Data types in R

Jaime Davila

1/31/22

Introduction

The following worksheet follows closely the content from chapter 20 of R for data Science

Vectors

One of the most basic types in R is a vector which is basically an ordered collection of elements. We can define a vector in R by using the function `c()` and we can access the different elements of a vector by using `[]`. For example:

```r
(dec.numbers <- c(1.1,2.2,3.3,4.4,5.5))
```

```
## [1] 1.1 2.2 3.3 4.4 5.5
```

```r
dec.numbers[1]
```

```
## [1] 1.1
```

```r
dec.numbers[5]
```

```
## [1] 5.5
```

```r
dec.numbers[2]
```

```
## [1] 2.2
```

Two key things we can do we vectors is calculate their number of elements by using **`length`**, and determine the type of a vector by using **`typeof`**:

```r
length(dec.numbers)
```

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

```r
typeof(dec.numbers)
```

```
## [1] "double"
```

The different types of vectors can be summarized in the following figure from R for data science. We will explore it in detail in the next couple of sections.

Atomic data types

Let's start by creating some vectors using our favorite characters from "Encanto"

```r
(madrigal.names <- c("Alma","Mirabel","Bruno","Luisa","Antonio"))
```

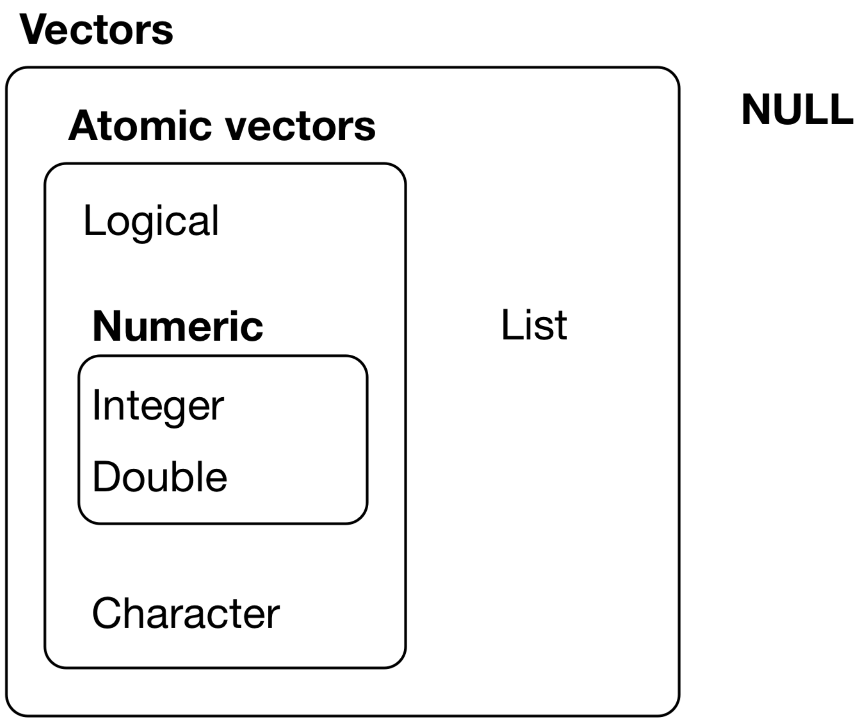

Figure 1: Types of vectors (R for data science ch 20).

```
## [1] "Alma"    "Mirabel" "Bruno"   "Luisa"   "Antonio"
```

```
(madrigal.ages <- c(75, 15,50,19,5))
```

```
## [1] 75 15 50 19  5
```

```
(madrigal.heights <- c(5.5, 5.2,5.4,6.5,3.10))
```

```
## [1] 5.5 5.2 5.4 6.5 3.1
```

```
(madrigal.haspower <- c(FALSE,FALSE,TRUE,TRUE,TRUE))
```

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

Numbers

Notice that in R the default numeric type is `double`. Notice how the type of of `madrigal.ages` is the same as for `madrigal.heights` (double)

```
typeof(madrigal.ages)
```

```
## [1] "double"
```

```
typeof(madrigal.heights)
```

```
## [1] "double"
```

If you want to force R to considers the ages as integers you can use the letter L after the number as follows

```
(madrigal.ages <- c(75L,15L,50L,19L,5L))
```

```
## [1] 75 15 50 19  5
```

```
typeof(madrigal.ages)
```

```
## [1] "integer"
```

Booleans

Booleans are either TRUE or FALSE (or NA). Notice how R can compare vectors of numbers using a expression (for example `>=`) and the result is a vector of booleans.

```
(madrigal.adult <- (madrigal.ages>=18))
```

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

Characters

Each element of a character vector is a string. Notice how strings are represented using the quotes (" "). They can be as short as the empty string or as long as you want them to be:

```
two.words = c("","Incomprehensibilities")
typeof(two.words)
```

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

Common operations in vectors

Coercion In R we can convert between different types by using the functions with the prefix `as.`.

1. Experiment using `as.logical`, `as.integer`, `as.double`, and `as.character` using the 4 vectors of attributes from the Madrigal family.

```
as.integer(madrigal.names)
```

```
## Warning: NAs introduced by coercion
```

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

```
as.logical(madrigal.names)
```

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

```
as.character(madrigal.ages)
```

```
## [1] "75" "15" "50" "19" "5"
```

```
as.integer(madrigal.heights)
```

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

Subsetting The basic function for subsetting vectors is `[]`. Notice how you can use it in a number of different ways:

```
madrigal.names[1]
```

```
## [1] "Alma"
```

```
madrigal.names[c(1,4)]
```

```
## [1] "Alma"  "Luisa"
```

```
madrigal.names[madrigal.ages>=18]
```

```
## [1] "Alma"  "Bruno" "Luisa"
```

2. From our collection of Madrigal family members what are the names of the ones that have powers? What's the age of the Madrigal family members who are over 6ft?

```
madrigal.names[madrigal.haspower]
```

```
## [1] "Bruno"   "Luisa"   "Antonio"
```

```
madrigal.names[!madrigal.haspower]
```

```
## [1] "Alma"    "Mirabel"
```

```
madrigal.ages[madrigal.heights>=6]
```

```
## [1] 19
```

Lists

List are data types that allow you to store different types in a single structure Let's start with a simple example.

```
(mirabel.lst = list(name="Mirabel", age=15L, height=5.2, haspowers=FALSE,
                    songs=c("The Family Madrigal", "Waiting on a Miracle")))
```

```
## $name
## [1] "Mirabel"
##
## $age
## [1] 15
##
## $height
## [1] 5.2
##
## $haspowers
## [1] FALSE
##
## $songs
## [1] "The Family Madrigal"  "Waiting on a Miracle"
```

Notice we can access each element of the list by using the dollar sign and the name of each field.

```
mirabel.lst$name
```

```
## [1] "Mirabel"
```

```
mirabel.lst$age
```

```
## [1] 15
```

3. Explore the use of `[[]]` using `mirabel.lst`. Using `[[]]` extract the height and age of Mirabel. Also extract the second song that she sings in the movie.

```r
mirabel.lst[[3]]
```

```
## [1] 5.2
```

```r
mirabel.lst[[2]]
```

```
## [1] 15
```

```r
mirabel.lst[[5]][2]
```

```
## [1] "Waiting on a Miracle"
```

Augmented vectors

Tables (tibbles) One of the most important data types in R is the data frame/tibble which is used to store tables. Let's make a data frame, a tibble, and a list out of our information

```r
madrigal.df <- data.frame(name = madrigal.names,
                          age = madrigal.ages,
                          height = madrigal.heights,
                          has.power = madrigal.haspower)
madrigal.df
```

```
##      name age height has.power
## 1    Alma  75    5.5     FALSE
## 2 Mirabel  15    5.2     FALSE
## 3   Bruno  50    5.4      TRUE
## 4   Luisa  19    6.5      TRUE
## 5 Antonio   5    3.1      TRUE
```

```r
madrigal.tbl <- tibble(name = madrigal.names,
                       age = madrigal.ages,
                       height = madrigal.heights,
                       has.power = madrigal.haspower)
madrigal.tbl
```

```
## # A tibble: 5 x 4
##   name      age height has.power
##   <chr>   <int>  <dbl> <lgl>
## 1 Alma       75    5.5 FALSE
## 2 Mirabel    15    5.2 FALSE
## 3 Bruno      50    5.4 TRUE
## 4 Luisa      19    6.5 TRUE
## 5 Antonio     5    3.1 TRUE
```

```r
madrigal.lst <- list (name = madrigal.names,
                      age = madrigal.ages,
                      height = madrigal.heights,
                      has.power = madrigal.haspower )
madrigal.lst
```

```
## $name
## [1] "Alma"    "Mirabel" "Bruno"   "Luisa"   "Antonio"
##
## $age
## [1] 75 15 50 19  5
##
```

```
## $height
## [1] 5.5 5.2 5.4 6.5 3.1
##
## $has.power
## [1] FALSE FALSE  TRUE  TRUE  TRUE
```

4. We can access a column of a tibble in data frame by using the `$` sign. Obtain detailed information from column `height` from `madrigal.tbl` and `madrigal.df`.

```
madrigal.tbl$height
```

```
## [1] 5.5 5.2 5.4 6.5 3.1
```

```
madrigal.df$height
```

```
## [1] 5.5 5.2 5.4 6.5 3.1
```

5. We can explore the structure of a variable by using the command `str`. Use `str` to find any differences between `madrigal.lst` and `madrigal.tbl`? Use this information to describe how tibbles are constructed using lists.

```
str(madrigal.tbl)
```

```
## tibble [5 x 4] (S3: tbl_df/tbl/data.frame)
##  $ name     : chr [1:5] "Alma" "Mirabel" "Bruno" "Luisa" ...
##  $ age      : int [1:5] 75 15 50 19 5
##  $ height   : num [1:5] 5.5 5.2 5.4 6.5 3.1
##  $ has.power: logi [1:5] FALSE FALSE TRUE TRUE TRUE
```

```
str(madrigal.lst)
```

```
## List of 4
##  $ name     : chr [1:5] "Alma" "Mirabel" "Bruno" "Luisa" ...
##  $ age      : int [1:5] 75 15 50 19 5
##  $ height   : num [1:5] 5.5 5.2 5.4 6.5 3.1
##  $ has.power: logi [1:5] FALSE FALSE TRUE TRUE TRUE
```

6. What happens if you try to create a `data.frame` or a `tibble` with two columns of different sizes?

```
#example.tbl = tibble(age=c(1,2), name=c("a","b","c"))
```

Factors Factors are a very useful data type for representing categorical data in R. Let's represent the generation of the family member using a factor with 3 levels, "first", "second", and "third"

```
madrigal.levels <- c("first","second","third")
madrigal.generation <- factor (c("first","third","second","third","third"),
                               levels=madrigal.levels)
madrigal.generation
```

```
## [1] first  third  second third  third
## Levels: first second third
```

```
str(madrigal.generation)
```

```
##  Factor w/ 3 levels "first","second",..: 1 3 2 3 3
```

```
typeof(madrigal.generation)
```

```
## [1] "integer"
```

```
levels(madrigal.generation)
```

```
## [1] "first"  "second" "third"
```

Behind the scenes, R stores each level as a integer (that's why the type is an integer) and the command `levels` shows the order of each of these categories. Notice how the order of such categories is important and on occasion is used to force a particular order when doing plots.

7. Notice that factors can be a source of frustration since on occasion they can look like strings when they are used in a data frame. Determine the type of the field `name` in `madrigal.df` and `madrigal.tbl`.

```
typeof(madrigal.df$name)
```

```
## [1] "integer"
```

```
typeof(madrigal.tbl$name)
```

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

```
levels(madrigal.df$name)
```

```
## [1] "Alma"    "Antonio" "Bruno"   "Luisa"   "Mirabel"
```

Matrices Matrices are extensions of vectors that havedimensions (number of rows and number of columns). Let's create a matrix where the columns represent the characters of the movie and the rows are the songs. To do that we will use the commands:

- `rbind` (row bind): To bind the vector rows
- `rownames`: To give names to the rows which are the movie songs
- `colnames`: To give names to the columns which are the movie characters

```
songs.mat <- rbind (c(1,0),
                    c(1,0),
                    c(0,1))
rownames(songs.mat) <- c("The Family Madrigal",
                        "Waiting for a miracle",
                        "Surface Pressure")
colnames(songs.mat) <- c("Mirabel","Luisa")

dim(songs.mat)
```

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

```
songs.mat
```

```
##                         Mirabel Luisa
## The Family Madrigal           1     0
## Waiting for a miracle         1     0
## Surface Pressure              0     1
```

8. In order to access the contents of a matrix we can use `[]`. What do the following commands do? How do you get the elements of the second column of this matrix? Can you create a submatrix with the first and second rows?

```
songs.mat[1]
```

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

```
songs.mat[1,2]
```

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

```
songs.mat[1,]
```

```
## Mirabel   Luisa
##       1       0
```

```
songs.mat[,1]
```

```
##   The Family Madrigal Waiting for a miracle      Surface Pressure
##                     1                     1                     0
```

```
songs.mat[,2]
```

```
##   The Family Madrigal Waiting for a miracle      Surface Pressure
##                     0                     0                     1
```

```
songs.mat[c(1,3),]
```

```
##                     Mirabel Luisa
## The Family Madrigal       1     0
## Surface Pressure          0     1
```