Introduction to data.table

Markus Steinmaßl, Andreas Burgstaller, Vanessa Eibl

Universität Salzburg

Statistics, Visualization and More Using "R"

April 8, 2019

Overview

What is data.table?

- R-package for manipulating data sets
- provides an enhanced version of data.frame
- data.table inherits from data.frame, therefore a data.table object is also a data.frame object
- Functions, that take data.frame objects only, can be applied to data.table objects, since a data.table IS a data.frame

Why is data.table useful?

- data.table has a simple and compact syntax.
- It keeps dataset operations (i.e subset, group, join) together in one function, which allows for internal optimization.
- data.table provides the opportunity to build indices, which can speed up computations especially for large data sets
- Therefore it is very fast and well suited for manipulating huge data sets.

The data.table A[B] syntax

The syntax for manipulating a data.table object *DT* is:

```
DT[ i , j , by ]
```

How to read:
Take *DT*, subset/reorder rows using *i*, then calculate *j*, grouped by *by*

This is similar to SQL:

```
   R:          i                    j          by
 SQL:  where | order by   select | update  group by
```

Data.table vs. dplyr

- dplyr is another popular package for data manipulation.
- It uses operations with "verb" names, e.g select(), filter(), group_by() to manipulate data.frames.
- This makes dplyr intuitive and expressive
- Drawback: operations are executed one after another
  $\rightarrow$ no optimization between operations

Data.table vs. dplyr - Indexing

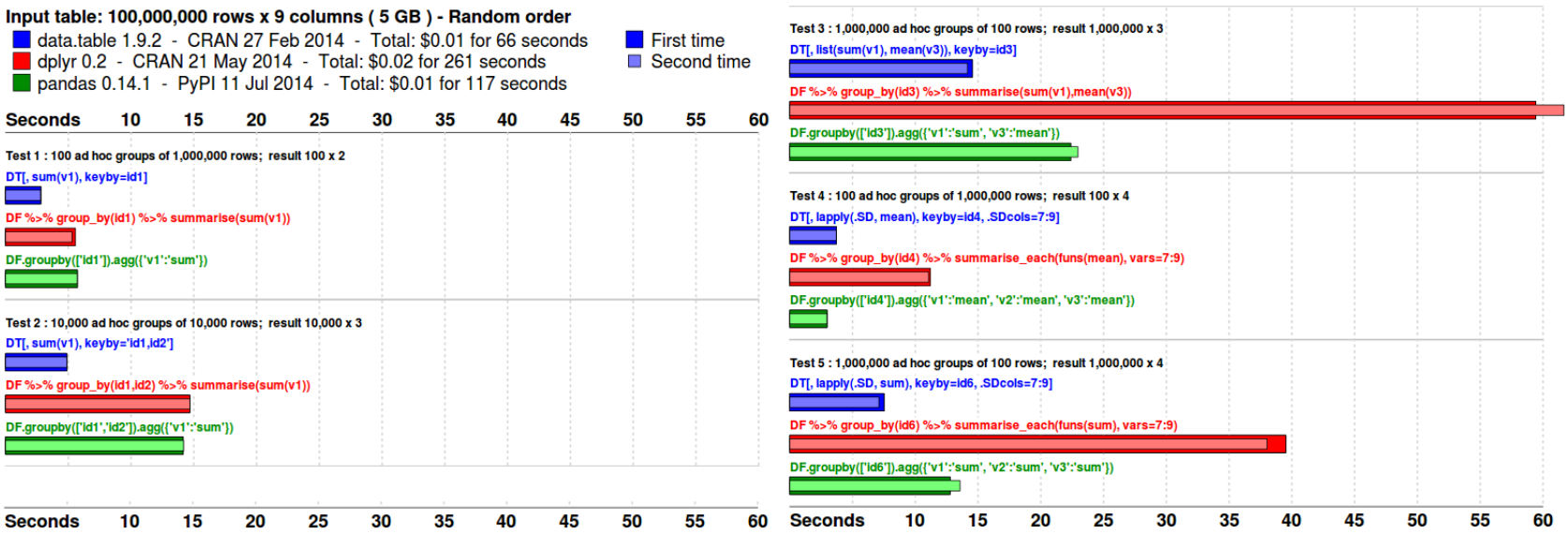
- Unlike dplyr, data.table provides memory efficient sorting and indexing
- `setkey()` sorts a data.table by reference (i.e no deep copys are made)
- there can only be one key at a time
- `setindex()` creates an index on specified columns
- multiple indices possible
- since version 1.9.3 automatic indexing is implemented and enabled by default

Data.table vs. dplyr - Automatic indexing

- The first time you subset a data.table, an index is automatically generated while performing a vector scan, and therefore subsequent subsets are very fast.
- See `https://gist.github.com/arunsrinivasan/dacb9d1cac301de8d9ff` for more information on automatic indexing and comparison to dplyr

Data.table vs. dplyr - Benchmark for aggregation

Especially for grouping operations, data.table is a lot faster than dplyr.



https://raw.githubusercontent.com/wiki/Rdatatable/data.table/bench/grouping.1E8.png

Data.table vs. dplyr - Summary

- dplyr operates directly on data.frames
- dplyr is optimized to be intuitive and expressive
- data.table creates a new data structure by extending data.frame
- data.table is optimized for fast and memory efficient functionality
- dplyr might be better suited for medium sized data, where as data.table is best on large data sets

Installing and loading the data.table package

Before we can work with the data.table package, we need to install it to R and load it for the current session

```
install.packages("data.table")
library(data.table)
```

Creating a data.table object...

...by using data.table's file reader method `fread()`.

- `fread()` is similar to base R's `read.table()`,
- but faster!
- `fread()` detects controls such as sep, colClasses and nrows automatically

Example:

```
DT <- fread("filename.csv")
```

- analogously `fwrite()` is a function to write a data.table to a file.

Creating a data.table object...

...by combining vectors of the same length using `data.table()`

This is the same as for `data.frame()` and works because data.table inherits from data.frame

Example:

```
x <- seq(1,10)
y <- rep(4,10)
DT <- data.table(id = x,y)
```

Here the first column is called "id" and the second one is called "y".

Creating a data.table object...

...by copying an existing data.frame or matrix using `data.table()`

Example:

```
DF <- data.frame(id = seq(1,5), x = c(-2,3,4,0,10))
DT <- data.table(DF)

M <- matrix(seq(1,6), nrow = 3)
colnames(M) <- c("id", "x")
DT <- data.table(M)
```

Creating a data.table object...

...by transforming an existing data.frame using `setDT()`

Example:

```
DF <- data.frame(id = seq(1,5), x = c(-2,3,4,0,10))
setDT(DF)
```

- DF is now a data.table object. We will explore this further in the first exercises
- `setDT()` can also be applied to lists.

Creating a data.table object - Summary

1. Using data.table's file reading method *fread()*

```
DT <- fread("filename.csv")
```

2. Building data.table from vectors

```
DT <- data.table(id = seq(1,5), x = c(-2,3,4,0,10))
```

3. Copying existing object, e.g data.frame or matrix

```
DF <- data.frame(id = seq(1,5), x = c(-2,3,4,0,10))
DT <- data.table(DF)
```

4.Transforming existing object, e.g data.frame or list

```
DF <- data.frame(id = seq(1,5), x = c(-2,3,4,0,10))
setDT(DF)
```

Exercises

Exercises - Block 1

Do exercises 1 - 4.

Be careful when to use `data.table()` and `setDT()`.

Data set

Which variables are in the data set?

colnames(RTR)

```
 [1] "id"              "mtime"           "mymd"
 [4] "long"            "lat"             "iso_adm2"
 [7] "op_name"         "nw_cat"          "device"
[10] "device_platform" "rtr_speed_dl"    "rtr_speed_ul"
```

Notation differences to data frame

Square brackets [] behave differently in data.table compared to data frames.
Square brackets are like a function but no need to type a word like myfunction.
All manipulation is handled by the brackets.
First listing in the brackets is the row not the column as in data frames.
This means DT[3:5] is equivalent to DF[3:5,]

```
.() makes the returned value a data.table.
Without, it would be a vector.
DT[,.(3:5)]
```

DT[i,j,by] - Subsetting rows using i

Subset rows using numbers:

```
DT[1:3,]
```

or

```
DT[1:3]
```

Subset rows using column names:

```
ans <- RTR[device == "Nexus 4"]
```

DT[i,j,by] - Manipulate columns with j

Select one or more columns:

```
RTR[,.(device, device_platorm)]
```

Compute on one or more columns:

```
Returns the mean of upload and download speed
RTR[,.(mean(rtr_speed_up), mean(rtr_speed_dl))]

Assigne a new column name to the computed variable
RTR[,.(Aggregate = mean(rtr_speed_up),
speed.dl=mean(rtr_speed_dl)]

Aggregate speed.dl
1:  7247.022 19035.52
```

DT[i,j,by] - Manipulate columns with j

Summarize Create a new data.table with new columns based on the summarized values of rows

```
RTR[ , .( x=max( rtr_speed_ul ) )]
1:  49034
```

DT[i,j,by] - Changing columns using :=

:= updates the data.table by reference. It is used at position j.
If you want a new data.table with the changes, you need to copy it first.

Delete a column:

```
Compute a new column based on an expression
 RTR[ ,.c:= 100+200]
RTR2 = RTR
RTR2[, mtime := NULL]
```

Updating a column:

```
 RTR2[, rtr_speed_dl := round(rtr_speed_dl ,2)]
```

Adding [] prints the result to the screen Otherwise R will change it without printing it to the console

Manipulating multiple columns and convert column type

You can delete multiple columns with a vector of column names:

```
Cols.chosen = c("op_name", "iso_adm2")
RTR2[,Cols.chosen:=NULL]
```

Convert a column type (and save as a new column):

```
RTR[,dateAsText:=as.character(mymd)]
```

DT[i,j,by] - grouping using by

Execute j by group

Calculate mean for download speed for every network category:

RTR[,.(speed.dl.Mean = mean(rtr_speed_dl)), by=nw_cat]

```
   device_platform  speed_dl.Mean
1:         Android       7133.920
2:             iOS       7488.003  rows
```

DT[i,j,by] - grouping using by

Extract unique rows based on columns specified in "by"
Leave out "by" to use all columns

```
unique(RTR, by=c("a","b"))
```

Combine data.tables

Join two data.tables based on rows with equal values
or on rows with equal and unequal values
or rolling join which keeps only the most recent preceding match of a according to date columns

```
dt_a[dt_b, on = .(b=y)]
dt_a[dt_b, on =.(b=y, c>z)]
dt_a[dt_b, on=.(id, date=date), roll = TRUE]
```

Combine rows and columns

Binding rows and columns in data.table is alike to data.frame
No specific data.table function

```
Bind rows
rbind(dt_a, dt_b)
Bind columns
cbind(dt_a, dt_b)
```

Exercises

Exercises - Block 2

Do exercises 5 to 10

Subqueries

We can use subqueries to build a chain in data.table.

```
DT[ , y:=a+b ][ , .( y , a , b ) ]
```

First, we create a new column and then we select only relevant columns.

Add multiple columns with ‘:=‘()

```
RTR[, ':=' (max = max(rtr_speed_dl), v = 700)]
```

It is not possible to use a generated column for calculating an other column in the same query

```
RTR[, ':=' (max = max(rtr_speed_dl), v = 700)][, t:=max+v]
```

Exercises

Exercises 11

- Create a column called "mean_by_device", where you calculate the mean for every single device
- Display a table with the columns "rtr_speed_dl" , "mean_by_device" , and the difference between them (rtr_speed_dl - mean_by_device) by using subqueries

Some useful commands

- .N is a counting tool that holds the number of observations in a current group.
- .I counts every row and saves it as a vector. So you can add an id to a data set
- .GRP command allows you to add an unique id to groups
- ‘:=‘ () is used to add multiple columns to your dataset
- %in% is an match operator to compare multiple values

Example for .N

For example, if we want to calculate the number of rows per group:

- `RTR[ , .N , by = device_platform ]`

```
#    device_platform     N
# 1:         Android 75093
# 2:             iOS 35244
```

Example for .I

Add an id to a dataset

- `RTR[ , new_id := .I , ]`
- `str(RTR)`

```
Classes 'data.table' and 'data.frame': 110337 obs.
of 13 variables:
 $ id              : int  1 2 3 4 5 6 7 8 9 10 ...
 ...
 $ rtr_speed_dl    : num  3226 2068 1948 11604 6277 ...
 $ rtr_speed_ul    : num  419 572 1039 2881 2484 ...
 $ new_id          : int  1 2 3 4 5 6 7 8 9 10 ...
```

Example for .GRP

For example, if we want to give a new id for every unique value of a column:

- `RTR[,.(.GRP),by=device_platform]`

```
#     device_platform GRP
# 1:         Android   1
# 2:             iOS   2
```

Example for %in%

```
RTR[id %in% c(1,3,110,200) , , ]
```

is the same as

```
RTR[id == 1 | id == 3 | id == 110 , , ]
```

Exercises

Exercises 12 + 13

Do exercise 12 + 13

Finish

Thank you

for listening