Data analysis and visualization using R (1)

using and writing functions in R

Michiel Noback

September 2019

Contents

Descriptive statistics

Descriptive stats functions

- R provides a wealth of descriptive statistics functions
- They are listed on the next two slides
- The ones with an asterisk are described in more detail.

Descriptive statistics functions (1)

function	purpose
mean( )	mean
median( )	median
min( )	minimum
max( )	maximum
range( )	min and max

Descriptive statistics functions (2)

function	purpose
var( )	variance s^2
sd( )	standard deviation s
summary( )	6-number summary
quantile( ) *	quantiles
IQR( ) *	interquantile range

The quantile() function

- Gives the data values corresponding to the specified quantiles
- The function defaults to the quantiles 0%   25%   50%   75%   100%

```
quantile(ChickWeight$weight)
```

```
    0%    25%    50%    75%   100% 
 35.00  63.00 103.00 163.75 373.00 
```

```
quantile(ChickWeight$weight, probs = seq(0, 1, 0.2))
```

```
   0%   20%   40%   60%   80%  100% 
 35.0  57.0  85.0 126.0 181.6 373.0 
```

Interquantile range IQR()

- Gives the range between 25% and 75% quantiles

```
IQR(ChickWeight$weight)
```

```
[1] 100.75
```

```
## same as
quantile(ChickWeight$weight)[4] - quantile(ChickWeight$weig
```

```
   75%
100.75
```

```
## same as
diff(quantile(ChickWeight$weight, probs = c(0.25, 0.75)))
```

```
   75%
100.75
```

boxplot() is summary() visualized

- Boxplot is a graph of the 5-number summary, but summary() also gives the mean

```r
summary(ChickWeight$weight)
```

```
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max. 
   35.0    63.0   103.0   121.8   163.8   373.0 
```

```r
boxplot(ChickWeight$weight)
```

General purpose functions

Remove objects from memory

- When working with large datasets it may be usefull free some memory one in a while (i.e. intermediate results)
- use `ls()` to see what is in menory
- use `rm()` to delete single items: `rm(genes)`, `rm(x, y, z)`
- clear all by typing `rm(list = ls())`

File system operations

- `getwd()` returns the current working directory
- `setwd(</path/to/folder>)` sets the current working directory
- `dir()`, `dir(path)` lists the contents of the current directory or of *path*
- *path* can be defined as
  - Windows: `"E:\\emile\\datasets"`
  - Linux/Mac: `"~/datasets"` or `"/home/emile/datasets"`

Glueing text pieces: paste()

- Use paste() to combine elements into a string

```r
paste(1, 2, 3)
```

```
[1] "1 2 3"
```

```r
paste(1, 2, 3, sep="-")
```

```
[1] "1-2-3"
```

```r
paste(1:12, month.abb)
```

```
 [1] "1 Jan"  "2 Feb"  "3 Mar"  "4 Apr"  "5 May"  "6 Jun"
[11] "11 Nov" "12 Dec"
```

Investigate structure: `str()`

- Use `str()` to investigate the structure of a complex object

```
str(chickwts)
```

```
'data.frame':   71 obs. of  2 variables:
 $ weight: num  179 160 136 227 217 168 108 124 143 140 ..
 $ feed  : Factor w/ 6 levels "casein","horsebean",..: 2 2
```

A local namespace: `with()`

- When you have a piece of related code operating on a single dataset, use `with()` so you don't have to type its name all the time.

```
with(airquality, {
  mdl <- lm(Solar.R ~ Temp)
  plot(Solar.R ~ Temp)
  abline(mdl)
})
```

- Local variables such as `mdl` will not end up in the global environment
- Note the use of the tilde `~` character to specify a ***formula***.
- You can read the formula `A ~ B` as ***"A as a function of B"***

Convert numeric vector to factor: `cut()`

- Sometimes it is useful to work with a factor instead of a numeric vector.
- For instance, when working with a Body Mass Index (bmi) variable it may be nice to split this into a factor for some analyses.
- The function `cut()` is used for this.

cut() demo

- Suppose you have the following fictitious dataset

```r
## body mass index
bmi <- c(22, 32, 21, 37, 28, 34, 26, 29,
         41, 18, 22, 27, 32, 31, 26)
## year income * 1000 euros
income <- c(23, 14, 20, 13, 47, 15, 38, 29,
            12, 25, 33, 24, 19, 42, 38)
my.data <- data.frame(bmi = bmi, income = income)
```

- You can of course look at income as a function of bmi using a scatter plot:

```r
with(my.data, plot(income ~ bmi))
```

. . . but wouldn't it be nice to look at the bmi categories as defined by the WHO? - use `cut()`

```
my.data$bmi.class <- cut(bmi,
    breaks = c(0, 18.5, 25.0, 30.0, Inf), right = F,
    labels = c("underweight", "normal", "overweight", "obes
    ordered_result = T)
with(my.data, boxplot(income ~ bmi.class))
```

File I/O revisited

Data file structure

Whatever the contents of a file, you always need to address (some of) these questions:

- Are there comment lines at the top?
- Is there a header line with column names?
- What is the column separator?
- Are there quotes around character data?
- How are missing values encoded?
- How are numeric values encoded?
- Are there dates (a special challenge)
- What is the type in each column?
    - character / numeric / factor / date/time

Some `read.table()` arguments

arg	specifies	example
sep	field separator	sep = “:”
header	is there a header	header = F
dec	decimal format	dec = “,”
comment.char	comment line start	comment.char = ""
na.strings	NA value	na.strings = “-”
as.is	load as character	as.is = c(1,4)
stringsAsFactors	load strings as factors	stringsAsFactors = F

The data reading workflow

Always apply this sequence of steps and repeat until you are satisfied with the result:

1. `read.table()` with arguments that seem OK
2. Check the result at least with `str()` and `head()` and verify that the columns have the correct data type.
   - Factors where numeric expected indicate missed “NA” values!
3. Adjust the read.table parameters
4. Rinse and repeat

Writing data to file

- writing a data frame / matrix / vector to file:
  - `write.table(myData, file="file.csv")`
- Standard is a comma-separated file with both column- and row names, unless otherwise specified:
  - `col.names = F`
  - `row.names = F`
  - `sep = ";"`
  - `sep = "\t" # tab-separated`

Saving R objects to file

Use the `save()` function to write R objects to file for later use. This is especially handy with intermediate results in long analysis workflows.

```r
x <- stats::runif(20)
y <- list(a = 1, b = TRUE, c = "oops")
save(x, y, file = "xy.RData")
```

Writing plot to file

- Use `width` and `height` to specify size
- Default unit is pixels
- Use other unit: `units = "mm"`

```r
png("/path/to/your/file.png",
    width = 700, height = 350, units = "mm")
plot(cars)
dev.off() # don't forget this one!
```

Pattern matching

What is pattern matching

Pattern matching is the process of finding, locating, extracting and replacing patterns in character data that usually cannot be literally described.

For instance, it is easy enough to look for the word “Chimpansee” in a vector containing animal species names:

```r
animals = c("Chimpanzee", "Cow", "Camel")
animals == "Chimpanzee"
```

```
[1]  TRUE FALSE FALSE
```

What are you going to do if there are multiple variants of the word you are looking for? This?

```r
animals = c("Chimpanzee", "Chimp", "chimpanzee", "Pan trogl
animals == "Chimpanzee" | animals == "Chimp" | animals == "
```

```
[1]  TRUE  TRUE  TRUE FALSE
```

The solution here is not using literals, but describe ***patterns***.

Look at the above example. How would you describe a pattern that would correctly identify all Chimpanzee occurrences?

Is you pattern something like this?

A letter C in upper-or lower case followed by 'himp' followed by nothing or 'anzee'

In programming we use ***regular expressions*** to describe such patterns:

`[Cc]himp(anzee)?`

```r
grepl("[Cc]himp(anzee)?", animals)
```

```
[1]  TRUE  TRUE  TRUE FALSE
```

Functions using regex

- **finding** Does an element contain a pattern (TRUE/FALSE)?
  `grepl(pattern, string)`
- **locating** Which elements contain a pattern (INDEX)?
  `grep(pattern, string)`
- **locating** Where is the pattern is found in the string?
  `regexpr(pattern, string)`
- **extracting** Get the content of matching elements
  `grep(pattern, string, value = TRUE)`
- **replacing** Replace the first occurrence of the pattern
  `sub(pattern, replacement, string)`
- **replacing** Replace all occurrences of the pattern
  `gsub(pattern, replacement, string)`