SIPHER Consortium

Processing large-scale data efficiently: An introduction to the R package 'data.table'.

Research Data Scotland

Bayes Centre
University of Edinburgh
15th August 2024

Andreas Hoehn

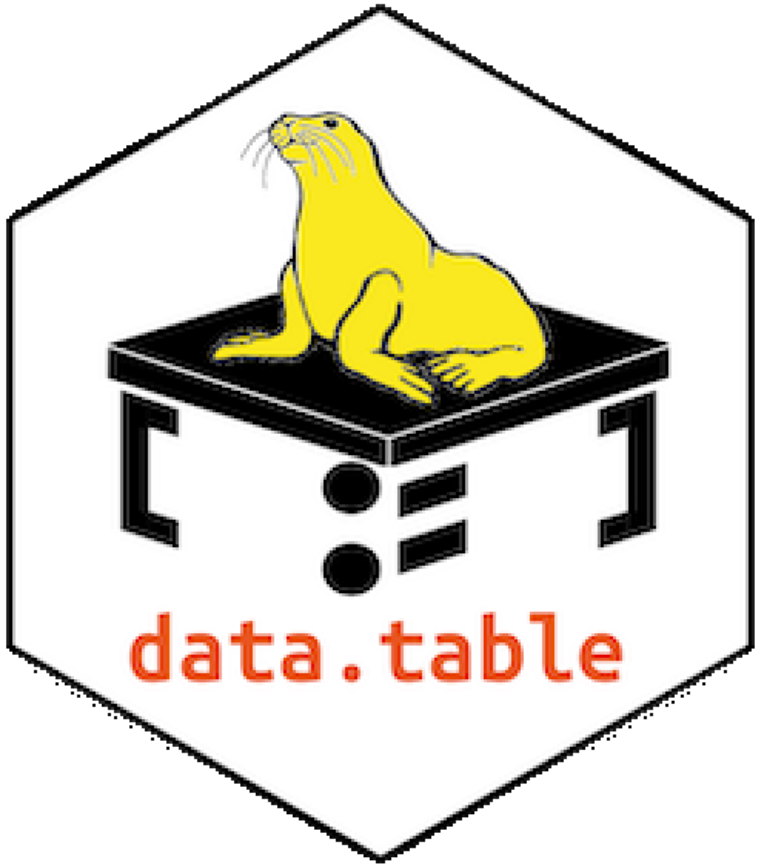
University of Glasgow
andreas.hoehn@glasgow.ac.uk
AndreasXHoehn
0000-0002-7170-1205

Outline

Introducing ‘data.table’

Hands-on Session

Q&A



Source: https://rdatatable.gitlab.io/data.table/

Introducing ‘data.table’

What will be faster: A Ferrari or a Honda?

Introducing ‘data.table’

What will be faster: A Ferrari or a Honda?

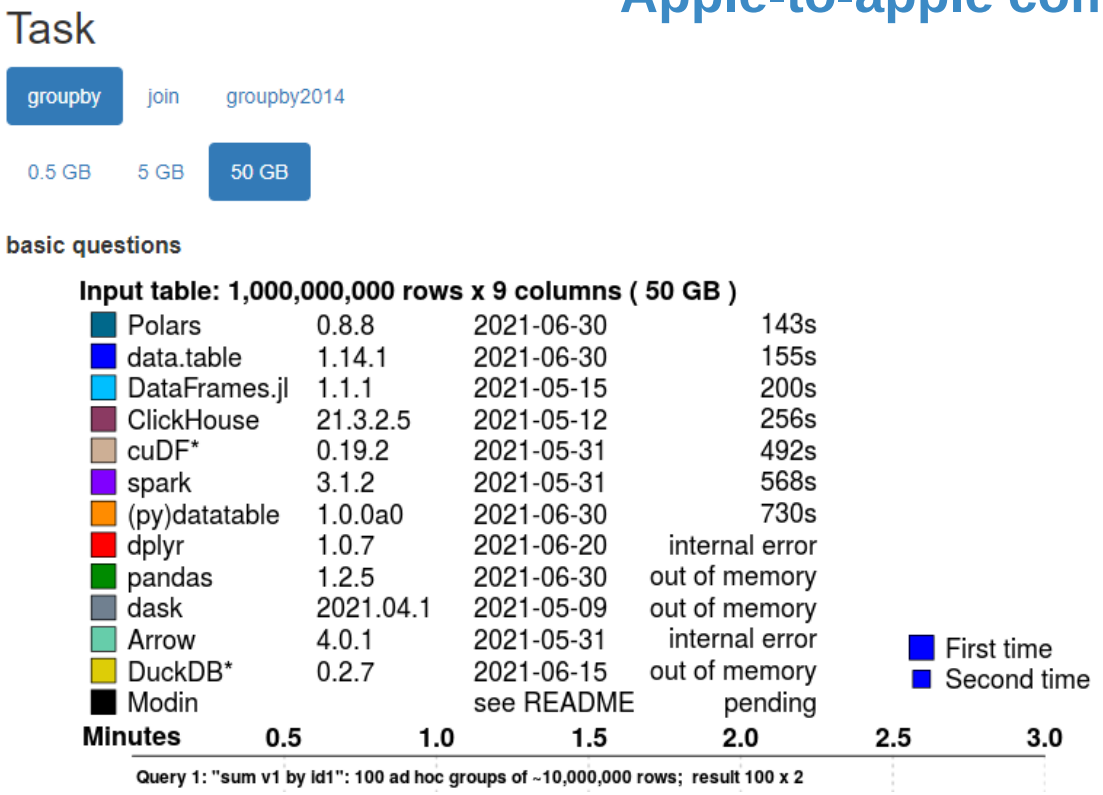
Photo by Stefano Probst on Unsplash

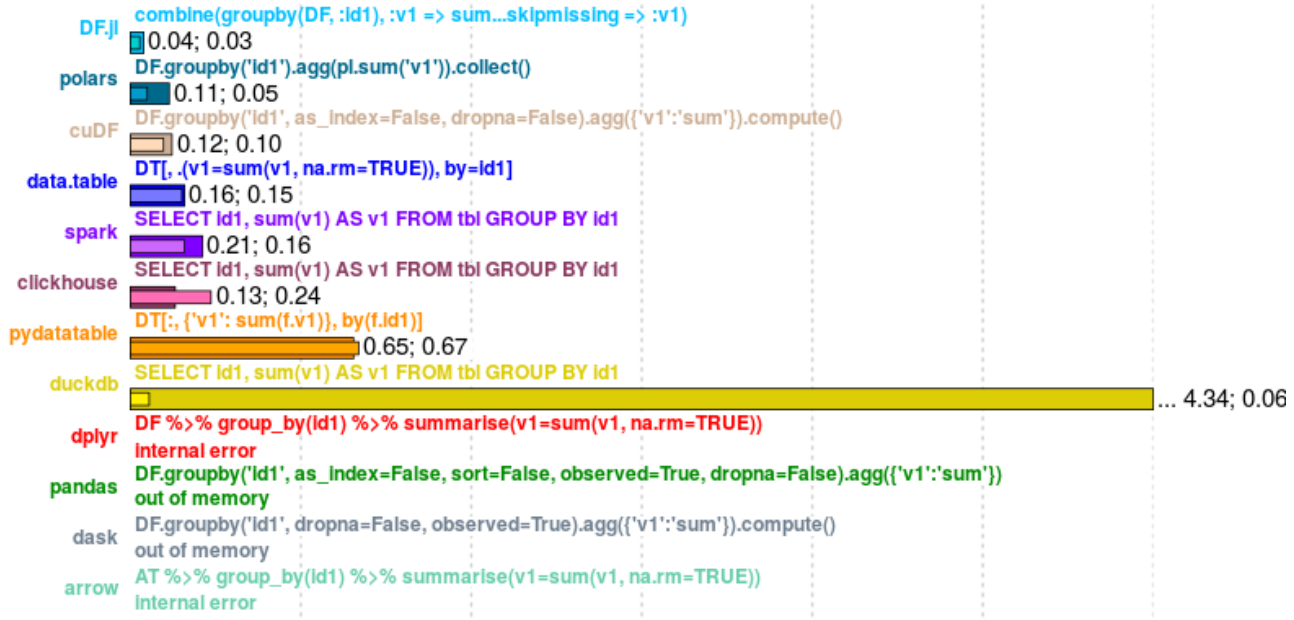
vs.

Photo by Brad armore on Unsplash

Introducing ‘data.table’

Apple-to-apple comparisons are required!





Source: https://h2oai.github.io/db-benchmark/

Introducing ‘data.table’

Scene Setting

1) Routinely collected data has always been large - and will only get larger (e.g. high-dimensional smart data, real-time data, omics data, CTGAN synthetic data)

2) Despite lots of progress, memory (RAM) and processing (CPU) are still limited resources

While R is a fantastic programming language, “standard” R (e.g. through base, dplyr etc.) is not the most efficient/safest/futureproof/portable/updateable/….. way of working

Even worse! “Standard” R’s limitations further fuel the problems that come from 1) and 2)

Introducing ‘data.table’

“Standard” R’s limitations

1) By default, R uses 1 core – but many more are available on most machines (→ CPU + time)

2) By default, R has a “copy-on-modify behaviour” (→ RAM + time)

3) Dependencies, portability, backward compatibility: Updates vs. old code (→ lots of time ….)

4) R Code can get long, especially when using long chains of pipes (→ even more time …)

Introducing ‘data.table’

“Standard” R’s limitations vs. data.table

1) By default, R uses 1 core – but many more are available on most machines (→ CPU + time)
data.table parallelises whenever this is easily done, running compiled C/C++ underneath

2) By default, R has a “copy-on-modify behaviour” (→ RAM + time)
data.table modifies on reference (“ := “) and uses pointers, requiring less working RAM

3) Dependencies, portability, backward compatibility: Updates vs. old code (→ lots of time ….)
data.table has no dependencies other than R >= Version 3.1 (10+ years old)

4) R Code can get long, especially when using long chains of pipes (→ even more time …)
data.table has short and expressive code, similar to high-level programming languages

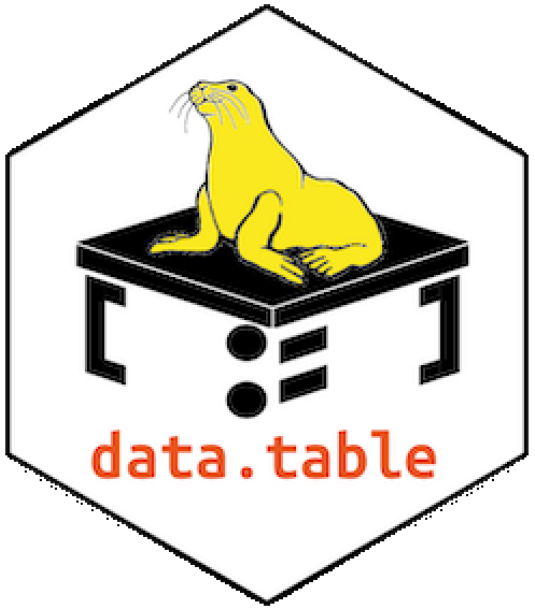
Introducing ‘data.table’

What is data.table?

1) A selection of functions, optimised to work efficiently with large amounts of data (e.g., fwrite, fsave, or for reshaping)

2) A separate dialect for R, with some similarities to SQL or C/C++

3) A unique chance to think about our computing, with the aim of improving processes and futureproofing our work.



Source: https://rdatatable.gitlab.io/data.table/

Introducing ‘data.table’

Standard ‘data.table’ notation

Standard format 1: data[i , j] → basic format when not operating on groups

Standard format 2: data[i , j , by] → when operating by group

“data” is our dataset
“i” subset of “data” based on row information (“observations”)
“j” states what to execute for the columns (“variables”)
“by” defines whether “i” and “j” should be done by groups

Introducing ‘data.table’

How to read data.table?

data[sex == “Male”,] → subset where sex is “Male”, nothing to execute

data[, V2 := V1+1] → nothing to subset, create a new variable “V2” which is “V1+1”

data[sex == “Male”, V2 := V1+1] → subset where sex is “Male”, then create V2…

data[sex == “Male”, .(V2 = max(V1))] → subset where sex is “Male”, then return a new variable

data[, .(V2 = max(V1)), by = c(“sex”)] → no subset, return new variable by group (“sex” variable)

Introducing ‘data.table’

Any guesses?

```
data2 <- data[age >= 16, .(sex = sex, income_median = median(income)), by = c(“ID”)]
```

Introducing ‘data.table’

Any guesses?

```
data2 <- data[age >= 16, .(sex = sex, income_median = median(income)), by = c(“ID”)]
```

… assign to a new object data2 something that comes out of data
… subset for those aged 16 or older
… returns a dataset which will contain “sex” (unchanged), “income_median”, and the group by “ID”
… the new variable “income_median” is the income of all recorded medians of the subset
… which was established separately for all “IDs” (as there are multiple records per “ID”)

Hands-on Session

All course materials are available on GitHub!

No account required, a .zip bundle can be downloaded

Repository: 2024_RDS_DT

https://github.com/AndreasxHoehn

Hands-on Session

Learning Objectives:

1) Benchmarking time: ‘microbenchmark::microbenchmark()’

2) Benchmarking memory: ‘object.size()’ and variable types

3) Tracing the location of objects within ‘tracemem()’

4) Introduction to ‘data.table’ - basic functions, subsets, creating new variables, group by operations, reshaping data etc.

Further Resources

data.table on cran: https://cran.r-project.org/web/packages/data.table/

Benchmarking data.table operations: https://tysonbarrett.com//jekyll/update/2019/10/06/datatable_memory/]

Benchmarking joins: https://tysonbarrett.com/jekyll/update/2019/10/11/speed_of_joins/

A good basic data.table intro: https://atrebas.github.io/post/2020-06-17-datatable-introduction/

The official data.table FAQ: https://cran.r-project.org/web/packages/data.table/vignettes/datatable-faq.html

Q&A

Questions – Comments – Feedback?

SIPHER Consortium

THANK YOU FOR LISTENING

Find out more

Website: www.sipher.ac.uk

Email: sipher@Glasgow.ac.uk

Twitter: @SipherC