Statistical Computing (36-350)

Lecture 13: Split/Apply/Combine with `plyr`

Cosma Shalizi

Massive thanks to Vince Vu

9 October 2013

Agenda

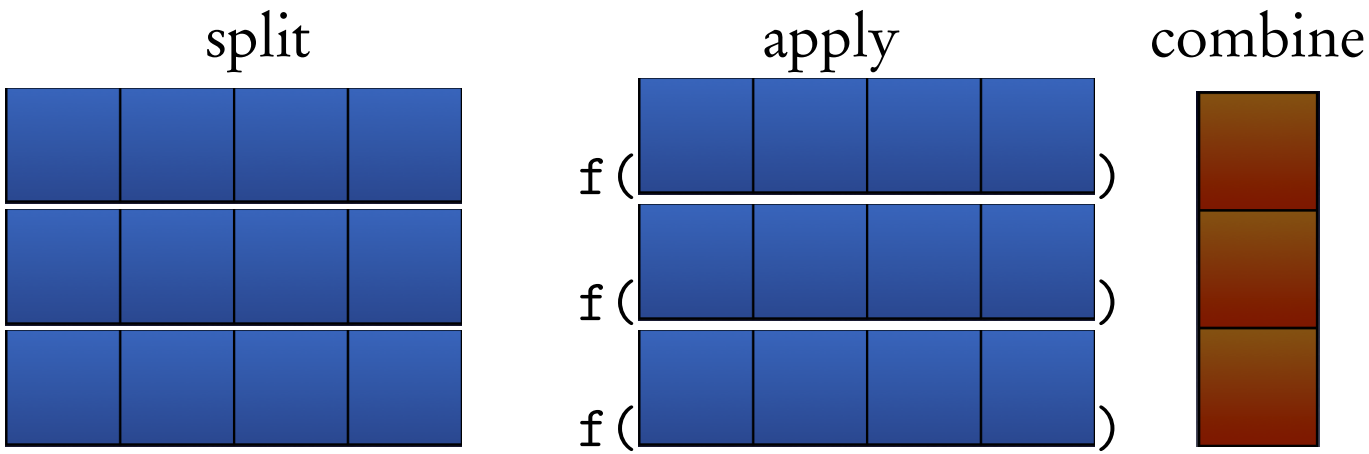
- Abstracting split, apply, combine plyr usage
- Examples

Recommended reading: `http://plyr.had.co.nz/`

Previously on Split/Apply/Combine

Many problems can be solved this way:

- Divide the big problem into smaller ones (split)
- Solve each piece independently and in the same way (apply)
- Put the piecemeal solutions together (combine)


split
apply
combine
f(
)
f(
)
f(
)

Example from last time

```
x <- split(strikes, strikes$country)
y <- lapply(x, strikes_vs_left, coefficients.only=TRUE)
coefs <- do.call(rbind, y)
```

split the data by country
fit the same linear model for each country
combine the results into an array

*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()` data frames (tables)

`mapply()` multiple vectors (special case of 2d array)

*apply in base R

Grew up without any particular plan

*apply in base R

Grew up without any particular plan
Functions are useful, but

- Output is inconsistent (lists or array)
- Too much to remember
- Too much to write

The plyr model

Abstract the pattern:

- Input data structure (split)
- Processing function (apply)
- Output data structure (combine)

The plyr model

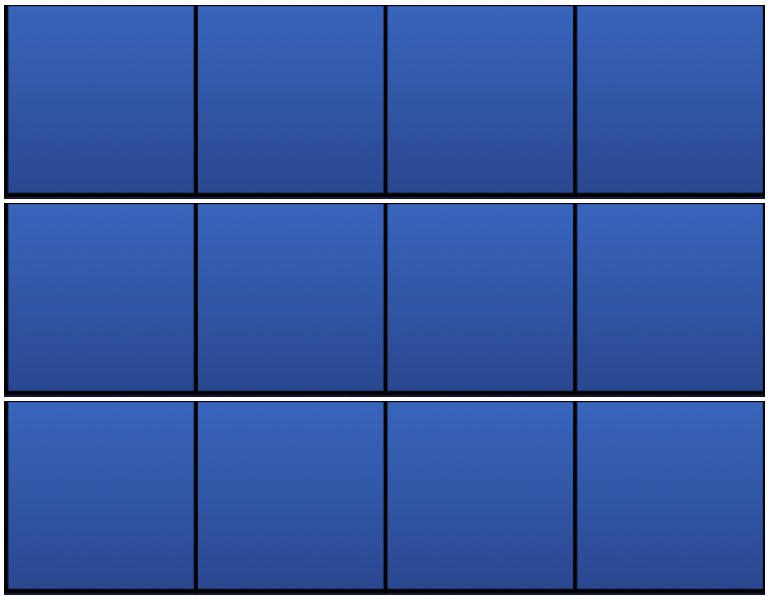
Abstract the pattern:

- Input data structure (split)
- Processing function (apply)
- Output data structure (combine)

Functions named and designed consistently
`*ply()` — replace * with 2 characters:

- first character: input type array, data frame or list (`a`, `d`, `l`)
- second character: output type array, data frame, list, or discard (`a`, `d`, `l`, `_`)

Input Data Structure



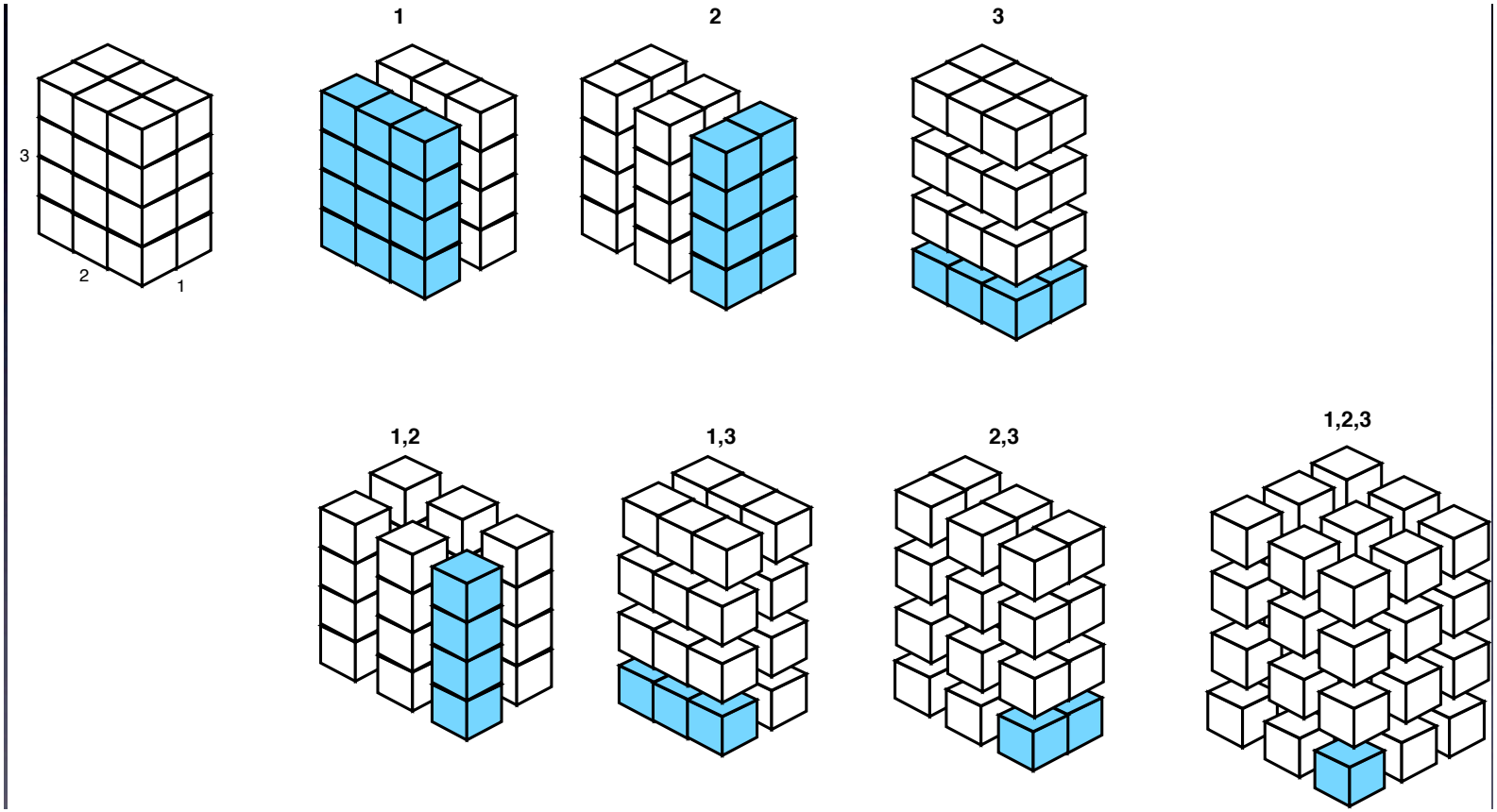
Each type (array, list, data frame) has its own ways of being split

Inputs: d-dimensional Arrays

d dimensions that can be subscripted independently
$\therefore$ can be split $2^d - 1$ different ways
2D arrays can be split 3 ways: rows, columns, cells

Splitting 3D Arrays

$2^3 - 1 = 7$ ways to split



from Wickham (2011)

a*ply()

```
y <- 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

Returns a (*: a = array, d = data frame, l = list)

Input: Lists — `l*ply()`

Lists can only be split one way

```
y <- l*ply(.data, .fun, ...)
```

Pop Quiz

```
x <- list(alice="Wonderland",babur="Samarkand")
```

What's the difference between
`x[[1]]` and `x[1]`?

`x[[1]]` is the 1st component of `x`
It is a string, namely `"Wonderland"`
It has no components

`x[[1]]` is the 1st component of `x`
It is a string, namely `"Wonderland"`
It has no components
`x[1]` is a subset of `x`
It is a list, which happens to be of length 1
It has components: what is `x[1][[1]]`?

Input: Data Frames

Can be split into groups according to the values of variables in the columns
Groups need not be of equal size
e.g., split census tracts by state
e.g., split census tracts by urban/suburban/rural
e.g., split census tracts by state *and* type

d*ply()

```
y <- d*ply(.data, .variables, .fun, ...)
```

`.data`	a data frame
`.variables`	variables used to define groups
`.fun`	the function to be applied
`...`	additional arguments to the function

Returns array, data frame, list, nothing
`.variables` can be of two forms
`.(var1, var2)` or
`c('var1','var2')`
searches `.data` for those variables first, then the parent environment

Data Frames Have Two Natures

Data frame is a list of vectors
$\therefore$ Can be split into separate columns
$\therefore$ Can be used with `l*ply()`

Data Frames Have Two Natures

Data frame is a list of vectors
$\therefore$ Can be split into separate columns
$\therefore$ Can be used with `l*ply()`
Data frame responds to array-like indexing
$\therefore$ Can be split like a 2D array
$\therefore$ Can be used with `a*ply()`

Processing Function

Function that is applied to each piece
Should:

- Take a piece as its first argument
- Return same type as eventual output (but there are exceptions)
- Sometimes cause side effects (plot, save, ...)

Output Data Structure

Defines how results are combined and labeled

- Array (`a`)
- List (`l`)
- Data frame (`d`)
- Discarded (`_`) — for side effects, e.g., plotting

Output Arrays

Output organized in the expected way.
Processing function should return an object of same type each time it is called.
If processing function returns a list, then output will be a list-array (list with dimensions)

Output Data Frames

Output will contain results with additional label columns indicating which group the result corresponds to.

Applying the pattern to your problem

- check data type of
  - input data structure
  - output data structure
- Use a built-in function, or write a processing function and test it on one piece
- Call appropriate `**ply()`

Examples

Install `plyr`

```
install.packages("plyr", dependencies = T)
```

Load `plyr`

```
library(plyr)
```

(use `require` in code, returns TRUE or FALSE as appropriate)

Examples

Regularly sampled spatial data

```
x <- array(STUFF, dim = c(10, 10, 100))
```

10×10 grid of locations
100 measurements at each location
Problem: Standardize measurements at each location
Standardize one location:

```
z <- scale(x[1, 1, ])
```

Iteration

Iteration

```
y <- array(dim = dim(x))
for(i in 1:dim(x)[1]) {
    for(j in 1:dim(x)[2]) {
        y[i, j, ] <- scale(x[i, j, ])
  }
}
```

Iteration

```
y <- array(dim = dim(x))
for(i in 1:dim(x)[1]) {
    for(j in 1:dim(x)[2]) {
        y[i, j, ] <- scale(x[i, j, ])
  }
}
```

Base R:

Iteration

```
y <- array(dim = dim(x))
for(i in 1:dim(x)[1]) {
    for(j in 1:dim(x)[2]) {
        y[i, j, ] <- scale(x[i, j, ])
  }
}
```

Base R:

```
y <- apply(x, 1:2, scale)
```

Iteration

```
y <- array(dim = dim(x))
for(i in 1:dim(x)[1]) {
    for(j in 1:dim(x)[2]) {
        y[i, j, ] <- scale(x[i, j, ])
  }
}
```

Base R:

```
y <- apply(x, 1:2, scale)
```

plyr

```
y <- aaply(x, 1:2, scale)
```

Ragged spatial data

```
x <- data.frame(loc.x = FOO,
                loc.y = BAR,
                value = BAZ)
```

Irregularly sampled (x,y) locations
Different number of measurements at each location
Standardize measurements at each location

Handle one location:

```
df <- subset(x, loc.x = 1 & loc.y = 1)
z <- scale(df$value)
```

Handle one location:

```
df <- subset(x, loc.x = 1 & loc.y = 1)
z <- scale(df$value)
```

Iteration

Handle one location:

```
df <- subset(x, loc.x = 1 & loc.y = 1)
z <- scale(df$value)
```

Iteration

```
Left as an exercise for the student
```

Handle one location:

```
df <- subset(x, loc.x = 1 & loc.y = 1)
z <- scale(df$value)
```

Iteration

```
Left as an exercise for the student
```

Base R

Handle one location:

```
df <- subset(x, loc.x = 1 & loc.y = 1)
z <- scale(df$value)
```

Iteration

```
Left as an exercise for the student
```

Base R

```
Left as an exercise
```

Handle one location:

```
df <- subset(x, loc.x = 1 & loc.y = 1)
z <- scale(df$value)
```

Iteration

```
Left as an exercise for the student
```

Base R

```
Left as an exercise
```

plyr

Handle one location:

```
df <- subset(x, loc.x = 1 & loc.y = 1)
z <- scale(df$value)
```

Iteration

```
Left as an exercise for the student
```

Base R

```
Left as an exercise
```

plyr

```
y <- ddply(x, .(loc.x, loc.y), function(df) { return(scale(df$value)) } )
```

Only want to scale one column of the split-off data frame

Used an anonymous function; could also define a function previously

Continuing the parliamentary politics/strikes connection

```
strikes <- read.csv("http://www.stat.cmu.edu/~cshalizi/uADA/12/hw/06/strikes.csv"
strikes <- strikes[,-7] # drop "centralization" variable
```

country	year	strike.volume	unemployment	inflation	left.parliament	density
Australia	1983	313	9.8	10.1	60	48.5
Australia	1984	241	8.9	4	55.4	47.6
Australia	1985	226	8.2	6.7	55.4	45.9
Austria	1951	43	3.5	27.5	43.6	NA
Austria	1952	39	4.7	13.6	43.6	NA
Austria	1953	20	5.8	-1.6	46.7	NA

If left parties gain an extra 1% of the gov't., how much more strike activity?
Try this as a linear model
Analysis in a function

```
strikes_vs_left <- function(df,coefficients.only=FALSE) {
  fit <- lm(strike.volume ~ left.parliament, data=df)
  if (coefficients.only) {
   return(coefficients(fit))
  } else {
    return(fit)
  }
}
```

Input: a data frame, `strikes`
Split by: `country`
Output desired: an array of regression coefficients $\Rightarrow$ an array
$\therefore$ use `daply`
Processing function: `strikes_vs_left`

Iteration

Iteration

```
coefs <- matrix(nrow=nlevels(strikes$country),ncol=2)
for (i in 1:nlevels(strikes$country)) {
  x <- subset(strikes, country==levels(strikes$country)[i])
  coefs[i,] <- strikes_vs_left(x,coefficients.only=TRUE)
}
rownames(coefs) <- levels(strikes$country)
```

Iteration

```
coefs <- matrix(nrow=nlevels(strikes$country),ncol=2)
for (i in 1:nlevels(strikes$country)) {
  x <- subset(strikes, country==levels(strikes$country)[i])
  coefs[i,] <- strikes_vs_left(x,coefficients.only=TRUE)
}
rownames(coefs) <- levels(strikes$country)
```

Base R

Iteration

```
coefs <- matrix(nrow=nlevels(strikes$country),ncol=2)
for (i in 1:nlevels(strikes$country)) {
  x <- subset(strikes, country==levels(strikes$country)[i])
  coefs[i,] <- strikes_vs_left(x,coefficients.only=TRUE)
}
rownames(coefs) <- levels(strikes$country)
```

Base R

```
x <- split(strikes, strikes$country)
y <- lapply(x, strikes_vs_left, coefficients.only=TRUE)
coefs <- do.call(rbind, y)
```

Iteration

```
coefs <- matrix(nrow=nlevels(strikes$country),ncol=2)
for (i in 1:nlevels(strikes$country)) {
  x <- subset(strikes, country==levels(strikes$country)[i])
  coefs[i,] <- strikes_vs_left(x,coefficients.only=TRUE)
}
rownames(coefs) <- levels(strikes$country)
```

Base R

```
x <- split(strikes, strikes$country)
y <- lapply(x, strikes_vs_left, coefficients.only=TRUE)
coefs <- do.call(rbind, y)
```

plyr

Iteration

```
coefs <- matrix(nrow=nlevels(strikes$country),ncol=2)
for (i in 1:nlevels(strikes$country)) {
  x <- subset(strikes, country==levels(strikes$country)[i])
  coefs[i,] <- strikes_vs_left(x,coefficients.only=TRUE)
}
rownames(coefs) <- levels(strikes$country)
```

Base R

```
x <- split(strikes, strikes$country)
y <- lapply(x, strikes_vs_left, coefficients.only=TRUE)
coefs <- do.call(rbind, y)
```

plyr

```
coefs <- daply(strikes, .(country), strikes_vs_left, coefficients.only=TRUE)
```

How many complete observations per country?

```
nrow.omitting.nas <- function(df) { nrow(na.omit(df)) }

daply(strikes, .(country), nrow.omitting.nas)
```

How many complete observations per year?

```
daply(strikes, .(year), nrow.omitting.nas)
```

Group Summaries

For each country, take median of each variable

Omit a year with NAs, but only for that variable

e.g., omit 1951 for Austria for `density` but not `inflation`

In: data frame

Out: data frame (country by variables)

Processing:

```
colMedians <- function(df) { apply(df,2,median,na.rm=TRUE) }

medians <- ddply(strikes, .(country), colMedians)
```

Doesn't work!

```
> head(medians)
    country year strike.volume unemployment inflation left.parliament  density
1 Australia 1968           326          2.5       5.9            41.0     <NA>
2   Austria 1968            11          2.0       4.0            47.9     <NA>
3      <NA> <NA>          <NA>         <NA>      <NA>            <NA>     42.1
4    Canada 1968           470          5.6       4.0            59.0     <NA>
5   Denmark 1968            38          6.0       6.5            46.9     <NA>
6   Finland 1968           155          2.2       7.1            27.0     <NA>
```

Problem: `country`, a factor variable, is still part of each split data frame; medians don't make sense
Slightly inelegant solution:

```
colMedians <- function(df) { apply(df[,-1],2,median,na.rm=TRUE) }
medians <- ddply(strikes, .(country), colMedians)
```

```
    country   year strike.volume unemployment inflation left.parliament  density
1 Australia 1968.0         326.0          2.5      4.30             41.0   48.10
2   Austria 1968.0          11.0          2.0      4.00             47.9   60.90
3   Belgium 1965.5         186.5          2.9      3.85             33.0   42.10
4    Canada 1968.0         470.0          5.6      3.70             59.0   32.45
5   Denmark 1968.0          38.0          6.0      6.50             46.9   62.20
6   Finland 1968.0         155.0          2.2      7.10             27.0   61.10
```

More elegant: have `colMedians` figure out which columns are numeric (or logical), drop the rest.
Similar in effect to base R

```
aggregate(strikes[,-1],by=list(strikes$country),FUN=median,na.rm=TRUE)
```

Don't Force It

Don't use split/apply/combine as a fancy way of writing `for`

```
l_ply(1:708, function(i) {
  # several hundred lines of code follow
})
```

Use the pattern (and the tools) when:

- The problem naturally breaks the data into smaller pieces
- You can solve the problem on each piece in the same way, and independently of the other pieces
- You need to re-integrate the piecemeal solutions

Summary

1. `plyr` simplifies using split/apply/combine, abstracting away from implementation details
2. You focus on figuring out the input type, the output type, and the processing function
3. Try writing a processing function for one piece, then generalize