THE #RDATATABLE PACKAGE

+ new developments in v1.10.0

Arun Srinivasan

JAN 20'17, AMSTERDAM

@ARUN_SRINIV

WHO AM I?

- Electronics Engineer
- Bioinformatician / Comp. Biologist
- data.table user, co-developer since late 2013
- Last: Data scientist @Open Analytics, Belgium
- Current: Lead engineer, Millennium, UK

MOST UNDERRATED PACKAGE

Conor Nash
@conornash

Follow

Data.table is the most underrated R package. It has saved me *days* in waiting for analyses to complete.

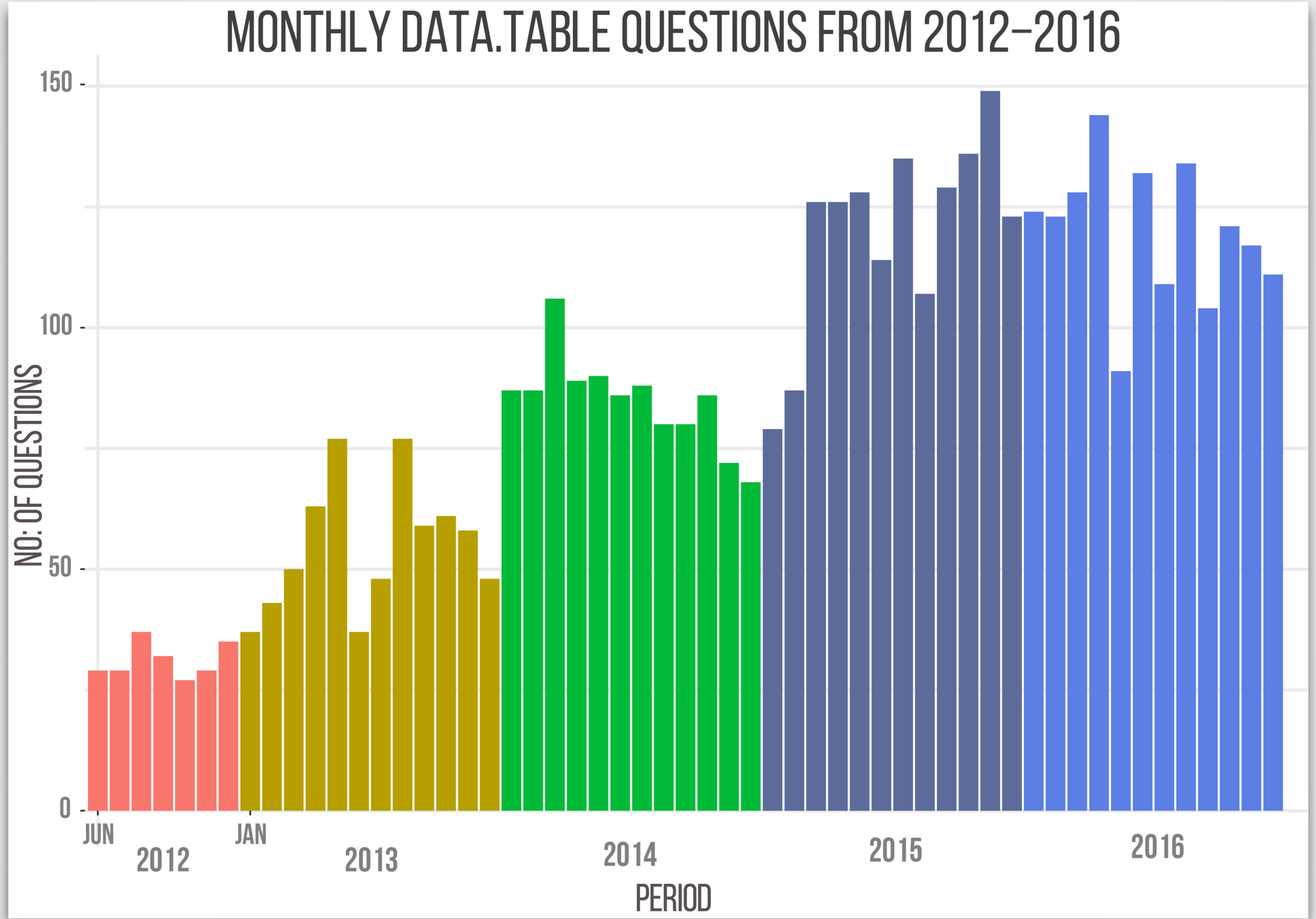
MOST UNDERRATED PACKAGE

Mehdi Nemlaghi
@Mehdi_Nemlaghi

Follow

@freakonometrics "setkey" function is so powerful, so innovative for #rstats. Imho, "Data.table" package is kind of underrated...

- Homepage: http://r-datatable.com
- Since 2006 on CRAN, 36 releases so far
- >5900 unit tests, ~90% coverage (using covr)
- >330 packages import/depend/suggest data.table
  - ~19 packages per month since Sep'16
- 9th most starred R package on Github (METACRAN)
- ~5000 Q on SO. 3rd amongst R packages


MONTHLY DATA.TABLE QUESTIONS FROM 2012–2016
150
100
50
0
NO. OF QUESTIONS
JUN
2012
JAN
2013
2014
2015
2016
PERIOD

POWERFUL

Alexander Flyax
@aflyax

Follow

somebody should just write a version of #Rstat's data.table for #python. end of story. nothing as powerful exists at the moment.

GREAT SADNESS

Jim Savage
@khakieconomist

Follow

With great sadness I was forced to start using data.table today.

DATA.TABLE DATA.TABLE DATA.TABLE

Joey Reid
@JoeyPReid

Follow

data.table
data.table
data.table
data.table
ggplot2
rstan
knitr

#7FavPackages

TALK OVERVIEW

- data.table's philosophy
  - concise + straightforward code
  - fast + memory efficient
- optimisations, and new features in v1.10.0

TALK OVERVIEW

- data.table's philosophy
  - concise + straightforward code
  - fast + memory efficient
- optimisations, and new features in v1.10.0

DATA FRAMES

- are columnar data structures

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

2 column data.frame

DATA FRAMES

- are columnar data structures
  - 2D — rows and columns

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

2 column data.frame

DATA FRAMES

- are columnar data structures
  - 2D — rows and columns

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

2 column data.frame

DATA FRAMES

- are columnar data structures
  - 2D — rows and columns
- subset rows — X[X$id != “a”,]

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

DATA FRAMES

- are columnar data structures
  - 2D — rows and columns
- subset rows — X[X$id != “a”,]
- select columns — X[, “val”]

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

DATA FRAMES

- are columnar data structures
  - 2D — rows and columns
- subset rows — X[X$id != “a”,]
- select columns — X[, “val”]
- subset rows & select columns — X[X$id != “a”, “val”]

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

DATA FRAMES

- are columnar data structures
  - 2D — rows and columns
- subset rows — X[X$id != “a”,]
- select columns — X[, “val”]
- subset rows & select columns — X[X$id != “a”, “val”]
- that’s pretty much it…

X

	id	val
1	b	4
2	a	2
3	a	3
4	c	1
5	c	5
6	b	6

1. HOW TO COMPUTE ON COLUMNS?

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code != “abd”,
get sum(valA)

1.9

1. HOW TO COMPUTE ON COLUMNS?

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

```
sum(DF[DF$code != “abd”, “valA”])
```

1.9

2. GROUPED AGGREGATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code != “abd”,
get sum(valA) and sum(valB)
for each id

	id	valA	valB
1	1	0.7	18
2	2	1.2	23

2. GROUPED AGGREGATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

```
aggregate(cbind(valA, valB) ~ id,
          DF[DF$code != "abd", ],
          sum)
```

	id	valA	valB
1	1	0.7	18
2	2	1.2	23

3. SIMPLE UPDATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	1.5	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code == “abd”,
update valA
with NA

3. SIMPLE UPDATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	NA	5
4	2	apq	0.9	10
5	2	apq	0.3	13

For code == “abd”,
update valA
with NA

3. SIMPLE UPDATE

DF

	id	code	valA	valB
1	1	abc	0.1	11
2	1	abc	0.6	7
3	1	abd	NA	5
4	2	apq	0.9	10
5	2	apq	0.3	13

```
DF[DF$code == "abd", "valA"] <- NA
```

CAN WE BE MORE CONSISTENT?

```
sum(DF[DF$code != “abd”, “valA”])
```

How to get *sum* of both *valA* and *valB*?
Or *sum* of *valA* and *valB* combined?

```
aggregate(cbind(valA, valB) ~ id,
          DF[DF$code != “abd”, ],
          sum)
```

New function. Formula interface.
Unwanted columns are subsetted.
How to get sum(valA) and mean(valB)?

```
DF[DF$code == “abd”, “valA”] <- NA
```

Entire expression is now to the left
of the “<-“ operator

ENHANCED DATA FRAMES

- Three main enhancements:
  1. Allow column names to be seen as variables within [...]
  2. Since they're variables, we can do computations on them directly, i.e, within [...]
  3. Additional argument by

DATA TABLES

- are columnar data structures as well

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

2 column data.table

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

2 column data.table

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

2 column data.table

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]
- select columns — X[, val]

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]
- select columns — X[, val]
- *compute on* columns — X[, mean(val)]

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

mean
3.5

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]
- select columns — X[, val]
- *compute on* columns — X[, mean(val)]
- subset rows & select / *compute on* columns — X[id != “a”, mean(val)]

X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

mean
4.0

DATA TABLES

- are columnar data structures as well
  - 2D — rows and columns
- subset rows — X[id != “a”,]
- select columns — X[, val]
- *compute on* columns — X[, mean(val)]
- subset rows & select / *compute on* columns — X[id != “a”, mean(val)]
- *virtual* 3rd dimension — group by

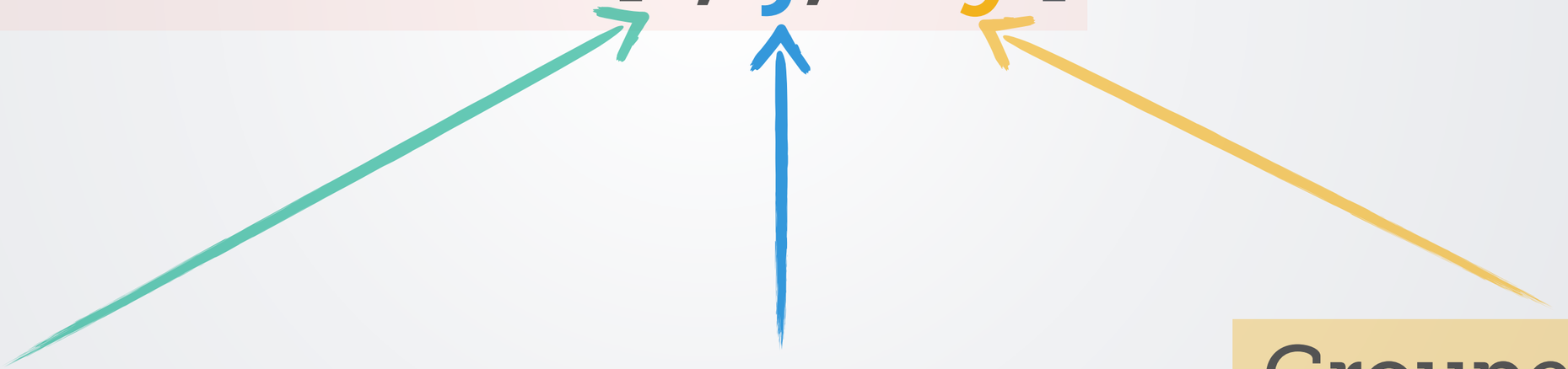
X

	id	val
1:	b	4
2:	a	2
3:	a	3
4:	c	1
5:	c	5
6:	b	6

DATA TABLES

- think in terms of basic units — rows, columns and groups
- data.table syntax provides *placeholder* for each of them

General form: DT[i, j, by]



On which rows

What to do?

Grouped by what?

EQUIVALENT DATA TABLE CODE

```
sum(DF[DF$code != “abd”, “valA”])
```

```
DT[code != “abd”, sum(valA)]
```

```
aggregate(cbind(valA, valB) ~ id,
          DF[DF$code != “abd”, ],
          sum)
```

```
DT[code != “abd”,
   .(sum(valA), sum(valB)),
   by = id]
```

```
DF[DF$code == “abd”, “valA”] <- NA
```

```
DT[code == “abd”, valA := NA]
```

JOINS - AS SUBSETS

A

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	5
4:	2	apq	0.9	10
5:	2	apq	0.3	13

B

	id	code
1:	1	abd
2:	2	apq

Join data.tables A and B on id, code

```
A[B, on = .(id, code)]
```

on which rows?

JOINS - AS SUBSETS

A

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	5
4:	2	apq	0.9	10
5:	2	apq	0.3	13

B

	id	code
1:	1	abd
2:	2	apq

	id	code	valA	valB
1:	1	abd	1.5	5
2:	2	apq	0.9	10
3:	2	apq	0.3	13

A[B, on = .(id, code)]

on which rows?

Why joins as subsets?

UPDATE WHILE JOIN

A

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	5
4:	2	apq	0.9	10
5:	2	apq	0.3	13

B

	id	code	mul
1:	1	abd	2.0
2:	2	apq	0.5
3:	3	abc	1.7

Update valA with valA*mul while matching on id, code

UPDATE WHILE JOIN

A

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	3.0	5
4:	2	apq	0.45	10
5:	2	apq	0.15	13

B

	id	code	mul
1:	1	abd	2.0
2:	2	apq	0.5
3:	3	abc	1.7

Update valA with valA*mul while matching on id, code

```
A[B, on = .(id, code),
    valA := valA * mul]
```

on which rows?
what to do?

AGGREGATE WHILE JOIN

A

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	5
4:	2	apq	0.9	10
5:	2	apq	0.3	13

B

	id	code	mul
1:	1	abd	2.0
2:	2	apq	0.5
3:	3	abc	1.7

Get min(valA) from A while matching with B on id, code

AGGREGATE WHILE JOIN

A

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	5
4:	2	apq	0.9	10
5:	2	apq	0.3	13

B

	id	code	mul
1:	1	abd	2.0
2:	2	apq	0.5
3:	3	abc	1.7

Get min(valA) from A while matching with B on id, code

```
A[B, on = .(id, code),
    min(valA),
    by = .EACHI]
```

on which rows?
what to do?
grouped by what?

AGGREGATE WHILE JOIN

A

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	5
4:	2	apq	0.9	10
5:	2	apq	0.3	13

B

	id	code	mul
1:	1	abd	2.0
2:	2	apq	0.5
3:	3	abc	1.7

	id	code	valA
3:	1	abd	1.5
4:	2	apq	0.3
5:	3	abc	NA

```
A[B, on = .(id, code),
    min(valA),
    by = .EACHI]
```

on which rows?
what to do?
grouped by what?

UPDATE+AGGREGATE WHILE JOIN

A

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	1.5	5
4:	2	apq	0.9	10
5:	2	apq	0.3	13

B

	id	code	mul
1:	1	abd	2.0
2:	2	apq	0.5
3:	3	abc	1.7

Update valA with cumsum(valA)*mul while matching on id, code

UPDATE+AGGREGATE WHILE JOIN

A

	id	code	valA	valB
1:	1	abc	0.1	11
2:	1	abc	0.6	7
3:	1	abd	3.0	5
4:	2	apq	0.45	10
5:	2	apq	0.60	13

B

	id	code	mul
1:	1	abd	2.0
2:	2	apq	0.5
3:	3	abc	1.7

Update valA with cumsum(valA)*mul while matching on id, code

```
A[B, on = .(id, code),
    valA := cumsum(valA) * mul
    by = .EACHI]
```

TALK OVERVIEW

- data.table's philosophy
  - concise + straightforward code
  - fast + memory efficient
- optimisations, and new features in v1.10.0

NON-EQUI JOINS

A

	x	y	val
1:	1	2	0.2
2:	2	3	0.1
3:	2	4	0.7
4:	2	6	0.4
5:	4	5	0.5
6:	4	5	0.6
7:	4	10	0.3

B

	x	y
1:	2	6
2:	1	12

For each row in B replace A$z where A$x >= B$x & A$y >= B$y with NA

NON-EQUI JOINS

A

	x	y	val
1:	1	2	0.2
2:	2	3	0.1
3:	2	4	0.7
4:	2	6	NA
5:	4	5	0.5
6:	4	5	0.6
7:	4	10	NA

B

	x	y
1:	2	6
2:	1	12

For each row in B replace A$z where A$x >= B$x & A$y >= B$y with NA

```
A[B, on = .(x >= x, y >= y),
    val := NA]
```

NEW LOGO

Get yours on stickermule

Github #1237

FWRITE - PARALLEL FILE WRITER

		Laptop SSD 4core/16gb 10m rows		Server 32core/256gb 100m rows		
		Time Sec	Size GB	RamDisk Time	HDD Time	Size GB
`fwrite(DT,"fwrite.csv")`	csv	2	0.8	9	61	7.5
`write_feather(DT, "feather.bin")`	bin	5	1.0	27	75	9.1
`save(DT,file="save1.Rdata",compress=F)`	bin	11	1.2	90	137	12.0
`save(DT,file="save2.Rdata",compress=T)`	bin	70	0.4	647	679	2.8
`write.csv(DT,"write.csv.csv",**)`	csv	63	0.8	749	824	7.3
`readr::write_csv(DT,"write_csv.csv")`	csv	132	0.8	1997	1571	7.3

[**] row.names=F,quote=F

SOURCE: http://blog.h2o.ai/2016/04/fast-csv-writing-for-r/

FWRITE - HANDLES LIST COLUMNS

sep2 For columns of type `list` where each item is an atomic vector, `sep2` controls how to separate items *within* the column. `sep2[1]` is written at the start of the output field, `sep2[2]` is placed between each item and `sep2[3]` is written at the end. `sep2[1]` and `sep2[3]` may be any length strings including empty `""` (default). `sep2[2]` must be a single character and (when `list` columns are present and therefore `sep2` is used) different from both `sep` and `dec` . The default (`|`) is chosen to visually distinguish from the default `sep` . In speaking, writing and in code comments we may refer to `sep2[2]` as simply "sep2".

Matt Dowle @MattDowle · Jan 18

fwrite can write list columns. Its secondary separator, sep2, is by default c("","|",""). #rstats #rdatatable

2 7 23

FWRITE - HANDLES DATE/TIME FORMAT

fwrite has 4 options for writing dates and times in parallel: ISO, squash, epoch or as write.csv #rstats #rdatatable

R: data.table. How to save dates properly with fwrite?

I have a dataset. I can choose to load it on R from a Stata file or from a SPSS file. In both cases it's loaded properly with the haven package. The dates are recognized properly. But when ...

stackoverflow.com

FSORT - PARALLEL SORT

length	size in RAM	threads	base R	v1.10.0
500m	3.8GB	8	65s	3.9s
1b	7.6GB	32	140s	3.5s
10b	76GB	32	25m	48s

SOURCE: https://www.r-project.org/dsc/2016/slides/ParallelSort.pdf

%BETWEEN%

x %between% c(2000, 20000) length(x) = 500e6, int, ~1.9GB		
v1.9.6	15.7s	7.2GB
v1.10.0 (C, parallelised)	1.1s (4 threads)	3.8GB
	run time	peak memory

GFORCE OPTIMISATIONS

DT[id > 1e3L, lapply(.SD, median), by=id] dim(DT) = 20e6 * 15	
v1.9.6	58s
v1.10.0	19s
	run time

OTHER DEVELOPMENTS

- 205 issues closed, 75+ bug fixes, 30+ features, including ...
  - parallel subsets
  - parallel setkey / setorder (the reordering part)
  - new functions `rowid()`, `rleid()`, `inrange()` etc.

QUESTIONS?

Give data.table a go :-)