Lecture 7. Factors and Tables

R and Data Visualization

BIG2006, Hanyang University, Fall 2022

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.

We will begin by looking at the extra information contained in factors and then focus on the functions used with factors.

Factors and Levels

An R *factor* might be viewed simply as a vector with a bit more information added.

That extra information consists of a record of the distinct values in that vector, called *levels*.

```
x <- c(5,12,13,12)
xf <- factor(x)
xf # the distinct values (5, 12, and 13) are the levels.
```

```
## [1] 5  12 13 12
## Levels: 5 12 13
```

```r
str(xf)
```

```
##  Factor w/ 3 levels "5","12","13": 1 2 3 2
```

```r
unclass(xf) # the data has been recorded by level
```

```
## [1] 1 2 3 2
## attr(,"levels")
## [1] "5"  "12" "13"
```

```r
length(xf)
```

```
## [1] 4
```

- The core of `xf` is not (5,12,13,12) but rather (1,2,3,2).

- Our data consists first of a level-1 value, then level-2 and level-3 values, and finally another level-2 value.

- We can anticipate future new levels.

```
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
```

- We cannot sneak in an “illegal" level.

```
> xff[2] <- 28
Warning message:
In `[<-.factor`(`*tmp*`, 2, value = 28) :
invalid factor level, NAs generated
```

Common Functions Used with Factors

With factors, we have another member of family of `apply` functions, `tapply`.

We will look at that function, as well as two other functions commonly used with factors: `split()` and `by()`.

The tapply() Function

- `tapply(x, f, g)` has `x` as a vector, `f` as a factor or list of factors, and `g` as a function.
  - Each factor in `f` must have the same length as `x`.

```
ages <- c(25,26,55,37,21,42)
affils <- c("R","D","D","R","U","D")
tapply(ages,affils,mean)
```

```
##  D  R  U
## 41 31 21
```

Note: `tapply()` treated the vector ("R","D","D","R","U","D") as a factor with levels "D", "R", and "U".

- Two or more factors?

```r
d <- data.frame(list(gender=c("M","M","F","M","F"),
    age=c(47,59,21,32,33),
    income=c(55000,88000,32450,76500,123000)))
d
```

```
##   gender age income
## 1      M  47  55000
## 2      M  59  88000
## 3      F  21  32450
## 4      M  32  76500
## 5      F  33 123000
```

```r
d$over25 <- ifelse(d$age > 25, 1, 0)
```

```r
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
```

```r
tapply(d$income,list(d$gender,d$over25),mean)
```

```
##       0         1
## F 32450 123000.00
## M    NA  73166.67
```

Note: We specified two factors, genders and indicator variable (for age over or under 25). Thus, `tapply`() partitioned the income data into four groups and applied to `mean`() to each group.

The split() Function

- split(x, f): splits a vector into groups

```
split(d$income,list(d$gender,d$over25))
```

```
## $F.0
## [1] 32450
##
## $M.0
## numeric(0)
##
## $F.1
## [1] 123000
##
## $M.1
## [1] 55000 88000 76500
```

- We want to determine the indices of the vector elements corresponding to male, female, and infant.

```r
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
```

The by() Function

- It works like `tapply()`, but it is applied to objects rather than vectors, i.e., an object-oriented wrapper for `tapply()` applied to data frames.

```
head(warpbreaks)
```

```
##   breaks wool tension
## 1     26    A       L
## 2     30    A       L
## 3     54    A       L
## 4     25    A       L
## 5     70    A       L
## 6     52    A       L
```

```r
by(warpbreaks[, 1:2], warpbreaks[,"tension"], summary)[1:2]
```

```
## $L
##      breaks      wool
##  Min.   :14.00   A:9
##  1st Qu.:26.00   B:9
##  Median :29.50
##  Mean   :36.39
##  3rd Qu.:49.25
##  Max.   :70.00
##
## $M
##      breaks      wool
##  Min.   :12.00   A:9
##  1st Qu.:18.25   B:9
##  Median :27.00
##  Mean   :26.39
##  3rd Qu.:33.75
##  Max.   :42.00
```

Working with Tables

- Contigency table in statistics.

```
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
```

Note: `tapply()` temporarily breaks `u` into subvectors, and then applies the `length()` to each subvector. 5 occurred twice with "a" and not at all with "bc"; hence the entries 2 and NA in the first row of the output.

- NA, meaning that in no cases, so should be changed to 0.
- `table()` creates contingency tables correctly.

```
table(fl)
```

```
##       fl.2
## fl.1 a bc
##   5  2  0
##   12 1  1
##   13 2  1
```

- A three-dimensional table, involving voters' genders, race, and political views (liberal or conservative):

```r
v <- data.frame(gender=c("M","M","F"),race=c("W","B","A"),
                pol=c("L","L","C"))
table(v) # print a 3D table as a series of 2D tables
```

```
## , , pol = C
##
##       race
## gender A B W
##      F 1 0 0
##      M 0 0 0
##
## , , pol = L
##
##       race
## gender A B W
##      F 0 0 0
##      M 0 1 1
```

Other Factor- and Table-Related Functions

R includes a number of other functions that are handy for working with tables and factors.

We will discuss two of them: `aggregate()` and `cut()`.

The aggregate() Function

- Splits the data into subsets, computes summary statistics for each subsets and returns the result in a group by form.

```
agg_mean = aggregate(iris[,1:4],by=list(iris$Species),
                     FUN=mean, na.rm=TRUE)
agg_mean
```

```
##      Group.1 Sepal.Length Sepal.Width Petal.Length Petal.Width
## 1     setosa        5.006       3.428        1.462       0.246
## 2 versicolor        5.936       2.770        4.260       1.326
## 3  virginica        6.588       2.974        5.552       2.026
```

The cut() Function

- A common way to generate factors, especially tables

```
set.seed(20220912)
z <- round(runif(8),3)
z
```

```
## [1] 0.432 0.134 0.209 0.670 0.541 0.536 0.332 0.540
```

```
seq(from=0,to=1,by=0.1)
```

```
##  [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
```

```
binmarks <- seq(from=0,to=1,by=0.1)
cut(z,binmarks,labels=FALSE)
```

```
## [1] 5 2 3 7 6 6 4 6
```

Note: This says that z[1], 0.432, fell into bin 5, which was (0.4,0.5].

Reference

- Matloff, N. The Art of R Programming: A Tour of Statistical Software Design. No Starch Press. Chapter 6.