REVOLUTION ANALYTICS

Data Aggregation with Base R

Revolution Analytics

Overview

At the end of this session, you should be able to:

- Use R to form cross-tabulation
- Use R to apply arbitrary functions to subsets of data sets

Outline

Contingency tables and `table()`

R has a variety of functions that are useful for data aggregation.

The simplest of these is `table()`:

```
table(mtcars[, "gear"])
```

```
##
##  3  4  5
## 15 12  5
```

Cross-tabulation

```
table(mtcars[, c("gear", "cyl")])
```

```
##      cyl
## gear  4  6  8
##    3  1  2 12
##    4  8  4  0
##    5  2  1  2
```

```
table(mtcars[, c("cyl", "gear")])
```

```
##    gear
## cyl  3  4  5
##   4  1  8  2
##   6  2  4  1
##   8 12  0  2
```

`table()` builds a contingency table of counts

Working with `xtabs()`

The function `xtabs()` allows you to do the same thing, but using a formula:

```
xtabs(~gear + cyl, mtcars)
```

```
##      cyl
## gear  4  6  8
##    3  1  2 12
##    4  8  4  0
##    5  2  1  2
```

Compare to previous output

Formulae revisited.

Same concept and specification as used in `lm()` in one of our previous sessions.

- Two sides separated by a “~” symbol
- Left side: dependent variables (in these examples, it is empty)
- Right side: independent variables, separated by a “+” in simple cases

In this example: the left side is empty, so it simply counts each time a value of `gear` appears with a value of `cyl`.

```
## Not evaluated again
xtabs(~gear + cyl, data = mtcars)
```

Contingency tables with `xtabs()`

The formula syntax makes it easy to calculate multi-dimensional contingencies:

```
xtabs(~gear, mtcars)
xtabs(~gear + carb, mtcars)
xtabs(~gear + cyl + carb, mtcars)
```

Order matters

Exercise

Your turn:

- How many cars in mtcars have am=1 and carb=2?
- How many cars in mtcars have hp<100, am=1 and carb=2?

Summary: Frequency tables

- use `table()` with the objects you actually want to cross-tabulate
- use `xtabs()` with a formula interface
- What about other types of aggregation?

Outline

Function aggregate()

With aggregate() you can calculate other functions beyond simple sums, e.g. mean horsepower:

```
aggregate(hp ~ cyl + gear, data = mtcars, FUN = mean)
```

```
##   cyl gear    hp
## 1   4    3  97.0
## 2   6    3 107.5
## 3   8    3 194.2
## 4   4    4  76.0
## 5   6    4 116.5
## 6   4    5 102.0
...
```

`aggregate()` function

Use is similar to `xtabs()`

- formula interface that now has a left side
  - Left side is the dependent variable you operate on at each combination of the independent variables
- an additional FUN argument
  - FUN is the function that you want to apply to the dependent variable.

FUN is arbitrary.

So you can calculate median HP, min HP, max HP – whatever you want.

```
aggregate(hp ~ cyl + gear, data = mtcars, FUN = median)
```

```
##   cyl gear    hp
## 1   4    3  97.0
## 2   6    3 107.5
## 3   8    3 180.0
## 4   4    4  66.0
## 5   6    4 116.5
## 6   4    5 102.0
...
```

```
## aggregate(hp ~ cyl + gear, data = mtcars, FUN = min)
## aggregate(hp ~ cyl + gear, data = mtcars, FUN = max)
```

FUN can return more than one value

```
aggregate(hp ~ cyl + gear, data = mtcars, FUN = quantile, probs = c(0.25,
  0.75))
```

```
##   cyl gear hp.25% hp.75%
## 1   4    3  97.00  97.00
## 2   6    3 106.25 108.75
## 3   8    3 175.00 218.75
## 4   4    4  64.25  93.50
## 5   6    4 110.00 123.00
## 6   4    5  96.50 107.50
...
```

FUN can be used on multiple variables

```
aggregate(cbind(hp, mpg) ~ cyl + gear, data = mtcars, FUN = mean)
```

```
##   cyl gear    hp   mpg
## 1   4    3  97.0 21.50
## 2   6    3 107.5 19.75
## 3   8    3 194.2 15.05
## 4   4    4  76.0 26.93
## 5   6    4 116.5 19.75
## 6   4    5 102.0 28.20
...
```

Outline

Using `tapply()`

In many ways `tapply()` is to `aggregate()` as `table()` is to `xtabs`.

The function `tapply` applies a function to a vector, indexed by the levels of a list of variables.

If the result of applying the function is a single number, the output will be a contingency table.

tapply Example 1

```
# Similar to Pivot Tables (PT) in Excel tapply(X, INDEX, FUN =
# NULL, ..., simplify = TRUE) ... where X are the PT Values, INDEX
# are the PT Row and Column Labels and FUN is the what we choose
# to summarize the Value field by in the PT

tapply(mtcars$mpg, mtcars[, c("gear", "carb")], mean)
```

```
##      carb
## gear     1     2    3     4    6  8
##    3 20.33 17.15 16.3 12.62   NA NA
##    4 29.10 24.75   NA 19.75   NA NA
##    5    NA 28.20   NA 15.80 19.7 15
```

Comparison

```
aggregate(mpg ~ gear + carb, data = mtcars, FUN = mean)
```

```
##    gear carb   mpg
## 1     3    1 20.33
## 2     4    1 29.10
## 3     3    2 17.15
## 4     4    2 24.75
## 5     5    2 28.20
## 6     3    3 16.30
...
```

`aggregate()` vs `tapply()`

- Output: dataframe vs. N-dimensional array
- Inclusion of missing values:
  - aggregate removes missing values
  - tapply does not

See `as.data.frame.table()` for dataframe output that includes `NA` values

Exercise: Practice with tapply

Your turn:

- Create a contingency table of counts for `mtcars` based on `am`, `gear`, and `carb`.
- Calculate the same table, but now as a sum of horse power (`hp`) instead of counts.
- Use `tapply()` to create a 3-d contingency table of the sums of `mpg` in `mtcars`. Use the columns `vs`, `am`, and `cyl` as contingency columns.
- Write a function that returns both the minimum and maximum values of a vector. Apply this function to `mpg` in `mtcars` over `gear` and `am`.

Summary

- Use `table()` or `xtabs()` to create frequency counts.
- Use `tapply()`, `aggregate()`, or `by()` to perform arbitrary computations on data.

Questions?

REVOLUTION ANALYTICS

Thank you

Revolution Analytics is the leading commercial provider of software and support for the popular open source R statistics language.

www.revolutionanalytics.com
1.855.GET.REVO
Twitter: @RevolutionR