useR Vignette:

Grouping & Summarizing Data in R

Greater Boston useR Group
March 3, 2011

by

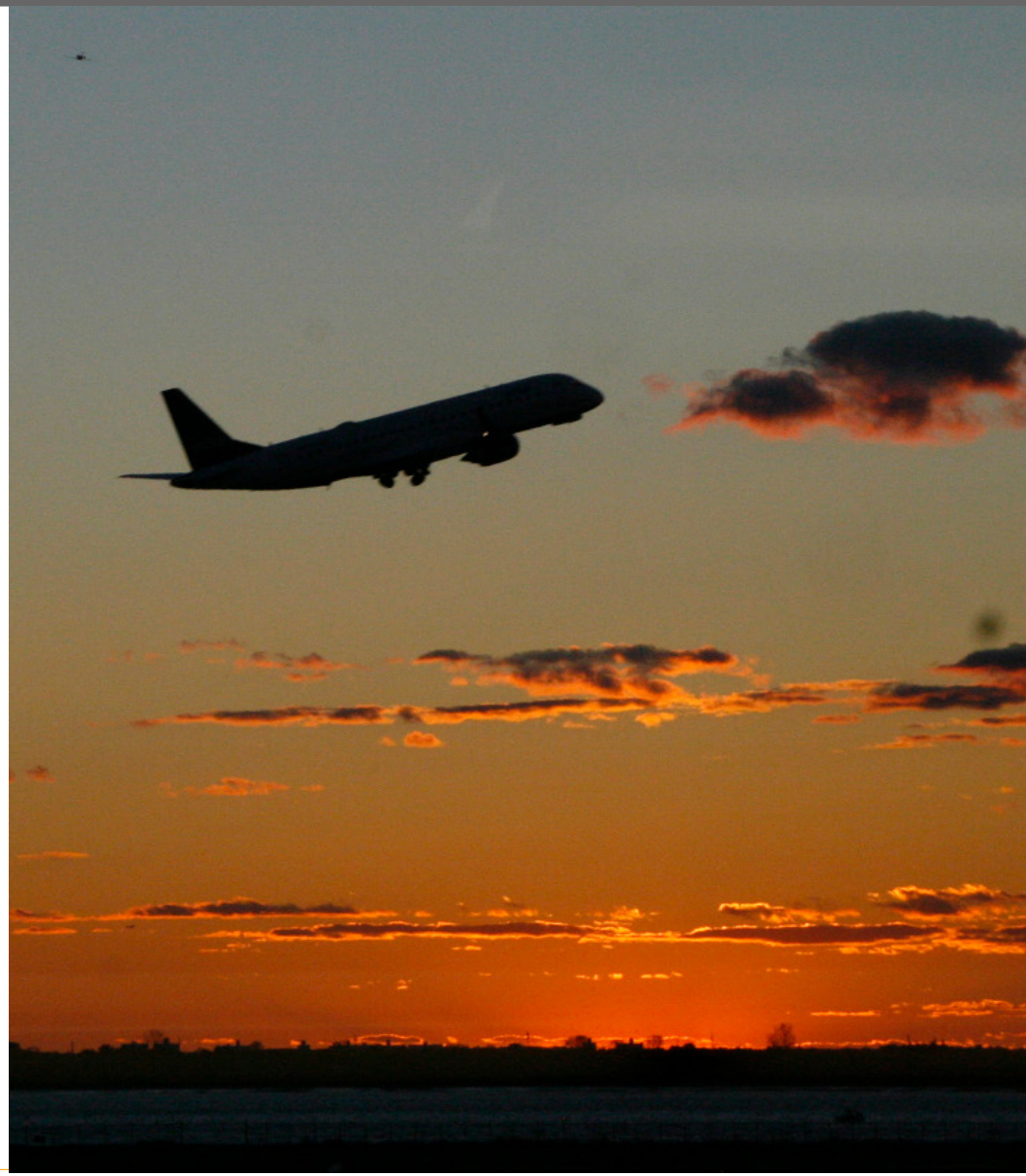
Jeffrey Breen
jbreen@cambridge.aero

Photo: http://flic.kr/p/5HHnRN

#	comp	mass	time	rt_index	chg	S1	S2	S3	S4
0	1	300.130	13.271	218.720	2	35.200	71.000		
1	3	300.142	16.546	277.850	2	37.800	38.900	48.500	63.700
2	4	300.150	21.879	369.890	1	26.700	22.600		
3	5	300.156	14.974	250.627	2	11.400		26.700	25.700
4	7	300.160	23.724	405.368	3	1390.000	1080.000	463.000	552.000
5	9	300.170	27.365	459.410	3	27.700	21.400		
6	10	300.174	18.146	303.145	1	191.000	20.300	16.000	
7	11	300.184	22.930	390.230	1	21.600	23.800		
8	12	300.189	19.380	322.203	2		420.000	690.000	353.000
9	13	300.190	20.750	350.840	2	123.000		114.000	
10	14	300.218	26.257	441.103	2	472.000	306.000	316.000	
11	16	300.480	18.599	309.823	3	18.500		26.000	26.000
12	18	300.480	14.272	237.073	3		23.500		39.000
13	19	300.500	27.109	455.170	3	40.900	37.000	21.000	24.200
14	22	300.640	29.241	490.978	2	80.200	86.800	90.400	100.000
15	23	300.649	17.936	300.046	1	44.500	45.200	26.500	34.100
16	24	300.665	15.088	253.020	2	119.000	366.000	634.000	345.000
17	26	300.690	14.023	232.220	2	446.000	399.000	618.000	607.000
18	27	300.682	21.130	357.260	1	43.800	15.400		
19	28	300.690	20.134	337.710	2	27.400	21.200	13.800	22.200
20	29	300.698	22.291	377.470	2	1400.000	1290.000	73.200	95.400
21	31	300.700	18.390	306.732	2	182.000	156.000	200.000	245.000
22	33	300.850	26.615	447.026	3	130.000	139.000	110.000	116.000
23	36	301.155	19.741	329.060	1	22.000			11.500
24	37	301.158	36.577	619.438	1	16.800	38.600	49.200	32.000
25	38	301.160	18.014	301.190	2	183.000	202.000	90.100	111.000
26	40	301.163	14.308	237.775	2	75.300	76.600		110.000
27	41	301.164	16.116	270.677	1	24.900		19.700	52.100
28	42	301.168	16.362	274.773	2		24.200	29.500	
29	44	301.181	20.429	344.366	2	54.300	51.600	55.100	80.500
30	45	301.187	21.619	365.506	2	104.000	104.000	90.600	87.100
31	48	301.194	19.293	320.552	2	47.700	56.300	78.200	83.400
32	49	301.200	24.271	412.480	2	129.000	127.000	86.900	103.000
33	50	301.200	22.727	385.908	2		204.000	283.000	341.000

Outline

- Overview
- Sample data: airfares
- A few options
  - Be naïve: subset()
  - Use a loop
  - tapply()
  - aggregate()
  - doBy's summaryBy()
  - plyr's ddply()
  - more more more...



Overview: group & summarize data

- Very common to have detail-level data from which you need summary-level statistics based on some grouping variable or variables
  - Sales by region, market share by company, discount and margin by product line and rep, etc.
- Hadley Wickham coined term “split-apply-combine” to describe this analysis pattern
  - *c.f.* SQL's “GROUP BY”, SAS has “by”, MapReduce
- There's more than one way to do it in R
- Often discussed on StackOverflow.com &c.

Sample data: BOS-NYC airfares

Individual airfares paid from Boston Logan (BOS) to New York City airports (EWR, JFK, LGA) last year:

```
> nrow(df)
[1] 1852

> head(df)
  Origin Dest Carrier   Fare
1    BOS  EWR      CO  56.32
2    BOS  EWR      9L   0.00
3    BOS  EWR      CO 102.00
4    BOS  EWR      CO 109.00
5    BOS  EWR      CO 130.00
6    BOS  EWR      CO 147.50

> tail(df)
     Origin Dest Carrier   Fare
1847    BOS  LGA      DL 208.87
1848    BOS  LGA      DL 223.79
1849    BOS  LGA      US 100.46
1850    BOS  LGA      UA 125.89
1851    BOS  LGA      US 167.63
1852    BOS  LGA      US 186.68

> unique(df$Dest)
[1] "EWR" "JFK" "LGA"
```

Naïve approach – split by hand

```
> ewr = subset(df, Dest=='EWR')
> jfk = subset(df, Dest=='JFK')
> lga = subset(df, Dest=='LGA')

> # counts:
> nrow(ewr)
[1] 392
> nrow(jfk)
[1] 572
> nrow(lga)
[1] 888
> # averages:
> mean(ewr$Fare)
[1] 267.6365
> median(ewr$Fare)
[1] 210.85
> mean(jfk$Fare)
[1] 147.3658
> median(jfk$Fare)
[1] 113.305
> mean(lga$Fare)
[1] 190.2382
> median(lga$Fare)
[1] 171
```

Automating naïveté with a loop

```
results = data.frame()

for ( dest in unique(df$Dest) )
{
  tmp = subset(df, Dest==dest)
  count = nrow(tmp)
  mean = mean(tmp$Fare)
  median = median(tmp$Fare)
  results = rbind(results, data.frame(dest, count, mean, median) )
}
```

```
> results
  dest count     mean  median
1  EWR   392 267.6365 210.850
2  JFK   572 147.3658 113.305
3  LGA   888 190.2382 171.000
```

Rule of Thumb: if you're using a loop in R, you're probably doing something wrong

Base R's tapply()

Applying functions repeatedly sounds like a job for Base R's *apply() functions:

```
> tapply(df$Fare, df$Dest, FUN=length)
EWR JFK LGA
392 572 888

> tapply(df$Fare, df$Dest, FUN=mean)
     EWR      JFK      LGA
267.6365 147.3658 190.2382

> tapply(df$Fare, df$Dest, FUN=median)
    EWR     JFK     LGA
210.850 113.305 171.000
```

I'm honestly not thrilled with the output format, but I'm sure we could wrestle into a *data.frame* which includes the grouping variable thanks to the names() function.

Base R's aggregate()

```
> aggregate(Fare~Dest, data=df, FUN="mean")
  Dest     Fare
1  EWR 267.6365
2  JFK 147.3658
3  LGA 190.2382

> aggregate(Fare~Dest, data=df, FUN="median")
  Dest    Fare
1  EWR 210.850
2  JFK 113.305
3  LGA 171.000

> aggregate(Fare~Dest, data=df, FUN="length")
  Dest Fare
1  EWR  392
2  JFK  572
3  LGA  888
```

- *data.frame* in, *data.frame* out (works for time series *ts, mts* too)
- Uses formula notation & *data.frame* environment aware (no $'s)

doBy package's summaryBy()

More capable and simpler (at least for me)

- Accepts formula to access multiple columns for values and groupings
- Accepts anonymous functions which can use c() to perform multiple operations
- doBy package also provides formula-based lapplyBy() and my favorite sorting function, orderBy()

```
> summaryBy(Fare~Dest, data=df, FUN=function(x)
c(count=length(x), mean=mean(x), median=median(x)))
  Dest Fare.count Fare.mean Fare.median
1  EWR        392  267.6365     210.850
2  JFK        572  147.3658     113.305
3  LGA        888  190.2382     171.000
```

Hadley Wickham's plyr package

- Provides a standard naming convention:
  - *X* + *Y* + "ply"
    - *X* = input data type
    - *Y* = output data type
    - Types:
      - "a" = *array*
      - "d" = *data.frame*
      - "l" = *list*
      - "m" = *matrix*
      - "_" = no output returned
  - Example: ddply() expects and returns a *data.frame*
- Most plyr functions wrap other plyr, Base functions

ddply() in action

Like summaryBy(), can use multiple-part grouping variables and functions

```
> ddply(df, 'Dest', function(x) c(count=nrow(x), mean=mean(x$Fare),
median=median(x$Fare)))
  Dest count     mean  median
1  EWR   392 267.6365 210.850
2  JFK   572 147.3658 113.305
3  LGA   888 190.2382 171.000

> ddply(df, c('Dest', 'Carrier'), function(x) c(count=nrow(x),
mean=mean(x$Fare), median=median(x$Fare)))
   Dest Carrier count     mean  median
1   EWR      9L    33 181.9697 131.500
2   EWR      CO   326 279.7623 264.250
3   EWR      XE    33 233.5152 152.500
4   JFK      AA     6 129.6600 140.120
5   JFK      B6   112 132.2796 108.245
[...]
```

“Are we there yet?”

Good news: plyr provides “.parallel” & “.progress” bar options for long-running jobs

```
> ddply(df, 'Dest', function(x) c(count=nrow(x), mean=mean(x$Fare),
median=median(x$Fare)), .progress='text')
  |=================================================================| 100%
  Dest count     mean  median
1  EWR   392 267.6365 210.850
2  JFK   572 147.3658 113.305
3  LGA   888 190.2382 171.000
```

Bad news: you may need them

- Has been getting faster, major work planned for summer 2011 (Hadley's goal: “as fast as *data.table*” !)
- “immutable” *idata.frame* in plyr 1.0 can help now
- Great speed & alternatives discussion: http://stackoverflow.com/questions/3685492/r-speeding-up-group-by-ope

Other options & approaches

- Loops that don't (necessarily) suck: foreach
  - Works with parallel backends (SMP, MPI, SNOW, etc.)
- Have data in a database?
  - DBI & friends to access and group (RMySQL, RPostgresSQL, ROracle, RJDBC, RODBC, etc.)
  - For MySQL, Postgres, look at dbApply() to aggregate
- sqldf will create a temporary database for you
- Data > Memory? http://www.bigmemory.org/
  - or just use Hadoop for everything: RHIPE

References and further reading

- StackOverflow.com discussions (has an active “[r]” tag for searching)
  - “for each group summarise means for all variables in dataframe (ddply? Split?)”
    - http://stackoverflow.com/questions/1407449/for-each-group-summarise-means-for-all-variable
  - “How to split a data frame by rows, and then process the blocks?”
    - http://stackoverflow.com/questions/1395191/how-to-split-a-data-frame-by-rows-and-then-proc
  - “R Grouping functions: sapply vs. lapply vs. apply. vs. tapply vs. by vs. aggregate vs. …”
    - http://stackoverflow.com/questions/3505701/r-grouping-functions-sapply-vs-lapply-vs-apply-vs
  - “how to aggregate this data in R”
    - http://stackoverflow.com/questions/2900058/how-to-aggregate-this-data-in-r
- JD Long: A Fast intro to Plyr
  - http://www.cerebralmastication.com/2009/08/a-fast-intro-to-plyr-for-r/
- Kane & Emerson: “Scalable Strategies for Computing with Massive Data: The Bigmemory Project”
  - http://www.slideshare.net/joshpaulson/big-memory