Working with list-columns in `data.table`

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 `dplyr::group_nest()` function and show a more efficient approach when using a data table. Results using `bench::mark()` 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: data.table, dplyr, list-columns, nesting

Introduction

The use of *list-columns* in data frames and tibbles provides 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. It is often called "nested" data, where information is, in essence, nested within a column of 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. We could also nest students within classrooms, players within teams, and measures

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

within individuals.

In turn, nesting can reduce the difficulty to appropriately analyze the data stored in the list-column. Using functions like `lapply()` or `purrr::map*()` makes further analysis of the nested data more intuitive.

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). Herein, I show how one can create list-columns in a data table using `purrr::map()` and the `by` argument in `data.table`. I further highlight the `dplyr::group_nest()` function and show a more efficient approach when using a data table. Results using `bench::mark()` show the speed and efficiency of using `data.table` to work with list-columns.

This tutorial relies on several powerful packages, including `data.table`, `dplyr`, `bench`, `tidyr`, `papaja`, `stringr`, `ggplot2`, `ggbeeswarm`, `performance`, and `rvest` (Aust & Barth, 2018; Clarke & Sherrill-Mix, 2017; Dowle & Srinivasan, 2019; Hester, 2019; Lüdecke, Makowski, & Waggoner, 2019; Wickham, 2016, 2019b, 2019a; Wickham et al., 2019; Wickham & Henry, 2019).

Example with NBA Data

The Data

To demonstrate the use of *list-columns* in `data.table`, data from NBA Stuffer will be scraped to get information on players from the 2017-2018 and 2018-2019 seasons. First, the HTML data are read in, the tables with player data by year are then extracted using a custom function, indicators are added, and then combined into a single data table for the player data.

```
url_2018 <- "https://www.nbastuffer.com/2017-2018-nba-player-stats/"
url_2019 <- "https://www.nbastuffer.com/2018-2019-nba-player-stats/"
players_2018 <- read_html(url_2018)
players_2019 <- read_html(url_2019)

extract_fun <- function(html){
  html_nodes(html, "table") %>%
    .[2] %>%
    html_table(fill = TRUE) %>%
    .[[1]]
}

player_2018 <-
  extract_fun(players_2018) %>%
  mutate(year = 2018,
         AGE = as.numeric(AGE))
player_2019 <-
  extract_fun(players_2019) %>%
  mutate(year = 2019)
```

```
players <-
  bind_rows(player_2018, player_2019) %>%
  clean_names() %>%
  rename(ppg = ppg_points_points_per_game,
         apg = apg_assists_assists_per_game) %>%
  data.table()
```

Below is a subset of this data set.

```
##          full_name  mpg  ppg apg
## 1:    Aaron Brooks  5.9  2.3 0.6
## 2:    Aaron Gordon 32.9 17.6 2.3
## 3: Aaron Harrison 25.9  6.7 1.2
## 4:   Aaron Jackson 34.5  8.0 1.0
## 5:     Abdel Nader 10.9  3.0 0.5
## 6:   Adreian Payne  8.5  4.2 0.0
```

Nesting Players within Teams

In `dplyr` the `group_nest()` function is valuable when creating list-columns based on a grouping variable. It takes the data by group and puts it all in a list-column. Figure 1 highlights the process of taking a data frame and creating a nested data frame with a list-column. That is, all data from variables `x`, `y`, and `z` relating to each group is split into a distinct data frame and stored within the `data` column.

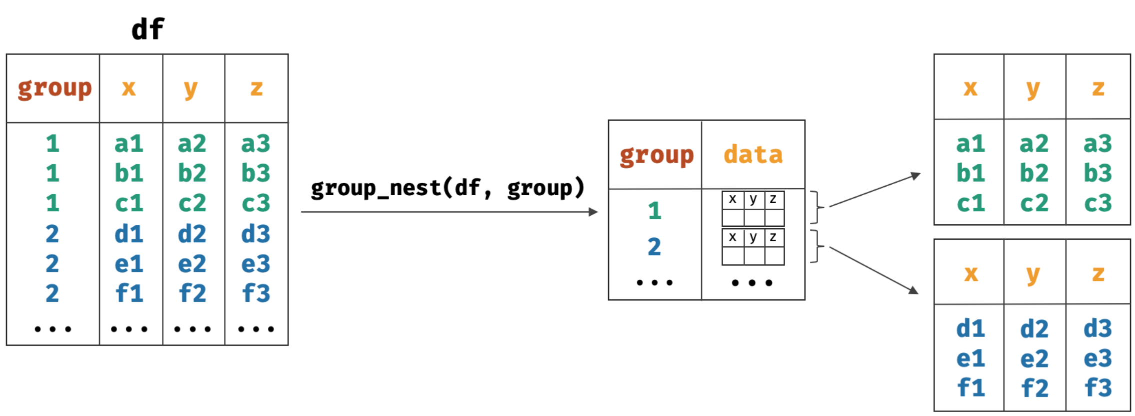

Figure 1. Speed comparisons for each nesting approach.

Overall, this function is efficient and fast but by using `data.table` it can be faster. This will be shown using the following function:

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

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

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

In essence, this function takes a data table, then creates a list of the data table per group specified in the `by` argument.

```
group_nest_dt(players, team) %>%
  head()
```

```
##    team         data
## 1:  Min <data.table>
## 2:  Orl <data.table>
## 3:  Dal <data.table>
## 4:  Hou <data.table>
## 5:  Bos <data.table>
## 6:  Ind <data.table>
```

This is nearly identical to the `dplyr::group_nest()` function, in terms of output, but has data tables in the list-column instead of tibbles.

```
group_nest(players, team) %>%
  head()
```

```
## # A tibble: 6 x 2
##   team  data
##   <chr> <list>
## 1 Atl   <tibble [44 x 30]>
## 2 Bos   <tibble [37 x 30]>
## 3 Bro   <tibble [41 x 30]>
## 4 Cha   <tibble [34 x 30]>
## 5 Chi   <tibble [43 x 30]>
## 6 Cle   <tibble [49 x 30]>
```

Importantly, Figure 3 presents the timings from `bench::mark()` across the two approaches, showing `group_nest_dt()` is somewhat faster. The memory allocated is very similar, with `group_nest_dt()` allocating 451KB and `group_nest()` allocating 335KB.

This nesting approach can be used with multiple grouping variables too. For example, we can nest by both `team` and `year`, as is done below.

```
group_nest_dt(players, team, year) %>%
  head()
```

```
##    team year         data
## 1:  Min 2018 <data.table>
## 2:  Orl 2018 <data.table>
## 3:  Dal 2018 <data.table>
```

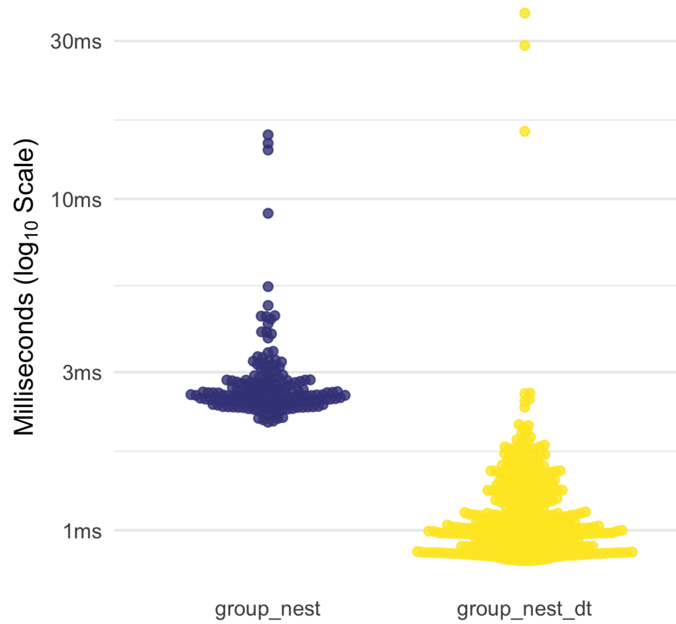

Figure 2. Speed comparisons for each nesting approach.

```
## 4:  Hou 2018 <data.table>
## 5:  Bos 2018 <data.table>
## 6:  Ind 2018 <data.table>
```

Modeling within the Nest

Often, the nested data can provide an intuitive format to run several models to understand key features of the data within the groups. Below, the relationship between points-per-game and assists-per-game for each team and year is modeled and then the R^2 of the models are extracted.

```
players_nested <- group_nest_dt(players, team, year) %>%
  .[, ppg_apg    := purrr::map(data, ~lm(ppg ~ apg, data = .x))] %>%
  .[, r2_list    := purrr::map(ppg_apg, ~performance::r2(.x))] %>%
  .[, r2_ppg_apg := purrr::map_dbl(r2_list, ~.x[[1]])]
head(players_nested)
```

```
##    team year         data ppg_apg      r2_list r2_ppg_apg
## 1:  Min 2018 <data.table>    <lm> <r2_generic>  0.4662060
## 2:  Orl 2018 <data.table>    <lm> <r2_generic>  0.4357684
## 3:  Dal 2018 <data.table>    <lm> <r2_generic>  0.4305347
## 4:  Hou 2018 <data.table>    <lm> <r2_generic>  0.6967150
## 5:  Bos 2018 <data.table>    <lm> <r2_generic>  0.6043402
## 6:  Ind 2018 <data.table>    <lm> <r2_generic>  0.6060465
```

This produces two list-columns (`ppg_apg` and `r2_list`) and a numeric vector (`r2_ppg_apg`) all organized by team and year. This information is then readily available to plot. For example, we can look at the change in how related points-per-game and assists-per-game are by team and year.

```
library(ggrepel)
theme_set(theme_minimal() +
          theme(panel.grid.major.x = element_blank(),
                legend.position = "none"))

ex_fig <- players_nested %>%
  dcast(team ~ year, value.var = "r2_ppg_apg") %>%
  ggplot(aes(`2018`, `2019`, group = team)) +
    geom_point() +
    geom_text_repel(aes(label = team)) +
    geom_abline(slope = 1) +
    coord_fixed(ylim = c(0,1),
                xlim = c(0,1))
```

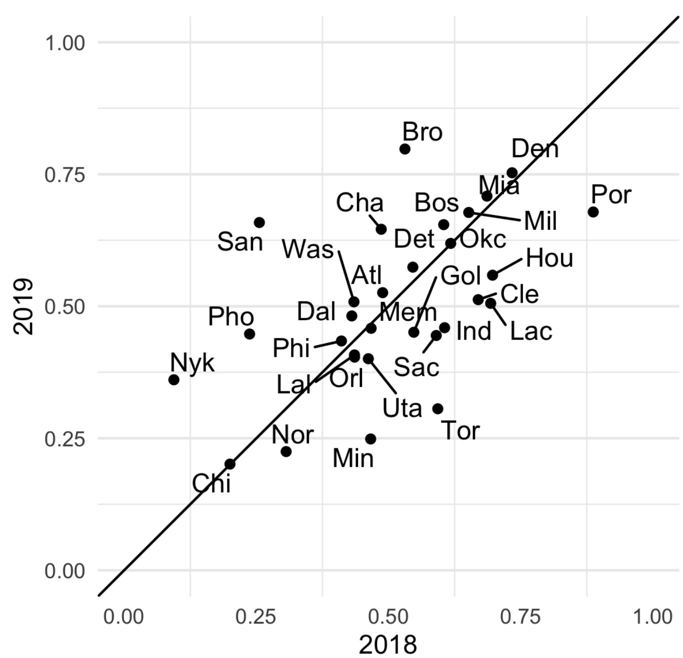

Figure 3. Speed comparisons for each nesting approach.

Discussion

List-columns are a useful approach to organizing

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

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

Hester, J. (2019). *Bench: High precision timing of r expressions*. Retrieved from https://CRAN.R-project.org/package=bench

Lüdecke, D., Makowski, D., & Waggoner, P. (2019). *Performance: Assessment of regression models performance*. Retrieved from https://easystats.github.io/performance/

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

Wickham, H. (2019a). *Rvest: Easily harvest (scrape) web pages*. Retrieved from https://CRAN.R-project.org/package=rvest

Wickham, H. (2019b). *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