**When Hadley Wickham published** [**dtplyr 1.0.0**](https://www.tidyverse.org/blog/2019/11/dtplyr-1-0-0/) **last week, I was super excited. Moreover, I wanted to know, if it really is a useful package. Therefore I performed some tiny research being presented in this blog entry.**

Ok. So *dtplyr* gives you the speed of *data.table* with the nice API of *dplyr*. This is the main feature. You will see the this in the following code. The speedup of *dtplyr* should allow converting variables in large scale data sets by simple calls. So I first constructed an example call which is partly taken from Hadley Wickham`s [blog entry](https://www.tidyverse.org/blog/2019/11/dtplyr-1-0-0/):

n <- 100

# Create a word vector

words <- vapply(1:15, FUN = function(x) {

paste(sample(LETTERS, 15, TRUE), collapse = "")

}, FUN.VALUE = character(1))

# Create an example dataset

df <- data.frame(

word = sample(words, n, TRUE),

letter = sample(LETTERS, n, TRUE),

id = 1:n,

numeric\_int = sample(1:100, n, TRUE),

numeric\_double = sample(seq(from = 0, to = 2, by = 1/(10^6)), n, TRUE),

missing = sample(c(NA, 1:100), n, TRUE)

)

# Convert the numeric values

df %>%

filter(![is.na](http://is.na)(missing)) %>%

group\_by(letter) %>%

summarise(n = n(), delay = mean(numeric\_int))

The call shown is the one used within *dplyr.* In *dtplyr* a lazy evaluation term needs to be added. At the end of the *dplyr* call a %>% as\_tibble() is needed to start the evaluation. In *data.table* the call looks totally different.

# dtplyr call

ds\_lazy <- lazy\_dt(ds)

ds\_lazy %>% fil.... %>%

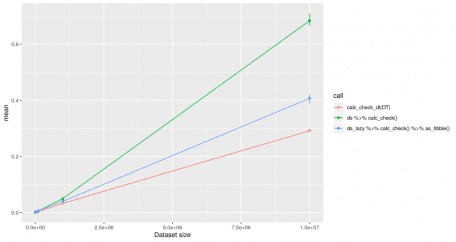
as\_tibble()

DT <- as.data.table(ds)

# data.table call

DT\_int[, .(.N, mean(numeric\_int[![is.na](http://is.na)(missing)])), by = letter]

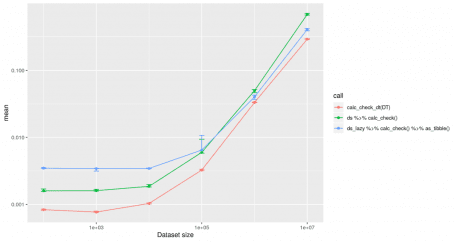
To evaluate the execution I created testing datasets of different sizes. The smallest dataset will contain 100 rows. The largest dataset 10E7 rows. Afterward I ran the data conversion operations and logged the time of execution using the *bench* package and 1000 iterations. All calculations were performed on an [AWS t3.2xlarge](https://aws.amazon.com/ec2/instance-types/t3/) machine hosting the [*rocker/tidyverse*](https://hub.docker.com/r/rocker/tidyverse) docker image. This is the result of the calculations plotted in a linear manner:



The execution time of data transformations using dplyr, dtplyr and data.table plotted in a linear manner.

One can clearly see that *dtplyr* (ds\_lazy …) and *data.table* (red line, calc\_check\_dt) increase nearly linear over time. Both seem to be really good solutions when it comes to large datasets. A lot of computation time is needed to convert a *data.frame* into a *data.table.* This shall be considered. When using *data.table* all datasets should be data.tables right from the start.

From the linear plot you cannot see at what point it is useful to use dtplyr against dplyr. This can thus be seen in the following plot:



The execution time of dtplyr, dplyr and data.table plotted for the conversion of different datasets in a logarithmic manner.

The above image clearly shows that it is worth using *dtplyr* over *dplyr* in case there are more than **10E4** rows inside the dataset. So if you never work with datasets of this size it might not be worth considering *dtplyr* at all. On the other hand, you can see the *data.table* package itself is always the fastest solution.

As in my project we do not just deal with numeric transformations of data, but also character transformations, I wanted to know if the *dtplyr* package could be helpful for this. The conversion I performed for character strings looks like this:

# dplyr code

df %>%

filter(letter == "A") %>%

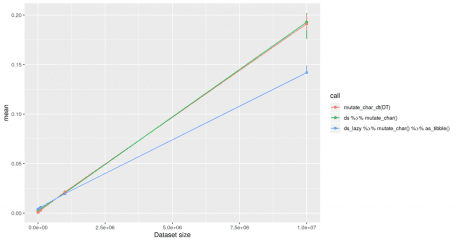
mutate(word\_new = paste0(word, numeric\_int)) %>%

select(word\_new)

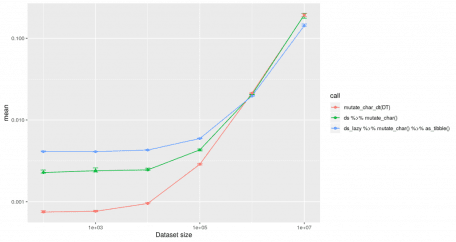
# data.table code

DT[letter == "A", word\_new:=paste0(word, numeric\_int)][letter == "A", "word\_new"]

The calculation results are similar to those gained with numeric conversions



Inside this plot the difference between *dplyr* and *data.table* is nearly invisible. All methods increase the computing time in linear relation to the data set size. *dtplyr* seems to use more efficient code than my *data.table* code.



The execution time of large scale data conversions for character evaluations with dtplyr, dplyr and data.table.

For character transformations the performance of *dplyr* is pretty good. It is just useful to use *dtplyr* for datasets with **>10E6** rows. In case of the largest dataset tested in this approach with 10E7 rows, *dtplyr* is twice as fast as *dplyr.*

**Conclusion**

The code you need to write to use *dtplyr* is really simple. Especially if you are like me *dplyr* native and do not want to start using *data.table*. But there will be a lot of use-cases where it is not necessary or not even useful to use the *dtplyr* package. So before you start writing fancy *dtplyr* pipelines, think about the necessity.