05 - Looking Under the Hood with apply

R Workshop

- Data Formatting and Reshaping -

Outline

- Summaries from matrices
- Sweeping out summary statistics
- Summaries for ragged arrays
- Applying models to lists

Teacher Salaries

- estimated average annual salaries of teachers in public elementary/secondary schools
- summarized by state/jurisdiction
- Source: 2008 Digest of Education Statistics

```
> head(salaries)
           1969-70 1979-80 1989-90 1999-2000 2004-05 2005-06 2006-07
Alabama      36845   34341   39916     44241   40665   41390   43389
Alaska       57067   71549   69377     56026   55828   54938   54658
Arizona      47075   39585   47270     44498   45691   45827   45941
Arkansas     34083   32340   35935     40258   43124   43874   44245
California   55743   47384   61089     57494   61344   61372   63640
Colorado     41941   42611   49450     46018   46803   45588   45833
```

Teacher Salaries

- What if we wanted overall summaries by year? By state?
- Plan: apply the same function to each column/row
- If we only wanted means/sums, use `rowMeans`, `rowSums`, `colMeans`, `colSums`

aaply & apply

- What about more complicated summaries?
  - `aaply(.data, .margins, .fun = NULL, ...)`
  - `apply(X, MARGIN, FUN, ...)`

Your Turn

- Find the min & max average salaries during these school years by state.
- Find the min & max average salaries within each school year.
- Compare the output from aaply and apply. Are the formats the same?

Does it matter?

- Three ways to find row means

```
rowMeans(salaries)
apply(salaries, 1, mean)
aaply(salaries, 1, mean)
```

```
      test replications elapsed relative user.self sys.self
3    aaply          100   2.409  301.125     2.311    0.095
2    apply          100   0.279   34.875     0.272    0.006
1 rowMeans          100   0.008    1.000     0.008    0.001
```

sweeping

- What if we wanted to re-express the salaries based on the median salary for that school year?
- Plan: calculate medians and subtract from the corresponding column
- Not directly corresponding function in `plyr` (need to define our own function)

sweeping

- Calculating the medians for each school year

```
ymeds <- apply(salaries, 2, median)
```

- We could write a `for` loop to cycle through columns

```
adj <- matrix(NA, nrow = dim(salaries)[1],
              ncol = dim(salaries)[2],
              dimnames = dimnames(salaries))
for(i in 1:dim(salaries)[2]){
  adj[,i] <- salaries[,i] - ymeds[i]
}
```

sweeping

- Avoid the `for` loop!

```
sweep(x, MARGIN, STATS, FUN = "-", check.margin = TRUE, ...)
```

```
swept <- sweep(salaries, 2, ymeds, FUN = "-")
head(swept)
```

	1969-70	1979-80	1989-90	1999-2000	2004-05	2005-06	2006-07
Alabama	-6766	-4862	-6685	-1289	-5498	-4185	-2552
Alaska	13456	32346	22776	10496	9665	9363	8717
Arizona	3464	382	669	-1032	-472	252	0
Arkansas	-9528	-6863	-10666	-5272	-3039	-1701	-1696
California	12132	8181	14488	11964	15181	15797	17699
Colorado	-1670	3408	2849	488	640	13	-108

```
aaply(salaries, .margin=2, function(x) x-median(x))
```

Your Turn

- Convert the salaries for each year into proportions based on the maximum average salary for that school year.

Does it matter?

- Three ways to “sweep” out proportions

```
   test replications elapsed  relative user.self sys.self
3 aaply          100   0.631 30.047619     0.613    0.017
2 apply          100   0.028  1.333333     0.028    0.001
1 sweep          100   0.021  1.000000     0.021    0.000
```

Wages Data

- data on labor-market experience of male high school dropouts
- 6402 observations
- 888 respondents

```
head(wages)
  id   lnw exper ged postexp black hispanic hgc hgc.9 uerate
1 31 1.491 0.015   1   0.015    no      yes   8    -1   3.21
2 31 1.433 0.715   1   0.715    no      yes   8    -1   3.21
3 31 1.469 1.734   1   1.734    no      yes   8    -1   3.21
4 31 1.749 2.773   1   2.773    no      yes   8    -1   3.30
5 31 1.931 3.927   1   3.927    no      yes   8    -1   2.89
6 31 1.709 4.946   1   4.946    no      yes   8    -1   2.49
```

Wages Data

- We would like to find the average unemployment rate experienced by each individual.

```
head(table(wages$id), 10)
31  36  53 122 134 145 155 173 206 207
 8  10   8  10  12   9  11   6   3  11
```

- Plan: split data by respondent and find the mean `uerate` for each

tapply

- We can use ddply

```
avgUE_plyr <- ddply(wages, .(id), summarise,
                    avgUE = mean(uerate))
```

- `tapply(X, INDEX, FUN = NULL, ...)`

```
avgUE <- tapply(wages$uerate, wages$id, mean)
head(avgUE)
      31       36       53      122      134      145
3.213750 5.097000 4.431250 5.296000 5.720000 5.195556
```

Does it matter?

- Comparing `ddply` and `tapply`

```
    test replications elapsed relative user.self sys.self
2  ddply          100  75.450  16.8115    73.442    0.415
1 tapply          100   4.488   1.0000     4.463    0.016
```

lmList vs. dlply vs. by

Suppose we want to fit a regression model for each respondent (split-apply)

```
sepLM <- lmList(lnw ~ exper | id, data = wages)

sepLM_alt1 <- dlply(wages, .(id),
                    function(x) lm(lnw ~ exper, data = x))

sepLM_alt2 <- by(data = wages, INDICES = wages$id,
                 function(x) lm(lnw ~ exper, data = x))
```

```
    test replications elapsed relative user.self sys.self
3     by          100 269.706 1.036012   264.937    1.035
2  dlply          100 262.306 1.007586   258.972    0.896
1 lmList          100 260.331 1.000000   258.000    0.937
```

lmList

- lmList relies on two functions: split and lapply

```
# Split
sepDF <- split(wages, wages$id)

# Apply
fitLM <- lapply(sepDF, function (x) lm(lnw ~ exper, data = x))
```

- returns a list
- easy to extract coefficients, residuals, etc.

lapply

- To understand lapply, we look to a loop

```
# Split
sepDF <- split(wages, wages$id)

# Apply
fitLM_loop <- vector("list", length(sepDF))
for(i in seq(1, length(sepDF))){
  df <- sepDF[[i]]
  fitLM_loop[[i]] <- lm(lnw ~ exper, data = df)
}
```

lapply vs. sapply

- Pulling off coefficients for each respondent

```
coefs <- lapply(fitLM, coef)
class(coefs)

coefs_alt1 <- t(sapply(fitLM, coef))
class(coefs_alt1)
```

- lapply returns a list
- sapply returns a matrix (can also return vector or array)

plyr vs. base

- Now the whole split-apply combine process

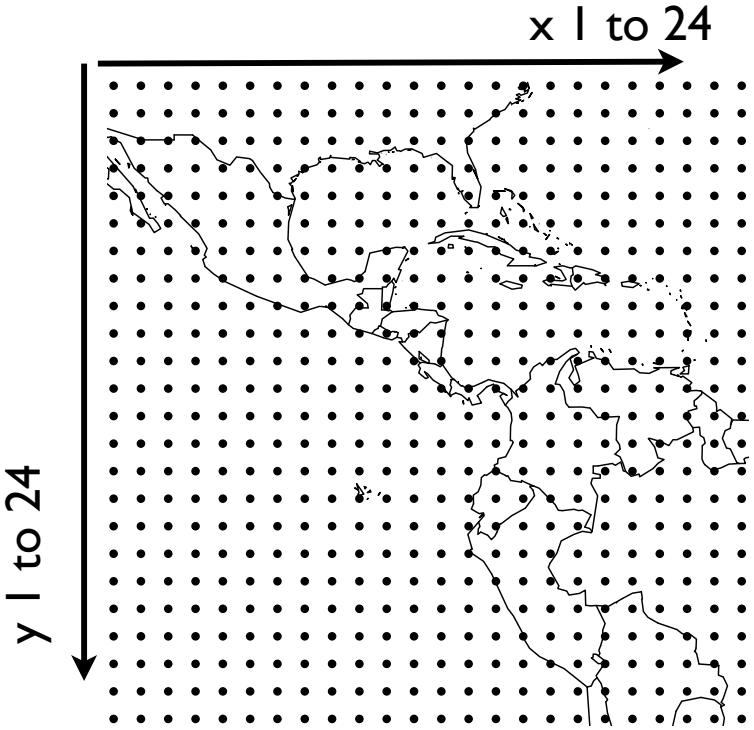
```
# using plyr
sepLM_plyr <- dlply(wages, .(id),
                    function(x) lm(lnw ~ exper, data = x))
coefs_plyr <- ldply(sepLM_plyr, coef)

# using base apply functions
sepDF <- split(wages, wages$id)
sepLM_alt <- lapply(sepDF,
                    function(x) lm(lnw ~ exper, data = x))
coefs_alt <- t(sapply(sepLM_alt, coef))
```

```
  test replications elapsed relative user.self sys.self
2 base           20  47.810 1.000000    47.702    0.111
1 plyr           20  51.477 1.076699    51.107    0.173
```

NASA Meteorological Data

- 24 x 24 grid across Central America
- satellite captured data:
  temperature,
  near surface temperature
  pressure
  ozone
  cloud coverage:
    low
    medium
    high



- for each location monthly averages for Jan 1995 to Dec 2000

Your Turn

- Compute the trend in temperature for each location using year and month as covariates.
  - Define a factor that denotes the location
  - Use base apply functions
  - Use rlm function in MASS package
- How many locations experienced increasing temperatures?
- Extract residual standard errors from each model. Are there areas of higher variability?

Additional Resources

- https://nsaunders.wordpress.com/2010/08/20/a-brief-introduction-to-apply-in-r/ (overview of base apply functions)
- *Data Manipulation with R* by Phil Spector (Chapter 8)
- *The Art of R Programming* by Norman Matloff (computational speed)