Factors and Tables

Bok, Jong Soon
javaexpert@nate.com
www.javaexpert.info

Factors and Tables

- Factors form the basis for many of R's powerful operations, including many of those performed on tabular data.
- The motivation for factors comes from the notion of nominal, or categorical, variables in statistics.
- These values are non-numerical in nature, corresponding to categories such as Democrat, Republican, and Unaffiliated, although they may be coded using numbers

Factors and Levels (Cont.)

- An R *factor* might be viewed simply as a vector with a bit more information added (though, as seen below, it's different from this internally).
- That extra information consists of a record of the distinct values in that vector, called *levels*.
- Here's an example:

Console C:/R Home/

```
> x <- c(5, 12, 13, 12)
> xf <- factor(x)
> xf
[1] 5  12 13 12
Levels: 5 12 13
```

Factors and Levels (Cont.)

- The core of **xf** is not **(5,12,13,12)** but rather **(1,2,3,2)**.
- The latter means that our data consists first of a level-1 value, then level-2 and level-3 values, and finally another level-2 value.
- So the data has been recoded by level.
- The levels themselves are recorded too, of course, though as characters such as **"5"** rather than **5**.

Console C:/R Home/

```
> str(xf)
 Factor w/ 3 levels "5","12","13": 1 2 3 2
>
> unclass(xf)
[1] 1 2 3 2
attr(,"levels")
[1] "5"  "12" "13"
```

Factors and Levels (Cont.)

- The length of a factor is still defined in terms of the length of the data rather than, say, being a count of the number of levels:

```
Console C:/R Home/
>
> length(xf)
[1] 4
```

Factors and Levels (Cont.)

- **We can anticipate future new levels, as seen here:**

Console C:/R Home/

```
> x <- c(5, 12, 13, 12)
> xff <- factor(x, levels=c(5, 12, 13, 88))
> xff
[1] 5  12 13 12
Levels: 5 12 13 88
>
> xff[2] <- 88
> xff
[1] 5  88 13 12
Levels: 5 12 13 88
```

Factors and Levels (Cont.)

- **By the same token, you cannot sneak in an "illegal" level.**

```
Console C:/R Home/
> xff
[1] 5  88 13 12
Levels: 5 12 13 88
>
> xff[2] <- 28
Warning message:
In `[<-.factor`(`*tmp*`, 2, value = 28) :
  invalid factor level, NA generated
```

Common Functions Used with Factors

- With factors, have yet another member of the family of apply functions, `tapply()`.
- look at that function, as well as two other functions commonly used with factors: `split()` and `by()`.

The tapply() Function

- Suppose we have a vector `x` of `ages` of voters and a factor `f` showing some no-numeric trait of those voters, such as party affiliation (Democrat, Republican, Unaffiliated).
- We might wish to find the `mean` ages in `x` within each of the party groups.

```
Console C:/R Home/
> ages <- c(25, 26, 55, 37, 21, 42)
> ages
[1] 25 26 55 37 21 42
>
> affils <- c("R", "D", "D", "R", "U", "D")
> affils
[1] "R" "D" "D" "R" "U" "D"
```

The tapply() Function (Cont.)

- In typical usage, the call `tapply(x,f,g)` has `x` as a vector, `f` as a factor or list of factors, and `g` as a function.
- The function `g()` in our little example above would be R's built-in `mean()` function.

```
Console C:/R Home/
> tapply(ages, affils, mean)
 D  R  U
41 31 21
```

- The operation performed by `tapply()` is to (temporarily) split `x` into groups, each group corresponding to a level of the factor, and then apply `g()` to the resulting subvectors of `x`.

The tapply() Function (Cont.)

- **Suppose that have an economic data set that includes variables for gender, age, and income.**
- **Here, the call `tapply(x,f,g)` might have `x` as income and `f` as a pair of factors: one for gender and the other coding whether the person is older or younger than 25.**
- **We may be interested in finding mean income, broken down by gender and age.**
- **If we set `g()` to be `mean()`, `tapply()` will return the mean incomes in each of four subgroups:**
  - Male and under 25 years old
  - Female and under 25 years old
  - Male and over 25 years old
  - Female and over 25 years old

The tapply() Function (Cont.)

Console C:/R Home/

```
> d <- data.frame(list(gender=c("M","M","F","M","F","F"),
+     age=c(47,59,21,32,33,24),
+     income=c(55000,88000,32450,76500,123000,45650)))
>
> d
  gender age income
1      M  47  55000
2      M  59  88000
3      F  21  32450
4      M  32  76500
5      F  33 123000
6      F  24  45650
>
> d$over25 <- ifelse(d$age > 25, 1, 0)
> d
  gender age income over25
1      M  47  55000      1
2      M  59  88000      1
3      F  21  32450      0
4      M  32  76500      1
5      F  33 123000      1
6      F  24  45650      0
>
> tapply(d$income, list(d$gender, d$over25), mean)
      0         1
F 39050 123000.00
M    NA  73166.67
```

The tapply() Function (Cont.)

- We specified two factors, gender and indicator variable for age over or under 25.
- Since each of these factors has two levels, `tapply()` partitioned the income data into four groups, one for each combination of gender and age, and then applied to `mean()` function to each group.

The split() Function

- `tapply()` splits a vector into groups and then applies a specified function on each group.
- `split()` stops at that first stage, just forming the groups.
- The basic form is `split(x,f)`.
- With `x` and `f` playing roles similar to those in the call `tapply(x,f,g)`.
- `x` being a vector or data frame.
- `f` being a factor or a list of factors.
- The action is to split `x` into groups, which are returned in a list.

The split() Function (Cont.)

Console C:/R Home/

```
> d
  gender age income over25
1      M  47  55000      1
2      M  59  88000      1
3      F  21  32450      0
4      M  32  76500      1
5      F  33 123000      1
6      F  24  45650      0
>
> split(d$income, list(d$gender, d$over25))
$F.0
[1] 32450 45650

$M.0
numeric(0)

$F.1
[1] 123000

$M.1
[1] 55000 88000 76500
```

- The output of `split()` is a list.
- So the last vector, was named `M.1` to indicate that it was the result of combining `M` in the first factor and `1` in the second.

The split() Function (Cont.)

Console C:/R Home/

```
> g <- c("M", "F", "F", "I", "M", "M", "F")
> split(1:7, g)
$F
[1] 2 3 7

$I
[1] 4

$M
[1] 1 5 6
```

- We wanted to determine the indices of the vector elements corresponding to male, female, and infant.
- The data in that little example consisted of the seven-observation vector `("M","F","F","I","M","M","F")`, assigned to **g**.

Working with Tables

Console C:/R Home/

```
> u <- c(22, 8, 33, 6, 8, 29, -2)
> fl <- list(c(5, 12, 13, 12, 13, 5, 13),
+     c("a", "bc", "a", "a", "bc", "a", "a"))
>
> tapply(u, fl, length)
   a bc
5  2 NA
12 1  1
13 2  1
```

- **tapply()** temporarily breaks **u** into subvectors.
- Applies the **length()** function to each subvector.
- Those subvector lengths are the counts of the occurrences of each of the $3 \times 2 = 6$ combinations of the two factors.
- In statistics, this is called a *contingency table*.

Working with Tables (Cont.)

- There is one problem in this example.
- The `NA` value.
- It really should be `0`.
- It is meaning that in no cases did the first factor have level 5 and the second have level "bc".
- The `table()` function creates *contingency tables* correctly.

Working with Tables (Cont.)

```
Console C:/R Home/
>
> fl
[[1]]
[1]  5 12 13 12 13  5 13

[[2]]
[1] "a"  "bc" "a"  "a"  "bc" "a"  "a"

>
> table(fl)
    fl.2
fl.1 a bc
  5  2  0
  12 1  1
  13 2  1
```

- The first argument in a call to **`table()`** is either a factor or a list of factors.
- The two factors here were **`(5,12,13,12,13,5,13)`** and **`("a","bc","a","a","bc","a","a")`**.

Working with Tables (Cont.)

ct.dat - Notepad

File Edit Format View Help

```
"Vote for X"      "Voted For X Last Time"
"Yes"             "Yes"
"Yes"             "No"
"No"              "No"
"Not Sure"        "Yes"
"No"              "No"
```

- The file ct.dat consists of election-polling data, in which candidate X is running for reelection.
- The ct.dat file looks like left.

Working with Tables (Cont.)

- **In the usual statistical fashion, each row in this file represents one subject under study.**
- **In this case, we have asked five people the following two
  questions:**
  - Do you plan to vote for candidate X?
  - Did you vote for X in the last election?

Working with Tables (Cont.)

- **Let's read in the file:**

```
Console C:/R Home/
> ct <- read.table("ct.dat", header=T)
> ct
  Vote.for.X Voted.For.X.Last.Time
1        Yes                   Yes
2        Yes                    No
3         No                    No
4   Not Sure                   Yes
5         No                    No
```

Working with Tables (Cont.)

- Can use the table() function to compute the contingency table for this data.

Console C:/R Home/

```
> cttab <- table(ct)
> cttab
           Voted.For.X.Last.Time
Vote.for.X No Yes
  No        2   0
  Not Sure  0   1
  Yes       1   1
```

Working with Tables (Cont.)

- Can get one-dimensional counts, which are counts on a single factor, as follows.

Console C:/R Home/

```
> table(c(5, 12, 13, 12, 8, 5))

 5  8 12 13
 2  1  2  1
```

Working with Tables (Cont.)

- Here's an example of a three-dimensional table, involving voters' genders, race (white, black, Asian, and other), and political views (liberal or conservative).

Console C:/R Home/

```
> gender <- c("M","M","F","M","F","F")
> race <- c("W","W","A","O","B","B")
> pol <- c("L","L","C","L","L","C")
> v <- data.frame(gender, race, pol, stringsAsFactors = FALSE)
> v
  gender race pol
1      M    W   L
2      M    W   L
3      F    A   C
4      M    O   L
5      F    B   L
6      F    B   C
```

Working with Tables (Cont.)

Console C:/R Home/

```
> vt <- table(v)
>
> vt
, , pol = C

      race
gender 0 A B W
     F 0 1 1 0
     M 0 0 0 0

, , pol = L

      race
gender 0 A B W
     F 0 0 1 0
     M 1 0 0 2
```

- R prints out a three-dimensional table as a series of two-dimensional tables.
- In this case, it generates a table of gender and race for conservatives and then a corresponding table for liberals.
- For example, the second two dimensional table says that there were two white male liberals.

Matrix/Array-Like Operations on Tables

- Can access the table cell counts using matrix notation.

Console C:/R Home/

```
> cttab
          Voted.For.X.Last.Time
Vote.for.X No Yes
  No        2   0
  Not Sure  0   1
  Yes       1   1
>
> class(cttab)
[1] "table"
>
> cttab[1,1]
[1] 2
>
> cttab[1,]
 No Yes
  2   0
```

Matrix/Array-Like Operations on Tables (Cont.)

- Can multiply the matrix by a scalar.
- For instance, here's how to change cell counts to proportions.

```
Console C:/R Home/
> cttab / 5
           Voted.For.X.Last.Time
Vote.for.X   No Yes
  No        0.4 0.0
  Not Sure  0.0 0.2
  Yes       0.2 0.2
```

Matrix/Array-Like Operations on Tables (Cont.)

- In statistics, the ***marginal*** values of a variable are those obtained when this variable is held constant while others are summed.
- In the voting example, the marginal values of the `Vote.for.X` variable are `2 + 0 = 2`, `0 + 1 = 1`, and `1 + 1 = 2`.
- We can of course obtain these via the matrix `apply()` function:

```
Console C:/R Home/
> apply(cttab, 1, sum)
      No Not Sure      Yes
       2        1        2
```

Matrix/Array-Like Operations on Tables (Cont.)

- R supplies a function **addmargins()** for previous purpose : that is, to find marginal totals.

```
Console C:/R Home/
> addmargins(cttab)
             Voted.For.X.Last.Time
Vote.for.X No Yes Sum
  No        2   0   2
  Not Sure  0   1   1
  Yes       1   1   2
  Sum       3   2   5
```

- we got the marginal data for both dimensions at once, conveniently superimposed onto the original table.

Matrix/Array-Like Operations on Tables (Cont.)

- We can get the names of the dimensions and levels through `dimnames()`, as follows:

```
Console C:/R Home/
> dimnames(cttab)
$Vote.for.X
[1] "No"          "Not Sure" "Yes"

$Voted.For.X.Last.Time
[1] "No"  "Yes"
```

Other Factor- and Table-Related Functions

- R includes a number of other functions that are handy for working with tables and factors.
- We'll discuss two of them here: `aggregate()` and `cut()`.

The aggregate() Function

- The aggregate() function calls `tapply()` once for each variable in a group.
- For example, in the abalone data, we could find the median of each variable,
  broken down by gender, as follows: