Linguistic Data: Quantitative Analysis and Visualisation

Ilya Schurov, Olga Lyashevskaya, George Moroz, Alla Tambovtseva

19 January 2019

More on vectors

Operations on vectors

Let's create two numeric vectors: `v` and `w`:

```
v <- c(2, 7, 1, 0, 9, 5, 8)
w <- c(0, 0, 1, 2, 1, 5, 0)
v
```

```
## [1] 2 7 1 0 9 5 8
```

```
w
```

```
## [1] 0 0 1 2 1 5 0
```

Using vectors in R is convenient because many operations are performed element-wise. Let's look what we will get if subtract 1 from the vector `v`:

```
v - 1
```

```
## [1]  1  6  0 -1  8  4  7
```

1 is subtracted from every element of `v`! So, we don't need any special constructions to tell R that we work with each element of a vector. One more example:

```
v * 2
```

```
## [1]  4 14  2  0 18 10 16
```

And what if we work with two vectors at the same time?

```
v + w
```

```
## [1]  2  7  2  2 10 10  8
```

Addition is done pairwise, so elements of `v` and `w` with the same index are added: 2 and 0, 7 and 7, and so on.

```
v * w
```

```
## [1]  0  0  1  0  9 25  0
```

Of course, if lengths of vectors are different, we will get something wrong:

```
# the 2nd element is doubled
# and the warning says that lengths are different
# c(1, 4, 5) + c(6, 7)
```

Problem 1

Researchers listen to six people of different social status and record their speech. The number of minutes after 2 PM at the moment of starting is saved in the vector `start`, the number of minutes after 2 PM at the moment of finishing is stored in `finish`.

```
# in minutes
start <- c(20, 30, 40, 10, 15, 12)
finish <- c(30, 38, 42, 15, 17, 18)
```

Calculate the duration of each person's speech in seconds.

Solution

```
# duration in seconds
dur_min <- (finish - start) * 60
dur_min
```

```
## [1] 600 480 120 300 120 360
```

Problem 2

Vector `speakers` contains number of speakers of a concrete language in different communities, the vector `total` includes total number of people in these communities. Calculate the percentage of speakers of this language in every community.

```
speakers <- c(25, 32, 10, 43, 50)
total <- c(100, 120, 45, 50, 65)
```

Solution

```
perc <- (speakers/total) * 100
perc
```

```
## [1] 25.00000 26.66667 22.22222 86.00000 76.92308
```

Sequences in R

In R we can create sequences of subsequent integers automatically:

```
# from 1 to 10
1:10
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

Note that both endpoints, 1 and 10, are included. Sequences with negatives are also possible:

```
-10:20
```

```
##  [1] -10  -9  -8  -7  -6  -5  -4  -3  -2  -1   0   1   2   3   4   5   6
## [18]   7   8   9  10  11  12  13  14  15  16  17  18  19  20
```

Sequences of integers can be helpful for choosing several elements from a vector.

```
# recall: how to choose the first value
v[1]
```

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

```
# with sequences: from the 1st to 4th one
v[1:4]
```

```
## [1] 2 7 1 0
```

```
# from 4th to 6th one
v[4:6]
```

```
## [1] 0 9 5
```

But what if we need several elements that do not stand side by side? Arrange their indices in a vector and put it in square brackets:

```
# 1st, 4th, 6th
v[c(1, 4, 6)]
```

```
## [1] 2 0 5
```

If we want to choose all elements except certain ones, we can put a minus before a vector of indices:

```
# all without the 2nd and the 4th
v[-c(2, 4)]
```

```
## [1] 2 1 9 5 8
```

Not only sequences of subsequent integers are possible, we can use `seq()` to create more general sequences and specify the step, the interval between values in a vector:

```
# the same as 1:10
seq(1, 10)
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10
```

```
# step equals 2
seq(1, 10, 2)
```

```
## [1] 1 3 5 7 9
```

```
# step equals 0.5, decimals are possible
seq(1, 10, 0.5)
```

```
##  [1]  1.0  1.5  2.0  2.5  3.0  3.5  4.0  4.5  5.0  5.5  6.0  6.5  7.0  7.5
## [15]  8.0  8.5  9.0  9.5 10.0
```

Filtering elements by value

In most cases we do not choose elements by indices, but filter vectors by values. It will be useful for real data as well because the same logic will be applied for filtering rows in tables.

Recall the vector `v`:

```
v
```

```
## [1] 2 7 1 0 9 5 8
```

Let's choose values of `v` that are greater than 2:

```
v[v > 2]
```

```
## [1] 7 9 5 8
```

Why it is not enough to type `v > 2`? Let's look:

```
v > 2
```

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

This expression returns a vector of values `TRUE` and `FALSE`. If an element satisfies the condition, there will be `TRUE`, otherwise, there will be `FALSE`. So, when we run `v[v > 2]`, R takes only those values in `v` that are with `TRUE`.

Let's test more conditions:

```
# equality
v[v == 5]
```

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

Note that we should use double = to test the equality, a single one is used only for assignment, not for testing conditions.

```
# inequality, !=
v[v != 5]
```

```
## [1] 2 7 1 0 9 8
```

In programming we use != for testing inequality. Generally, ! in programming is used for negation.

Now let's combine two conditions. Suppose we want to choose elements from v that are greater than 1 and less than 7 at the same time:

```
# & - and, two conditions hold true at the same time
v[v > 1 & v < 7]
```

```
## [1] 2 5
```

Now choose elements that are either less than 2 or greater than 5:

```
# | - or, at least one condition holds true
v[v < 2 | v > 5]
```

```
## [1] 7 1 0 9 8
```

Note: this operator | corresponds to inclusive OR, not to the exclusive one. So, two conditions can be true at the same time. In our case it is impossible (a number cannot be less than 2 and greater than 5 at the same time), but generally it is ok.

Vectors of repeated values

Sometimes, especially if we create a data set from scratch, we have to create a set of repeated values. For example, we have data on 15 female respondents and we need a column for gender. We can repeat the letter "F" 15 times using the function rep():

```
# the value goes first, then the number of times
rep("F", 15)
```

```
##  [1] "F" "F" "F" "F" "F" "F" "F" "F" "F" "F" "F" "F" "F" "F" "F"
```

The same can be done with numbers:

```
rep(2, 10)
```

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

Suppose we know that respondents in our data alternate: *female*, *male*, *female*, *male*, and so on. So, we should repeat "F" and "M" several times. We can arrange these letters in a vector and repeat:

```
# repeat a vector 5 times
rep(c("F", "M"), 5)
```

```
##  [1] "F" "M" "F" "M" "F" "M" "F" "M" "F" "M"
```

And what if we want to repeat "F" 5 times and then "M" 5 times? We can add the option each:

```
# repeat each value in a vector 5 times
rep(c("F", "M"), each=5)
```

```
##  [1] "F" "F" "F" "F" "F" "M" "M" "M" "M" "M"
```

Type conversion

Last time we discussed that vectors can be of different type: numeric, character, logical... How to convert the type of elements in a vector? In R there is a set of functions starting with the prefix `as.` that are used for conversion: `as.numeric()`, `as.character()`, etc. Suppose we have a character vector `a` and we want to make it numeric:

```
a <- c("1", "0", "1", "0", "1")
as.numeric(a)
```

```
## [1] 1 0 1 0 1
```

Note: remember that objects in R cannot be changed unless we reassign values explicitly. So, if we look at `a`, we will see that it is still character:

```
a
```

```
## [1] "1" "0" "1" "0" "1"
```

But we can reassign values and save changes:

```
a <- as.numeric(a)
a
```

```
## [1] 1 0 1 0 1
```

The same can be done for numeric vectors, now we can convert a numeric vector to a character one:

```
b <- c(1, 2, 3, 0)
b <- as.character(b)
b
```

```
## [1] "1" "2" "3" "0"
```

```
# make sure it is character
class(b)
```

```
## [1] "character"
```

Now let's discuss factor vectors, a special type of vectors in R. At the previous lecture you discussed that categorical (nominal) data in R are called *factor data*. Suppose we want to encode a categorical variable `region`: value 1 stands for Europe, 2 - for Asia, 3 for Africa, etc.

```
region <- c(1, 3, 2, 4, 4, 1)
```

And now we want R to understand that these values are not pure numbers that we can compare, but just codes we decided to use instead of text names of regions. So, we can make the vector factor:

```
region <- as.factor(region)
class(region)
```

```
## [1] "factor"
```

Levels are unique values in a vector, so values that are used to encode categories of a non-numeric variable.

Why is it important to use correct data types in R? If R knows that a vector is not numeric, it will not allow us to treat this vector as numeric. For example, technically we could plot a histogram of `region` before making it factor, but it would not be substantially correct (we cannot sort values in `region` because they correspond to continents, so we should not plot histograms for categoriacal data). However, if we convert `region` to factor, R will not plot a histogram for it. It will return an error `'x' should be numeirc`.

```
hist(region)
```

Thus, if our data types are correctly specified, R will prevent us from performing operations that are substantially inacceptable.

Now let's look how function `summary()` works for different types of vectors. For numeric vectors it returns descriptive statistics: minimum, maximum, 1st and 3rd quartiles (values that 25% of elements or and 75% of elements do not exceed), median (value that 50% of elements do not exceed) and mean (simple average).

```
summary(a)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##     0.0     0.0     1.0     0.6     1.0     1.0
```

For a character (text) vector it returns only its length and class:

```
summary(b)
```

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

For factor vectors it works as `table()` function, so it returns absolute frequencies for each unique element (factor level) in a vector:

```
summary(region)
```

```
## 1 2 3 4
## 2 1 1 2
```

That's all! Let's proceed to real data.