Basic concepts with R (part 2)

Rodrigo Esteves de Lima-Lopes
State University of Campinas
rll307@unicamp.br

Contents

1 Introduction

In this tutorial we are going to advance a little more about some basic R syntax and general conventions.

2 Vectors

2.1 Definition

A vector is a kind of structured data that form the basis of the R programming. A vector is:

- A one-dimensional data set
  - One **row** x **multiple columns**
- A group of data of the same nature:
  - All are either *character*, *logical*, *integer* or *double*.

2.2 Building a vector

To create a vector, let's execute the following code:

```
my.vector <- c(1,2,3,4,5,6)
my.vector
```

```
## [1] 1 2 3 4 5 6
```

If we look at the code above, we see that we use the function **c ()**, *combine*, to create a line with numbers. **my.vector** is a vector with *5* elements. Let's see its structure:

```
class(my.vector)
```

```
## [1] "numeric"
```

```
str(my.vector)
```

```
##  num [1:6] 1 2 3 4 5 6
```

That is, **my.vector** is a numerical data structure. Another example would be a character vector:

```
my.vector.2 <- c("b","r","a","s","i","l")
my.vector.2
```

```
## [1] "b" "r" "a" "s" "i" "l"
```

```
class(my.vector.2)
```

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

```
str(my.vector.2)
```

```
##  chr [1:6] "b" "r" "a" "s" "i" "l"
```

The quotation marks in the code above tell R to treat the letters as characters. If we don't tell so, it will treat them as a variable saved in the system's memory. This vector of characters represents each item in quotes when R prints it. The following is a logical vector:

```
my.vector.3 <- c(TRUE,FALSE)
my.vector.3
```

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

```
class(my.vector.3)
```

```
## [1] "logical"
```

```
str(my.vector.3)
```

```
##  logi [1:2] TRUE FALSE
```

In the above commands we learnt:

- The first line creates the vector, using the command **c()**
- The second line tells us the class, **class()**, of our variable
  - Very useful when we are trying to find out what a variable is
- The third line gives us the structure, **str()**, of the variable

2.3 Naming Vectors

Vectors can be named, there are specific functions for that. Naming a vector can help me access this data more clearly, in addition to making the variable more intuitive.

```
my.vector <- c(1,2,3,4,5,6,7)
names(my.vector) <- c("Domingo","Segunda","Terça","Quarta","Quinta","Sexta","Sábado")
my.vector
```

```
## Domingo Segunda   Terça  Quarta  Quinta   Sexta  Sábado
##       1       2       3       4       5       6       7
```

That is, using the function **names()** and the function **c ()** I assigned a name to each column: Domingo contains the number 1 and so on.

If we may want to rename other datasets with the days of the week, there is a smarter way. We keep those days in a variable that can be reused:

```r
my.vector <- c(1,2,3,4,5,6,7)
days <- c("Domingo","Segunda","Terça","Quarta","Quinta","Sexta","Sábado")
names(my.vector) <-days
my.vector
```

```
## Domingo Segunda   Terça  Quarta  Quinta   Sexta  Sábado
##       1       2       3       4       5       6       7
```

2.4 Operations with vectors

2.5 Basic Operations

Vectors can be used to perform some basic operations. For example, I can add these vectors:

```r
v1 <- c(10,29,333)
v2 <- c(57,68,702)
v1+v2
```

```
## [1]   67   97 1035
```

Vectors v1 and v2 were added individually, in each column. In other words, column 1 of v1 was added to column 1 of v2 and so on. We can also perform:

```r
v1/v2
```

```
## [1] 0.1754386 0.4264706 0.4743590
```

```r
v1*v2
```

```
## [1]    570   1972 233766
```

```r
v1^v2
```

```
## [1] 1.000000e+57 2.773728e+99          Inf
```

```r
v1-v2
```

```
## [1]  -47  -39 -369
```

2.6 Some Functions

There are a number of functions that can be performed with vectors using R preloaded functions. Just so we can remember a function is applied as follows:

function_name (function input)

```r
sum(v1) # Sum the contents of the vector
```

```
## [1] 372
```

```r
sd(v1) # standard deviation
```

```
## [1] 181.2484
```

```r
max(v1) # Maximum value
```

```
## [1] 333
```

```r
min(v1) # Minimum value
```

```
## [1] 10
```

```r
prod(v1) # The product of all elements
```

```
## [1] 96570
```

3 Indexing and slicing vectors

Vectors are naturally indexed, that is, we can access data using some operators. Maybe in simple vectors like we have here it doesn't make much of a difference, but if we start working with large amounts of data, it is quite useful.

Let's go back to our vector:

```
my.vector
```

```
## Domingo Segunda   Terça  Quarta  Quinta   Sexta  Sábado
##       1       2       3       4       5       6       7
```

We can access our data using the following command:

```
my.vector[1] # the first element of the vector
```

```
## Domingo
##       1
```

```
my.vector[2] # the second element of the vector
```

```
## Segunda
##       2
```

We can also access it by the name of the column, which makes everything easier.

```
my.vector["Domingo"]
```

```
## Domingo
##       1
```

It is also possible to create a multiple indexing process, which allows us to slice it. These slices can also be sent to another variable.

```
my.vector[c(1,2)]
```

```
## Domingo Segunda
##       1       2
```

```
my.vector[c("Sexta","Sábado")]
```

```
##  Sexta Sábado
##      6      7
```

```
my.vector.2<-my.vector[c("Sexta","Sábado")]
my.vector.2
```

```
##  Sexta Sábado
##      6      7
```

Another way to select parts of a vector is by a range.

```
my.vector.4 <- c(1,2,3,4,5,6)
my.vector.5 <- c("b","r","a","s","i","l")
names(my.vector.4) <- my.vector.5
my.vector.4[1:3]
```

```
## b r a
## 1 2 3
```

It is possible to create comparison functions.

```
names.V <- c("a","b","c")
names(v1) <- names.V
names(v2) <- names.V
v1[v1<100]
```

```
##  a  b
## 10 29
```

```
v2[v2<100]
```

```
##  a  b
## 57 68
```

```
v1[v1>100]
```

```
##   c
## 333
```

```
v2[v2>100]
```

```
##   c
## 702
```

```
v1[v1==100]
```

```
## named numeric(0)
```

```
v2[v2==100]
```

```
## named numeric(0)
```

Notice the difference:

```
v1<100
```

```
##     a     b     c
##  TRUE  TRUE FALSE
```

```
v2<100
```

```
##     a     b     c
##  TRUE  TRUE FALSE
```

Now, let us look at this:

```
filter.1<-v1<100
filter.2<-v2<100
filter.1
```

```
##     a     b     c
##  TRUE  TRUE FALSE
```

```
filter.2
```

```
##     a     b     c
##  TRUE  TRUE FALSE
```

```
v1[filter.1]
```

```
##  a  b
## 10 29
```

```
v2[filter.2]
```

```
##  a  b
## 57 68
```

```
v1[!filter.1]
```

```
##   c
## 333
```

```
v2[!filter.2]
```

```
##   c
## 702
```

R operators are:

- '>' greater than
- '<' less than
- '<=' less than or equal to
- '>=' greater than or equal to
- '**!**=' not equal to
- '&' AND
- '==' exactly equal to
- '**!x**' Not x
- '|' OR