Data and statistics with R

Jüri Lillemets

2019-09-04

Contents

Preface

Purposes of the course: quantitative data management, processing and analysis with R, tidying, description, visualization, statistical analysis.

Getting help: ? command, Stackoverflow

> R is a **language** and **environment** for **statistical computing** and **graphics**.

Based on S language orginating from 1970's. Developed during 1990's and became public around 2000.

Language and environment.

- Programming language similar to any other but developed paricularly for data analysis.
- Flexible and extensible environment as opposed to many statistical packages.
- Command line interface.

Statistical computing and graphics.

- Includes many statistical procedures for various fields.
- Constantly extended by the community with novel methods.
- Abundant possibilities for plotting data

Chapter 1

Basic data management in R

1.1 Basic R use

1.1.1 Arithmetics

R is a functional programming language. One of the implications of this is that R evaluates expressions and returns the result instead of assigning values to variables through statements. This functionality of R includes basic arithmetic functions.

```
5 + 2 # Addition
5 - 2 # Subtraction
5 * 2 # Multiplication
5 / 2 # Division
```

Some less common arithmetic operators can also be used.

```
5 %% 2 # Return only the remainder of division
```

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

```
5 %/% 2 # Return the result of division without remainder
```

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

Some more advanced mathematical expressions can also be evaluated using R. Most of these require input to be explicitly passed to functions instead of using operators as in the previous example.

```
5 ^ 2 # 5 to the power of 2
```

```
## [1] 25
```

```
sqrt(25) # Square root of 25
```

```
## [1] 5
```

```
exp(1) # Exponent
```

```
## [1] 2.718282
```

```
log(5) # Natural logarithm
```

```
## [1] 1.609438
```

Finally, the order of operations can be set with parenthesis.

```
5 - 2 * 3
```

```
## [1] -1
```

```
5 - (2 * 3)
```

```
## [1] -1
```

Thus, for users comfortable with typing, R can also function as a very handy and advanced calculator.

1.1.2 Relational operators

For the comparison of two objects, relational operators are used in R.

```
5 > 2 # 5 is greater than 2
5 < 2 # 5 is less than 2
2 >= 2 # 2 is equal to or greater than 2
2 >= 5 # 5 is equal to or less than 2
5 == 2 # 5 is equal to 2
2 != 2 # 2 is not equal to 2
```

If the sides of the operator contain uneven number of elements, these will be recycled.

```
5 > c(2, 5) # 5 is greater than 2, 5 is greater than 5
```

Naturally, all of these operators also work on character objects.

```
"String" == "String"
```

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

```
"String" != "String"
```

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

A very useful operator `%in%` can also be considered as a relational operator, evaluating whether or not an identical element is included in another object.

```
5 %in% c(2, 5) # 5 is an element of a vector containing 2 and
```

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

```
"String" %in% c(2, 5)
```

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

1.1.3 Logical operators

Boolean logic is useful to compare objects of type logical (i.e. `TRUE` and `FALSE`)

```
TRUE | FALSE # True or false is true
```

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

```
TRUE & FALSE # True and false are both true
```

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

```
!TRUE # The opposite of true
```

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

Of course, these operators are not very useful for comparison of logical objects explicitly but are helpful together with expressions.

```
5 == 2 | 5 > 2 # 5 equals 2 or 5 is larger than 2
```

1.1.4 Assignment

Data in R is stored in objects. In most cases, values are assigned to objects by assignment operators. In a lot of programming languages, `=` is used for assignment. This works in R, too (which is why `==` must not be confused with `=`). However, for historical reasons it is customary to use `<-` for assignment in R. Object names can contain numbers, letters and punctuation marks but never start with a number.

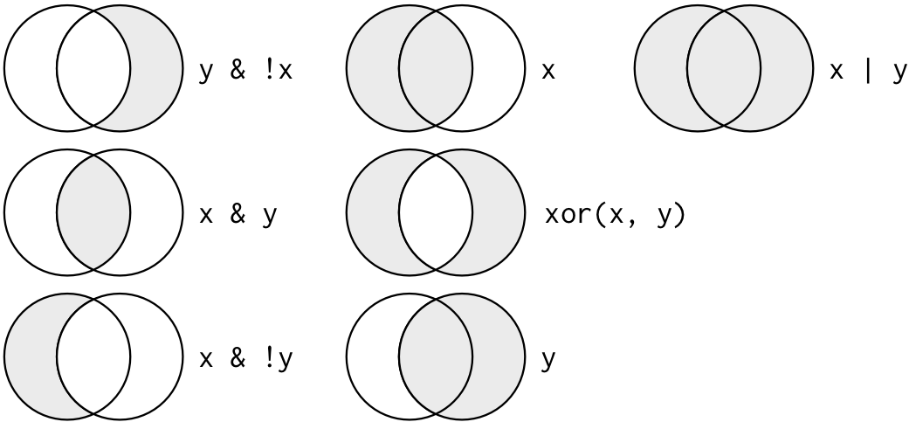

Figure 1.1: https://r4ds.had.co.nz/transform.html#introduction-2

```
(a <- 5)
```

```
## [1] 5
```

```
(a = 5)
```

```
## [1] 5
```

When an assignment is passed to R in parenthesis, it will also be printed in addition to the assignment.

Note that the assignment operator must always be enclosed with spaces, otherwise R might evaluate the assignment as a comparison if you accidentally type a space.

```
a <- 5 # Assign 5 to a
a< -5 # a is greater than -5
```

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

Another way to assign values is to use the `assign()` function but this usually better to be avoided.

1.1.5 Set operations

Although rarely necessary, operations on sets can be helpful to return specific elements from vectors.

```
a <- c(2, 2, 5, "String")
b <- c(1:3)
union(a, b) # All elements of a and b
```

```
## [1] "2"      "5"      "String" "1"      "3"
```

```
intersect(a, b) # Elements in both a and b
```

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

```
setdiff(a, b) # Elements in a but not in b
```

```
## [1] "5"      "String"
```

```
setdiff(a, b) # a and b contain the same elements
```

```
## [1] "5"      "String"
```

While `union()` and `setdiff()` may seem to be the same as `c()` and `identical()` respectively, the set functions presented here behave differenly because they ignore duplications and ordering of elements.

1.2 Data structures

1.2.1 Levels of measurement

Traditional (S. S. Stevens)

- Nominal
- Ordinal
- Interval
- Ratio

Conventional scales

- Categorical/qualitative
  - Binary/dichotomous/boolean/logical
  - Nominal
  - Ordinal/ranked
- Numeric/quantitative
  - Continuous
  - Discrete
    - Interval
    - Ratio

In R

- Logical
- Character
- (Factor)
- Integer
- Double

Missing values in R

- NA - Not available/applicable
- NaN - Not a number
- Inf - positive infinite
- -Inf - Negative infinite

1.2.2 Types of objects in R

Dimensions	Homogenous	Heterogenous
1	Vector, c()	List, list()
2	Matrix, matrix()	Data frame, data.frame()
>2	Array, array()	

To determine type and class of an object: `typeof()`, `class()`, `is.*...()*`

To treat object as if it was of a certain type: `as.*...()*`.

1.2.3 Subsetting

Vectors

```
a <- letters[1:12]
a[2]
```

```
## [1] "b"
```

```
a[-2]
```

```
##  [1] "a" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

```
a[c(1, 3, 5)]
```

```
## [1] "a" "c" "e"
```

Lists

```
a <- list(Letters = letters[1:12], Numbers = 1:10)
a$Letters
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

```
a[1]
```

```
## $Letters
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

```
a[[1]]
```

```
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

```
a[[1]][1]
```

```
## [1] "a"
```

Matrices

```
(a <- matrix(1:9, 3, 3))
```

```
##      [,1] [,2] [,3]
## [1,]    1    4    7
## [2,]    2    5    8
## [3,]    3    6    9
```

```
a[1]
```

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

```
a[1, ]
```

```
## [1] 1 4 7
```

```
a[, 1]
```

```
## [1] 1 2 3
```

Data.frames

```
mtcars$mpg
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

```
mtcars[, 1]
```

```
##  [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2
## [15] 10.4 10.4 14.7 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4
## [29] 15.8 19.7 15.0 21.4
```

```
mtcars[1:10, ]
```

```
##                    mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Duster 360        14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Merc 240D         24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Merc 230          22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 280          19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
```

Using logical operators.

```
mtcars$mpg > 20
mtcars[mtcars$mpg > 20, ]
```

Using indexes.

```
order(mtcars$mpg)
```

```
##  [1] 15 16 24  7 17 31 14 23 22 29 12 13 11  6  5 10 25 30  1  2  4 32 21
## [24]  3  9  8 27 26 19 28 18 20
```

```
mtcars[order(mtcars$mpg), ]
```

```
##                      mpg cyl  disp  hp drat    wt  qsec vs am gear carb
## Cadillac Fleetwood  10.4   8 472.0 205 2.93 5.250 17.98  0  0    3    4
## Lincoln Continental 10.4   8 460.0 215 3.00 5.424 17.82  0  0    3    4
## Camaro Z28          13.3   8 350.0 245 3.73 3.840 15.41  0  0    3    4
## Duster 360          14.3   8 360.0 245 3.21 3.570 15.84  0  0    3    4
## Chrysler Imperial   14.7   8 440.0 230 3.23 5.345 17.42  0  0    3    4
## Maserati Bora       15.0   8 301.0 335 3.54 3.570 14.60  0  1    5    8
## Merc 450SLC         15.2   8 275.8 180 3.07 3.780 18.00  0  0    3    3
## AMC Javelin         15.2   8 304.0 150 3.15 3.435 17.30  0  0    3    2
## Dodge Challenger    15.5   8 318.0 150 2.76 3.520 16.87  0  0    3    2
## Ford Pantera L      15.8   8 351.0 264 4.22 3.170 14.50  0  1    5    4
## Merc 450SE          16.4   8 275.8 180 3.07 4.070 17.40  0  0    3    3
## Merc 450SL          17.3   8 275.8 180 3.07 3.730 17.60  0  0    3    3
## Merc 280C           17.8   6 167.6 123 3.92 3.440 18.90  1  0    4    4
## Valiant             18.1   6 225.0 105 2.76 3.460 20.22  1  0    3    1
## Hornet Sportabout   18.7   8 360.0 175 3.15 3.440 17.02  0  0    3    2
## Merc 280            19.2   6 167.6 123 3.92 3.440 18.30  1  0    4    4
## Pontiac Firebird    19.2   8 400.0 175 3.08 3.845 17.05  0  0    3    2
## Ferrari Dino        19.7   6 145.0 175 3.62 2.770 15.50  0  1    5    6
## Mazda RX4           21.0   6 160.0 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag       21.0   6 160.0 110 3.90 2.875 17.02  0  1    4    4
## Hornet 4 Drive      21.4   6 258.0 110 3.08 3.215 19.44  1  0    3    1
## Volvo 142E          21.4   4 121.0 109 4.11 2.780 18.60  1  1    4    2
## Toyota Corona       21.5   4 120.1  97 3.70 2.465 20.01  1  0    3    1
## Datsun 710          22.8   4 108.0  93 3.85 2.320 18.61  1  1    4    1
## Merc 230            22.8   4 140.8  95 3.92 3.150 22.90  1  0    4    2
## Merc 240D           24.4   4 146.7  62 3.69 3.190 20.00  1  0    4    2
## Porsche 914-2       26.0   4 120.3  91 4.43 2.140 16.70  0  1    5    2
## Fiat X1-9           27.3   4  79.0  66 4.08 1.935 18.90  1  1    4    1
## Honda Civic         30.4   4  75.7  52 4.93 1.615 18.52  1  1    4    2
## Lotus Europa        30.4   4  95.1 113 3.77 1.513 16.90  1  1    5    2
## Fiat 128            32.4   4  78.7  66 4.08 2.200 19.47  1  1    4    1
## Toyota Corolla      33.9   4  71.1  65 4.22 1.835 19.90  1  1    4    1
```

`order()` returns indexes while `sort()` returns the elements.

1.3 Workspace management

To set working directory for the session: `setwd()`. On Windows filesystem, \ needs to be escaped (\\) or replaced with `/`. When working directory is set, all file paths must be relative to the specified directory. To return current working directory: `getwd()`.

To list all objects on workspace: `ls()`. To remove an object: `rm()`; and to remove all objects: `rm(list = ls())`.

1.4 Data sources and managing data

Natively R supports only plain text (e.g. `.csv`) and its native (`.Rdata` and `.Rds`) data formats. Most widely used plain text data format, the Comma-Separated Values (`.csv`) can be loaded by a dedicated function `read.csv()` by providing a location on disk or a url. For an alternative `.csv` where values are separated by semicolons is the `read.csv2()` function. For other plain text formats, `read.table()` allows to specify various attributes. When values are separated by tabs use `sep = \t`.

```
mtCars <- read.csv('some_file.csv`, stringsAsFactors = F)
```

When some data is used exclusively in R, the native R data formats should be used as these allow more efficient data compression. All objects currently on workspace can be saved with `save.image()` function and loaded afterwards using `load()`. In this case, `.Rdata` file should be used.

```
save.image('some_data.Rdata')
load('some_data.Rdata')
```

Sometimes it is necessary to only save a single object on workspace. Then the extension should be `.Rds` and the corresponding commands are `saveRDS` and `readRDS`.

```
saveRDS(a, 'some_data.Rds')
readRDS('some_data.Rds')
```

All common data formats native to other software can also be loaded into R, but this requires relevant libraries to be loaded. The `foreign` package contains functions to load data of other statistical packages, e.g. SAS, SPSS and Stata. For Excel formats, `readxl::read_excel` loads `.xlx` as well as `.xlsx` files while the `openxlsx` library provides functions to meticulously edit and save Excel workbooks.

1.4.1 Understanding a dataset

Once a dataset is loaded into R, it's a good idea to get an understanding of it. While an entire object can be viewed using `View()`, this is not feasible for anything but small tables. Instead, `structure()` (`str()`) displays an overview of all columns in a data frame, `names()` lists the names of all columns and `summary()` gives some statistics on the values of each column.

```
str(mtcars)
```

```
## 'data.frame':    32 obs. of  11 variables:
##  $ mpg : num  21 21 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 ...
##  $ cyl : num  6 6 4 6 8 6 8 4 4 6 ...
##  $ disp: num  160 160 108 258 360 ...
##  $ hp  : num  110 110 93 110 175 105 245 62 95 123 ...
##  $ drat: num  3.9 3.9 3.85 3.08 3.15 2.76 3.21 3.69 3.92 3.92 ...
##  $ wt  : num  2.62 2.88 2.32 3.21 3.44 ...
##  $ qsec: num  16.5 17 18.6 19.4 17 ...
##  $ vs  : num  0 0 1 1 0 1 0 1 1 1 ...
##  $ am  : num  1 1 1 0 0 0 0 0 0 0 ...
##  $ gear: num  4 4 4 3 3 3 3 4 4 4 ...
##  $ carb: num  4 4 1 1 2 1 4 2 2 4 ...
```

```
summary(mtcars)
```

```
##       mpg             cyl             disp             hp       
##  Min.   :10.40   Min.   :4.000   Min.   : 71.1   Min.   : 52.0  
##  1st Qu.:15.43   1st Qu.:4.000   1st Qu.:120.8   1st Qu.: 96.5  
##  Median :19.20   Median :6.000   Median :196.3   Median :123.0  
##  Mean   :20.09   Mean   :6.188   Mean   :230.7   Mean   :146.7  
##  3rd Qu.:22.80   3rd Qu.:8.000   3rd Qu.:326.0   3rd Qu.:180.0  
##  Max.   :33.90   Max.   :8.000   Max.   :472.0   Max.   :335.0  
##       drat             wt             qsec             vs        
##  Min.   :2.760   Min.   :1.513   Min.   :14.50   Min.   :0.0000  
##  1st Qu.:3.080   1st Qu.:2.581   1st Qu.:16.89   1st Qu.:0.0000  
##  Median :3.695   Median :3.325   Median :17.71   Median :0.0000  
##  Mean   :3.597   Mean   :3.217   Mean   :17.85   Mean   :0.4375  
##  3rd Qu.:3.920   3rd Qu.:3.610   3rd Qu.:18.90   3rd Qu.:1.0000  
##  Max.   :4.930   Max.   :5.424   Max.   :22.90   Max.   :1.0000  
##        am              gear            carb      
##  Min.   :0.0000   Min.   :3.000   Min.   :1.000  
##  1st Qu.:0.0000   1st Qu.:3.000   1st Qu.:2.000  
##  Median :0.0000   Median :4.000   Median :2.000  
##  Mean   :0.4062   Mean   :3.688   Mean   :2.812  
##  3rd Qu.:1.0000   3rd Qu.:4.000   3rd Qu.:4.000  
##  Max.   :1.0000   Max.   :5.000   Max.   :8.000
```

```
names(mtcars)
```

```
##  [1] "mpg"  "cyl"  "disp" "hp"   "drat" "wt"   "qsec" "vs"   "am"   "gear"
## [11] "carb"
```

```
head(mtcars)
```

```
##                    mpg cyl disp  hp drat    wt  qsec vs am gear carb
## Mazda RX4         21.0   6  160 110 3.90 2.620 16.46  0  1    4    4
## Mazda RX4 Wag     21.0   6  160 110 3.90 2.875 17.02  0  1    4    4
## Datsun 710        22.8   4  108  93 3.85 2.320 18.61  1  1    4    1
## Hornet 4 Drive    21.4   6  258 110 3.08 3.215 19.44  1  0    3    1
## Hornet Sportabout 18.7   8  360 175 3.15 3.440 17.02  0  0    3    2
## Valiant           18.1   6  225 105 2.76 3.460 20.22  1  0    3    1
```

```
tail(mtcars)
```

```
##                 mpg cyl  disp  hp drat    wt qsec vs am gear carb
## Porsche 914-2  26.0   4 120.3  91 4.43 2.140 16.7  0  1    5    2
## Lotus Europa   30.4   4  95.1 113 3.77 1.513 16.9  1  1    5    2
## Ford Pantera L 15.8   8 351.0 264 4.22 3.170 14.5  0  1    5    4
## Ferrari Dino   19.7   6 145.0 175 3.62 2.770 15.5  0  1    5    6
## Maserati Bora  15.0   8 301.0 335 3.54 3.570 14.6  0  1    5    8
## Volvo 142E     21.4   4 121.0 109 4.11 2.780 18.6  1  1    4    2
```

1.5 R style guide

See: adv-r.had.co.nz

1.6 Some R principles

1.6.1 Objects and functions

John Chambers:

> To understand computations in R, two slogans are helpful:
>
> - Everything that exists is an object.
> - Everything that happens is a function call.

1.6.2 Function arguments

Function arguments are read by (1) complete name, (2) partial name, or (3) position.

```
# All of these give the same result
quantile(x = mtcars$mpg, probs = .9, na.rm = F)
quantile(probs = .9, na.rm = F, x = mtcars$mpg)
quantile(pr = .9, nam = F, x = mtcars$mpg)
quantile(mtcars$mpg, .9, F)
```

1.6.3 Vectors need to be explicity defined as such

Any vector needs to be passed inside the `c()` function.

```
a <- 1:12
a[1,2,3,5,8] # This is does not work
a[c(1,2,3,5,8)] # This is does work
```

1.6.4 Characters must be in quotation marks

Othewise R will interpret them as references to objects and attempts to find them.

```
a <- c(a, b, c) # This attempts to add existing objects a, b and c
a <- c('a', 'b', 'c') # This adds character elements
```

1.6.5 Environments

R searches for objects in an environment where an operation is done. When it does not find an object there, it will incrementally search in higher environments.

This means that when you give an object a name that already exists in base R and then refer to it, R will use the object in workspace.

```
sum(1, 10)
```

```
## [1] 11
```

```
sum <- function(...) Reduce(`-`, ...)
sum(1, 10)
```

```
## [1] 9
```

It also means that R first look for objects inside function calls and when not found, look at workspace.

```
a <- "Cow"
say <- function() print(a)
say()
```

```
## [1] "Cow"
```

```
say <- function() {a <- "Sheep"; print(a)}
say()
```

```
## [1] "Sheep"
```

1.6.6 Recycling

```
(a <- 1:2)
```

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

```
(b <- 1:3)
```

```
## [1] 1 2 3
```

```
a + b
```

```
## Warning in a + b: longer object length is not a multiple of shorter object
## length

## [1] 2 4 4
```

1.6.7 Lazy evaluation

```r
someFun <- function(x, y) print(x)
someFun(x = "Hello world!")
```

```
## [1] "Hello world!"
```