Data Science and R

2.2. Factors, Generating Random Data

SUNG KYUN KWAN UNIVERSITY

Admin

- For lab exercises, if you are asked to give an output result and you can provide code to give that output, then just the code is enough
- Assignment 1 should be out by the end of this week (1 week)
- Next Monday Sept 12th is a holiday
  - There will be a recorded lecture (and lab)

Recap Matrices

- 2-D representation of data with the same data type (numbers, booleans or characters)
- Creation via `cbind(),rbind, matrix()`
- Indexing via numbers and logical operations
  - `my_m[1,3]  # returns element in row 1 col 3`
  - `my_m[2:5] # returns elements 2 to 5 by column`
  - `my_m[my_m > 3]  # returns all elements greater than 3`
  - `my_m > 3  # returns TRUE for elements greater than 3 and FALSE otherwise`
- Matrix functions: `rownames(), colnames(), rowSum(), colSum()`

Factors

- Factors are used to work with categorical variables – those that have a fixed and known set of possible values
  - Gender – “Female”, “Male”, “Other”
  - Marital status – “Married”, “Single”, “Separated”, “Divorced”, “Widowed”, “Complicated”
  - Grades – “A”, “B”, “C”, “D”, “E” (or “Pass”/“Fail”)
  - Months – “Jan”, “Feb”, . . . , “Dec”
- Factors are integer vectors in R
- Both numeric and character variables can be made into factors
- Factors are useful for visualisation and analysis later on when data in the same category or “group” will be treated as one entity
- The different fixed values or categories are referred to as levels, which can be ordered or unordered

Factors: Creation

- Factors can be created using the `factor()` function
  - `gender_data <- c("male", "female", "female", "male", "other")`
  - `gender_factor <- factor(gender_data)`
  - `gender_factor <- factor(c("male", "female", "female", "male", "other"))`
  - `levels(gender_factor)  # female male other`
  - `nlevels(gender_factor) # 3`
- R will automatically assign 1 to `female`, 2 to `male` and 3 to `other`
- `# 1=female, 2=male, 3=others internally (`**`alphabetically`**`)`
- Changing factor levels (factor is still NOT ordered):
  - `gender_factor <- `**`relevel`**`(gender_factor, "male") # now male comes first and the others are pushed down`
  - `gender_factor # male female female male other`
  - `Levels: male female other # 1= male, 2=female, 3=other`

Factors: Ordering

- Ordering of categories might be important (e.g. low, medium, high)

```
heat <- c("low", "high", "high", "medium", "low")
heat_factor <- factor(heat)
levels(heat_factor)  # high low medium - alphabetical
max(heat_factor)  # Returns an error!
```

- Reassign factor to the old factor plus levels information

```
heat_factor <- factor(heat, levels = c("low", "medium", "high"), ordered =
TRUE)
heat_factor
[1] low high high medium low
Levels: low < medium < high
```

- Or use `ordered()` to an existing unordered factor

```
heat_factor <- ordered(heat_factor, levels = c("low", "medium", "high"))
str(heat_factor) # structure
Ord.factor w/ 3 levels "low"<"medium"<..: 1 3 3 2 1
```

Converting Numerics to Factors

- Let’s say we have ages of a group of people as follows: `55,27,22,60,18,20,35,38,26,67,78,19,44,30,28,21,15,70,55,21` stored in the vector `age`
- Create age group categories – `A:<=20, B:21-30, C:31-40, D:41-50, E:51-60, F>60`
  - `ageCat <- cut(age, breaks = c(0,20,30,40,50,60,100), labels=c("A","B","C","D","E","F"))`
  - 0 is the lower bound and 100 is the upper bound
  - By default the intervals are left-open, (a,b] meaning the border values go to the left category
  - `head(age)    # 55 27 22 60 18 20`
  - `head(ageCat) # E B B E A A`
- You could also specify the number of categories and R will determine the intervals for these categories, but obviously you have more control when specifying them yourself
  - `ageCat <- cut(age, breaks = 6, labels=c("A","B","C","D","E","F"))`

Generating Random Data

- Use the `sample()` function to generate random data
- **`sample(x, size, replace=FALSE, prob=NULL)`**
  - `x`: vector or elements from which to choose from, e.g. coin toss c("H","T"), 1:10, c(0,1)
  - `size`: how many items you want to generate
  - `replace`: could the element occur more than once? Set to `FALSE` by default, so must be set to `TRUE` if `size > length(x)`
- Examples
  - **`sample(5)`** `# generates 5 numbers between 1 and 5, with no numbers repeating, same as sample(1:5)`
  - **`sample(1:10, 5, replace=T)`** `# generates 5 numbers between 1 and 10, the numbers do not have to be unique, i.e. any sampled number can be repeated`
  - **`sample(1:10, 20, replace=T)`** `# generates 20 numbers between 1 and 10 and obviously there will be repeated numbers`
  - **`sample(1:10, 20)`** `# will throw an error as replace is FALSE, meaning numbers cannot be repeated but this is not possible as size > length(x)`
  - **`sample(1:3, 100, replace=T, prob=c(0.5, 0.3, 0.2))`** `# generates 100 integers between 1 and 3 with 50% probability given to 1, 30% to 2 and 20% to 3`

Tips

- Common mistake: forgetting to add `c()` when creating factors
- Computation for cases such as `1 <= x <= 10` need to be evaluated as follows: `x >= 1 & x <= 10`
- Useful commands:
  - `ls()  # lists current objects (also in Environment panel)`
  - `rm(object)  # removes (deletes an object)`
  - `summary(object) # summary statistics on object`
  - `rm(list=ls()) # delete all the variables (clears environment)`