Data Programming with R

Isabella Gollini

Lecture 5 - Factors and Tables

Factors and Tables

- What are factors?
- Factors and levels
- `tapply`, `split` and `by`
- Working with tables
- Tables and matrices
- `aggregate` and `cut`
- RStudio projects and Keyboard Shortcuts.

What are factors?

- Categorical data arise in 3 different forms:
  - Nominal (e.g. blue, green, brown)
  - Ordinal (e.g. A, B, C)
  - Interval (e.g. 1-10, 11-20, 21-30)
- We often code the different **levels** as numbers (e.g. male = 1, female = 2). Often these numbers don't mean anything!
- When the data are ordinal however (e.g. small = 1, medium = 2, large = 3), these numbers do mean something and we require some special methods to analyse them.
- Factors in R store categorical data: they are simply vectors with some additional information on the levels.

Factors and levels

Here's an example of a factor:

```
x <- c(5, 12, 13, 12)
xf <- factor(x)
xf
```

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

The distinct values are the levels. Looking at the **structure** of `xf`:

```
str(xf)
```

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

The values of `xf` are now actually `1`, `2`, `3`, `2`, rather than the values in `x`. Notice that the levels are characters.

More factors and levels

If we know in advance there will be more levels than just in the present data we can specify them:

```
xf2 <- factor(x, levels = c(5, 12, 13, 88))
xf2
```

```
## [1] 5  12 13 12
## Levels: 5 12 13 88
```

If we haven't specified a level then we can't use it in a factor:

```
xf[2] <- 100
```

```
## Warning in `[<-.factor`(`*tmp*`, 2, value = 100): invalid factor level, NA
## generated
```

```
xf
```

```
## [1] 5    <NA> 13   12
## Levels: 5 12 13
```

Functions for factors: `tapply`

Let's suppose we have a list of chick weight at 21 days together with a diet the chick received:

```
weight <- c(205, 331, 175, 117, 272, 251)
diet <- c('Diet_A', 'Diet_B', 'Diet_B', 'Diet_A', 'Diet_C', 'Diet_B')
```

Say we want to calculate the mean weight within each Diet group.

The command is `tapply(x, f, g)` where `x` is a vector, `f` is a factor, and `g` is a function. Here `g` is the function `mean`, `x` is the `weight` vector and `diet` is the factor:

```
tapply(weight, diet, mean)
```

```
##   Diet_A   Diet_B   Diet_C
## 161.0000 252.3333 272.0000
```

- Note: here we haven't specified `diet` as a factor but `tapply` applies `as.factor` to it.
- The `tapply` function splits the vector of ages up into different diets/groups and runs mean on each group.

Functions for factors: `split`

The `split` function does the first part of `tapply`, it just splits things into groups

```
split(weight, diet)
```

```
## $Diet_A
## [1] 205 117
##
## $Diet_B
## [1] 331 175 251
##
## $Diet_C
## [1] 272
```

- Note that it returns a list. `tapply` takes that list and uses `lapply` on it.

- If you type `tapply` without brackets at the R prompt you will see the contents of the function. Note that `tapply` uses `split` and `lapply`.
- See the screencast for more detail on `tapply` and `split`.

Functions for factors: `by`

- The function `by` works a bit like `tapply` but is applied to objects rather than vectors.
- The general code is `by(x, f, g)` where `x` is a data object, `f` is a factor on which to split the data into groups and `g` is a function to be applied to each group.

Going back to the birthweight example:

```
library(MASS)
by(birthwt, birthwt$race, function(m) glm(m[,1] ~ m[,2], family = binomial))
```

- This will produce a logistic regression of age (column 2 of `birthwt`) against low birth weight (column 1) by `race`.
- Replacing `by` with `tapply` will give an error as `tapply` works on a vector, not a data frame.

Tables

- The `table` function
- Multidimensional tables
- Operations on tables
- `aggregate` and 'cut

Working with tables

We can create a table using the `table` function:

```
birthwt2 <- data.frame(birthwt$low, birthwt$race)
head(birthwt2)
```

```
##   birthwt.low birthwt.race
## 1           0            2
## 2           0            3
## 3           0            1
## 4           0            1
## 5           0            1
## 6           0            3
```

```
table(birthwt2)
```

```
##            birthwt.race
## birthwt.low  1  2  3
##           0 73 15 42
##           1 23 11 25
```

The `table` function will take a data frame of 2 (or more) columns and turn them into a contingency table

If we just wanted counts on one of the variables we could use, for example:

```
table(birthwt$low)
```

```
##
##   0   1
```

```
## 130  59
```

Multi-dimensional tables

Quite often tables have more than 2 dimensions:

```
birthwt3 <- data.frame(birthwt$low, birthwt$race, birthwt$smoke)
table(birthwt3)
```

```
## , , birthwt.smoke = 0
##
##             birthwt.race
## birthwt.low  1  2  3
##           0 40 11 35
##           1  4  5 20
##
## , , birthwt.smoke = 1
##
##             birthwt.race
## birthwt.low  1  2  3
##           0 33  4  7
##           1 19  6  5
```

- R presents these as a series of 2D tables. Be careful - the output can get messy!
- Notice that a table looks a lot like a matrix. A multi-dimensional table is just an array.
- We can use all the matrix/array functions on tables

Functions on tables

We can access part of a table in exactly the same way as a matrix/array.

```
tab1 <- table(birthwt2)
tab1[1, 2]
```

```
## [1] 15
```

We can perform scalar multiplication on a table:

```
tab1 / sum(tab1)
```

```
##             birthwt.race
## birthwt.low          1          2          3
##           0 0.38624339 0.07936508 0.22222222
##           1 0.12169312 0.05820106 0.13227513
```

This turns the table counts into proportions. We can find marginal counts by using `apply`:

```
apply(tab1, 1, sum)
```

```
##   0   1
## 130  59
```

A nicer way of doing this is with `addmargins`

`aggregate`

The `aggregate` function calls `tapply` for each variable in a group.

For example:

```
aggregate(birthwt$age, list(birthwt$race), mean)
```

```
##   Group.1        x
## 1       1 24.29167
## 2       2 21.53846
## 3       3 22.38806
```

- This is the **mean age** broken down by **race**. We can aggregate over multiple variables if required.

`cut`

- `cut` is a useful function for generating a factor from a list of bins. The general use is `cut(x, b)` where `x` is a numeric variable and `b` is a set of bins

You can optionally give labels to the bins too:

```
cut(birthwt$bwt, c(0, 2000, 4000, 6000), labels = c('Low', 'Medium', 'High'),
    ordered_result = TRUE)
```

- Here we are cutting the raw birth weights into 3 categories, 0-2000g, 2000-4000g, and 4000-6000g and giving them three labels.
- The argument `ordered_result` allows you to creare an ordered factor
- If you do not specify the labels then R will create labels for you. Alternatively use `labels = FALSE`.

Some useful extras

- RStudio Shortcuts
- The working directory
- RStudio Projects

RStudio Shortcuts

RStudio provides a few keyboard shortcuts

- For more RStudio shortcuts go to `Tools` > `Keyboard Shortcuts Help`
- To add your own shortcuts `Tools` > `Modify Keyboard Shortcuts ...`

The working directory

- R will create, read & save files to the working directory unless told otherwise.

We can see the directory R currently believes to be the working directory with the command:

```
getwd()
```

- R has a number of useful functions to play with files in a directory, e.g. `list.files()`, `file.choose()`, `file.info()` etc, etc.
- See `help(files)` for many more commands.

You can set the working directory with the command:

```
setwd('path/to/somewhere')
```

where the path is the directory with which you want to work. You can also set the working directory using the 'Session' menu in RStudio.

- Working directory is specific of the machine that you are using
- This is fine when you work on a single project in a single machine. RStudio provides a better solution!

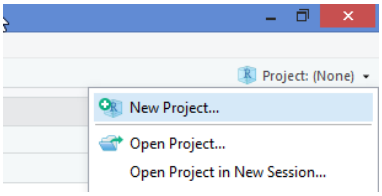
RStudio Projects

An RStudio project is a context for work on a specific project

- automatically sets working directory to project root
- has separate workspace and command history
- works well with version control via git or svn

Create a project from a new/existing directory via the File Menu or the New Project button

Switch project, or open a different project in a new RStudio instance via the Project menu.



Lessons from this week

- Factors are useful for storing nominal/ordinal data in a compressed format. R makes sure they are handled correctly in other functions.
- Tables are just matrices/arrays - all the usual functions `apply`.
- Some useful functions `tapply`, `aggregate`, `cut`.
- Work with RStudio projects and Keyboard Shortcuts.