



# Data Tables

## An introduction (and pitch)

Gene Leynes  
Chicago R User Group  
February 7, 2013

(Or How I learned to stop  
worrying and love data.table)

# Simple message: forget data.frame use data.table everywhere

- Data tables are fast stable and ready for prime time
  - Used to be buggy, but not anymore
- Data table package is maintained by Matthew Dowle, written by Dowle, Tom Short and Steve Lianoglou
- Used at Google
- Logic is more portable

## R Reference Card

by Tom Short, EPRI PEAC, tshort@epri-peac.com 2004-11-07  
Granted to the public domain. See [www.Rpad.org](http://www.Rpad.org) for the source and latest version. Includes material from *R for Beginners* by Emmanuel Paradis (with permission).

### Getting help

Most R functions have online documentation.  
**help(topic)** documentation on topic  
**?topic** id.  
**help.search("topic")** search the help system  
**apropos("topic")** the names of all objects in the search list matching the regular expression "topic"  
**help.start()** start the HTML version of help  
**str(a)** display the internal "structure" of an R object  
**summary(a)** gives a "summary" of a, usually a statistical summary but it is generic meaning it has different operations for different classes of a  
**ls()** show objects in the search path; specify pat="pat" to search on a pattern  
**ls.str()** str() for each variable in the search path  
**dir()** show files in the current directory

character or factor columns are surrounded by quotes (' field separator; eol is the end-of-line separator; na is missing values; use col.names=0 to add a blank coliar get the column headers aligned correctly for spreadsheet  
**sink(file)** output to file, until sink()  
Most of the I/O functions have a file argument. This can often be a string naming a file or a connection. file="" means the stan output. Connections can include files, pipes, zipped files, and R On windows, the file connection can also be used with des "clipboard". To read a table copied from Excel, use  
x <- read.delim("clipboard")  
To write a table to the clipboard for Excel, use  
write.table(x,"clipboard",sep="\t",col.names=NA)  
For database interaction, see packages RODBC, DBI, RMySQL, ROracle. See packages XQL, hdf5, netCDF for reading other file  
**Data creation**  
**c(...)** generic function to combine arguments with the default vector; with recursive=TRUE descends through lists elements into one vector  
**from:** to generates a sequence; "" has operator priority; 1:4 +  
**seq(from,to)** generates a sequence by- specifies increment specifies desired length  
**seq(along=x)** generates 1, 2, ..., length(along); us loops

# Pros and Cons

## PROS

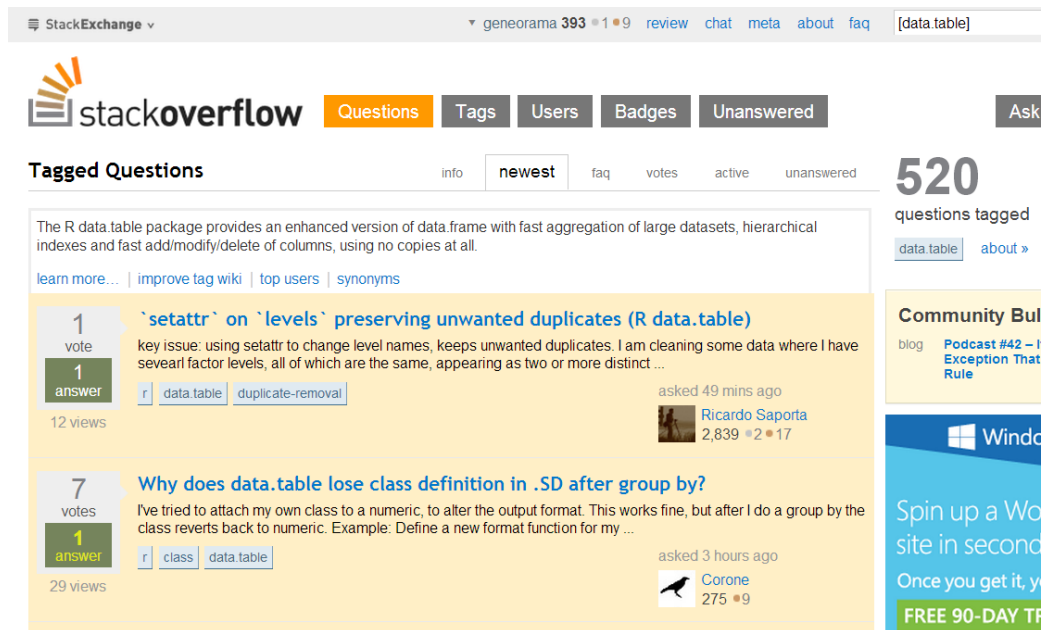
- Data tables are also data frames, so methods that use `data.frame` can also use `data.table`
- Data tables are much faster, especially on large data sets. This might not matter to you now, but it will eventually. When it does matter, you don't want to be forced to learn something new.
- Data tables support complex queries that lead to more compact and readable code
- Queries are easier to read in data tables compared to any other type of summary / aggregates / queries
- Similar to SQL (especially because of `group by`)

## CONS

- Data tables are definitely harder to use
- Data tables are not as widely adopted as data frames (yet)
  - Can be a challenge for sharing code
  - Fewer users means few testers
  - Fewer users also means fewer examples

# Great Resources

- Stack overflow is the best place for quickly finding answers to just about any data.table problem
- Inside the package
  - Introduction is brief, but
  - FAQ is very useful
- Mailing list is also nice
- <http://datatable.r-forge.r-project.org/>



The screenshot shows the Stack Overflow interface. At the top, there's a search bar with the text "[data.table]". Below the search bar, the Stack Overflow logo is visible, along with navigation tabs for "Questions", "Tags", "Users", "Badges", and "Unanswered". The main content area is titled "Tagged Questions" and shows a list of questions related to "data.table". The first question is titled "`setattr` on `levels` preserving unwanted duplicates (R data.table)" and has 1 vote and 1 answer. The second question is titled "Why does data.table lose class definition in .SD after group by?" and has 7 votes and 1 answer. On the right side, there's a sidebar with a "520 questions tagged" section and a "Community Bulletin" section.

# Main Features

```
MyDataFrame[ i , j ]
```

```
MyDataTable[ i , j , k]
```

## (i) Keys work differently

- Can have multiple keys
- Keys don't have to be unique like in data.frame's row names

## (j) The column argument (optional)

- Can be an “aggregate” like query
- Can be an assignment based on existing columns

## (k) By or Keyby argument (optional)

- Can have multiple keys
- Keys don't have to be unique like in

# Funny Thing...

- The simple things become a little harder (at first)
- But the complex stuff gets a whole lot easier
  
- Awesome FAQ:

## 1.7 This is really hard. What's the point?

`j` doesn't have to be just column names. You can write any R *expression* of column names directly as the `j`; e.g., `DT[,mean(x*y/z)]`. The same applies to `i`; e.g., `DT[x>1000, sum(y*z)]`. This runs the `j` expression on the set of rows where the `i` expression is true. You don't even need to return data; e.g., `DT[x>1000, plot(y,z)]`. When we get to compound table joins we will see how `i` and `j` can themselves be other `data.table` queries. We are going to stretch `i` and `j` much further than this, but to get there we need you on board first with FAQs 1.1-1.6.

# Simple things take some getting used to...

- Indexing is a whole different ball of wax
- You don't need quotes, and quotes will cause problems
- Column indexing by number works, but only if "with=FALSE"

```
## WRONG WAY:  
dat[, 3]
```

```
## [1] 3
```

```
dat[, "race"]
```

```
## [1] "race"
```

```
## RIGHT WAY:  
dat[1:10, race]
```

```
## [1] "WH" "LW" "BK" "BK" "LW" "BK" "BK" "BK" "LW" "BK"
```

```
dat[1:10, 3, with = F]
```

```
##      age_at_booking  
## 1:           26  
## 2:           37  
## 3:           18  
## 4:           32  
## 5:           49  
## 6:           26  
## 7:           41  
## 8:           56  
## 9:           40  
## 10:          20
```

# Example: Cook County Jail Data

- Some data that I was working on for the D3 visualization workshop (I used ggplot, I'm still learning D3)

```
> str(dat)
Classes 'data.table' and 'data.frame': 19207 obs. of 11 variables:
 $ charges_citation      : chr  "720 ILCS 5 12-3.4(a)(2) [16145]" "625 ILCS 5 6-101
 [12935]" "720 ILCS 5 12-3(a)(1) [10529]" "720 ILCS 550 5(c) [5020200]" ...
 $ race                  : chr  "WH" "LW" "BK" "BK" ...
 $ age_at_booking        : int  26 37 18 32 49 26 41 56 40 20 ...
 $ gender                 : chr  "M" "M" "M" "F" ...
 $ booking_date          : POSIXct, format: "2013-01-20" ...
 $ jail_id               : chr  "2013-0120171" "2013-0120170" "2013-0120169" "2013-
 0120167" ...
 $ bail_status           : chr  NA NA NA NA ...
 $ housing_location      : chr  "05-" "05-" "05-L-2-2-1" "17-WR-N-A-2" ...
 $ charges               : chr  NA NA NA NA ...
 $ bail_amount           : int  5000 10000 5000 50000 5000 5000 25000 5000 25000 10000
 ...
 $ discharge_date_earliest: POSIXct, format: NA ...
 - attr(*, ".internal.selfref")=<externalptr>
```



# What does a complex query this look like?

```
mysummary = dat[
  i = !is.na(charges) &
      !is.na(booking_date) &
      !is.na(bail_amount),
  j = list(
    count = .N,
    coverage = diff(range(booking_date)),
    bailave = mean(bail_amount),
    bailsd = sd(bail_amount),
    bailmin = min(bail_amount),
    bailmax = max(bail_amount)
  ),
  by = list(race, gender, age_at_booking)
]
mysummary
```

```
> mysummary
  race gender age_at_booking count coverage bailave bailsd bailmin bailmax
1:  WH     F             36     7 333.0000 days  72285.71 103842.4   1000 300000
2:  BK     F             19    24 630.0417 days 170604.17 604854.3   2000 3000000
3:  WH     M             20    36 267.0417 days 138250.00 295434.6   1000 1500000
4:  LT     M             27    27 829.0417 days 154444.44 205671.2    5000 900000
5:  LT     M             28    25 960.0417 days 292100.00 607990.9    7500 3000000
---
467:  W     M             17     1  0.0000 days 100000.00      NA 100000 100000
468:  B     M             40     1  0.0000 days 1000000.00     NA 1000000 1000000
469:  LB    M             23     1  0.0000 days 250000.00     NA 250000 250000
470:  W     M             51     1  0.0000 days 5000000.00     NA 5000000 5000000
471:  AS    M             60     1  0.0000 days 5000000.00     NA 5000000 5000000
```

# Tips and Tricks

- Printing the DT object gives you top and bottom rows of the data, and column names, which is often more useful than `str()` or `summary()`
- `rbindlist`: very helpful
- Two commands for setting the key
  - `setkey` single key setting, uses objects
  - `setkeyv` multi key setting, uses vector (v=vector)
- Setting the key also sorts the data
- Date time
  - They have something, but it says “still experimental”
  - I prefer to keep it date / times separate
- Numeric keys can cause problems
  - This is an R pitfall related to precision and rounding
- Good code format goes a long way toward readability, I like seeing `i`, `j`, and `k` on separate lines.