When you write code in R, you might face the question: „Is there a faster way to do this?“. Over the years I worked at STATWORX, I have done a lot of little benchmarks to find an answer to this kind of question. Often, I just did a quick check to see if there is any time difference between two methods, used the faster one and moved on. Of course, I forgot about my tests over time and may have wondered twice about the same problem. To break this vicious circle, I created an overview of all the benchmarks I have done so far with the possibility to add more in the future. This overview can be found on [Github](https://github.com/Dschaykib/benchmarks).

**Creating an overview of all results**

The overview needed to cater to multiple purposes:

* function as a quick lookup table (which is the fastest way to do a specific task)
* show the alternatives that were tested
* give an idea of what was tested

Since the tested functions are often not that complicated (e.g. range(x) vs max(x) - min(x)), the benchmarks I did so far mostly had two varying parameters (e.g., the size or the number of different values). After some feedback from two of my colleagues, I settled for this table:

| **DATE** | **TEST** | **COMMENT** | **BEST** | **TIME\_FACTOR** | **BEST\_RUNS** | **DETAILS** | **DURATION** |
| --- | --- | --- | --- | --- | --- | --- | --- |
| 2019-11-29 08:53:33 | Accsess a colum in a data frame, table or tibble. | varying size of data | $ tbl | 66.9% | 4/4 | [link](https://github.com/Dschaykib/benchmarks/tree/master/benchmarks/column_access) | 00:00:06 |
| 2019-11-29 08:53:36 | assign with <- or = | varying size of vector | equal sign | 27.4% | 6/6 | [link](https://github.com/Dschaykib/benchmarks/tree/master/benchmarks/assignment) | 00:00:01 |

Since this is a work in progress, there is a good chance the format will change again in the future. But for now, this is shown in the table:

* The **DATE** of the last time, the benchmark run.
* A short description **TEST** of the benchmark.
* In the **COMMENTS** I tried to give a hint of what the setups looked like.
* The **BEST** option out of all tested alternatives compared by their mean time.
* The **TIME\_FACTOR** presents the mean time that can be saved with the best option compared with the mean of the alternatives over all grid setups. **Note:** The time factor can be negative if the best option is not the best in the cases where it takes more time. For these cases, have a look at the details and dependencies of the grid parameters.
* **BEST\_RUNS** is the number of cases were **BEST** solution was actually the best one in relation of all different varying setups that were used (e.g. sample size).
* **DURATION** is the time the whole benchmark with all setups took.

**Making the benchmark setup multi-usable**

As I said before, I planned to make the overview extendable for new benchmarks in the future. Therefore, I created some helper functions and templates to make it easier to include new benchmarks. The main parts for this were:

* a template folder and script for new benchmarks
* a function that saves the result in my desired output
* a function that creates the overview by reading in all existing results
* a script that runs all benchmarks.

For adding a new benchmark, I have to copy the template folder and include the new setup I want to test. The save\_benchmark() function will create the same output as for the previous benchmarks and the update\_bench\_overview() function will add it to the overview.

The main issue is the visualization of different grid parameters and their results. The good thing is that if I get an idea on how to improve this visualization, I could add it to save\_benchmark() and rerun all benchmarks with the run\_all\_bench.R script. At the moment, a plot for each grid parameter is created, which indicates how the change influenced the timing. Also, the summaries for each run are shown, so one can see what exactly is going on.

**How to set up a new benchmark**

The template for further benchmarks has different sections that can be easily adjusted. Since this is a work in progress, it might change in the future. So if you have any good ideas or think I missed something, let me know and raise an issue on [Github](https://github.com/Dschaykib/benchmarks/issues).

**It all starts with settings**

There are three libraries I need for my functions to run. If the next benchmark needs other packages, I can add them here.

# these are needed

library(microbenchmark)

library(helfRlein)

library(data.table)

source("functions/save\_benchmark.R")

# add more here

The next step is to describe the benchmark. Where are the results saved? What is the benchmark all about? What parameters are changing? All this information is later used to create the plots and tables to make it more understandable.

# test setup --------------------------------------------------------------

# folder for results

folder <- "benchmarks/00\_template\_folder/"

# test description

description <- "A short description of what is tested."

# number of repetitions

reps <- 100L

comments <- "what parameters changed"

start\_time <- Sys.time()

**The more parameters, the merrier**

How valid are the benchmark results? The more different settings it was tested in, the better the generalization. Is there maybe even a dependency, which is the best alternative? That can all be set up in this section, where you can define the different grid settings. I’d advise you to use variable names that can easily be understood, e.g., number\_of\_rows, unique\_values, or sample\_size. These names are also used in the plots at the end – so choose wisely!

# grid setup --------------------------------------------------------------

# if there are different values to test

grid <- as.data.table(expand.grid(

param\_1 = 10^c(2:3),

param\_2 = c(5,10,20)))

result\_list <- as.list(rep(NA, dim(grid)[1]))

best\_list <- as.list(rep(NA, dim(grid)[1]))

**The benchmark core**

Looping over all grid settings, creating the starting values for each run, and adding all alternatives functions – this is the main part of the function: the benchmark itself.

for (i in c(1:nrow(grid))) {

# i <- 1

i\_param\_1 <- grid[i, param\_1]

i\_param\_2 <- grid[i, param\_2]

# use grid parameters to define tested setup

x <- rnorm(n = i\_param\_1, mean = i\_param\_2)

tmp <- microbenchmark(

"Alternative 1" = mean(x),

"Alternative 2" = sum(x) / length(x),

times = reps,

control = list(warmup = 10L),

unit = "ms")

#tmp <- data.table(summary(tmp), i = grid[i, ])

result\_list[[i]] <- tmp

# select best by mean

tmp\_sum <- summary(tmp)

best\_list[[i]] <- as.character(tmp\_sum$expr[tmp\_sum$mean == min(tmp\_sum$mean)])

}

**All that is not saved will be lost**

During all the previous steps, the intermediate results are stored in lists, which are the input values for the save\_benchmark() function. As mentioned before, it creates tables for each benchmark run and plots with an overview of the effects of each grid parameter. Lastly, it updates the main README file with the newest results.

## saving all data

save\_benchmark(result\_list = result\_list,

best\_list = best\_list,

folder = folder,

start\_time = start\_time,

description = description,

grid = grid,

reps = reps,

comments = comments)

**How do the results look like**

After running the benchmark, a new README file is automatically created. This file contains an overview of the tested alternatives (as you named them), the used grid parameters, plots with the impact of these grid parameters, and the tabled summary of every single result.

For example, here, you can see that the number of unique values has a positive effect (faster) on the time it takes to filter, but the number of rows has a negative impact (slower).

If you are interested in not only the overview but the actual data, have a look at result\_list.rds. This list contains all results of microbenckmark() for each grid combination.

The last two created files are last\_result.rds and log\_result.txt. The first is used to create the current overall README.md, and the second is just a logfile with all previous results.