338e-09
#> 2016-10-31 2.376229e-09
#> 2016-11-30 3.631201e-09
#> 2016-12-30 5.418911e-09
#> 2017-01-31 7.385977e-08
#> 2017-02-28 7.689037e-10
#> 2017-03-31 1.270954e-09
#> 2017-04-28 4.947615e-09
#> 2017-05-31 3.384573e-08
#> 2017-06-30 3.168153e-09
#> 2017-07-31 5.767073e-09
#> 2017-08-30 6.115365e-09
chart.StackedBar(w_Markowitz_rolling, main = "Evolution of rolling-window Markowitz por
tfolio", ylab = "w", space=0, border = NA)
```



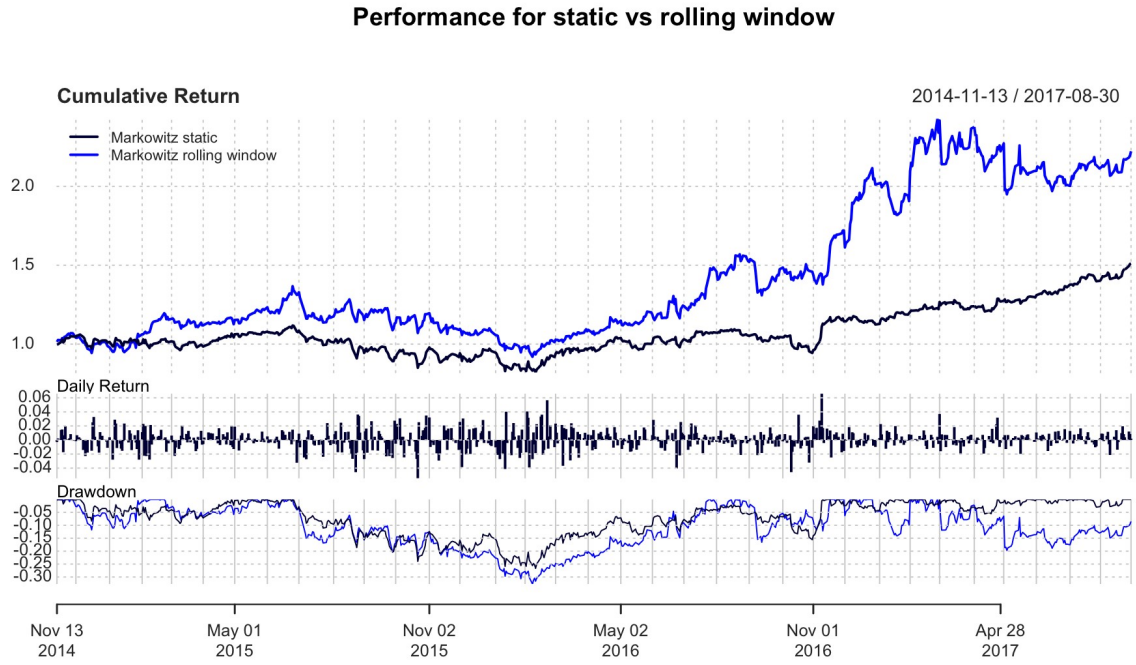

```
# compute portfolio returns
ret_rolling <- Return.portfolio(X_lin_tst, weights = w_Markowitz_rolling)
```

We can now compare the static and rolling window versions:

```
# performance
ret_Markowitz <- cbind(ret_static, ret_rolling)
colnames(ret_Markowitz) <- c("Markowitz static", "Markowitz rolling window")

table.AnnualizedReturns(ret_Markowitz)
#>                          Markowitz static Markowitz rolling window
#> Annualized Return                 0.1592                   0.3310
#> Annualized Std Dev                0.2152                   0.3213
#> Annualized Sharpe (Rf=0%)         0.7397                   1.0302
chart.CumReturns(ret_Markowitz, main = "Daily wealth for static vs rolling window",
                 wealth.index = TRUE, legend.loc = "topleft", colorset = rich6equal)
```




```
charts.PerformanceSummary(ret_Markowitz, main = "Performance for static vs rolling wind
ow",
                          wealth.index = TRUE, colorset = rich6equal)
```



Conclusion

We can conclude with the following points:

- It seems that using linear returns or log-returns does not make any significant difference for daily, weekly, and monthly returns.
- We have backtested a number of different portfolio designs.
- Backtests can be initially done on a single set of training and test data.
- More serious backtesting is done on a rolling-window basis: the package **portfolioBacktest** (https://github.com/dppalomar/portfolioBacktest) makes this very simple.
- However, we have only used one set of market data and a serious backtesting requires multiple data sets.
- The package **PermornaceAnalytics** has the function `Return.portfolio()` that is useful to compute returns of a portfolio without daily rebalancing.