Introduction to R: **Summarizing Data**

Session 3, Part B

Nick Graetz[1]

[1] University of Pennsylvania, Population Studies Center

9/4/2020

In this lecture

1. Aggregation functions
2. Aggregating with `dcast()`
3. Aggregation vs reshape with `dcast()`
4. Summarizing data with data.table

Aggregating data

Aggregating (or collapsing) data refers to taking multiple rows (observations) and combining them into a single row according to some function (e.g., sum, or mean, or median, etc.).

There are a number of different functions in base R that are explicitly for aggregating, e.g.:

- `apply()`
- `tapply()`
- `aggregate()`

Aggregating data

Because this is such a common task, and potentially a quite complicated one, there are also a number of other packages that provide additional functions.

- `dcast()` in the `data.table` library can be used to aggregate, in addition to being used to reshape. It's one of the easiest functions to work with for simple aggregations, so this is what we will do today.
- We won't do more complicated aggregations at this point, but when you get there, the functions in the `plyr` and `dplyr` libraries are very useful when working with data frames.

Aggregating data

In all the examples using `dcast()` so far, we were careful to include all identifying variables in the casting formula such that these variables uniquely identified one single data value.

AGGREGATING DATA

In all the examples using `dcast()` so far, we were careful to include all identifying variables in the casting formula such that these variables uniquely identified one single data value.

If you don't include all identifying variables, something different happens:

```
> head(long, 2)
   cnty year sex variable  value
1: King 2010   1      pop 965486
2: King 2010   2      pop 971999
> dcast(long, cnty + sex ~ variable, value.var = "value")
Aggregate function missing, defaulting to 'length'
        cnty sex pop deaths
1:      King   1   4      4
2:      King   2   4      4
3:    Pierce   1   4      4
4:    Pierce   2   4      4
5: Snohomish   1   4      4
6: Snohomish   2   4      4
```

The variable that we didn't include (`year`) has been dropped. The four values present for each combination of `cnty`, `sex`, and `variable` (one for each of the four years) have been *aggregated*, in this case using the default function, `length()`, since no aggregation function was specified.

Aggregating data

We can aggregate using a wide range of functions:

```
> dcast(long, cnty + sex ~ variable, value.var = "value", fun.aggregate = sum)
        cnty sex     pop deaths
1:      King   1 3971435  24155
2:      King   2 3989386  24464
3:    Pierce   1 1600737  11904
4:    Pierce   2 1629950  11241
5: Snohomish   1 1460946   9319
6: Snohomish   2 1455889   9374

> dcast(long, cnty + sex ~ variable, value.var = "value", fun.aggregate = mean)
        cnty sex      pop  deaths
1:      King   1 992858.8 6038.75
2:      King   2 997346.5 6116.00
3:    Pierce   1 400184.2 2976.00
4:    Pierce   2 407487.5 2810.25
5: Snohomish   1 365236.5 2329.75
6: Snohomish   2 363972.2 2343.50

> dcast(long, cnty + sex ~ variable, value.var="value", fun.aggregate=quantile, p=0.25)
        cnty sex      pop  deaths
1:      King   1 978914.8 5951.50
2:      King   2 983941.2 6058.50
3:    Pierce   1 396580.0 2931.25
4:    Pierce   2 404838.8 2756.75
5: Snohomish   1 360971.0 2284.00
6: Snohomish   2 359777.0 2324.00
```

AGGREGATION

And by changing the casting formula, we can change what is aggregated over and how the final data are shaped:

```
> dcast(long, cnty ~ variable, value.var = "value", fun.aggregate = sum)
        cnty     pop deaths
1:      King 7960821  48619
2:    Pierce 3230687  23145
3: Snohomish 2916835  18693

> dcast(long, variable ~ cnty, value.var = "value", fun.aggregate = sum)
   variable    King  Pierce Snohomish
1:      pop 7960821 3230687   2916835
2:   deaths   48619   23145     18693

> dcast(long, . ~ variable, value.var = "value", fun.aggregate = sum)
   .      pop deaths
1: . 14108343  90457

> dcast(long, variable ~ ., value.var = "value", fun.aggregate = sum)
   variable        .
1:      pop 14108343
2:   deaths    90457
```

Reshaping vs aggregating

Remember, `dcast()` can do two (very) different things, depending on how it's used:

- reshape your data
- aggregate (collapse) your data

If the variables in the casting formula uniquely identify each data value, `dcast()` will **reshape**.

If not, `dcast()` will **aggregate** (and possibly reshape at the same time).

Summarizing data with data.table

There is actually a third component to manipulating data tables... the *by* column.

The command `DT[i, j, by]` can be read "Take *DT*, subset rows using *i*, then calculate *j* grouped by *by*"

```
> dt <- as.data.table(long)
> dt[cnty == "King", .(mean = mean(value)), by = .(year, varia
   year variable      mean
1: 2010      pop  968742.5
2: 2011      pop  985656.5
3: 2012      pop 1003787.0
4: 2013      pop 1022224.5
5: 2010   deaths    5879.0
6: 2011   deaths    6047.0
7: 2012   deaths    6148.0
8: 2013   deaths    6235.5
```

Summarizing data with data.table

A useful tool to combine with `by` is the special symbol, `.SD`. It stand for **S**ubset of **D**ata and holds the current group defined using `by`.

```
> dt <- as.data.table(iris)
> str(dt)
Classes 'data.table' and 'data.frame':  150 obs. of  5 variables:
 $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
 $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
 $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
 $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
 $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1
 - attr(*, ".internal.selfref")=<externalptr>
> dt[, lapply(.SD, mean), by = "Species"]
      Species Sepal.Length Sepal.Width Petal.Length
1:     setosa        5.006       3.428        1.462
2: versicolor        5.936       2.770        4.260
3:  virginica        6.588       2.974        5.552
   Petal.Width
1:       0.246
2:       1.326
3:       2.026
```

SUMMARIZING DATA WITH DATA.TABLE

`.SDcols` lets you subset the columns included in `.SD`:

```
> dt[, lapply(.SD, mean), by = "Species", .SDcols = grep("Sepa
+     names(dt), value = T)]
      Species Sepal.Length Sepal.Width
1:     setosa        5.006       3.428
2: versicolor        5.936       2.770
3:  virginica        6.588       2.974
```