Basic R and data wrangling

Seija Sirkiä

2016-10-31

Preface

This is one of the many versions of a basic R course material I have prepared over the years. It is intended for an audience with some programming background but no R experience. The context covers basic syntax, the most common data sctructures and types and how to read in csv and csv-like data. The subject of modifying, cleaning, reformatting etc. data, generally called data wrangling, as well as data visualisation, is considered using the collection of packages known as "hadley-verse".

Some other sources of information

This document is intended to go hand in hand with a few cheatsheets found on the RStudio web page:

- https://www.rstudio.com/wp-content/uploads/2016/10/r-cheat-sheet-3.pdf
- https://www.rstudio.com/wp-content/uploads/2016/02/advancedR.pdf
- https://www.rstudio.com/wp-content/uploads/2015/02/data-wrangling-cheatsheet.pdf
- https://www.rstudio.com/wp-content/uploads/2015/12/ggplot2-cheatsheet-2.0.pdf

Why R?

Well. You tell me!

About RStudio

Does way more things than I even know how to use. But go ahead and find these buttons/texts/things in there somewhere: Environment. Console. Project. Files. (new) R script. Help. Import Dataset. *Another* Help. Run.

Simple calculations and assignment

```r
3*5+4.7
```

```
## [1] 19.7
```

```r
radius <- 10
area <- pi*radius^2
radius <- 7
```

Pay attention to the Environment tab while running these. Remember there are rules for variable names. Also, there's a hotkey for the arrow!

Vectors

- Vectors come in 3 flavors: numeric, character and logical
- Vectors are created using the function `c()`
  - combines elements and/or existing vectors of the same type
- Vectors are also created by the operator `:` (only makes sense with integer numbers) and of course as a result of functions and calculations
- All kinds of vectors can contain missing values, denoted by the symbol `NA`

Vectorized operation and the recycling rule

As a rule of thumb, everything about vectors, such as calculations, comparisons etc. happens element by element (unless it specifically is something about the whole vector, such as sum). The *recycling rule* applies:

```
(1:6) + (0:2)
```

```
## [1] 1 3 5 4 6 8
```

```
(1:5) * c(-1,1)
```

```
## Warning in (1:5) * c(-1, 1): longer object length is no
## shorter object length
```

```
## [1] -1  2 -3  4 -5
```

Selecting vector elements

Selecting elements happens using brackets `[]` and

- by integer index (starting from 1)
  - multiple indices need to be given as a vector
  - negative indices mean "everything but"
- by a logical vector
  - such as a result from a comparison
- by a name, if it has them
  - again, multiple names as a character vector

All of these can also be used in *replacing* elements! Just assign something new to the selection.

Factors

An ‘R specific’ data structure that looks like a character vector, but is in fact an integer vector with values 1 to something, with special labels corresponding to the values:

```
group <- c("Trt","Ctrl","Ctrl","Trt")
summary(group)
```

```
##     Length     Class      Mode
##          4 character character
```

```
group <- factor(group)
summary(group)
```

```
## Ctrl  Trt
##    2    2
```

More about factors

```r
levels(group)
```

```
## [1] "Ctrl" "Trt"
```

```r
nlevels(group)
```

```
## [1] 2
```

```r
as.numeric(group)
```

```
## [1] 2 1 1 2
```

```r
group==levels(group)[1]
```

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

Be careful when making comparisons, or changing the type, or combining two factors!

R function documentation

Every function has a documentation page that you can access e.g. with the command `?functionname`. It will tell you all the possible arguments to that function and their default values, what (if anything) it returns, mention some related functions and give you examples of usage. Tab completion in RStudio draws its info from that documentation.

Note that when calling R functions you can

- leave the arguments unnamed if you are giving them in the assumed order, or
- give them in some other order if you name them
- or any unambiguous combination of those

Data frames

Possibly the most important data structure in R. Related to the statistics concept of data matrix

- Technically, it is a list of vectors (of any type) and/or factors of equal length
- Because of the equal lengths, it also looks and behaves like a table
  - columns are *variables*
  - rows are *cases* or *subjects*

There is a rather large collection of example data frames that you can access with the call `data()` (for a list of available data sets) and `data(name)` (to bring it in to use).

Reading data from a formatted text file

- A common way to bring data in to R is via a csv file.
- The workhorse function is `read.table()` and besides the csv standard format it can read any text data as long as that data in the file “looks like” a data frame.
    - You just need to tell it how exactly it is formatted
- In RStudio the Import Dataset wizard will automate a lot of the work

About packages

By default, an R installation includes only a handful of recommended packages. More can be installed from CRAN (and some other places too) using the function `install.packages()` or using the Packages tab on RStudio.
Except for a very few base ones, packages also need to be *attached* in order to be used. This happens using the command `library(pkgname)` or again using the Packages tab on RStudio. This needs to be done in every new R session (unless you do tricks).

Tidy data

- You want to have single variables in single columns, and a single observation on a single row. You really do.
- E.g. you do *not* want to have measurements made at different time points (or different dosage, etc) in several columns, and just one row per subject. You want to have a single column for the outcome, and different time points on different rows.
- In other words, **you do not want to see variable values in column names**! Only in the column itself.

Luckily, with tidyr you can switch easily between formats.

Adding variables to a data frame

- Base R way: just assign to a new name within the frame
- Hadley-verse way: mutate (and transmute)

```r
dfname$newvar <- dfname$oldvar1 + dfname$oldvar2
dfname <- mutate(dfname, newvar = oldvar1 + oldvar2,
                         another = newvar^2)
```

Non-trivial ways and reasons to create new variables

- Cutting (or binning) continuous variables in to factors
  - `cut` from base R, `cut_interval`, `cut_number` and `cut_width` from ggplot2
- Relabeling or combining factor levels
  - replacement like `levels(myfactor) <- c("A","B","A")` is possible
  - `recode` from package car?
- Splitting factor levels
  - `ifelse`?
- Cumulative sums, rankings, etc
  - dplyr's window functions

Subsetting data frames

(Generally, you don't want to remove rows or columns, but create a copy with a subset.)

- Base R way of selecting columns using the list logic works
- Base R way of selecting rows using logical vectors also works
- But base R function `subset()` is more user friendly
- But dplyr's functions `select` and `filter` do the same, better, and more (check the helper functions for `select`!)

Groupwise calculations

- Base R includes the rough idea via `apply`-family of functions (`tapply`, `sapply`, `lapply`), and `aggregate`
- `group_by` and `summarise` in dplyr will again do the same better, and more
  - also works with the window functions and mutate

Combining two data sets

- To simply stick two data frames end to end by column or by row:
  - Base R: `cbind` and `rbind`
  - dplyr: `bind_cols` and `bindrows`
- To join according to a common variable (or several)
  - Base R: `merge`
  - dplyr: `left_join` and all the other versions of join

Plotting in base R

- Simple, though a bit clumsy
- Uses for example the generic function `plot` and formulas

```
plot(Sepal.Length~Sepal.Width,data=iris)
plot(Sepal.Length~Species,data=iris)
```

See also: `hist`, `boxplot`, `barplot`...

Plotting with ggplot2

- Not entirely simple, but very powerful
- Uses a concept of “grammar of graphics” (inspired by Tufte) and layers

```
ggplot(iris,aes(x=Sepal.Width,y=Sepal.Length)) +
  geom_points()
ggplot(iris,aes(x=Species,y=Sepal.Length)) +
  geom_boxplot()
```