Minimum Variance Portfolio in Action

Patrik Münch (patrik.muench@student.unisg.ch)
Christoph Schenker (christoph.schenker@student.unisg.ch)

May 18, 2020

Abstract

This document presents a code which allows to compare the portfolio performance of several stocks between an equal-weight and a mean-variance optimized portfolio and graphically illustrates the results.

Overview

The minimum variance portfolio is one of the most basic investment theories developed by Harry Markowitz in 1952 (Markowitz, 1952). It states that for each combination of securities, an efficient risk/return relationship ("efficient frontier") exists based on which a rational investor would choose his portfolio weights given his or her individual risk/return preference (Markowitz, 1952, p. 85-91). This ground-breaking work got Harry Markovitz the Nobel Prize in Economics in 1990 and is the base for numerous following models in modern portfolio theory such as the Capital Asset Pricing Model (Sharpe, 1964; *The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel*, 1990).

In this paper, we build on this model to analyze how the portfolio return differs between a standard equal-weight portfolio, i.e. investing the same amount of money into each asset in a portfolio, and a minimum variance optimized portfolio. While the weights of the equal-weight portfolio are constant over time, we recalculate (or rebalance) the mean-variance portfolio every week. We do this in a five-step approach: First, we gather the daily closing price from Thomson Reuters Eikon for nine stocks with a time range from 13.03.1986 until 01.03.2019. Second, we transform them into weekly log returns, which is the most commonly used approach of calculatig returns in

financial research. Based on these numbers, we calculate the covariance matrix of the stock returns in step three. In essence, the covariance matrix summarises the relationship between the returns of each assets in a two-dimensional matrix. This allows us to calculate the respective portfolio weights in step four. Lastly, we calculate the portfolio values and plot their performance and weight development for both equal-weight and minimum variance optimized portfolios.

The expected return for both portfolios is

$$E = \sum_{i=1}^{N} X_i \mu_i$$

Whereas the variance is

$$V = \sum_{i=1}^{N} \sum_{j=1}^{N} \sigma_{ij} X_i X_j$$

With X being the weight of the asset i or j.

In our analysis, the performance differences are based on different X_i's in the strategies. This, because for the equal-weight portfolios the weight of each stock is the same whereas for the minimum variance portfolio the asset weights differ and are proportional to the inverse of the covariance matrix:

$$X = \frac{\sum^{-1} 1}{1' \sum^{-1} 1}$$

With $\sum^{-1}$ being the inverted covariance matrix, 1 being a vector of 1's with a length of N and $1'$ being a transposed vector of 1's.

Having discussed the mathematics of this optimisation approach on a high level, the following paragraphs will now give a deep-dive on the code and explain in a step-by-step approach what each chunk is doing.

Code description

1 R environment preparation

In the first step, we download and attach the respective R packages which are required to run the code. We set up the code so that it checks for missing packages and directly installs them instead of working e.g. with the *require()* function and make the user install the packages manually. The following paragraph shows which code we used to do so:

```
# Install missing packages for the code
list_of_packages <- c("xts", "lubridate", "Quandl", "ggplot2",
    "tidyr", "tidyverse", "scales", "tseries", "reshape2", "fBasics",
    "gridExtra", "rstudioapi", "formatR")
new_packages <- list_of_packages[!(list_of_packages %in% installed.packages()[,
    "Package"])]
if (length(new_packages)) install.packages(new_packages)
# Load and attach packages
library(xts)
library(lubridate)
library(Quandl)
library(ggplot2)
library(tidyr)
library(tidyverse)
library(scales)
library(tseries)
library(reshape2)
library(fBasics)
library(gridExtra)
library(rstudioapi)
library(formatR)
```

Then, we remove all current variables in the R environment to prevent any interferences with our

code:

```
rm(list = ls())
```

In a final step to prepare the environment, we let the user choose the working directory in which he saved the input csv file. Through this, we ensure that the code finds the required input sources and also saves the output into the same folder where the input is at a later stage:

```
# Make user select the directory with the input csv document
WorkingDir <- NULL
while(is.null(WorkingDir)){
  WorkingDir <- selectDirectory(
    caption = "Please select the directory in which the input file is saved",
    label = "Here is the input folder",
    path = getActiveProject()
  )
}
setwd(WorkingDir)
```

Now, we are ready to load and prepare the data as we have the required packages and working directory for the input file ready.

2 Data preparation

In the next step, we prepare our raw data on which the analysis will be based. First, we define different parameters which are used several times in the code to increase consistency:

```
# Set parameters
reb_int      <- "weekly"
source       <- "csv"
export_plot  <- TRUE
```

Then we define the stock tickers which we consider in our analysis. Given that we downloaded data for 23 different stocks from Eikon, we can change the tickers to any of these 23 (see the csv

input file for all tickers) to run the subsequent analysis:

```
# Obtain stock data
ticker_csv   <- c("SAMSUNG",
                  "SASOL",
                  "DYCOM",
                  "NESTLE",
                  "NOVARTIS",
                  "APPLE",
                  "ROYALD",
                  "NAVISTAR",
                  "UBS")
```

Next, we read in the source data through creating the function *source_data*. This initial data consists of a csv file which is in the folder the user chose in the beginning. The data is then converted into the *xts* time series format and into weekly data points. Also we set the rebalancing interval and pick the stocks based on the *ticker_csv* variable from the step before:

```
# Define function to source data
source_data <- function(source) {
    toDate <- function(x) as.Date(x, origin = "1986-03-86", format = "%d.%m.%y")
    Time_series_zoo <- read.zoo("RS_Time series_v3.csv", header = TRUE, sep = ","
                                , FUN = toDate)
    Time_series_xts <- as.xts(Time_series_zoo)

    # Convert daily to weekly data
    Time_series_weekly_xts <- to.weekly(Time_series_xts, OHLC = FALSE)

    # Set rebalancing interval and pick stocks
    stock_prices <- Time_series_weekly_xts[, ticker_csv]
    return(stock_prices)
    }
```

```
stock_prices <- source_data(source)
```

The following output depicts the head of this time series object. As we can see, the data is on a weekly basis and stock price information is included for the nine stocks in the variable *ticker_csv*:

```
##            SAMSUNG SASOL  DYCOM NESTLE NOVARTIS  APPLE ROYALD NAVISTAR   UBS
## 1986-03-14    0.15  3.75 2.8574  2.357    2.164 0.4665   4.19  91.2500 6.613
## 1986-03-21    0.15  3.62 2.7516  2.397    2.194 0.4933   4.51  86.2500 6.684
## 1986-03-28    0.17  3.42 2.6810  2.307    2.157 0.5045   4.49 110.0001 6.426
## 1986-04-04    0.17  3.19 2.6810  2.346    2.138 0.4777   4.30 102.5001 6.510
## 1986-04-11    0.18  3.22 2.7516  2.419    2.216 0.4821   4.35 108.7501 6.853
## 1986-04-18    0.19  3.18 2.7516  2.518    2.296 0.5313   4.45 116.2501 7.152
```

We are now going to convert this data into log returns:

```
# Define function for calculating log returns
calc_log_returns <- function(stock_prices) {
  log_returns <- stock_prices
  for (i in 1:ncol(stock_prices)) {
    log_returns[, i] <- diff(log(stock_prices[, i]))
  }
  return(log_returns)
}
# Calculate log returns
log_returns  <- calc_log_returns(stock_prices)
log_ret_cl   <- log_returns["1998-01-01/2018-12-31"]
nr_cols      <- ncol(log_returns)
ticker_sort <- sort(ticker_csv)
```

The following output depicts the data conversion:

```
##               SAMSUNG        SASOL       DYCOM      NESTLE     NOVARTIS
## 1986-03-14         NA           NA          NA          NA           NA
```

```
## 1986-03-21 0.00000000 -0.035281814 -0.03772956  0.01682833  0.013768001
## 1986-03-28 0.12516314 -0.056833475 -0.02599270 -0.03826998 -0.017007994
## 1986-04-04 0.00000000 -0.069619634  0.00000000  0.01676377 -0.008847555
## 1986-04-11 0.05715841  0.009360443  0.02599270  0.03064248  0.035832956
## 1986-04-18 0.05406722 -0.012500163  0.00000000  0.04011070  0.035464710
##                    APPLE       ROYALD    NAVISTAR         UBS
## 1986-03-14            NA           NA          NA          NA
## 1986-03-21  0.055859488  0.073596420 -0.05635294  0.01067920
## 1986-03-28  0.022450332 -0.004444452  0.24323122 -0.03936435
## 1986-04-04 -0.054584919 -0.043237679 -0.07061750  0.01298720
## 1986-04-11  0.009168641  0.011560822  0.05918882  0.05134706
## 1986-04-18  0.097175272  0.022728251  0.06669132  0.04270553
```

Given that we now have the basic data for our analysis we are going to define the date inputs for the time window we will base our analysis on. In this case, we decided to choose January 1, 2012 as starting date with the portfolio being active for seven years, ten calibration years and 52 weeks per year. The calibration years refers to the time horizon of historical data which the model uses at each rebalancing point, thus every week, to recalculate the new weights of the mean variance portfolio. The analysis output will be saved in the respective directory path of the variable *folder*:

```
# Input for the dates
start_date   <- "2012-01-01"
port_span    <- 7 # in years
cali_years   <- 10 # in years
wpy          <- 52
folder       <- paste("Output/", year(start_date), "-", year(start_date) +
                        port_span - 1,
                      "_", cali_years, "cali_", reb_int, sep = "")
```

Based on these parameters, we define a function to assign the appropriate dates in the correct format to our data set. In doing so, we also determine the correct dates for the applied calibration years. This is crucial as the mechanics of our model rely on a consistent date format among the

calibration and the portfolio span.

```
# Define function for calculating dates: This function also checks that port_start
#and cali_start have the same length and cuts the longer if not
calc_dates <- function(stock_prices, start_date, port_span, cali_years, reb_int) {
    index <- as.character(index(stock_prices))
    port_start_date <- as.Date(index[year(index) == year(start_date)][1])
    port_start_row  <- which(index == as.character(port_start_date))
    port_end_row    <- port_start_row + port_span * wpy -1
    port_end_date   <- as.Date(index[port_end_row])
    port_start      <- as.Date(index[port_start_row:port_end_row])
    port_rows       <- match(as.character(port_start), index)
    cali_start_row  <- port_start_row - wpy * cali_years
    cali_start_date <- as.Date(index[cali_start_row])
    cali_end_row    <- cali_start_row + port_span * wpy - 1
    cali_end_date   <- as.Date(index[cali_end_row])
    cali_start      <- as.Date(index[cali_start_row:cali_end_row])
    cali_rows       <- match(as.character(cali_start), index)
    all_start       <- as.Date(index[cali_start_row:port_end_row])
  if (length(port_start) == length(cali_start)) {
    return(list(all_start, port_start, cali_start, port_rows, cali_rows))
  } else {
    v <- data.frame(cali_start = length(cali_start), port_start =
                      length(port_start))
    l <- abs(length(cali_start) - length(port_start))
    max <- apply(v, MARGIN = 1, max)
    if (max == length(cali_start)) {
      cali_start <- cali_start[-((length(cali_start) - l +
                                    1):(length(cali_start)))]
    } else {
```

```
      port_start <- port_start[-((length(port_start) - l + 1
                                  ):(length(port_start)))]
    }
    return(list(all_start, port_start, cali_start, port_rows, cali_rows))
  }
}
```

Through these steps, we then assign the relevant dates and subsets of the stock returns into the respective variables. Through this, we align the overall data with *port_data* and *cali_data* which are used as the dates to calculate and rebalance our portfolio in the following steps.

```
# Calculate dates
dates <- calc_dates(stock_prices, start_date, port_span, cali_years, reb_int)
all_start    <- dates[[1]]
port_start   <- dates[[2]]
cali_start   <- dates[[3]]
port_rows    <- dates[[4]]
cali_rows    <- dates[[5]]
# Calculates data subsets according to parameters
port_data    <- log_returns[port_start]
cali_data    <- log_returns[cali_start]
all_data     <- log_returns[all_start]
```

After this step, we now have prepared the time series adequately to start our mean-variance analysis.

3 Covariance matrix calculation

In the third step, we calculate the covariance matrices which we need to calculate the weights of the minimum variance portfolio. The respective formulas are described in the *Overview* chapter of this document:

```
# Define function to calculate the covariance matrix
#(required as input for portfolio optimization)
calc_all_cov_forecast <- function (log_returns, port_start, cali_rows, port_rows) {
  all_forecast <- list()
  all_cors <- list()
  for (i in 1:length(port_start)) {
    dataset <- log_returns[cali_rows[i]:port_rows[i], ]
    all_forecast[[i]] <- cov(dataset)
    all_cors[[i]] <- cor(dataset)
  }
  names(all_forecast) <- port_start
  return(list(all_forecast, all_cors))
}
# Calc covariance matrices
meanvar_output        <- calc_all_cov_forecast(log_returns, port_start, cali_rows,
                                               port_rows)
all_forecast_meanvar  <- meanvar_output[[1]]
fore_cor_meanvar      <- meanvar_output[[2]]
```

4 Portfolio weights calculation

Given that we now have the coviance matrices, we can calculate the portfolio weights. First, we program the functions to calculate the minimum variance portfolio weights. The function *calc_pf_weights* allows to calulate the optimal weights for a portfolio at a given point in time. This is then used in the following function, *calc_all_pf_weights*, which calculates all portfolio weights over the entire analysed time horizon. In our case this allows to compute the portfolio weights at every rebalancing point (thus each week):

```
# Define functions to calculate (all) portfolio weights based on goal of
#mean-variance
calc_pf_weights <- function(a_cov_mat) {
```

```
  ones <- c(rep(1, nrow(a_cov_mat)))
  weights <- (solve(a_cov_mat) %*% ones)
  weights <- weights / sum(weights)
  return(weights)
}
calc_all_pf_weights <- function(all_cov_matrices) {
  all_pf_weights <- list()
  for (i in 1:length(all_cov_matrices)) {
    all_pf_weights[[i]] <- calc_pf_weights(all_cov_matrices[[i]])
  }
  names(all_pf_weights) <- port_start
  return(all_pf_weights)
}
```

The same approach is taken to return the weights for the equal-weight portfolio with the difference being that now the weights are not based on the stock's variance but simply calculated through 1/n:

```
# Define functions to "calculate" equal portfolio weights
calc_pf_weights_eql <- function(a_cov_mat) {
  ones <- c(rep(1, nrow(a_cov_mat)))
  weights <- ((1 / ncol(log_returns)) %*% ones)
  weights <- weights / sum(weights)
  return(weights)
}
calc_all_pf_weights_eql <- function(all_cov_matrices) {
  all_pf_weights_eql <- list()
  for (i in 1:length(all_cov_matrices)) {
    all_pf_weights_eql[[i]] <- calc_pf_weights_eql(all_cov_matrices[[i]])
  }
  names(all_pf_weights_eql) <- port_start
```

```
  return(all_pf_weights_eql)
}
```

After setting up the functions, we can now calculate the portfolio weights:

```
# Calculate portfolio weights of both portfolios
pf_weights_meanvar    <- calc_all_pf_weights(all_forecast_meanvar)
pf_weights_eql    <- calc_all_pf_weights_eql(all_forecast_meanvar)
```

The following output depicts the first few set of weights of each portfolio (negative values are short positions):

```
head(pf_weights_meanvar)
```

```
## $`2012-01-06`
##                 [,1]
## SAMSUNG   0.10513964
## SASOL    -0.06094812
## DYCOM     0.02280808
## NESTLE    0.50989080
## NOVARTIS  0.30711882
## APPLE     0.08959549
## ROYALD    0.10914672
## NAVISTAR -0.03189845
## UBS      -0.05085299
##
## $`2012-01-13`
##                 [,1]
## SAMSUNG   0.10540091
## SASOL    -0.06003538
## DYCOM     0.02261647
## NESTLE    0.51218359
```

```
## NOVARTIS  0.30605063
## APPLE     0.09013277
## ROYALD    0.10463055
## NAVISTAR -0.03148853
## UBS      -0.04949101
##
## $`2012-01-20`
##                 [,1]
## SAMSUNG   0.10578602
## SASOL    -0.06103156
## DYCOM     0.02269727
## NESTLE    0.51098134
## NOVARTIS  0.30649265
## APPLE     0.09044317
## ROYALD    0.10691826
## NAVISTAR -0.03076471
## UBS      -0.05152243
##
## $`2012-01-27`
##                 [,1]
## SAMSUNG   0.10595357
## SASOL    -0.06136257
## DYCOM     0.02261009
## NESTLE    0.51103952
## NOVARTIS  0.30642283
## APPLE     0.09041677
## ROYALD    0.10691721
## NAVISTAR -0.03064744
## UBS      -0.05134999
##
```

```
## $`2012-02-03`
##                 [,1]
## SAMSUNG   0.10683979
## SASOL    -0.06136074
## DYCOM     0.02211408
## NESTLE    0.50975672
## NOVARTIS  0.31011055
## APPLE     0.08983278
## ROYALD    0.10459413
## NAVISTAR -0.03015005
## UBS      -0.05173726
##
## $`2012-02-10`
##                 [,1]
## SAMSUNG   0.10768381
## SASOL    -0.06089888
## DYCOM     0.02215413
## NESTLE    0.50870880
## NOVARTIS  0.31076984
## APPLE     0.08860230
## ROYALD    0.10412197
## NAVISTAR -0.03054225
## UBS      -0.05059972
```

```
head(pf_weights_eql)
```

```
## $`2012-01-06`
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
##           [,8]      [,9]
## [1,] 0.1111111 0.1111111
```

```
##
## $`2012-01-13`
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
##           [,8]      [,9]
## [1,] 0.1111111 0.1111111
##
## $`2012-01-20`
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
##           [,8]      [,9]
## [1,] 0.1111111 0.1111111
##
## $`2012-01-27`
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
##           [,8]      [,9]
## [1,] 0.1111111 0.1111111
##
## $`2012-02-03`
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
##           [,8]      [,9]
## [1,] 0.1111111 0.1111111
##
## $`2012-02-10`
##           [,1]      [,2]      [,3]      [,4]      [,5]      [,6]      [,7]
## [1,] 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111 0.1111111
##           [,8]      [,9]
## [1,] 0.1111111 0.1111111
```

One can easily observe that the minimum variance portfolio constantly adapts its weights, while the equal-weight portfolio naturally always uses the same weights. Based on these values, we are now going to calculate the performance and thus the returns for each portfolio.

5 Portfolio performance calculation

The function *calc_port_returns* calculates the respective portfolio returns accounting for the individual stock returns, portfolio weights and the changing weights for the mean- variance portfolio as defined by the rebalancing interval:

```
# Define function to calc portfolio returns and give the output of a return time
#series
calc_port_returns <- function (port_data, pf_weights, reb_int) {
  port_ret <- port_data[, -c(1:ncol(port_data))]
  for (i in 1:nrow(port_ret)) {
    weights <- unlist(pf_weights[[i]])
    port_ret[i] <- as.numeric(as.numeric(port_data[i, ]) %*% as.numeric(weights))
  }
  return(port_ret)
}
```

In the next step, we calculate the portfolio return for the two portfolios. First, we calculate the individual returns within the portfolios.

```
# Calculate all portfolio returns
port_ret_meanvar      <- calc_port_returns(port_data, pf_weights_meanvar, reb_int)
port_ret_eql       <- calc_port_returns(port_data, pf_weights_eql, reb_int)
```

Then we calculate the overall portfolio value based on an initial index (*start_value*) of 100 through multiplying the respective current portfolio value with the portfolio return of the given period. Also we calculate the performance difference between the two portfolios in a separate column (*port_all$DIFF*) of the data frame *port_all*:

```r
# Define function to calc portfolio returns and give the output of a return time
#series
calc_port_value <- function (port_data, ret_series_eql, ret_series_meanvar,
                             start_value) {
  port_all <- port_data[, -c(1:ncol(port_data))]
  port_all$EQL <- start_value
  port_all$MEANVAR <- start_value
  for (i in 2:nrow(port_all)) {
    port_all$EQL[i] <- as.numeric(port_all$EQL[i - 1]) *
      as.numeric(exp(ret_series_eql[i]))
    port_all$MEANVAR[i] <- as.numeric(port_all$MEANVAR[i - 1]) *
      as.numeric(exp(ret_series_meanvar[i]))
  }
  return (port_all)
}
# Calculate the portfolio value based on the returns
port_all           <- calc_port_value(port_data, port_ret_eql, port_ret_meanvar, 100)
port_all$DIFF      <- port_all$EQL - port_all$MEANVAR
```

Based on these values, we create the outputs in our sixth and last step.

6 Outputs

As first output, we generate a csv file containing the annualised mean and annualised standard deviation (the most basic measure of risk) of each portfolio. Hereby, *ann_fac* is the annualisation factor based on the 52 weeks of the year specified at the beginning. We save the output in a new output folder of the chosen work directory:

```r
# Table output
ann_fac <- wpy
m <- matrix(c(mean(port_ret_eql) * ann_fac,
```

```
              mean(port_ret_meanvar) * ann_fac,
              sd(port_ret_eql) * sqrt(ann_fac),
              sd(port_ret_meanvar) * sqrt(ann_fac)),
            nrow = 2, byrow = TRUE)
colnames(m)        <- c("EQL", "MEANVAR")
row.names(m)       <- c("Annualised mean", "Annualised standard deviation")
if(dir.exists(folder) == FALSE) (dir.create(file.path(folder), recursive = TRUE))
write.csv(m, file = paste(folder, "/stats.csv", sep = ""))
```

The following output depicts the output result for both strategies:

```
##                                       EQL    MEANVAR
## Annualised mean                0.03155214 0.07430193
## Annualised standard deviation 0.17562856 0.12569149
```

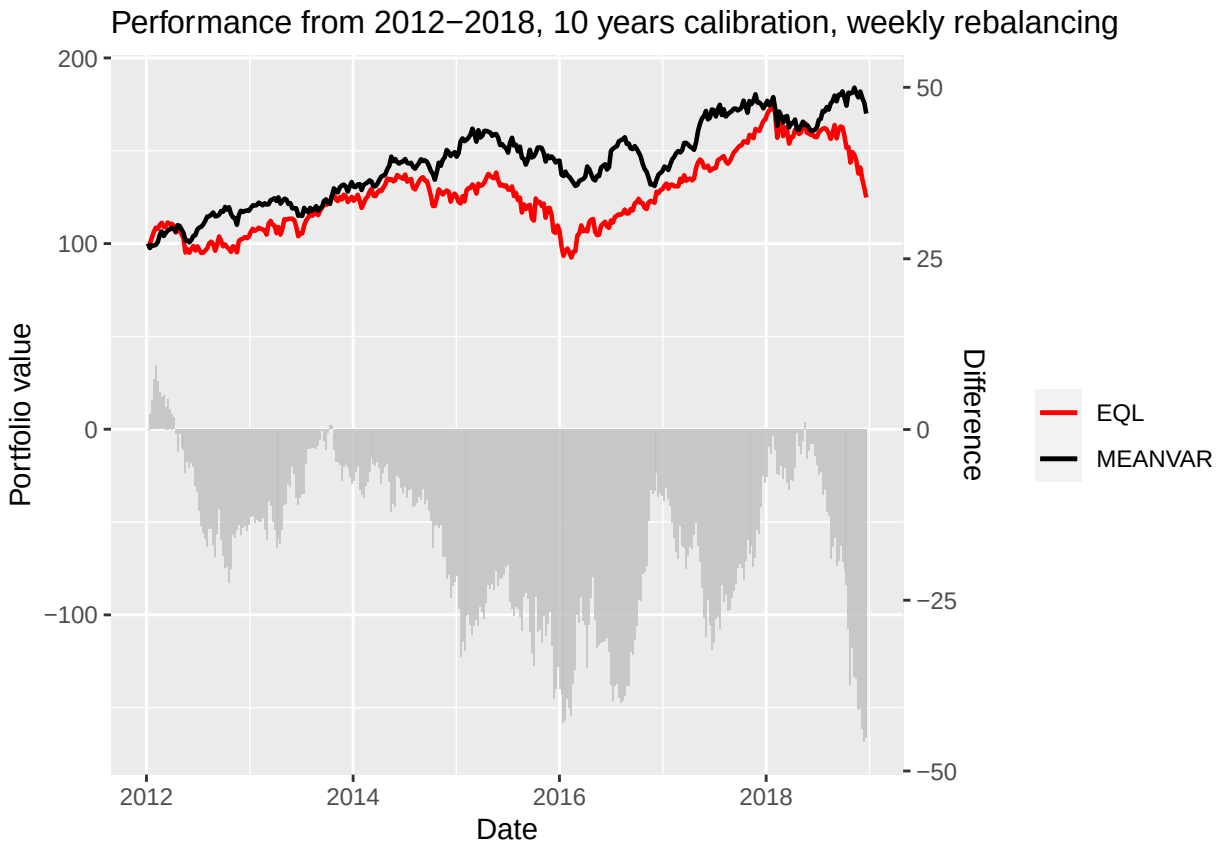
We can see that for the minimum variance strategy, the annualised mean is significantly higher with a simultaneously lower standard deviation, making the strategy the more successful one for the analysed stocks and time period.

In a next step, we plot the portfolio performance development and output it in pdf format. The output is saved in the new output folder we created above together with the table output. To create the plot, we read our data into a new data frame *p* to concentrate the data into one variable which eases code reading. Then we set up the plot and define the several axes and bars:

```
# Plot development of both portfolios (indexed at 100)
p <- data.frame(Date = index(port_all), EQL = port_all$EQL,
                MEANVAR = port_all$MEANVAR, DIFF = port_all$DIFF)
if (export_plot == TRUE) {pdf(file = paste(folder, "/port.pdf", sep = ""))}
ggplot(p) +
  geom_bar(aes(x = Date, y = DIFF * max(max(p$EQL), max(p$MEANVAR)) / 50),
           stat = "identity", fill = "gray", alpha = 0.8) +
  geom_line(aes(x = Date, y = EQL, color = "EQL"), size = 0.8) +
  geom_line(aes(x = Date, y = MEANVAR, color = "MEANVAR"), size = 0.8) +
```

```
  scale_y_continuous(name = "Portfolio value",
    sec.axis = sec_axis(~ . * 50 / max(max(p$EQL), max(p$MEANVAR)),
                        name = "Difference"),
    limits = c(min(p$DIFF) * max(max(p$EQL), max(p$MEANVAR)) /  50,
               max(max(p$EQL), max(p$MEANVAR)))) +
  scale_colour_manual(name = '', values = c('MEANVAR' = 'black', 'EQL' = 'red')) +
  labs(title = paste("Performance from ", year(start_date), "-",
                     year(start_date) + port_span - 1, ", ",
                     cali_years, " years calibration, ", reb_int,
                     " rebalancing", sep = "")) +
  theme(plot.title = element_text(size = 12))
dev.off()
```

The resulting and as pdf saved chart looks as follows for the portfolio development. One can clearly see that the dynamic approach to portfolio computation has performed better over the selected time horizon:



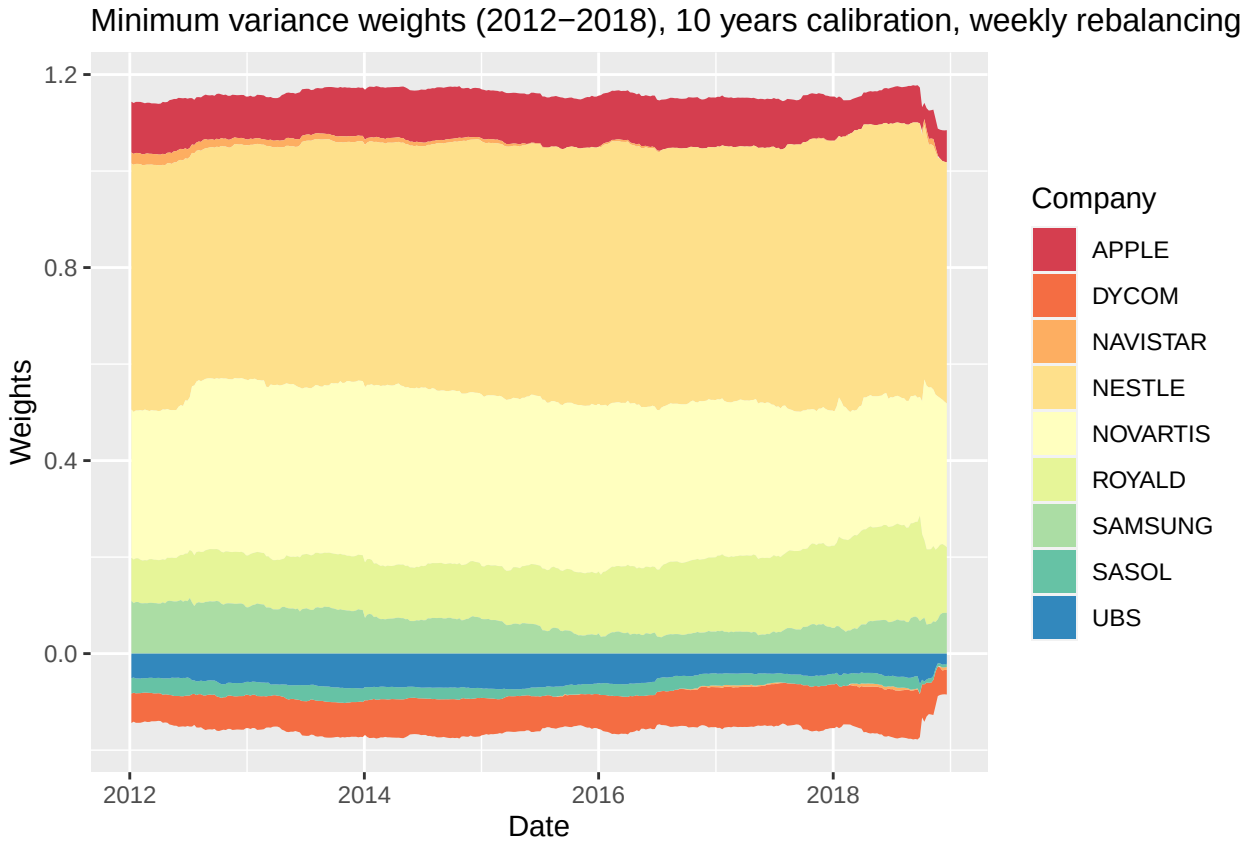
Last but not least, we plot the weight development of the mean-variance portfolio to get a better understanding on how dynamic the model has adjusted its portfolio weights over the time horizon:

```
# Plot weight development of the minimum variance portfolios
Date <- as.Date(names(all_forecast_meanvar))
if (export_plot == TRUE) {pdf(file = paste(folder, "/weights_meanvar.pdf",
                                           sep = ""))}
weight_mat_meanvar <- data.frame(matrix(NA, nrow = length(pf_weights_meanvar),
                                        ncol = length(ticker_sort)))
for (i in 1:nrow(weight_mat_meanvar)) {
  weight_mat_meanvar[i, ] <- unlist(pf_weights_meanvar[[i]])
}
colnames(weight_mat_meanvar) <- ticker_sort
weight_mat_meanvar$Date <- Date
weight_mat_meanvar_long <- gather(weight_mat_meanvar, Company, Weights,
```

```
                                ticker_sort[1]:ticker_sort[nr_cols],
                                factor_key = T)
ggplot(weight_mat_meanvar_long, aes(x = Date, y = Weights)) +
  geom_area(aes(fill = Company), position = 'stack') +
  scale_fill_brewer(palette = "Spectral") +
  labs(title = paste("Minimum variance weights from ", year(start_date), "-",
                     year(start_date) + port_span - 1, ", ",
                     cali_years, " years calibration, ", reb_int,
                     " rebalancing", sep = "")) +
  theme(plot.title = element_text(size = 12))
dev.off()
if (export_plot == TRUE) {pdf(file = paste(folder, "/weights_meanvar.pdf",
                                           sep = ""))}
weight_mat_meanvar <- data.frame(matrix(NA, nrow = length(pf_weights_meanvar),
                                        ncol = length(ticker_sort)))
for (i in 1:nrow(weight_mat_meanvar)) {
  weight_mat_meanvar[i, ] <- unlist(pf_weights_meanvar[[i]])
}
colnames(weight_mat_meanvar) <- ticker_sort
weight_mat_meanvar$Date <- Date
weight_mat_meanvar_long <- gather(weight_mat_meanvar, Company, Weights,
                                  ticker_sort[1]:ticker_sort[nr_cols],
                                  factor_key = T)
ggplot(weight_mat_meanvar_long, aes(x = Date, y = Weights)) +
  geom_area(aes(fill = Company), position = 'stack') +
  scale_fill_brewer(palette = "Spectral") +
  labs(title = paste("Minimum variance weights (", year(start_date), "-",
                     year(start_date) + port_span - 1, "), ",
                     cali_years, " years calibration, ", reb_int,
```

```
                    " rebalancing", sep = "")) +
  theme(plot.title = element_text(size = 12))
dev.off()
```

The resulting chart looks as follows. While changes in weights can be observed, this chart makes clear that the weights for the individual stocks have changed only gradually throughout the seven years. Probably, the inclusion of more volatile stocks in our portfolio would lead to a different result here with more substantial changes in weightings over time.




```
## null device
##           1
```

With this chart, we reached the end of the script. Of course, the user can play around with the different parameters (such as time period, frequency of rebalancing, stocks in the portfolio) to get different results and complement the current charts with different outputs. By doing so, one could further investigate the established thesis of whether the mean-variance approach to portfolio optimisation is truly superior to an equal-weights approach.

References

Markowitz, Harry M. 1952. "Portfolio Selection." *Journal of Finance* 7:77–91.

Sharpe, William F. 1964. "Capital asset prices: A theory of market equilibrium under conditions of risk." *The journal of finance* 19(3):425–442.

The Sveriges Riksbank Prize in Economic Sciences in Memory of Alfred Nobel. 1990.
URL: *https://www.nobelprize.org/prizes/economic-sciences/1990/summary/*