News from 1.6, 1.7 & 1.8

Matthew Dowle

LondonR, June 2012
Overview

- Real example
- Review of last presentation 2 years ago
- Package statistics
- New features
- Q&A
I have a data frame that is some 35,000 rows, by 7 columns. It looks like this:

```r
head(nuc)
```

<table>
<thead>
<tr>
<th>chr</th>
<th>feature</th>
<th>start</th>
<th>end</th>
<th>gene_id</th>
<th>pctAT</th>
<th>pctGC</th>
<th>length</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>CDS</td>
<td>67000042</td>
<td>67000051</td>
<td>NM_032291</td>
<td>0.600000</td>
<td>0.400000</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>CDS</td>
<td>67091530</td>
<td>67091593</td>
<td>NM_032291</td>
<td>0.609375</td>
<td>0.390625</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>CDS</td>
<td>67098753</td>
<td>67098777</td>
<td>NM_032291</td>
<td>0.600000</td>
<td>0.400000</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>CDS</td>
<td>67101627</td>
<td>67101698</td>
<td>NM_032291</td>
<td>0.472222</td>
<td>0.527778</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>CDS</td>
<td>67105460</td>
<td>67105516</td>
<td>NM_032291</td>
<td>0.631579</td>
<td>0.368421</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>CDS</td>
<td>67108493</td>
<td>67108547</td>
<td>NM_032291</td>
<td>0.436364</td>
<td>0.563636</td>
</tr>
</tbody>
</table>

gene_id is a factor, that has about 3,500 unique levels. I want to, for each level of gene_id get
the `min(start), max(end), mean(pctAT), mean(pctGC), and sum(length)`.

I tried using `lapply` and `do.call` for this, but it's taking forever +30 minutes to run. The code I'm using is:

```r
nuc_prof = lapply(levels(nuc$gene_id), function(gene){
  t = nuc[nuc$gene_id==gene, ]
  return(list(gene_id=gene, start=min(t$start), end=max(t$end), pctGC = mean(t$pctGC), pct = mean(t$pctAT), cds_length = sum(t$length)))
})
nuc_prof = do.call(rbind, nuc_prof)
```

I'm certain I'm doing something wrong to slow this down. I haven't waited for it to finish as I'm sure it can be faster. Any ideas?
`do.call` can be extremely slow on large objects. I think this is due to how it constructs the call, but I'm not certain. A faster alternative would be the `data.table` package. Or, as @Andrie suggested in a comment, use `tapply` for each calculation and `cbind` the results.

A note on your current implementation: rather than doing the subsetting in your function, you could use the `split` function to break up your `data.frame` into a list of `data.frame`s you can loop over.

```r
g <- function(tnuc) {
  list(gene_id=tnuc$gene_id[1], start=min(tnuc$start), end=max(tnuc$end),
       pctGC=mean(tnuc$pctGC), pct=mean(tnuc$pctAT), cdslength=sum(tnuc$length))
}
nuc_prof <- lapply(split(nuc, nuc$gene_id), g)
```

answered Jun 15 at 16:05

Joshua Ulrich

30.6k 3 34 67
Follow up answer

Since I'm in an evangelizing mood ... here's what the fast `data.table` solution would look like:

```r
library(data.table)
dt <- data.table(nuc, key="gene_id")

dt[,list(A=min(start),
      B=max(end),
      C=mean(pctAT),
      D=mean(pctGC),
      E=sum(length)), by=key(dt)]
```

NB: It isn't just the speed, but the simplicity. It's easy to write and easy to read.
Holy fudge buckets!!! data.table is awesome! That took about 3 seconds for the whole thing!!!

I think that congratulations are well in order for the frankly amazingly well written quick start guide and FAQ. Seriously. Where is the button to make all R and bioconductor packages like this one?

Davy Kavanagh, 15 Jun 2012
Review of presentation
2 years ago

( Including why grouping is fast )

Link to pdf
Since then
Rank all packages by users

<table>
<thead>
<tr>
<th>Rank</th>
<th>CRAN package</th>
<th>Users</th>
<th>AvgVote</th>
<th>NumVotes</th>
<th>Crantastic Rank</th>
<th>Inside-R Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ggplot2</td>
<td>77</td>
<td>4.0</td>
<td>50</td>
<td>7</td>
<td>2018</td>
</tr>
<tr>
<td>2</td>
<td>data.table</td>
<td>60</td>
<td>3.9</td>
<td>49</td>
<td>12</td>
<td>1581</td>
</tr>
<tr>
<td>3</td>
<td>plyr</td>
<td>55</td>
<td>3.7</td>
<td>32</td>
<td>24</td>
<td>1338</td>
</tr>
<tr>
<td>4</td>
<td>reshape</td>
<td>33</td>
<td>3.6</td>
<td>16</td>
<td>31</td>
<td>772</td>
</tr>
<tr>
<td>5</td>
<td>Sim.DiffProc</td>
<td>23</td>
<td>4.3</td>
<td>30</td>
<td>9</td>
<td>825</td>
</tr>
<tr>
<td>6</td>
<td>Sim.DiffProcGUI</td>
<td>23</td>
<td>4.3</td>
<td>30</td>
<td>10</td>
<td>825</td>
</tr>
<tr>
<td>7</td>
<td>lmec4</td>
<td>23</td>
<td>3.6</td>
<td>7</td>
<td>34</td>
<td>363</td>
</tr>
<tr>
<td>8</td>
<td>Hmisc</td>
<td>21</td>
<td>3.9</td>
<td>11</td>
<td>25</td>
<td>422</td>
</tr>
<tr>
<td>9</td>
<td>lattice</td>
<td>19</td>
<td>4.7</td>
<td>4</td>
<td>4</td>
<td>259</td>
</tr>
<tr>
<td>10</td>
<td>RODBC</td>
<td>17</td>
<td>4.2</td>
<td>11</td>
<td>19</td>
<td>520</td>
</tr>
</tbody>
</table>
15 reviews

- "It is much easier to subset, summarize, and investigate data.tables"
- "Improves programming and computing speed"
- "Great library for data.mining"
- "data.table is a perfect combination of usability and speed"
- "The more I use it, the better it gets"
- "A very useful package!"
- "The fast way to do SQL like operations in R"
Reviews continued

- "data.table is fast compared to dplyr and ave"
- "data.table rocks!"
- "Efficient and simple"
- "I use it on a regular basis and I managed to cut computing time dramatically."
- "Amazing package!"
- "Fast"
- "I don't know where I would be w/o data.table"
- "Fast splitting/sorting operations in frames"
datatable-help: posts per month
Other stats

- 24 articles "data.table" on R-bloggers
- 108 bugs fixed, 5 outstanding
- 66 feature requests implemented, 64 left
- 191 items in NEWS 1.6.0-1.8.1
- 2,600 lines of R
- 2,000 lines of C
- 653 unit tests

First released Aug 2008
<table>
<thead>
<tr>
<th>Chris Neff</th>
<th>Prasad Chalasani</th>
<th>Stavros Macrakis</th>
<th>gkaupas</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yike Lu</td>
<td>Helge Liebert</td>
<td>Branson Owen</td>
<td>Juliet Hannah</td>
</tr>
<tr>
<td>user1393348</td>
<td>Dieter Menne</td>
<td>RYogi</td>
<td>Iterator</td>
</tr>
<tr>
<td>Leon Baum</td>
<td>Yang Zhang</td>
<td>Prof Brian Ripley</td>
<td>Allan Engelhardt</td>
</tr>
<tr>
<td>Michael Weylandt</td>
<td>Steven Bagley</td>
<td>Ivo Welch</td>
<td>Sean Creighton</td>
</tr>
<tr>
<td>Christoph Jaeckel</td>
<td>Ivan Zhang</td>
<td>Simon Urbanek</td>
<td>ilprincipe</td>
</tr>
<tr>
<td>Muhammad Waliji</td>
<td>Joshua Ulrich</td>
<td>Luke Tierney</td>
<td>Dennis Murphy</td>
</tr>
<tr>
<td>Josh O'Brien</td>
<td>Eric</td>
<td>Eugene Tyurin</td>
<td>Vanja</td>
</tr>
<tr>
<td>Malcolm Cook</td>
<td>DM</td>
<td>Nicolas Servant</td>
<td>Alexander Peterhansl</td>
</tr>
<tr>
<td>Joseph Voelkel</td>
<td>Damian Betebenner</td>
<td>Jean-Francois Rami</td>
<td></td>
</tr>
<tr>
<td>user1165199</td>
<td>Jim Holtman</td>
<td>Jelmer Ypma</td>
<td></td>
</tr>
<tr>
<td>Karl Ove Hufthammer</td>
<td>Timothee Carayol</td>
<td>Thell Fowler</td>
<td></td>
</tr>
<tr>
<td>Ina</td>
<td>Johann Hibschman</td>
<td>Andreas Borg</td>
<td></td>
</tr>
</tbody>
</table>

In appearance order in NEWS. Special thanks to Chris Neff for weeks of help to solve difficult crash bug Jan 2012.
Hierarchical indexes

- 4 years old (released Aug 2008)
- `setkey(DT, id, date)`
- `setkey(DT, category, id, date)`
- `DT[X]` or `merge(DT,X)`
- But, it was integer columns only
- **NEW**: character and double now ok
  - fractional seconds in POSIXct ok
In R I find myself doing something like this a lot:

\[ \text{adatrame[adatframe$col==something]} <- \text{adatframe[adatframe$col==something]} + 1 \]

- DT[col1==something, col2:=col3+1]
- Easy to write, easy to read
Over allocation (add by reference)

data.frame

cbind

data.table

DT[,newcol:=1]
Delete in place

- `DT[,colname:=NULL]`
- Instant, regardless of size
- By reference, `memmove` internally
- Don't have to copy all but that column
data.table IS copied-on-change by <- as usual

No copy by set* functions (setkey, setnames, setattr)

No copy by :=

When you need a copy, call copy(DT)

Why copy a 20GB data.table, even once.
- Print method now prints head and tail
- Automatic optimization (sum -vs- mean)
- `rbenchmark()` replications default of 100 times overhead. Set to 1 and increase size of data, instead.
- Notice variable name repetition
- `:=` by group now in v1.8.1
Analogous to SQL

- Link to data.table FAQ

DT[where, select | update, group by] [having] [order by] [ ]...[ ]

Compound [] is key reason it's all inside [.data.table]
Not (that) much to learn

- One manual page: ?data.table

- Run example(data.table) at the prompt

- No methods, no functions, just use what you're used to in R
list columns

- Each **cell** can be a different type
- Each **cell** can be vector
- Each **cell** can itself be a data.table

- Combining list columns with i and by
list column example

data.table(x=letters[1:3],
           y=list(1:10,
                  letters[1:4],
                  data.table(a=1:3,b=4:6)))

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: a</td>
<td>1,2,3,4,5,6,</td>
</tr>
<tr>
<td>2: b</td>
<td>a,b,c,d</td>
</tr>
<tr>
<td>3: c</td>
<td>&lt;data.table&gt;</td>
</tr>
</tbody>
</table>
Questions?
Suggestions?
Feedback?

Thank you!

Homepage