Lecture 15: Better Apply Functions with `plyr`

Statistical Computing, 36-350

Wednesday November 11, 2015

Outline

- Quick review: split-apply-combine
- Quick review: apply functions in base R
- New apply functions from `plyr`

General strategy

These are tools that, when put together, give us three general steps:

- **Split** whatever data object we have into meaningful chunks
- **Apply** the function of interest to this division
- **Combine** the results into a new object

As in:

```
x = split(strikes, strikes$country) # Split
y = sapply(x, my.strike.lm)         # Apply and combine
```

`*apply()` in base R

- `apply()` : arrays
- `lapply()` : list or vector, output list
- `sapply()` : list or vector, simplify to vector
- `vapply()` : list or vector, safer simplify to vector
- `tapply()`, `by()`, `aggregate()` : data frames(tables)
- `mapply()` : multiple vectors(special case of 2d array)

Useful set, but some issues: inconsistent output

Back to the drawing board: `plyr`

We have three common data structures beyond vectors: arrays(matrices), data frames, lists

```
library(plyr)
```

Most popular R package of all time(most downloads)!

Functions within: `**ply()`, replace `**` with characters denoting types:

- First character: input type, one of `a`, `d`, `l`
- Second character: output type, one of `a`, `d`, `l`, or `_` (drop)

a*ply() : input arrays

`processed = a*ply(.data, .margins, .fun, ...)`

- `.data` : an array
- `.margins` : subscripts which the function gets applied over
- `.fun` : the function to be applied
- `...` : additional arguments to function

Looks like

`processed = apply(data, margin, function)`

Let's test it out!

```
my.boring.array = array(1:27, c(3,3,3))
rownames(my.boring.array) = c("Curly", "Larry", "Moe")
colnames(my.boring.array) = c("Groucho", "Harpo", "Zeppo")
dimnames(my.boring.array)[[3]] = c("Bart", "Lisa", "Maggie")
adply(my.boring.array, 1, sum)
```

```
##      X1  V1
## 1 Curly 117
## 2 Larry 126
## 3   Moe 135
```

```
alply(my.boring.array, 1, sum)
```

```
## $`1`
## [1] 117
##
## $`2`
## [1] 126
##
## $`3`
## [1] 135
##
## attr(,"split_type")
## [1] "array"
## attr(,"split_labels")
##      X1
## 1 Curly
## 2 Larry
## 3   Moe
```

```r
aaply(my.boring.array, 1, sum)
```

```
## Curly Larry   Moe
##   117   126   135
```

```r
adply(my.boring.array, 2:3, sum)
```

```
##        X1     X2 V1
## 1 Groucho   Bart  6
## 2   Harpo   Bart 15
## 3   Zeppo   Bart 24
## 4 Groucho   Lisa 33
## 5   Harpo   Lisa 42
## 6   Zeppo   Lisa 51
## 7 Groucho Maggie 60
## 8   Harpo Maggie 69
## 9   Zeppo Maggie 78
```

```r
alply(my.boring.array, 2:3, sum)
```

```
## $`1`
## [1] 6
##
## $`2`
## [1] 15
##
## $`3`
## [1] 24
##
## $`4`
## [1] 33
##
## $`5`
## [1] 42
##
## $`6`
## [1] 51
##
## $`7`
## [1] 60
##
## $`8`
## [1] 69
##
## $`9`
## [1] 78
##
```

```
## attr(,"split_type")
## [1] "array"
## attr(,"split_labels")
##        X1     X2
## 1 Groucho   Bart
## 2   Harpo   Bart
## 3   Zeppo   Bart
## 4 Groucho   Lisa
## 5   Harpo   Lisa
## 6   Zeppo   Lisa
## 7 Groucho Maggie
## 8   Harpo Maggie
## 9   Zeppo Maggie
```

```
aaply(my.boring.array, 2:3, sum)
```

```
##          X2
## X1        Bart Lisa Maggie
##   Groucho    6   33     60
##   Harpo     15   42     69
##   Zeppo     24   51     78
```

One more:

Return no values, but use outcomes in secondary functions

```
pdf("lotsaplots.pdf")
a_ply(my.boring.array, 2:3, plot)
graphics.off()
```

`d*ply()` : operate on data frames

```
processed = d*ply(.data, .(splitvariable), .fun, ...)
```

- `.data` : an array
- `.(...)` : arguments to split by
- `.fun` : the function to be applied
- `...` : additional arguments to function

Looks like

```
processed = tapply(data, data$splitvariable, function)
```

Strikes example again:

```
strikes = read.csv("http://www.stat.cmu.edu/~ryantibs/statcomp/lectures/strikes.csv")
my.strike.lm.better = function(country.df) {
  return(lm(strike.volume ~ left.parliament + unemployment + inflation,
            data=country.df)$coefficients)
}
```

Syntax: for data frame inputs, include the variables for the splitting in a stand-in function `.()` : syntax like `by()`

```
results = daply(strikes, .(country), my.strike.lm.better)
results[1:5,]
```

```
##
## country     (Intercept) left.parliament unemployment  inflation
##   Australia    157.9191       0.5658674   -1.1181489 30.4666061
##   Austria      600.6778     -11.2441604  -10.9216990 -0.5923972
##   Belgium     -243.4823      12.4516118    0.3578217 10.2673539
##   Canada       167.0712      13.4386408  -48.1790264 27.2180651
##   Denmark    -1331.7517      33.1605982   -8.1864655  5.1826201
```

```
results = ddply(strikes, .(country), my.strike.lm.better)
results[1:5,]
```

```
##     country (Intercept) left.parliament unemployment  inflation
## 1 Australia    157.9191       0.5658674   -1.1181489 30.4666061
## 2   Austria    600.6778     -11.2441604  -10.9216990 -0.5923972
## 3   Belgium   -243.4823      12.4516118    0.3578217 10.2673539
## 4    Canada    167.0712      13.4386408  -48.1790264 27.2180651
## 5   Denmark  -1331.7517      33.1605982   -8.1864655  5.1826201
```

```
results = dlply(strikes, .(country), my.strike.lm.better)
results[1:3]
```

```
## $Australia
##     (Intercept) left.parliament    unemployment       inflation
##     157.9191118       0.5658674      -1.1181489      30.4666061
##
## $Austria
##     (Intercept) left.parliament    unemployment       inflation
##     600.6777769     -11.2441604     -10.9216990      -0.5923972
##
## $Belgium
##     (Intercept) left.parliament    unemployment       inflation
##    -243.4822938      12.4516118       0.3578217      10.2673539
```

`l*ply()` : operate on lists

```
processed = l*ply(.data, , .fun, ...)
```

- `.data` : an array
- `.fun` : the function to be applied
- `...` : additional arguments to function

Looks like

```
processed = lapply(data, function, ...)
```

```
strikes.split = split(strikes, strikes$country)
results = laply(strikes.split, my.strike.lm.better)
results[1:5,]
```

```
##      (Intercept) left.parliament unemployment  inflation
## [1,]    157.9191       0.5658674   -1.1181489 30.4666061
## [2,]    600.6778     -11.2441604  -10.9216990 -0.5923972
## [3,]   -243.4823      12.4516118    0.3578217 10.2673539
## [4,]    167.0712      13.4386408  -48.1790264 27.2180651
## [5,]  -1331.7517      33.1605982   -8.1864655  5.1826201
```

```
results = ldply(strikes.split, my.strike.lm.better)
results[1:5,]
```

```
##         .id (Intercept) left.parliament unemployment  inflation
## 1 Australia    157.9191       0.5658674   -1.1181489 30.4666061
## 2   Austria    600.6778     -11.2441604  -10.9216990 -0.5923972
## 3   Belgium   -243.4823      12.4516118    0.3578217 10.2673539
## 4    Canada    167.0712      13.4386408  -48.1790264 27.2180651
## 5   Denmark  -1331.7517      33.1605982   -8.1864655  5.1826201
```

```
results = llply(strikes.split, my.strike.lm.better)
results[1:3]
```

```
## $Australia
##     (Intercept) left.parliament    unemployment       inflation
##     157.9191118       0.5658674      -1.1181489      30.4666061
##
## $Austria
##     (Intercept) left.parliament    unemployment       inflation
##     600.6777769     -11.2441604     -10.9216990      -0.5923972
##
## $Belgium
##     (Intercept) left.parliament    unemployment       inflation
##    -243.4822938      12.4516118       0.3578217      10.2673539
```

More dimensions

NHL data from the 2013-2014 season

```
print(load(url("http://www.stat.cmu.edu/~ryantibs/statcomp/lectures/nhl-2013.RData")))
```

```
## [1] "teamtable"
```

```
head(teamtable)
```

```
##      FAC_WIN FAC_LOSE PENL_TAKEN PENL_DRAWN HIT HIT_TAKEN CF CA FF FA SF
## MTL        4        1          2          1   3         6 12  5 10  4  4
## MTL1       3        2          0          0   0         3 11  9  7  5  6
## MTL2       6        7          2          2   9         7 12 13 11  9 10
## MTL3       5        2          2          1   5         6 21 12 16  8  9
## MTL4       2        1          3          0   0         0  2  1  2  1  2
## MTL5       3        2          2          2   0         1  3  1  1  1  1
##      SA GF GA Off Neu Def   TOI team opponent   season gcode scorediffcat
## MTL   3  1  0   2   3   0 632.0  MTL      TOR 20132014 20001            1
## MTL1  3  0  1   1   3   1 459.5  MTL      TOR 20132014 20001            2
## MTL2  8  1  0   3   4   6 641.0  MTL      TOR 20132014 20001            3
## MTL3  5  0  1   3   2   2 641.0  MTL      TOR 20132014 20001            4
## MTL4  1  0  0   2   1   0  91.0  MTL      TOR 20132014 20001            2
## MTL5  1  0  1   4   1   0 190.5  MTL      TOR 20132014 20001            3
##      gamestate home
## MTL          1    1
## MTL1         1    1
## MTL2         1    1
## MTL3         1    1
## MTL4         2    1
## MTL5         2    1
```

```
colnames(teamtable)
```

```
##  [1] "FAC_WIN"      "FAC_LOSE"     "PENL_TAKEN"   "PENL_DRAWN"
##  [5] "HIT"          "HIT_TAKEN"    "CF"           "CA"
##  [9] "FF"           "FA"           "SF"           "SA"
## [13] "GF"           "GA"           "Off"          "Neu"
## [17] "Def"          "TOI"          "team"         "opponent"
## [21] "season"       "gcode"        "scorediffcat" "gamestate"
## [25] "home"
```

```
teamtable = subset(teamtable, scorediffcat < 6)
```

Split by team name, add up performance measures:

```
teamperf = ddply(teamtable, .(team),
                 function(dd) colSums(dd[,c("GF","GA","HIT","HIT_TAKEN")]))
head(teamperf)
```

```
##   team  GF  GA  HIT HIT_TAKEN
## 1  ANA 298 240 2515      2512
## 2  BOS 287 197 2441      2466
## 3  BUF 150 243 1992      1537
## 4  CAR 204 226 1807      1965
## 5  CBJ 241 234 2901      2109
## 6  CGY 201 238 1708      1463
```

Split by team name and home/away, add up performance measures

```
teamperf = ddply(teamtable, .(team, home),
                 function(dd) colSums(dd[,c("GF","GA","HIT","HIT_TAKEN")]))
head(teamperf)
```

```
##   team home  GF  GA  HIT HIT_TAKEN
## 1  ANA    0 136 129 1079      1245
## 2  ANA    1 162 111 1436      1267
## 3  BOS    0 137 113 1106      1270
## 4  BOS    1 150  84 1335      1196
## 5  BUF    0  65 124 1039       906
## 6  BUF    1  85 119  953       631
```

Split by team name and game ID: the lethal case from last time

```
teamperf = daply(teamtable, .(team, gcode),
                 function(dd) colSums(dd[,c("GF","GA","HIT","HIT_TAKEN")]))
dim(teamperf)  # Worst case: 30 x 1323 rows.
```

```
## [1]   30 1323    4
```

```
head(teamperf)
```

```
## [1] NA NA NA NA NA NA
```

Summary

- Apply functions are very important in the context of the split-apply-combine paradigm
- `plyr` makes life easier by making the input and outputs types (array, data frame, or list?) as explicit as possible