



THE #RDATA TABLE PACKAGE

for fast, flexible and memory efficient data wrangling

Arun Srinivasan
CO-DEVELOPER, DATA.TABLE

~ A YEAR BEFORE

- Homepage: <http://r-datatable.com>, 50 contributors
- Since 2006 on CRAN, >40 releases so far
- Does not depend/import any other packages
- >7700 unit tests, ~93% coverage (using covr)
- >600 packages import/depend/suggest data.table
- ~18.3 packages per month since Jan'18
- 10th most starred R package on Github (METACRAN)
- >7400 questions on StackOverflow

NOW

- Homepage: <http://r-datatable.com>, ~~50~~ 69 contributors
- Since 2006 on CRAN, ~~>40~~ 48 releases so far
- Still does not depend/import any other packages
- ~~>7700~~ >9100 unit tests, ~~~93%~~ ~98% coverage (using covr)
- ~~>600~~ >690 packages import/depend/suggest data.table
- 15th most depended upon packages (METACRAN)
- ~~10th~~ 9th most starred R package on Github (METACRAN)
- ~~>7400~~ >8800 questions on StackOverflow
- >370 issues closed/fixed since then (277 so far in 2019)

TALK OVERVIEW

- Introduce data.table's syntax / general form with simple examples
- Slightly more involved example to help understand `.SD` and `.SDcols`
- Optimisations and new functionalities in data.table

TALK OVERVIEW

- Introduce data.table's syntax / general form with simple examples
- Slightly more involved example to help understand .SD and .SDcols
- Optimisations and new functionalities in data.table

EXAMPLE 1

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum(valA)**
for each value of **id**

```
DT[, .(valA=sum(valA)), by=id]
```

.() is an alias to *list()*

EXAMPLE 1

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum(valA)**
for each value of **id**

```
DT[, .(valA=sum(valA)), by=id]
```

.() is an alias to list()



EXAMPLE 1

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum(valA)**
for each value of **id**



```
DT[, .(valA=sum(valA)), by=id]
```

.() is an alias to list()

Group 1 Group 2

id:1	valA	id:2	valA
	1		5
	2		6
	3		8
	4		
	7		
	9		

EXAMPLE 1 CONTD ...

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum(valA)**
for each value of **id**

	id	valA
1:	1	26
2:	2	19

```
DT[, .(valA=sum(valA)), by=id]
```

EXAMPLE 2

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum(valA)**
for each value of **id**
where **code != "b"**

```
DT[code != "b", .(valA=sum(valA)), by=id]
```

EXAMPLE 2

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum(valA)**
for each value of **id**
where **code != "b"**

```
DT[code != "b", .(valA=sum(valA)), by=id]
```



EXAMPLE 2

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum(valA)**
for each value of **id**
where **code != "b"**

```
DT[code != "b", .(valA=sum(valA)), by=id]
```



EXAMPLE 2

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum(valA)**
for each value of **id**
where **code != "b"**

Group 1 Group 2

id:1	valA	id:2	valA
	1		5
	3		6
	4		8
	9		



```
DT[code != "b", .(valA=sum(valA)), by=id]
```

EXAMPLE 2 CONTD ...

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum(valA)**
for each value of **id**
where **code != "b"**

	id	valA
1:	1	17
2:	2	19

```
DT[code != "b", .(valA=sum(valA)), by=id]
```

GENERAL FORM

```
DT[code != "b", .(valA=sum(valA)), by=id]
```

DT[**i**, **j**, **by**]

On which
rows

What to do?

Grouped
by what?

TALK OVERVIEW

- Introduce data.table's syntax / general form with simple examples
- Slightly more involved example to help understand .SD and .SDcols
- Optimisations and new functionalities in data.table

.SD AND .SDCOLS

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum** of each of **val**
cols for each **id** where
code != "b"

```
DT[code != "b", lapply(.SD, sum), by=id,  
  .SDcols=patterns("^val")]
```

.SD AND .SDCOLS

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum** of each of **val**
cols for each **id** where
code != "b"

```
DT[code != "b", lapply(.SD, sum), by=id,  
  .SDcols=patterns("^val")]
```



.SD AND .SDCOLS

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum** of each of **val**
cols for each **id** where
code != "b"

```
DT[code != "b", lapply(.SD, sum), by=id,  
  .SDcols=patterns("^val")]
```



.SD AND .SDCOLS

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum** of each of **val**
cols for each **id** where
code != "b"

Group 1				Group 2			
id:1	code	valA	valB	id:2	code	valA	valB
	c	1	10		a	5	14
	c	3	12		a	6	15
	c	4	13		a	8	17
	c	9	18				

Subset
of Data

```
DT[code != "b", lapply(.SD, sum), by=id,  
  .SDcols=patterns("^val")]
```

.SD AND .SDCOLS

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum** of each of **val**
cols for each **id** where
code != "b"

Group 1

id:1	valA	valB
	1	10
	3	12
	4	13
	9	18

Group 2

id:2	valA	valB
	5	14
	6	15
	8	17

Subset
of Data

```
DT[code != "b", lapply(.SD, sum), by=id,  
  .SDcols=patterns("^val")]
```



.SD AND .SDCOLS CONTD ...

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum** of each of **val**
cols for each **id** where
code != "b"

	id	valA	valB
1:	1	17	53
2:	2	19	46

```
DT[code != "b", lapply(.SD, sum), by=id,  
  .SDcols=patterns("^val")]
```

.SD AND .SDCOLS CONTD ...

	id	code	valA	valB
1:	1	c	1	10
2:	1	b	2	11
3:	1	c	3	12
4:	1	c	4	13
5:	2	a	5	14
6:	2	a	6	15
7:	1	b	7	16
8:	2	a	8	17
9:	1	c	9	18

Get **sum** of each of **val**
cols for each **id** where
code != "b"

	id	valA	valB
1:	1	17	53
2:	2	19	46

```
DT[code != "b", lapply(.SD, sum), by=id,  
  .SDcols=patterns("^val")]
```

New vignette in devel version 1.12.3: `vignette("datatable-sd-usage")`

TALK OVERVIEW

- Introduce data.table's syntax / general form with simple examples
- Slightly more involved example to help understand .SD and .SDcols
- Optimisations and new functionalities in data.table

HOW TO BE PERFORMANT?

```
DT[code != "b", .(valA=sum(valA)), by=id]
```

DT[**i**, **j**, **by**]

On which
rows

What to do?

Grouped
by what?

No shortcuts ... Optimise every single stage

OPTIMISATIONS IN 'I'

Auto indexing:

```
dt <- data.table(x=sample(1e5, 2e8, TRUE), y=runif(2e8))  
200 million rows, 2 cols, ~3GB
```

Run 1: `dt[x %in% 1000:2000]`

4.9s

Run 2: Reuses **index** built during 1st run

.6s

Optimised for `==` and `%in%`

TODO: Other operators

run time

OPTIMISATIONS IN 'I'

Parallel subsets (columns are processed in parallel):

```
dt <- setDT(lapply(1:20, function(x) sample(100, 5e7, TRUE)))
```

50 million rows, 20 cols, all Integers, ~3.7GB

Parallel (default): `dt[V1 > 50L]`

.59s

Sequential: with `setDTthreads(1L)`

1.01s

**Every bit of optimisation matters
(1.7x speedup on 2 threads)**

run time

OPTIMISATIONS IN 'BY'

Radix order has been parallelised recently:

```
dt <- data.table(x=sample(1e5, 2e8, TRUE), y=runif(2e8))  
200 million rows, 2 cols, ~3GB
```

`dt[, .N, by=x]` (with 2 threads)

5.43s

Using 1 thread with `setDTthreads(1L)`

10.42s

~1.92x speedup

run time

OPTIMISATIONS IN 'J'

GForce: sum, min, max, mean, median, head, tail etc.

```
dt <- setDT(lapply(1:20, function(x) sample(100, 5e7, TRUE)))  
50 million rows, 20 cols, ~3.7GB
```

<code>dt[, lapply(.SD, mean), by=V1]</code>	2.8s
<code>dt[, lapply(.SD, base::mean), by=V1]</code>	6.8s

run time

MORE FUNCTIONALITIES

nafill:

	V1	V2	V3	V4
1:	1	c	NA	10
2:	2	b	2	NA
3:	NA	c	3	NA
4:	1	NA	4	NA
5:	2	NA	5	14

```
nafill(DT, "locf")
```

```
setnafill(DT, "locf")
```

*Operates in parallel
across cols, see
?nafill*

	V1	V2	V3	V4
1:	1	c	NA	10
2:	2	b	2	10
3:	2	c	3	10
4:	1	c	4	10
5:	2	c	5	14

MORE FUNCTIONALITIES

nafill:

```
dt <- setDT(lapply(1:20, function(x) sample(c(NA, 1:10), 5e7, TRUE)))
```

50 million rows, 20 cols, ~3.7GB

```
nafill(dt, "locf") # 2 threads
```

1.1s

```
nafill(dt, "locf") # 1 thread
```

2.7s

run time

Can also be used with 'by', e.g.,

```
DT[, if (cond) nafill(.SD, "locf") else .SD, by=col]
```

MORE FUNCTIONALITIES

froll: frollsum, frollmean

	V1	V2	V3
1:	1	6	11
2:	2	7	12
3:	3	8	13
4:	4	9	14
5:	5	10	15

```
setDT(frollsum(DT, 3))
```

*Operates in parallel
across cols, see
?froll*

	V1	V2	V3
1:	NA	NA	NA
2:	NA	NA	NA
3:	6	21	36
4:	9	24	39
5:	12	27	42

MORE FUNCTIONALITIES

coalesce:

	x	y	z
1:	11	NA	11
2:	NA	12	NA
3:	13	5	1
4:	NA	NA	14
5:	15	NA	NA
6:	NA	NA	NA

`coalesce(DT)`

11	12	13	14	15	NA
----	----	----	----	----	----

*Operates in parallel, see
?coalesce*

SUMMARY

- Auto indexing improvements in “i” for other operators could be a great feature to add.
- More performance might be squeezed out of (parallel) radix ordering for grouping operations by handling special cases more efficiently.
- Several new compute functionalities are being added. They can be hooked in “j” via GForce for quicker group-by operations.
- New vignettes added and several bugs fixed.

THANKS TO

- More contributions, activity, and issues fixed / features implemented.
- `fwrite()` can write to zip file directly - Philippe Chataignon
- Markus Bonsch on quite a few issues including auto indexing
- Hugh Parsonage, Frank, Uwe, Shrektan and others- routine filing of issues, following up on SO, and fixing issues/suggesting fixes etc.
- Renkun-ken and a few others for active dev testing and reporting

THANKS TO

- Pasha from H2O.ai and H2O.ai for their contributions and interest in data.table
- CRAN team, Github CI/R developers for providing the means to catch and fix issues quickly

THANKS TO

- Michael Chirico for active contributions on documentations, R-code and recently C-code too! `CJ()` has been moved to C and parallelised.
- Michael has also written a much requested vignette on `.SD` usage. Please check out `vignette("datatable-sd-usage")` from v1.2.3+

THANKS TO

- Jan has implemented several recent functionalities - `coalsce`, `nafill`, `frollmean`, `frollsum` etc. in C/OpenMP
- A lot other contributors that I've not been able to mention by name (see: <https://github.com/Rdatatable/data.table/graphs/contributors>)
- And of course, **Matt** for steering this ship for over 13 years :-) (fread, fwrite, parallel radix ordering etc.)

THANKS TO

you for your attention ...

Questions?