

plyr

split-apply-combine for mortals

why?

1. it's everywhere
2. less code, simple syntax
3. it runs faster

look familiar?

```
> d
  year count
1 2000   16
2 2000    4
3 2000   12
4 2001   15
5 2001    7
6 2001   12
7 2002   20
... 
```

why apply > for loop?

less code
subsetting
saving results
faster

```
> d
  year  count
1 2000    16
2 2000     4
3 2000    12
4 2001    15
5 2001     7
6 2001    12
7 2002    20
...
```

	year	mean
1	2000	10.66667
2	2001	11.33333
3	2002	13.66667

```
d.split <- split(d, d$year)
results <- vector("list", length =
  length(d.split))
```

```
for(i in 1:length(d.split)) {
  temp <- d.split[[i]]
  temp.mean <- mean(temp$count)
  results[[i]] <- data.frame(
    year = unique(temp$year),
    mean = temp.mean)
}
```

```
do.call("rbind", results)
```

```
apply(array, 1 or 2, func)
```

```
sapply(vector, func)
```

```
lapply(list, func)
```

```
tapply(vector, index, func)
```

```
aggregate(object, by, func)
```

```
...
```

```
d.split <- split(d, d$year)
```

```
result <- lapply(d.split,  
  function(x) mean(x$count))
```

```
result <- unlist(result)
```

```
result <- data.frame(year =  
unique(d$year), mean = result)
```

```
row.names(result) <- NULL
```

enter plyr

Hadley →



```
ddply(d, "year", summarize,  
      mean = mean(count))
```

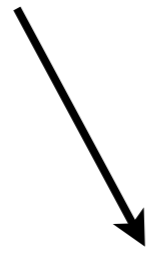
```
d.split <- split(d, d$year)
results <- vector("list", length =
  length(d.split))
```

```
for(i in 1:length(d.split)) {
  temp <- d.split[[i]]
  temp.mean <- mean(temp$count)
  results[[i]] <- data.frame(
    year = unique(temp$year),
    mean = temp.mean)
}
```

```
do.call("rbind", results)
```

input

output



ddp1y()

d – data frame

l – list

a – array

_ – discard

```
ddply(data, "split", function)
```

```
ddply(d, "year", summarise,  
      mean.count = mean(count))
```

	year	mean
1	2000	10.66667
2	2001	11.33333
3	2002	13.66667

```
ddply(d, "year", transform,  
      total.count = sum(count))
```

	year	count	total
1	2000	16	32
2	2000	4	32
3	2000	12	32
4	2001	15	34
5	2001	7	34
6	2001	12	34
7	2002	20	41
8	2002	15	41
9	2002	6	41

```
ddply(d, "year", function(x) {  
  browser()  
})
```

```
Browse[1]> x
```

	year	count
1	2000	16
2	2000	4
3	2000	12

```
Browse[1]> Q
```

```
>
```

```
library(doMC)
registerDoMC(2) # 2 cores

ddply(d, f, .parallel = TRUE))
```

```
# fail gracefully:
```

```
failwith(default, f)
```

remember

1. it's everywhere
2. less code, simple syntax
3. it runs faster (sometimes)

use it.