The Case for data.table

Eric Karsten

25 July, 2018

The Options

Base R

Pros:

- It's the first thing you learn

Cons:

- Code is unreadable
- Iterative processes require for loops (unclear) or
- calling the `apply` function (slow)

dplyr and the tidyverse

Pros:

- The syntax is very readable
- Major speed improvements over base R
- More complex aggregations possible

Cons:

- You will eventually come across aggregations you can't do
- It's a little slow when you start doing big stuff
- You can't poke around the backend code easily (it's in C++)

data.table

Pros:

- Almost every aggregation you can think of is possible
- Some speed improvements over dplyr (task-dependent)
- Significanlty more efficient use of RAM than dplyr or base R

Cons:

- Sytax is hard to read
- You have to know what you are doing to get the most efficiency out of it (this presentation will help with that!)
- You can't poke around the backend code easily (it's in C++)

Simple Example

Our Data

This is a super simple example so that you can see all the parts.

```
# you may need to `install.packages("data.table")`

library(data.table)

df <-
  data.table(
    Animal = c("Dog", "Cat", "Dog",
               "Raven", "Cat"),
    Weight = c(100, 40, 80, 16, 50),
    Height = c(23, 18, 40, 3, 16),
    Family = c("Mammal", "Mammal", "Mammal",
               "Bird", "Mammal"))
```

The Syntax

There are three parts to a data.table command:

1. The rows to be returned
2. The columns to be returned
3. The grouping by which to perform the operations

They are separated by commas, but commas need only be included when a subsequent field is occupied.

Some nice easy subsetting

Let's just select the dogs.

```
df[Animal == "Dog"]
```

```
##    Animal Weight Height Family
## 1:    Dog    100     23 Mammal
## 2:    Dog     80     40 Mammal
```

Let's return the height and weight of the mammals that weigh more than 35 lbs

```
df[Weight > 35, .(Animal, Height, Weight)]
```

```
##    Animal Height Weight
## 1:    Dog     23    100
## 2:    Cat     18     40
## 3:    Dog     40     80
## 4:    Cat     16     50
```

Using :=

Often times, we want to add new calculated columns to a dataframe. The correct way to do this in data.table is with a :=. Because this operation works by updating the previous object, we need to make a copy of it in order to avoid modifying our original data frame.

The following are equivalent for computing BMI (weight/height) [1]

```
df2 <- copy(df)[, BMI := Weight/Height]
df2 <- copy(df)[, `:=`(BMI = Weight/Height)]
```

```
##     Animal Weight Height Family      BMI
## 1:     Dog    100     23 Mammal 4.347826
## 2:     Cat     40     18 Mammal 2.222222
## 3:     Dog     80     40 Mammal 2.000000
## 4:   Raven     16      3   Bird 5.333333
## 5:     Cat     50     16 Mammal 3.125000
```

[1] BMI should technically be weight in Kg divided by height in m, but I'm not going to worry about the conversions

We could also have done this without efficient updating. This is much slower on big datasets and should not be done.

```
df[, .(Animal, Weight, Height, Family,
       BMI = Weight/Height)]
```

```
##     Animal Weight Height Family      BMI
## 1:     Dog    100     23 Mammal 4.347826
## 2:     Cat     40     18 Mammal 2.222222
## 3:     Dog     80     40 Mammal 2.000000
## 4:   Raven     16      3   Bird 5.333333
## 5:     Cat     50     16 Mammal 3.125000
```

Selective updating is the reason why := is so useful. Let's say we wanted to change the hieght of all ravens to be 5. We don't want to rewrite the whole table, so we simply call

```
df[Animal == "Raven", Height := 5]
```

```
##     Animal Weight Height Family
## 1:     Dog    100     23 Mammal
## 2:     Cat     40     18 Mammal
## 3:     Dog     80     40 Mammal
## 4:   Raven     16      5   Bird
## 5:     Cat     50     16 Mammal
```

Grouped operations

Let's say we want the average weight by animal. Here are two ways to do it with different results.

```
df[, .(mean_wt = mean(Weight)), by = .(Animal)]
```

```
##     Animal mean_wt
## 1:     Dog      90
## 2:     Cat      45
## 3:   Raven      16
```

```
df2 <- copy(df)[, mean_wt := mean(Weight), by = Animal]
```

```
##     Animal Weight Height Family mean_wt
## 1:     Dog    100     23 Mammal      90
## 2:     Cat     40     18 Mammal      45
## 3:     Dog     80     40 Mammal      90
## 4:   Raven     16      5   Bird      16
## 5:     Cat     50     16 Mammal      45
```

Chaining multiple operations in sequence

Oftentimes, you want to do several operations in sequence. Naturally, there is syntax that reflects this workflow. Here we are computing BMI and then making a table of average BMI by Animal.

```
df[, `:=`(BMI = Weight/Height)][
  , .(avg_BMI = mean(BMI)), by = Animal]
```

```
##     Animal  avg_BMI
## 1:     Dog 3.173913
## 2:     Cat 2.673611
## 3:   Raven 3.200000
```

It should be noted that this example could be done in one line, so chaining was not necessary.

Using .SD tricks

There are times when we want to perform more complex operations on our grouped data than applying a simple built-in function. One option is to write a function of our own, but this isn't always the clearest. The better option is to manipulate the grouped data.table using standard data.table operations!

The way we do this is by calling the .SD object (which is just a data.table of our selected data) and performing operations on it.

EX: We want to add a column for the height of the heaviest individual within each Animal species.

To do this, we need to first compute within each animal table what the biggest weight is, then, we need to return the height of the animal with that weight and assign it to the BigHeight variable for that animal group.

```
df[, BigHeight := .SD[, .(BiggestWeight = max(Weight),
                          Weight,
                          Height)][
                        Weight == BiggestWeight, Height]
   , by = Animal]
```

```
##     Animal Weight Height Family      BMI BigHeight
## 1:     Dog    100     23 Mammal 4.347826        23
## 2:     Cat     40     18 Mammal 2.222222        16
## 3:     Dog     80     40 Mammal 2.000000        23
## 4:   Raven     16      5   Bird 3.200000         5
## 5:     Cat     50     16 Mammal 3.125000        16
```

I challenge you to do this in either dplyr or base R in a cleaner or faster way!

Speed Tricks

Why is it so fast?

Both data.table and dplyr are executed in C++. This allows the developers to use some really efficient indexing to perform their operations.

The advantage data.table has over dplyr in the speed department is that it is able to update the data in RAM, it doesn't need to constantly be rewriting it. This saves a ton of time if you are doing thousands of small operations in a row. It also allows the program to run without eating up all your RAM!

How to maximize speed

Always use `:=` when possible. You never want to rewrite your data.table if you don't have to.

To delete a column quickly, just use `:= NULL` to get rid of it.

If you are doing several things on the same group, don't chain the operations together, instead do them all at once.

Merging isn't covered in this talk, but data.table can do really fast merging.

Proof

The task

We are going to use the microbenchmark library to find the average BMI of our animals, after adding 7 lbs to all the rabbits but to slow it down, we are going to randomly generate a ton of data.

```
library(microbenchmark)
library(dplyr)
n <- 1e5
dt_test <-
  data.table(Animal = sample(c("Rabbit", "Chicken",
                               "Zebra", "Snake",
                               "Giraffe", "Eagle"),
                             n, replace = T),
             Height = runif(n, 3, 7),
             Weight = runif(n, 33, 47))
df_test <- as.data.frame(dt_test)
tb_test <- as_tibble(dt_test)
```

The Benchmark Setup

```
data_table_BMI <- function(df) {
  df[Animal == "Rabbit", Weight := Weight + 7][
    , .(avg_BMI= mean(Weight/Height)), by = Animal]}
dplyr_BMI <- function(df) {
  df %>%
    mutate(Weight = if_else(Animal == "Rabbit",
                            Weight + 7,Weight)) %>%
    group_by(Animal) %>%
    summarise(avg_BMI = mean(Weight/Height))}
base_R_BMI <- function(df) {
  df$Weight = ifelse(df$Animal == "Rabbit",
                     df$Weight + 7,df$Weight)
  aggregate(df$Weight/df$Height,
            by=list(Animal=df$Animal),
            FUN=mean)}
```

Benchmark Results

```
res <-
  microbenchmark(
    data_table_BMI(dt_test),
    dplyr_BMI(tb_test),
    base_R_BMI(df_test))
```

```
## # A tibble: 3 x 4
##   Method       Mean Median    SD
##   <chr>       <dbl>  <dbl> <dbl>
## 1 Base R      67.5   67.0   5.08
## 2 data.table   7.42   7.19  0.93
## 3 dplyr       12.3   10.6   6.3
```

Concluding Thoughts

A Handy Guide

- If you data is less than 100 rows, skip the computer and do it with pencil and paper, it really doesn't even matter
- Less than 1000 and everything will feel fast
- Less than 1,000,000 and dplyr should be alright
- Less than 10,000,000 and data.table will work on your laptop
- Otherwise, it's time to go shopping for server space

Thank You

A much more comprehensive data.table guide is provided here: https://cran.r-project.org/web/packages/data.table/vignettes/datatable-intro.html