THE #RDATATABLE PACKAGE

for fast, flexible and memory efficient data wrangling

Arun Srinivasan

SATRDAY, SEP'16

@ARUN_SRINIV

THE DATA.TABLE PACKAGE

1. Fast, memory efficient, feature rich in-memory data wrangling tool with flexible and consistent syntax
2. v1.0 pushed to CRAN on April 2006.
3. >30 releases since then. v1.9.6 is the current version.
4. 260+ packages import/depend on data.table as of today.
5. >4000 [data.table] tagged Q on SO.
6. Github project page and benchmarks

DATA.TABLE GOALS

Goal 1: Reduce programming time
(fewer function calls, less variable name repetition)

Goal 2: Reduced computing time
(fast file reader, aggregations, equi joins, rolling joins,
overlapping range joins, update by reference)

EXERCISE 1

```
DF1 = data.frame(id = c(1,1,1,2,2),
                 code = c("abc", "abc", "abd", "apq", "apq"),
                 valA = c(0.1, 0.6, 1.5, 0.9, 0.3),
                 valB = c(11, 7, 5, 10, 13),
                 stringsAsFactors = FALSE)
```

1. Get **`sum(valA)`** for **`code != "abd"`**
2. Get **`sum(valA)`** and **`sum(valB)`** grouped by **`id`**
3. Get **`sum(valA)`** and **`mean(valB)`** grouped by **`id`** for **`code != "abd"`**
4. Replace **`valB`** with **`NA`** where **`code == "abd"`**
5. Get **`max(valA)-min(valA)`** grouped by **`code`**
6. Get **`sum(valA)`** and **`sum(valB)`** grouped by **`id`** and **`code`**

`[.DATA.FRAME`

The general form of `[.data.frame` is:

```
DF[i, j] + drop
```

`DF[DF$code == 3L,]` — i, subset rows

j, select columns — `DF[, c("vA", "vB")]`

	code	vA	vB
1	3	1	6
5	3	5	10

	vA	vB
1	1	6
2	2	7
3	3	8
4	4	9
5	5	10
6	6	11

Other methods

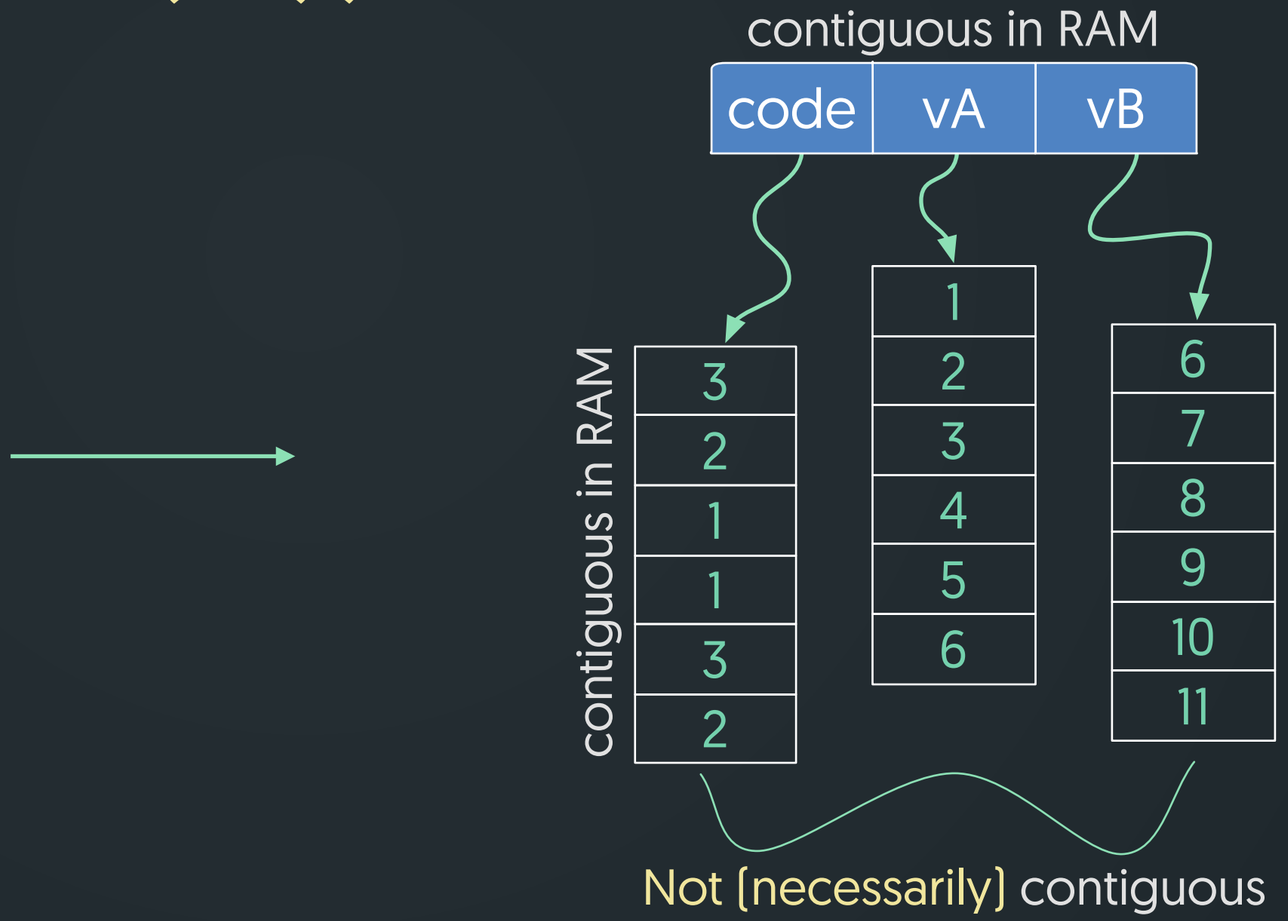
duplicated, unique
complete.cases, na.omit
split, lapply, rbind
tapply, ave, apply
get, mget
reshape, merge

INTERNAL REPRESENTATION

Column based storage, i.e., a vector of column pointers

```
.Internal(inspect(DF))
```

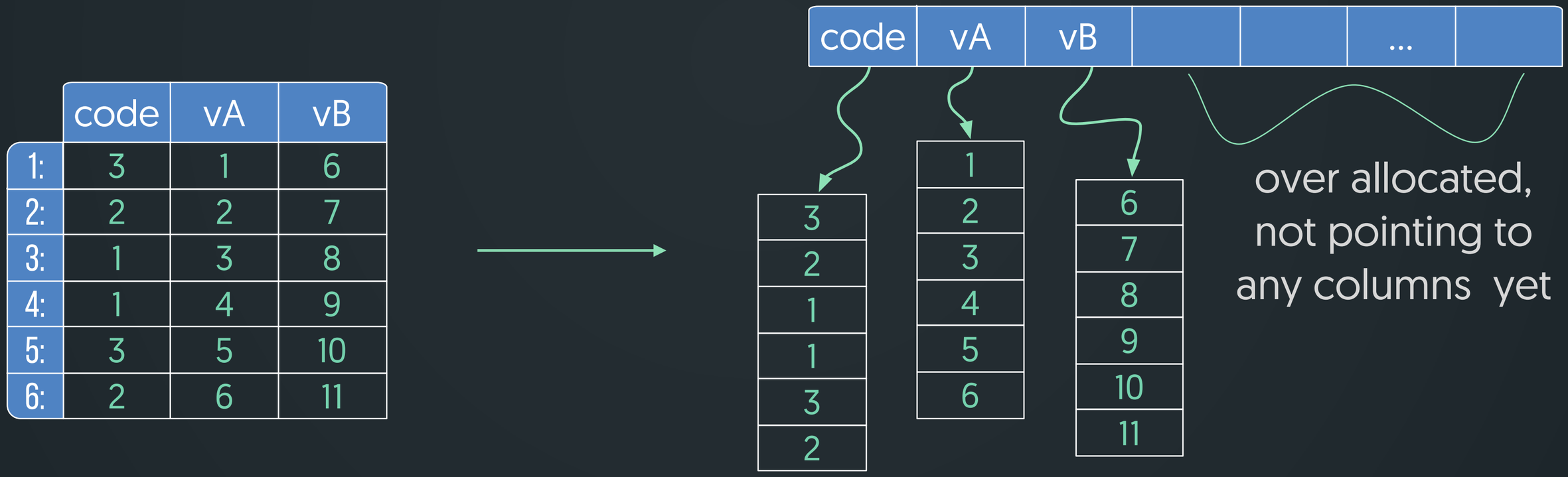
	code	vA	vB
1	3	1	6
2	2	2	7
3	1	3	8
4	1	4	9
5	3	5	10
6	2	6	11



INTERNAL REPRESENTATION

Column based storage. Over allocated vector of column pointers

```
.Internal(inspect(DT))
truelength(DT)
```



`[.DATA.TABLE`

The general form of `[.data.table` is:

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

R:	i	j	by
SQL:	WHERE	SELECT	GROUP BY

Other arguments

with, which
on, mult, nomatch
roll, rollends
.SD, .SDcols
allow.cartesian

LOAD ‘FLIGHTS’

```
require(data.table)
flights = fread("flights_2014.csv")
```

EXERCISE 2

1. Write down two differences in the way data.table prints to screen in comparison to data.frames - `print(flights)`.

Row subsets

LOGICAL

Extract all rows where code = 3

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT[code == 3L]
```

doesn't preserve row names

	code	vA	vB
1:	3	1	6
2:	3	5	10

```
DF[DF$code == 3L, ]
```

	code	vA	vB
1	3	1	6
5	3	5	10

INTEGER/NUMERIC

Subset 4th and 3rd rows in that order

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT[4:3]
```

doesn't preserve row names

	code	vA	vB
1:	1	4	9
2:	1	3	8

```
DF[4:3, ]
```

	code	vA	vB
4	1	4	9
3	1	3	8

USING ROW NAMES?

Subset last row using row names

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT["6"]
```

```
DF["6", ]
```

No rownames

	code	vA	vB
6	2	6	11

EXERCISE 3

Find all flights:

1. From **`JFK`** to **`MIA`**
2. From any **`origin`** airport except **`JFK`**

3. Order by **`origin`** airport and then in descending `order` by **`dest`** airport.
4. Get all rows where **`dest`** airport begins with **`I`** or **`J`**.
5. Get the last row.

Column subsets

SELECT - DATA.TABLE WAY

Select column vB and return a vector

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT[, vB]
```

```
???
```

columns are seen as if they are variables

6	7	8	9	10	11

SELECT - DATA.TABLE WAY

Select column vB and return a data.table

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

. is an alias to list

```
DT[, list(vB)]
```

or

```
DT[, .(vB)]
```

```
???
```

vB
6
7
8
9
10
11

Wrapping with `.()` or `list()` always returns a data.table

SELECT - DATA.TABLE WAY

Select all v* columns. Name vA to vC in result

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT[, .(vC = vA, vB)]
```

```
???
```

vC	vB
1	6
2	7
3	8
4	9
5	10
6	11

Wrapping with `.()` or `list()` always returns a data.table

CAN ALSO COMPUTE IN J

Calculate sum(vA) and return the result as a vector

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

j can accept expressions

```
DT[, sum(vA)]
```

21

```
sum(DF[, "vA"])
```

21

SELECT - DATA.FRAME WAY?

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

j resets to data.frame mode

```
DT[, "vA", with = FALSE]
```

```
DF[, "vA", drop = FALSE
```

vB
6
7
8
9
10
11

EXERCISE 4

From **`flights`**

1. Select **`arr_delay`** column and return as a vector
2. Same as (1) but return a data.table instead
3. Select both columns **`arr_delay`** and **`dep_delay`**
4. Get the total number of rows in 'flights' the data.table way.
5. Compute the median of both **`arr_delay`** and **`dep_delay`** columns
6. Same as (5), but in addition name them **`med_arr_delay`** and **`med_dep_delay`**

Subset rows *and* select columns

7. Select columns **`origin`**, **`dest`** and **`carrier`** for the **`month`** of June both the data.table and the data.frame way.
8. For those flights with **`JFK`** as the **`origin`** airport, calculate the total number of flights where the **`total delay`** is **`> 0`**.

Group by

GROUPED OPERATIONS

Calculate mean(vA) grouped by code

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

adhoc grouping (order preserved)

```
DT[, .(mean(vA)), by = .(code)]
```

	code	V1
1:	3	3
2:	2	4
3:	1	3.5

GROUPED OPERATIONS

Calculate mean(vA) grouped by code

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

shortcut when only one column

```
DT[, mean(vA), by = code]
```

```
DT[, mean(vA), by = c("code")]
```

by also accepts character vector

	code	V1
1:	3	3
2:	2	4
3:	1	3.5

GROUPED OPERATIONS

Altogether now: Take DT, subset rows in i, then calculate j grouped by by.

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT[code < 3, sum(vA),
   by = code]
```

??

GROUPED OPERATIONS

Altogether now: Take DT, subset rows in i, then calculate j grouped by by.

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT[code < 3, sum(vA),
      by = code]
```

	code	V1
1:	2	8
2:	1	7

How do you name the grouping and aggregated column?

EXPRESSIONS IN BY

by also accepts expressions in its `list()` form.

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT[, mean(vA),
by = .(code > 2)]
```

	code	V1
1:	TRUE	3
2:	FALSE	3.75

How do you name the grouping column to “`exp`”?

AGGREGATIONS ON MULTIPLE COLUMNS

.SD contains the Subset of Data for each group and is itself a data.table.
By default, contains all columns except grouping columns

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT[, lapply(.SD, mean),
      by = code]
```

	code	vA	vB
1:	3	3	8
2:	2	4	9
3:	1	3.5	8.5

The columns in .SD can be specified using argument .SDcols
which takes a character vector.

CHAINING

We can tack or chain operations together.

	code	vA	vB
1:	3	1	6
2:	2	2	7
3:	1	3	8
4:	1	4	9
5:	3	5	10
6:	2	6	11

```
DT[, .(mA = mean(vA),
by = code][order(-mA)]
```

DT[..][..][..]

	code	mA
1:	2	4
2:	1	3.5
3:	3	3

EXERCISE 5

1. Calculate average `dep_delay` for each `origin`
2. Find the first three `months` with lowest average `arr_delay`?
3. Get the total number of trips for each `origin,dest` pair in a) ascending and b) descending order.
4. Get the number of trips for each `origin,dest` pair over each `month`
5. Get the mean of arrival and departure delays for carrier == “AA” for every month for every origin,dest pair using `lapply, .SD` and `.SDcols`

REVISITING EXERCISE 1

```
DF1 = data.frame(id = c(1,1,1,2,2),
                 code = c("abc", "abc", "abd", "apq", "apq"),
                 valA = c(0.1, 0.6, 1.5, 0.9, 0.3),
                 valB = c(11, 7, 5, 10, 13),
                 stringsAsFactors = FALSE)
```

1. Get **`sum(valA)`** for **`code != "abd"`**
2. Get **`sum(valA)`** and **`sum(valB)`** grouped by **`id`**
3. Get **`sum(valA)`** and **`mean(valB)`** grouped by **`id`** for **`code != "abd"`**
4. Replace **`valB`** with **`NA`** where **`code == "abd"`**
5. Get **`max(valA)-min(valA)`** grouped by **`code`**
6. Get **`sum(valA)`** and **`sum(valB)`** grouped by **`id`** and **`code`**