Working with list-columns in `data.table`: Proposal for `rstudio::conf(2020)`

Tyson S. Barrett[1]

[1] Utah State University

Abstract

The use of *list-columns* in data frames and tibbles is well documented (e.g. Bryan, 2018), providing a cognitively efficient way to organize results of complex data (e.g. several statistical models, groupings of text, data summaries, or even graphics) with corresponding data. For example, we can store text of a verse in a list for each verse, chapter, book, and volume. This allows the text to be of variable sizes without overly complicating or adding redundancies to the structure of the data. In turn, this can reduce the difficulty to appropriately analyze the data stored in the list-column.

Because of its efficiency and speed, being able to use `data.table` to work with list-columns would be beneficial in many data contexts (i.e. to reduce memory usage in large data sets). I show how one can create list-columns in a data table using `purrr::map()` and the `by` argument in `data.table`. I further show the `tidyr::nest()` function and show a more efficient approach when using a data table. Results using `microbenchmark` and `pryr` show the speed and efficiency of using `data.table` to work with list-columns. An example walk-through is provided in the appendix herein.

Keywords: rstudio::conf, data.table, tidyr, list-columns

References

Aust, F., & Barth, M. (2018). *papaja: Create APA manuscripts with R Markdown.* Retrieved from https://github.com/crsh/papaja

Bryan, J. (2018). List columns (as part of "purrr tutorial"). Retrieved from https://jennybc.github.io/purrr-tutorial/ls13_list-columns.html

Correspondence concerning this article should be addressed to Tyson S. Barrett, 2800 Old Main, Logan, UT 84322. E-mail: tyson.barrett@usu.edu

Clarke, E., & Sherrill-Mix, S. (2017). *Ggbeeswarm: Categorical scatter (violin point) plots*. Retrieved from https://github.com/eclarke/ggbeeswarm

Dowle, M., & Srinivasan, A. (2019). *Data.table: Extension of ‘data.frame‘*. Retrieved from https://CRAN.R-project.org/package=data.table

Heiss, A. (2019). *ScriptuRs: Complete text of the lds scriptures*. Retrieved from https://CRAN.R-project.org/package=scriptuRs

Mersmann, O. (2018). *Microbenchmark: Accurate timing functions*. Retrieved from https://CRAN.R-project.org/package=microbenchmark

R Core Team. (2018). *R: A language and environment for statistical computing*. Vienna, Austria: R Foundation for Statistical Computing. Retrieved from https://www.R-project.org/

Wickham, H. (2016). *Ggplot2: Elegant graphics for data analysis*. Springer-Verlag New York. Retrieved from https://ggplot2.tidyverse.org

Wickham, H. (2018). *Pryr: Tools for computing on the language*. Retrieved from https://CRAN.R-project.org/package=pryr

Wickham, H. (2019). *Stringr: Simple, consistent wrappers for common string operations*. Retrieved from https://CRAN.R-project.org/package=stringr

Wickham, H., François, R., Henry, L., & Müller, K. (2019). *Dplyr: A grammar of data manipulation*. Retrieved from https://CRAN.R-project.org/package=dplyr

Wickham, H., & Henry, L. (2019). *Tidyr: Easily tidy data with ’spread()’ and ’gather()’ functions*. Retrieved from https://CRAN.R-project.org/package=tidyr

Appendix

To demonstrate list-columns in `data.table`, let's start with grabbing the New Testament text from the `scripturs` package and make it a data table. From this data table, I filtered in just the four "gospels" and selected a few important variables.

```
nt <- data.table(scriptuRs::new_testament)
gospels <-
  nt[book_title %in% c("Matthew", "Mark", "Luke", "John"),
     .(book_id, chapter_number, verse_number, book_title, text)]
```

Because we want to work with the individual words in the verses, we need to tokenize the corpus. There are two main ways we can go about this:

1. Using `purrr::map()` and then `unnest()`.
2. Using the `by` argument in `data.table`.

Turns out, these produce the exact same data table.

```
gospels2 <-
  gospels[, .(book_id, chapter_number, verse_number, book_title, text,
              tokens = purrr::map(text, ~tm::MC_tokenizer(.x)))] %>%
  unnest()
gospels3 <-
  gospels[, .(tokens = tm::MC_tokenizer(text)),
          by = .(book_id, chapter_number, verse_number, book_title, text)]
all.equal(gospels2, gospels3)
```

```
## [1] TRUE
```

And both take about the same amount of memory and time to complete.

```
exp1 <- expression(
  gospels[, .(book_id, chapter_number, verse_number, book_title, text,
              tokens = purrr::map(text, ~tm::MC_tokenizer(.x)))] %>%
    unnest()
)
exp2 <- expression(
  gospels[, .(tokens = tm::MC_tokenizer(text)),
          by = .(book_id, chapter_number, verse_number, book_title, text)]
)
```

```
##      kilobytes
## exp1   -10.336
## exp2    34.712
```

```
## Unit: seconds
##    expr      min       lq     mean   median       uq      max neval cld
##  unnest 3.779131 3.817389 3.842819 3.832526 3.842268 3.955123    10   a
##      by 3.744560 3.768794 3.830423 3.814809 3.895425 3.929232    10   a
```

As such, we'll use the `purrr::map()` approach since it gives us the intermediate data, that of **list-columns**. This time we'll use `:=` to simplify the workflow. This will modify-in-place, changing the `gospels` object without copying it.

```
gospels[, tokens := purrr::map(text, ~tm::MC_tokenizer(.x))]
gospels[, .(tokens)]
```

```
##                                       tokens
##    1:          THE,book,of,the,generation,of,...
##    2:   Abraham,begat,Isaac,and,Isaac,begat,...
##    3:        And,Judas,begat,Phares,and,Zara,...
##    4:  And,Aram,begat,Aminadab,and,Aminadab,...
##    5:        And,Salmon,begat,Booz,of,Rachab,...
##   ---
## 3775:        Peter,seeing,him,saith,to,Jesus,...
## 3776:              Jesus,saith,unto,him,If,I,...
## 3777:    Then,went,this,saying,abroad,among,...
## 3778: This,is,the,disciple,which,testifieth,...
## 3779:          And,there,are,also,many,other,...
```

With this updated `gospels` data table, we can unnest it with `tidyr::unnest()`.

```
unnested_gospels <- unnest(gospels)
```

We can also nest it again. If we use it raw with `tidyr::nest()`, it produces a data frame (no longer a data table).

```
nest(unnested_gospels, tokens, .key = "tokens") %>%
  class()
```

Instead, we are going to use a custom function that uses the power of the `by` argument in `data.table` to keep the data a data table while taking advantage of the efficiency and speed of `data.table`.

```
nest_dt <- function(dt, ..., .key = "data", by = "id"){
  stopifnot(is.data.table(dt))

  j <- substitute(list(...))
  by <- substitute(by)

  express <- dt[, list(eval(j)), by = eval(by)]
  setnames(express, old = "V1", new = .key)
  express
}
```

Since having a `data.table` may be important for our workflow, this custom function be compared with the regular `tidyr::nest()` function with the `data.table()`. Below, the `tidyr::nest()` without `data.table()` is also compared, showing that the results are not due to the addition of `data.table()`.

Table 1
Memory change from using each nesting approach.

	kilobytes
regular	64204.98
regular_dt	63962.05
custom	53.62

```
regular <- expression(
  nest(unnested_gospels, tokens, .key = "tokens")
)
regular_dt <- expression(
  nest(unnested_gospels, tokens, .key = "tokens") %>% data.table()
)
custom <- expression(
  nest_dt(unnested_gospels, tokens, .key = "tokens",
          by = .(book_id, chapter_number, verse_number, book_title, text))
)
```

Results of these comparisons for this data set show that using the `data.table` approach is far more efficient ($\approx$ 64,000 kB for `tidyr::nest()` vs. 54 kB for `data.table` with `by` argument; see Table 1) and far faster (see Figure 1), yet produces the list-columns just as is desired. Beyond the comparisons, however, this example shows that the benefits of list-columns can be found in data tables.

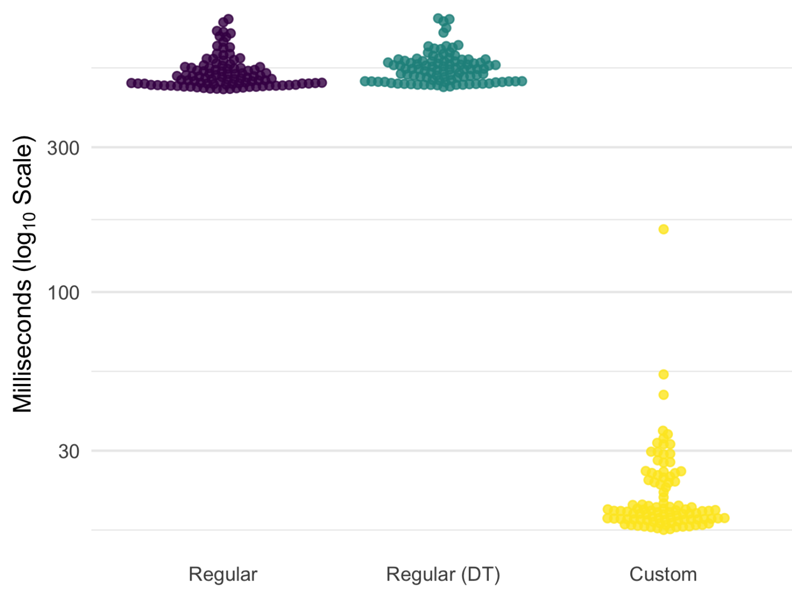

Figure 1. Speed comparisons for each nesting approach.