Data Analytics and Visualizations in R - Exercises

Contents

1 Basic R Data Structures

1.1 Types

What are the scalar types in R?

```
# Scalars are just vectors of length one
```

1.2 Weirdness of R

What is the major difference between atomic vectors and lists? How can you turn a list into an atomic vector?

In order to check if an object is of a certain type you can use `is.[type](object)` ,e.g. `is.integer(object)`

Can you use the `is.vector()` function to understand whether a data structure is a vector? If not, what are the functions that you can use for this purpose?

```
# Atomic vectors can contain elements of the same type. The elements of a list can have different types

# Yes, we can use the `is.vector()` function to understand is the structure is a vector.
# It will return TRUE or FALSE.

a <- list(c(1,2,3), "bla", TRUE)
print(a)
```

```
## [[1]]
## [1] 1 2 3
##
## [[2]]
## [1] "bla"
##
## [[3]]
## [1] TRUE
```

```
is.vector(a)
```

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

```
# Best response: If possible you should use type specific coercions like `as.numeric()` or `as.characte
# But since lists are heterogenous, this might not work. A more general function is `unlist()`,
# which returns the list into a vector of the most general type. Notice this difference:

a <- list(c(1,2,3), "bla", TRUE)
unlist(a)
```

```
## [1] "1"    "2"    "3"    "bla"  "TRUE"
```

```
as.character(a)
```

```
## [1] "c(1, 2, 3)" "bla"        "TRUE"
```

```
# Generally you can, but here comes the weird part: `is.vector()` will only return `TRUE` if the vector
# no attributes `names`. Therefore more specific functions like `is.atomic()` or `is.list()` functions
# be used to test if an object is actually atomic vectoror a list
```

1.3 Atomic Vector Concatenation

What happens when you try to generate an atomic vector with `c()`which is composed of different types of elements? What is the `mean()` of a logical vector?

```
# When we attempt to combine different types they will be coerced to the most flexible type. Types from
# to most flexible are: logical, integer, double and character.

str(c("a", 1)) # 1 corced to char
```

```
##  chr [1:2] "a" "1"
```

```
# As TRUE is encoded as 1 and FALSE as 0, the mean is the number of TRUEs devided by the vector length.
mean(c(TRUE, FALSE,FALSE))
```

```
## [1] 0.3333333
```

1.4 Vector Concatenation

Compare X and Y where X and Y are defined as follows. What is the difference?

```
x <- list(list(1,2), c(3,4))
y <- c(list(1,2), c(3,4))
```

```
# Answer
# X will combine seveal lists into one. Given a combination of atomic vectors and lists, y will coerce
# vectors to lists before combining them.
str(x)
```

```
## List of 2
##  $ :List of 2
##   ..$ : num 1
##   ..$ : num 2
##  $ : num [1:2] 3 4
```

```
str(y)
```

```
## List of 4
##  $ : num 1
##  $ : num 2
```

```
##  $ : num 3
##  $ : num 4
```

1.5 From Vectors to `data.frames`

First, create three named numeric vectors of size 10, 11 and 12 respectively in the following manner:

- One vector with the "colon" approach: *from:to*
- One vector with the `seq()` function: *seq(from, to)*
- And one vector with the `seq()` function and the by argument: *seq(from, to, by)*

For easier naming you can use the vector `letters` or `LETTERS` which contain the latin alphabet in small and capital, respectively. In order to select specific letters just use e.g. `letters[1:4]` to get the first four letters. Check their types. What is the outcome? Where do you think does the difference come from?

Then combine all three vectors in a list. Check the attributes of the vectors and the list. What is the difference and why?

Finally coerce the list to a `data.frame` with `as.data.frame()`. Why does it fail and how can we fix it? What happend to the names?

Hint: If list elements have no names, we can access them with the double brackets and an index, e.g. `my_list[[1]]`

```
# Answer

# A. create vectors
aa <- 1:10
names(aa) <- letters[aa]
aa
```

```
##  a  b  c  d  e  f  g  h  i  j
##  1  2  3  4  5  6  7  8  9 10
```

```
bb <- seq(1, 11)
names(bb) <- letters[bb]
bb
```

```
##  a  b  c  d  e  f  g  h  i  j  k
##  1  2  3  4  5  6  7  8  9 10 11
```

```
cc <- seq(1, 12, by=1)
names(cc) <- letters[cc]

typeof(aa)
```

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

```
typeof(bb)
```

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

```
typeof(cc)
```

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

```
# B. Combine all three vectors in a list

my_list <- list(aa, bb, cc)
my_list
```

```
## [[1]]
##  a  b  c  d  e  f  g  h  i  j
##  1  2  3  4  5  6  7  8  9 10
##
## [[2]]
##  a  b  c  d  e  f  g  h  i  j  k
##  1  2  3  4  5  6  7  8  9 10 11
##
## [[3]]
##  a  b  c  d  e  f  g  h  i  j  k  l
##  1  2  3  4  5  6  7  8  9 10 11 12
```

```
attributes(aa)
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j"
```

```
attributes(bb)
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k"
```

```
attributes(cc)
```

```
## $names
##  [1] "a" "b" "c" "d" "e" "f" "g" "h" "i" "j" "k" "l"
```

```
attributes(my_list)
```

```
## NULL
```

```
# C. Coerce to data.frames

# my_df <- as.data.frame(my_list)# fails

# Fixing the length
my_list[[1]] <- c(my_list[[1]], NA, NA)
my_list[[2]] <- c(my_list[[2]], NA)

my_df <- as.data.frame(my_list)
names(my_df) <- LETTERS[1:3]
my_df
```

```
##    A  B  C
## a  1  1  1
## b  2  2  2
## c  3  3  3
## d  4  4  4
## e  5  5  5
## f  6  6  6
## g  7  7  7
## h  8  8  8
## i  9  9  9
## j 10 10 10
## k NA 11 11
##   NA NA 12
```