Data type in and how to work with them....

D. Stefanovskiy

RANE

November, 2011

Statistical laws and subsequently statistical programs work with this data only of their type was declared beforehand. Remaining data types can be named differently:

- numerical,
- countable,
- sequenced,
- non-categorical.

Let's name them as numerical to be simple.

Basic commands in R

Suppose we have data on seven employees' height in a small company. That's how you can create a simple vector using this data: In R-project (or R) it is looks:

$$x \leftarrow c(174, 162, 188, 192, 165, 168, 172) \text{ or}$$
$$x = c(174, 162, 188, 192, 165, 168, 172)$$

x is a name of object R,
$\leftarrow$ — the operator of assignment,
$c()$ — function of creation of a vector (from English «concatenate» to collect).

Basic commands in R

Actually, R, and works mainly with objects and functions. An object can have its own structure:

$$str(x)$$

here x is a numeric (num, numerical) vector. In programming languages, there are also scalars, but no scalars in R. Single objects are treated as vectors consisting of a single element.

Basic commands in R

That's how you can check whether the vector is before us:

$$is.vector(x)$$

Simple rules

Generally speaking, R has many functions like is.smth() for a similar verification, and there are also functions like as.smth(), which will be used hereinafter. One can call objects in principle anyhow, but better follow some simple rules:

- Use for the names of Latin letters, digits and a point (object names should not begin with a dot or a digit);
- Remember that R is case-sensitive, X, and x are different names;
- Do not give the names of objects, already occupied by the common functions (such as c()), as well as key words (especially T, F, NA, NaN, Inf).

To create artificial vectors ":"is a very useful operator, and also functions seq() and rep().

To refer to categorical data R has several ways of different grades of "correctness". First, you can create a text (character) vector:

$sex \leftarrow c(\text{“}male\text{“}, \text{“}female\text{“}, \text{“}male\text{“}, \text{“}male\text{“}, \text{“}female\text{“}, \text{“}male\text{“}, \text{“}male\text{“})$

$is.character(sex)$

$is.vector(sex)$

$str(sex)$

Let us assume that sex is the description of sex workers in a small organization. That's how R displays this vector content:

sex

"Smart", or object-oriented commands in R understand something about the object "sex", for example, the command table():

$$sex$$
$$sex[1]$$
$$table(sex)$$

But the command plot(), sorry, cannot do anything good with the vector like this. In it is, in general, correct, because the program does not know anything about the properties of a human sex. In such cases, the user must inform R, that it should be treated as categorical data type.
Do it this way:

$$sex.f \leftarrow factor(sex)$$
$$sex.f$$

And now the command plot () already understands that it should do:

$$plot(sex.f)$$

Because we face a special type of object intended for categorical data - a factor with two levels (levels):

```
is.factor(sex.f)
is.character(sex.f)
str(sex.f)
```

Very many of the functions in R (for example, the same plot()) prefer factors instead of text vectors. Additionally some are able to convert text into vector factors, and some are not, therefore, one has to be careful. There are several other important properties of the factors that one needs to know in advance. First, a subset of factors is a factor with the same number of levels, even if they are not left in the subset:

$$sex.f[5:6]$$
$$sex.f[6:7]$$

One can get rid of an excess level only applying a special argument or by performing data conversion “to and back”:

$$sex.f[6:7, drop = TRUE]$$

The factors as opposed to text vectors can be easily converted into the numerical values:

```
as.numeric(sex.f)
```

To understand why one have do it becomes clear if we consider here is an example: Suppose besides a height, we also have data on employees weight and we want to draw graph in which at the same time height, weight and gender will be displayed. Here is how to do that:

$$\text{w} \leftarrow c(69, 68, 93, 87, 59, 82, 72)$$

plot(x, w, pch=as.numeric(sex.f), col=as.numeric(sex.f))

legend("topleft", pch=1:2, col=1:2, legend=levels(sex.f))

Here, of course, some explanations are needed: pch and col are parameters which were assigned respectively to determine the icon types and their colors on the graph. Thus, depending on which sex this point belongs to, it will be displayed as a circle or a triangle, and black or red color respectively. It is required, of course, that all three vectors correspond each other. Yet it must be noted that the display of sex, using icon and color together is excessive, for a “normal” graph one method is enough

Third, the factors can be sorted to convert them into some numerical data similarity. Let's insert a fourth variable: T-shirts size for the same hypothetical eight employees:

$$m \leftarrow c(\text{“}L\text{“}, \text{“}S\text{“}, \text{“}XL\text{“}, \text{“}XXL\text{“}, \text{“}S\text{“}, \text{“}M\text{“}, \text{“}L\text{“})$$
$$m.f \leftarrow factor(m)$$
$$m.f$$

As you can see, the levels are sorted in alphabetical order but we need to “S“ (small) to come first. Furthermore, we must somehow inform R that we have not simple categorical data but orderable categorical data. One should do that as follows:

$$\text{m.o} \leftarrow \mathit{ordered}(\mathit{m.f}, \mathit{levels} = \mathit{c}(\text{“}\mathit{S}\text{“}, \text{“}\mathit{M}\text{“}, \text{“}\mathit{L}\text{“}, \text{“}\mathit{XL}\text{“}, \text{“}\mathit{XXL}\text{“}))$$

$$\text{m.o}$$

Missing data

In addition to the vectors of numbers and text vectors, R supports also logical vectors, as well as special data types that can be very important for statistical calculations. First of all, it's skipped or missing data, which are denoted as NA. Such data are often arising in the real field and laboratory studies, surveys, polls, tests etc. It should be aware that the presence of missing data does not mean that the data are generally of poor quality. On the other hand, statistical programs must somehow work with that data also.

Let us consider the following example: suppose that we have the result of a survey of the same seven employees. They were asked how many hours they averagely sleep but one of the respondents refused to answer, another said “I do not know” and the third one was out of the office at the moment of in the survey. Than was how missing data were aroused:

$$h \leftarrow c(8, 10, NA, NA, 8, NA, 8)$$
$$h$$

This example shows that NA should be entered without quotation marks, and R is not in the least embarrassed that among the numbers some kind of text arrives. Note that the missing data are often as diverse as in our example. However, they are coded in the same way, and this must not be forgotten. Now, about how to work with the obtained vector h. If we just try to calculate the mean value (function mean()), we obtain:

$$mean(h)$$

And it is “ideologically correct” because the function may differently process NA, and by default, it simply indicates that something is wrong with the data. To calculate the average of “not missed” part of the vector one can use one of two ways:

$$mean(h, na.rm = TRUE)$$
$$mean(na.omit(h))$$

Which way is better depends on the situation. One problem more often arrives: how to make a substitution of missing data, for example, replace all NA with the average for the sample. A common solution is like the following:

$$h[is.na(h)] \leftarrow mean(h, na.rm = TRUE)$$
$$h$$

On the left side of the first expression indexing is on the run, it is selection of values in h to replace missing numbers (is.na ()). After the expression execution “old“ values disappear forever.

A matrix is extremely widespread form of data representation presented in a form of table. Concerning matrixes R one must generally know two important things: first, that they may be of different dimensions, and secondly, that there are really no matrixes R.

Let's start with the last. Matrix in R is simply a special type of vector which possesses some additional properties ("attributes"), allowing to interpret it as a set of rows and columns. Suppose we want to create a simple 2×2. To start with let's create it from a numerical vector:

$$m \leftarrow 1:4$$
$$m$$
$$ma \leftarrow matrix(m, ncol = 2, byrow = TRUE)$$
$$ma$$
$$str(ma)$$
$$str(m)$$

This example shows that the structure of the object m and ma are not too different. The only difference is in their screen display. Even more obvious unity between vectors and matrices can be traced if you create a matrix in a different way:

$$mb \leftarrow m$$
$$mb$$
$$attr(mb, \text{“}dim\text{“}) \leftarrow c(2,2)$$
$$mb$$

It looks like a trick. However, everything is simple: we assign a vector mb an attribute dim (from the word dimension) and set the value of this attribute in c(2,2), there are two rows and two columns. A reader is expected to guess why mb matrix is different from ma one (the answer at end of the article). We showed only two ways to create matrices, but in reality there are much more. It is very popular, for example, to make the matrices of vector-columns or rows using commands cbind() and rbind (). If the result must be turned in 90 degrees, use the command t().

The most common are matrices with two dimensions but nobody prevents to make a multi-dimensional matrix: $2 \times 2 \times 2$:

$$m3 \leftarrow 1 : 8$$
$$dim(m3) \leftarrow c(2, 2, 2)$$
$$m3$$

m3 is a three-dimensional matrix. Naturally one is not able to show it as a table, therefore R displays it on the screen as a series of tables. Similarly one can create also four-dimensional matrix (like built-in Titanic data from the previous article). Multi-dimensional matrices in R are called arrays.

Lists

Lists are another very important type of data representation. Their creation, especially at first, probably will not be necessary, but one needs to know their characters. It is needed mainly because very many functions in R return the very lists. At the very beginning of acquaintance let's create a list just for a practice:

$$l \leftarrow list(\text{“}R\text{“}, 1:3, TRUE, NA, list(\text{“}r\text{“}, 4))$$
$$l$$

We see that the list some kind of assortment. A vector and, accordingly, a matrix may consist only of elements of the same type, but a list can consist of anything. In particular, as one can see from the example, a list may include other lists. Now let's talk about indexing or selecting list items. Elements of the vector are selected by means of the square brackets function:

```
l[3]
```

The matrix elements can be chosen the same way, only several arguments are in use (for two-dimensional matrix it is the row number and the column number, in this order):

```
ma[2, 1]
```

But the list items are selected by three different methods. First, you can use square brackets:

```
l[1]
str(l[1])
```

It is very important here that the resulting object will be also list. Secondly double square brackets can be used:

```
l[[1]]
str(l[[1]])
```

In this case, the resulting object will be of the same type as it would be before to the unification to the list (and therefore the first object is a text vector and a fifth is a list). Third, you can use the list names. But t first you must assign them:

$$\text{names(l)} \leftarrow c(\text{“}\textit{first}\text{“}, \text{“}\textit{second}\text{“}, \text{“}\textit{third}\text{“}, \text{“}\textit{fourth}\text{“}, \text{“}\textit{fifth}\text{“})$$
$$\text{str(l\$first)}$$

To select by a name the dollar sign is used, and the resulting object will be the same as using double square brackets. In fact, the names in R can have vector elements and matrix rows and columns:

$$\text{names(w)} \leftarrow c(\text{“}Nic\text{“}, \text{“}John\text{“}, \text{“}Piter\text{“}, \text{“}Alex\text{“}, \text{“}Cat\text{“}, \text{“}Bob\text{“}, \text{“}Gorge\text{“})$$

w

w[“John“]

$$\text{rownames(ma)} \leftarrow c(\text{“}a1\text{“}, \text{“}a2\text{“})$$

$$\text{colnames(ma)} \leftarrow c(\text{“}b1\text{“}, \text{“}b2\text{“})$$

ma

Finally we came to the most important type of data - data tables (data frames). Those very data tables are the most similar to Excel spreadsheets and their analogues, and therefore are in the most often use. It is especially true for R beginners. Data tables are hybrid type of same length vectors representation, a one-dimensional list of vectors of the same length. Thus, each data table is a list of columns and within the same column, all data must be of one type.

We illustrate this by an example concerning vectors created in this article before:

$$\mathrm{d} \leftarrow \mathit{data.frame}(\mathit{weight} = w, \mathit{height} = x, \mathit{size} = m.o, \mathit{sex} = \mathit{sex}.f)$$

d

str(d)

Because the data table is a list, list indexing methods are applicable to it. Furthermore, data tables can be indexed as a two-dimensional matrix. Here are some examples:

```
d$weight
d[[1]]
d[,1]
d[,"weight"]
```

It is often necessary to select a few specific columns. This can be done but in different ways (exclude the column weight):

```
d[,2:4]
d[,-1]
```

The second method (negative indexing) in some cases is practically unreplacable. Indexing is directly connected to one more R data type – logical vectors. For example, how can we select from our table the data related to women only? Here's one way:

```
d$sex=="female"
d[d$sex=="female",]
```

Thus, after the selection “processing”, the data in the table contains only those rows that correspond to “TRUE, so the lines 2 and 5. A more complex selection case is the data tables sorting. To sort a vector it is sufficient to apply the command sort(), but if necessary, for example, to sort our data is first on sex, and then on the height, we have to user a more difficult operation:

```
d[order(d$sex, d$height), ]
```

From all ways data exchange from Excel with R through clipboard may be the most attractive. If to open in OpenOffice Calc a xls-file it is possible to copy any quantity of cells in the buffer, and then for putting them in R you may use next command:

$$\mathrm{t} \leftarrow \mathit{read.table}(\text{“}\mathit{clipboard}\text{“})$$
$$\mathrm{t} \leftarrow \mathit{read.table}(\text{“}\mathit{name\_file.txt}\text{“}, \mathit{sep} = \text{“;“})$$

Vectorized computation

Despite the fact that R is similar to many modern script programming languages, such as Perl and Python, it has many peculiar features. One of the interesting and very useful features of the R this so called vectorized computations. Their use is very simple. Suppose we want to transfer weight from pounds to grams:

```
w*1000
```

For such an operation it is often required to use looping constructs (loops) but here we use one operation only. Of course loops also will work in :

```
for(i in seq_along(w)) {
w[i] ← w[i] * 1000
}
w
```

Example

Now I want to show you the one program for example. This program take data from central bank of Czech Republic and create the matrix of correlation.
I will show it program in RStudio. Now please run Rstudio and create program on the computers yourself. And the last, please be very careful with integer type...
1)1!=1.0
2)1 not equal 1.0
3) not (1 >1.0 may be true)

Děkuji moc!
Přemýšlejte a ptejte se - pak se bude dařit.