Vectors & Subsetting in R

Saranjeet Kaur

24/01/2020

Reference: https://adv-r.hadley.nz/

1. **Vectors**: Atomic vectors (all elements of the same type) and Lists (all elements need not be of the same type).

2. Attributes of vector: dimension and class. Dimensions turns vectors into matrices and arrays (R considers 2D structures like matrices and data frames as vectors).

3. Atomic vectors: logical, numeric vectors (integer and double), character, complex and raw.

```
logic <- TRUE
typeof(logic)
```

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

```
doub <- 0.1212
typeof(doub)
```

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

```
inte <- 256L
typeof(inte)
```

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

```
stri <- "Welcome!"
typeof(stri)
```

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

The concatenation operator for forming longer vectors:

```
logic_vec <- c(TRUE, FALSE)
logic_vec
typeof(logic_vec)
length(logic_vec)
```

```
doub_vec <- c(8, 0.7, 1.33)
doub_vec
```

```
## [1] 8.00 0.70 1.33
```

```
typeof(doub_vec)
```

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

```r
length(doub_vec)
```

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

```r
inte_vec <- c(8L, 100L)
inte_vec
```

```
## [1]   8 100
```

```r
typeof(inte_vec)
```

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

```r
length(inte_vec)
```

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

```r
stri_vec <- c("hello", "all", "these are", "strings")
stri_vec
```

```
## [1] "hello"     "all"       "these are" "strings"
```

```r
typeof(stri_vec)
```

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

```r
length(stri_vec)
```

```
## [1] 4
```

Coercion:
Order: Character to double to integer to logical

```r
str(c("letter", 10))
```

```
##  chr [1:2] "letter" "10"
```

Exercise: check the output of the following (what do they coerce to?):
c(48, FALSE)
c("numeric", 5)
c(14L, TRUE)

Naming a vector:

```r
# While creating:
x <- c(a = 3, b = 5, c = 8)

# Using names():
x <- c(3, 5, 8)
names(x) <- c("a", "b", "c")

# Using setNames():
x <- setNames(c(3, 5, 8), c("a", "b", "c"))
```

Matrix and Array: The dimension attribute of vector allows it to behave like a 2D matrix or a multi-D array

```r
# A matrix:
```

```r
g <- matrix(1:6, nrow = 2, ncol = 3)
g
```

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

```r
# An array:

h <- array(1:12, c(2, 3, 2))
h
```

```
## , , 1
##
##      [,1] [,2] [,3]
## [1,]    1    3    5
## [2,]    2    4    6
##
## , , 2
##
##      [,1] [,2] [,3]
## [1,]    7    9   11
## [2,]    8   10   12
```

```r
# Using dim():

j <- 5:10
j
```

```
## [1]  5  6  7  8  9 10
```

```r
dim(j) <- c(3, 2)
j
```

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

```r
str(4:8) # 1d vector
```

```
##  int [1:5] 4 5 6 7 8
```

```r
str(matrix(2:5, ncol = 1)) # column vector
```

```
##  int [1:4, 1] 2 3 4 5
```

```r
str(matrix(5:9, nrow = 1)) # row vector
```

```
##  int [1, 1:5] 5 6 7 8 9
```

Exercise: Check the outputs of the following:

a1 <- array(1:4, c(4, 1, 1))
a2 <- array(1:4, c(1, 4, 1))
a3 <- array(1:4, c(1, 1, 4))

Lists: Each element can be any type.

```r
l1 <- list((2:6), "character", c(FALSE, FALSE, TRUE), c(7.5, 3.45))
l1
```

```
## [[1]]
## [1] 2 3 4 5 6
##
## [[2]]
## [1] "character"
##
## [[3]]
## [1] FALSE FALSE  TRUE
##
## [[4]]
## [1] 7.50 3.45
```

```r
typeof(l1)
```

```
## [1] "list"
```

```r
str(l1)
```

```
## List of 4
##  $ : int [1:5] 2 3 4 5 6
##  $ : chr "character"
##  $ : logi [1:3] FALSE FALSE TRUE
##  $ : num [1:2] 7.5 3.45
```

Sometimes also called as recursive vectors, because a list can contain other lists.

```r
l2 <- list(list(list(1)))
l2
```

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

```r
typeof(l2)
```

```
## [1] "list"
```

```r
str(l2)
```

```
## List of 1
##  $ :List of 1
##   ..$ :List of 1
##   .. ..$ : num 1
```

Data frames: Built on top of lists. In a data frame, the length of each of its vectors is the same, unlike a list.

```r
df <- data.frame(x = 2:6, y = LETTERS[2:6])
df
```

```
##   x y
## 1 2 B
## 2 3 C
## 3 4 D
## 4 5 E
## 5 6 F
```

```
typeof(df)
```

```
## [1] "list"
```

```
str(df)
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ x: int  2 3 4 5 6
##  $ y: Factor w/ 5 levels "B","C","D","E",..: 1 2 3 4 5
```

```
attributes(df)
```

```
## $names
## [1] "x" "y"
##
## $class
## [1] "data.frame"
##
## $row.names
## [1] 1 2 3 4 5
```

```
df1 <- data.frame(x = 2:6, y = LETTERS[2:6], stringsAsFactors = FALSE)
df1
```

```
##   x y
## 1 2 B
## 2 3 C
## 3 4 D
## 4 5 E
## 5 6 F
```

```
str(df1)
```

```
## 'data.frame':    5 obs. of  2 variables:
##  $ x: int  2 3 4 5 6
##  $ y: chr  "B" "C" "D" "E" ...
```

Subsetting:

```
# The [ operator:
x <- c(2.1, 4.2, 3.3, 5.4) # a simple atomic vector

## Positive integers: will return elements at the specified positions
x[c(3, 1)]  ## third and first element
```

```
## [1] 3.3 2.1
```

```
x[order(x)] ## increasing order
```

```
## [1] 2.1 3.3 4.2 5.4
```

```
x[c(1, 1)] ## duplicate indices, give duplicate values
```

```
## [1] 2.1 2.1
```

```
x[c(2.1, 2.9)] ## real numbers are truncated to integers
```

```
## [1] 4.2 4.2
```

```
## Negative integers: exclude elements at the specified positions
x[-c(2, 4)] ## exclude second and fourth element
```

```
## [1] 2.1 3.3
```

```
## Logical vectors: select elements where the corresponding logical value is TRUE
x[c(TRUE, TRUE, FALSE, FALSE)]
```

```
## [1] 2.1 4.2
```

```
x[x>= 4]
```

```
## [1] 4.2 5.4
```

```
## Nothing: returns the original vector
x[]
```

```
## [1] 2.1 4.2 3.3 5.4
```

```
## Zero: returns a zero-length vector
x[0]
```

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

```
## Matrices:

m <- matrix(1:9, nrow = 3)
colnames(m) <- c("A", "B", "C")
m
```

```
##      A B C
## [1,] 1 4 7
## [2,] 2 5 8
## [3,] 3 6 9
```

```
m[1:2, ] ## first two rows
```

```
##      A B C
## [1,] 1 4 7
## [2,] 2 5 8
```

```
m[c(TRUE, FALSE, TRUE), c("B", "A")]
```

```
##      B A
## [1,] 4 1
## [2,] 6 3
```

```
## Dataframes:
df <- data.frame(x = 1:3, y = 3:1, z = letters[1:3])
df[df$x == 2, ]
```

```
##   x y z
## 2 2 2 b
```

```
df[c(1, 3), ]
```

```
##   x y z
## 1 1 3 a
## 3 3 1 c
```

```
df[c("x", "z")] # subset like a list
```

```
##   x z
## 1 1 a
## 2 2 b
## 3 3 c
```

```r
df[, c("x", "z")] # subset like a matrix
```

```
##   x z
## 1 1 a
## 2 2 b
## 3 3 c
```

More subsetting operators:

```r
## The [[ operator:
## Helps in extracting single items, especially in lists
x <- list(1:5, "alphabet", 5:7)
x
```

```
## [[1]]
## [1] 1 2 3 4 5
##
## [[2]]
## [1] "alphabet"
##
## [[3]]
## [1] 5 6 7
```

```r
x[1]
```

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

```r
x[[1]][[3]]
```

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

```r
## The $ operator:
## x$y is roughly equivalent to x[["y"]]. Often used to access variables in a data frame.
df <- data.frame(x = 1:3, y = 3:1, z = letters[1:3])
df
```

```
##   x y z
## 1 1 3 a
## 2 2 2 b
## 3 3 1 c
```

```r
df$z
```

```
## [1] a b c
## Levels: a b c
```

Exercise: Given a linear model, e.g.,
mod <- lm(mpg ~ wt, data = mtcars),
extract the residual degrees of freedom.