Chapter 3

Daehyeog Lee

2023-08-30

Table of contents

Chapter 3. The R environment

3.1. What will this chapter tell me?

- Exploring how **R** works
- Key windows in **R**

3.2. Before you start

The **CRAN** (Comprehensive R Archive Network) is central to using **R**.

R can be expanded by downloading **packages** that add specific functionality to the program.

3.3. Getting started

Three windows of **R**: **Console**, **Editor**, **Graphics** ('**Quartz**' in Mac)

3.4. Using R

3.4.1. Commands, objects and functions

`Object <- function`

We call this **"Object is created from function"**.

Object is anything created in **R** (single values, collections of information, etc.).

Function is a method in **R** to create the object.

A *concatenate function*, or `c()`, groups things together.

```
metallica <- c("Lars", "James", "Jason", "Kirk")
metallica
```

```
[1] "Lars"  "James" "Jason" "Kirk"
```

```
metallica <- metallica[metallica != "Jason"]
metallica
```

```
[1] "Lars"  "James" "Kirk"
```

```
metallica <- c(metallica, "Rob")
metallica
```

```
[1] "Lars"  "James" "Kirk"  "Rob"
```

```
jeonlab <- c("prof_Jeon", "HD", "TH", "MH", "JT")
jeonlab
```

```
[1] "prof_Jeon" "HD"        "TH"        "MH"        "JT"
```

```
jeonlab <- c(jeonlab, "DH")
jeonlab
```

```
[1] "prof_Jeon" "HD"        "TH"        "MH"        "JT"        "DH"
```

```
jeonlab <- jeonlab[jeonlab != "DH"]
jeonlab
```

```
[1] "prof_Jeon" "HD"        "TH"        "MH"        "JT"
```

```
jeonlab <- c("DH", jeonlab)
jeonlab
```

```
[1] "DH"        "prof_Jeon" "HD"        "TH"        "MH"        "JT"
```

We can remove the element by using `!=`.

3.4.6. Getting Help

We can get help by executing the `help(function)` or `?function` command.

3.5. Getting data into R

3.5.1. Creating variables

```r
metallicaNames <- c("Lars", "James", "Kirk", "Rob")
metallicaAges <- c(47, 47, 48, 46)
```

String variables (`metallicaNames`) consist of data that are text. They should always be placed in quotes.

Numeric variables (`metallicaAges`) contain data that are numbers. They are never placed in quotes.

3.5.2. Creating dataframes

We can combine variables into a single object by creating a **dataframe**.

Dataframe are created by `data.frame()` function.

The `names()` function lists the variables in the dataframe.

```r
metallica <- data.frame(Name = metallicaNames, Age = metallicaAges)
metallica
```

```
   Name Age
1  Lars  47
2 James  47
3  Kirk  48
4   Rob  46
```

```r
metallica$Age
```

```
[1] 47 47 48 46
```

```r
metallica$Name
```

```
[1] "Lars"  "James" "Kirk"  "Rob"
```

```r
metallica$childAge <- c(12, 12, 4, 6)
metallica
```

```
   Name Age childAge
1  Lars  47       12
2 James  47       12
3  Kirk  48        4
4   Rob  46        6
```

```
names(metallica)
```

```
[1] "Name"     "Age"      "childAge"
```

In this dataframe, `metallica` contains two variables (Name and Age).

The `list()` function creates a list of separate objects.

We can also use `cbind()` function instead of the `data.frame()` function to combine the data.

```
metallicalist <- list(metallicaNames, metallicaAges)
metallicalist
```

```
[[1]]
[1] "Lars"  "James" "Kirk"  "Rob"

[[2]]
[1] 47 47 48 46
```

```
metallica
```

```
   Name Age childAge
1  Lars  47       12
2 James  47       12
3  Kirk  48        4
4   Rob  46        6
```

```
metallicacbind <- cbind(metallicaNames, metallicaAges)
metallicacbind
```

```
     metallicaNames metallicaAges
[1,] "Lars"         "47"
[2,] "James"        "47"
[3,] "Kirk"         "48"
[4,] "Rob"          "46"
```

The interesting feature of `cbind()` is that the numbers from the output are in quotes.

`cbind()` is most useful for combining variables of the same type, while `data.frame()` is useful for storing variables of different types together.

3.5.3. Calculating new variables from existing ones

We might want to create new variable from the existing ones.

```
metallica$fatherhoodAge <- metallica$Age - metallica$childAge
metallica
```

```
   Name Age childAge fatherhoodAge
1  Lars  47       12            35
2 James  47       12            35
3  Kirk  48        4            44
4   Rob  46        6            40
```

We created a new variable (**fatherhoodAge**), which is a difference of age between each member and their child.

3.5.4. Organizing your data

3.5.4.1. Creating a string variable

Let's create a variable called **name** as follows:

```
name <- c("Ben", "Martin", "Andy", "Paul", "Graham", "Carina",
          "Karina", "Doug", "Mark", "Zoe")
```

3.5.4.2. Creating a date variable

We can convert dates written as text into date objects using the `as.Date()` function.

`as.Date()` function takes strings of text and converts them into dates.

```
husband <- c("1973-06-21", "1970-07-16", "1949-10-08", "1969-05-24")
wife <- c("1984-11-12", "1973-08-02", "1948-11-11", "1983-07-23")
#agegap <- husband - wife # This gives an error message!
```

By using `as.Date()` function, we can subtract dates from one another.

```
husband <- as.Date(c("1973-06-21", "1970-07-16", "1949-10-08", "1969-05-24"))
wife <- as.Date(c("1984-11-12", "1973-08-02", "1948-11-11", "1983-07-23"))
agegap <- husband - wife
agegap
```

```
Time differences in days
[1] -4162 -1113   331 -5173
```

Below, we created a variable called **birth_date** containing the dates of birth.

```
birth_date <- as.Date(c("1977-07-03", "1969-05-24", "1973-06-21", "1970-07-16",
                        "1949-10-10", "1983-11-05", "1987-10-08", "1989-09-16",
                        "1973-05-20", "1984-11-12"))
```

3.5.4.3. Creating coding variables/factors

A **coding variable** (= grouping variable = factor) is a variable that uses numbers to represent different groups of data. It is a *numeric variable*, but these numbers represent names.

Let's create a new variable **job**.

```
job <- c(1,1,1,1,1,2,2,2,2,2)
job
```

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

We can make the above simpler by using the `rep()` function.

rep(1, 5) will repeat the number 1 five times.

```
job <- c(rep(1, 5), rep(2, 5))
job
```

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

This gives us the same result.

Now, we can use the `factor()` function to turn this variable **job** into a factor.

```
job <- factor(job, levels = c(1:2), labels = c("Lecturer", "Student"))
job
```

```
 [1] Lecturer Lecturer Lecturer Lecturer Lecturer Student  Student  Student
 [9] Student  Student
Levels: Lecturer Student
```

Having converted **job** to a factor, **R** will treat it as a nominal variable.

A final way to generate factors is to use the gl() function.

```
job <- gl(2, 5, labels = c("Lecturer", "Student"))
job
```

```
 [1] Lecturer Lecturer Lecturer Lecturer Lecturer Student  Student  Student
 [9] Student  Student
Levels: Lecturer Student
```

The end result is a fully-fledged coding variable (or factor)

We can also see the factor levels and their order by using the `levels()` function.

```
levels(job)
```

```
[1] "Lecturer" "Student"
```

3.5.4.4. Creating a numeric variable

Let's add some other variables like below.

```
friends <- c(5, 2, 0, 4, 1, 10, 12, 15, 12, 17)
alcohol <- c(10, 15, 20, 5, 30, 25, 20, 16, 17, 18)
income <- c(20000, 40000, 3500, 22000, 50000, 50000, 100, 3000, 10000, 10)
neurotic <- c(10, 17, 14, 13, 21, 7, 13, 9, 14, 13)
```

Then, we can bind these together in a dataframe by using the `data.frame()` function.

```
lecturerData <- data.frame(name, birth_date, job, friends,
                           alcohol, income, neurotic)
lecturerData
```

```
     name birth_date      job friends alcohol income neurotic
1     Ben 1977-07-03 Lecturer       5      10  20000       10
2  Martin 1969-05-24 Lecturer       2      15  40000       17
3    Andy 1973-06-21 Lecturer       0      20   3500       14
4    Paul 1970-07-16 Lecturer       4       5  22000       13
5  Graham 1949-10-10 Lecturer       1      30  50000       21
6  Carina 1983-11-05  Student      10      25  50000        7
7  Karina 1987-10-08  Student      12      20    100       13
8    Doug 1989-09-16  Student      15      16   3000        9
9    Mark 1973-05-20  Student      12      17  10000       14
10    Zoe 1984-11-12  Student      17      18     10       13
```

3.5.5 Missing values

We can fill out the missing data point by using **NA**.

Or, we can use the command *na.rm=TRUE* to tell **R** to ignore missing values before computing the mean.

```
mean(metallica$childAge, na.rm = TRUE)
```

```
[1] 8.5
```

3.6. Entering data with R Commander

We can create and modify the coding variables with R Commander.

R Commander can be executed by `library(Rcmdr)`.

3.7. Using other software to edit and enter data

3.7.1. Importing data

We can import *.csv* files and *.txt* files by using the `read.csv()` and `read.txt()`, respectively.

Or, we can select the file from the system's dialog box by executing the `file.choose()` function.

3.8. Saving data

We can export data from R by using `write.table()` (for *.txt* file) or `write.csv()` (for *.csv* file) command.

3.9. Manipulating data

3.9.1. Selecting parts of a dataframe

We can separate the columns from the original dataset by executing this command:

```
lecturerPersonality <- lecturerData[, c("friends", "alcohol", "neurotic")]
lecturerPersonality
```

```
   friends alcohol neurotic
1        5      10       10
2        2      15       17
3        0      20       14
4        4       5       13
5        1      30       21
6       10      25        7
7       12      20       13
8       15      16        9
9       12      17       14
10      17      18       13
```

Similarly, we can separate the rows from the original dataset.

Suppose we only want to see the data of the lecturers. We can do this by the following command:

```
lecturerOnly <- lecturerData[job == "Lecturer",]
lecturerOnly
```

```
    name birth_date      job friends alcohol income neurotic
1    Ben 1977-07-03 Lecturer       5      10  20000       10
2 Martin 1969-05-24 Lecturer       2      15  40000       17
3   Andy 1973-06-21 Lecturer       0      20   3500       14
4   Paul 1970-07-16 Lecturer       4       5  22000       13
5 Graham 1949-10-10 Lecturer       1      30  50000       21
```

Moreover, we can set the specific conditions from the dataset. Imagine that we wanted to select the personality variables but only for people who drink more than 10 units of alcohol.

We can do this by executing:

```
alcoholPersonality <- lecturerData[alcohol > 10, c("friends", "alcohol", "neurotic")]
alcoholPersonality
```

```
   friends alcohol neurotic
2        2      15       17
3        0      20       14
5        1      30       21
6       10      25        7
7       12      20       13
8       15      16        9
9       12      17       14
10      17      18       13
```

3.9.2. Selecting data with the `subset()` function

We can do the same thing by using the `subset()` function.

```
lecturerOnly <- subset(lecturerData, job == "Lecturer")
alcoholPersonality <- subset(lecturerData, alcohol > 10,
                             select = c("friends", "alcohol", "neurotic"))
lecturerOnly
```

```
    name birth_date      job friends alcohol income neurotic
1    Ben 1977-07-03 Lecturer       5      10  20000       10
2 Martin 1969-05-24 Lecturer       2      15  40000       17
3   Andy 1973-06-21 Lecturer       0      20   3500       14
4   Paul 1970-07-16 Lecturer       4       5  22000       13
5 Graham 1949-10-10 Lecturer       1      30  50000       21
```

```
alcoholPersonality
```

```
   friends alcohol neurotic
2        2      15       17
3        0      20       14
5        1      30       21
```

```
6        10       25         7
7        12       20        13
8        15       16         9
9        12       17        14
10       17       18        13
```

The result would be same as 3.9.1.

3.9.3. Dataframes and matrices

Dataframes, what we have handled so far, is one way to store data.

Another way to store data is a **matrix**.

The main difference between a dataframe and a matrix is that a matrix can contain only numeric variables. It cannot contain string variables or dates.

We can convert a dataframe to a matrix using the `as.matrix()` function.

```
alcoholPersonalityMatrix <- as.matrix(alcoholPersonality)
alcoholPersonalityMatrix
```

```
   friends alcohol neurotic
2        2      15       17
3        0      20       14
5        1      30       21
6       10      25        7
7       12      20       13
8       15      16        9
9       12      17       14
10      17      18       13
```

Defining *alcoholPersonality* inside of `as.matrix()` also returns the same outcome.

```
alcoholPersonalityMatrix <- as.matrix(lecturerData[alcohol > 10,
                                                   c("friends", "alcohol", "neurotic")])
alcoholPersonalityMatrix
```

```
   friends alcohol neurotic
2        2      15       17
3        0      20       14
```

```
5         1       30        21
6        10       25         7
7        12       20        13
8        15       16         9
9        12       17        14
10       17       18        13
```

3.9.4. Reshaping data

Data can be contained in either **wide** format or **long** (or **molten**) format.

In **wide** format, each person's data is contained in a single row of the data.

We can set the data into wide format by executing `unstack()` or `cast()` function.

In **long (molten)** format, data on different variables are placed in a single column.

We can set the data into wide format by executing `stack()` or `melt()` function.

Smart Alex's tasks

```
### Task_2 original ###
studentNum <- c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12",
                "13", "14", "15", "16", "17", "18", "19", "20")
gender <- c(rep(1, 10), rep(2, 10))
gender <- factor(gender, levels = c(1:2), labels = c("Male", "Female"))
electricShock <- c(15, 14, 20, 13, 13, NA, NA, NA, NA, NA, 6, 7, 5, 4, 8,
                   NA, NA, NA, NA, NA)
beingNice <- c(NA, NA, NA, NA, NA, 10, 9, 8, 8, 7, NA, NA, NA, NA, NA,
               12, 10, 7, 8, 13)
scores <-data.frame(studentNum, gender, electricShock, beingNice)
scores
```

```
   studentNum gender electricShock beingNice
1           1   Male            15        NA
2           2   Male            14        NA
3           3   Male            20        NA
4           4   Male            13        NA
5           5   Male            13        NA
6           6   Male            NA        10
7           7   Male            NA         9
```

```
8           8   Male            NA         8
9           9   Male            NA         8
10         10   Male            NA         7
11         11 Female            6         NA
12         12 Female            7         NA
13         13 Female            5         NA
14         14 Female            4         NA
15         15 Female            8         NA
16         16 Female           NA         12
17         17 Female           NA         10
18         18 Female           NA          7
19         19 Female           NA          8
20         20 Female           NA         13
```

```r
### Task_3 ###
subjectNum <- c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11",
                "12", "13", "14", "15", "16", "17", "18", "19", "20", "21",
                "22", "23", "24")
genders <- c(rep(1, 12), rep(2, 12))
genders <- factor(genders, levels = c(1:2), labels = c("Male", "Female"))
PF <- c(69, 76, 70, 76, 72, 65, 82, 71, 71, 75, 52, 34, 70, 74, 64, 43, 51,
        93, 48, 51, 74, 73, 41, 84)
OF <- c(33, 26, 10, 51, 34, 28, 27, 9, 33, 11, 14, 46, 97, 80, 88, 100, 100,
        58, 95, 83, 97, 89, 69, 82)
bullets <- data.frame(subjectNum, genders, PF, OF)
bullets
```

```
   subjectNum genders PF  OF
1           1    Male 69  33
2           2    Male 76  26
3           3    Male 70  10
4           4    Male 76  51
5           5    Male 72  34
6           6    Male 65  28
7           7    Male 82  27
8           8    Male 71   9
9           9    Male 71  33
10         10    Male 75  11
11         11    Male 52  14
12         12    Male 34  46
13         13  Female 70  97
14         14  Female 74  80
```

```
15         15  Female 64  88
16         16  Female 43 100
17         17  Female 51 100
18         18  Female 93  58
19         19  Female 48  95
20         20  Female 51  83
21         21  Female 74  97
22         22  Female 73  89
23         23  Female 41  69
24         24  Female 84  82
```

```r
### Task_2 revised ###
sbj <- c("1", "2", "3", "4", "5", "6", "7", "8", "9", "10", "11", "12", "13",
         "14", "15", "16", "17", "18", "19", "20")
treated <- c(rep(1, 10), rep(2, 10))
treated <- factor(treated, levels = c(1:2),
                  labels = c("electricShock", "beingNice"))
gend <- c(rep(1, 5), rep(2, 5), rep(1, 5), rep(2, 5))
gend <- factor(gend, levels = c(1:2), labels = c("Male", "Female"))
score <- c(15, 15, 20, 13, 13, 6, 7, 5, 4, 8, 10, 9, 8, 8, 7, 12, 10, 7, 8, 13)
result <- data.frame(sbj, treated, gend, score)
result
```

```
   sbj       treated   gend score
1    1 electricShock   Male    15
2    2 electricShock   Male    15
3    3 electricShock   Male    20
4    4 electricShock   Male    13
5    5 electricShock   Male    13
6    6 electricShock Female     6
7    7 electricShock Female     7
8    8 electricShock Female     5
9    9 electricShock Female     4
10  10 electricShock Female     8
11  11     beingNice   Male    10
12  12     beingNice   Male     9
13  13     beingNice   Male     8
14  14     beingNice   Male     8
15  15     beingNice   Male     7
16  16     beingNice Female    12
17  17     beingNice Female    10
18  18     beingNice Female     7
```

```
19  19     beingNice Female     8
20  20     beingNice Female    13
```