Vectors

Types of Vectors

In this chapter, I would learn about the type of vectors. There are basically 4 types of vectors. That is logical (TRUE- FALSE), Integer (non decimal values, NaN and Inf, -Inf), Double (decimal values) and Character.

Vector that is the same type of elements is Atomic Vector. Otherwise it is a list.

```
lgl_var <- c(TRUE, FALSE)
int_var <- c(1L, 6L, 10L)
dbl_var <- c(1, 2.5, 4.5)
chr_var <- c("these are", "some strings")
```

To get the type of the vector, we would use the function typeof()

```
typeof(lgl_var)
```

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

```
typeof(int_var)
```

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

```
typeof(dbl_var)
```

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

```
typeof(chr_var)
```

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

Missing values

Calculation with NA are mostly NA. Excepting some cases

```
NA + 10
```

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

```
NA *10
```

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

```
NA < 10
```

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

```
!NA
```

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

NA is something between TRUE and FALSE. Let's see it. When doing with TRUE and FALSE, | returns in TRUE & returns in FALSE. NA is in between.

```
NA| TRUE
```

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

```
NA & TRUE
```

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

```
NA | FALSE
```

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

```
NA & FALSE
```

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

```
NA ^ 0
```

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

Checking if NA equal NA or other numbers. Returns all in NA.

```
x <- c(NA, 1, NA, 2)
x == NA
```

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

Testing and Coercion

We can use the function is.*() to test the type of a vector. But it applies to is.logical(), is.integer(), is.double() and is.character()

```
is.logical(lgl_var)
```

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

```
is.integer(lgl_var)
```

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

```
is.double(dbl_var)
```

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

```
is.character(chr_var)
```

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

Check carefully if you use these tests below by reading their documentation:

```
is.vector(lgl_var)
```

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

```
is.numeric(chr_var)
```

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

```
is.atomic(dbl_var)
```

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

When combining different type of vector into one vector, the type of the vector will be corerced into this order: character > double > integer > logical.

```
typeof(c(lgl_var, dbl_var))
```

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

```
typeof(c(lgl_var, chr_var))
```

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

```r
typeof(c(chr_var, int_var))
```

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

Coercing into integer will make the element that is character becomes NA

```r
mixed_var <- c(chr_var, int_var)
typeof(mixed_var)
```

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

```r
as.integer(mixed_var)
```

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

```
## [1] NA NA  1  6 10
```

Attributes

We can set name pair attributes for any variable. Attribute can be strings or numeric.

```r
a <- 1:3
attr(a, "x") <- "abcdef"
attr(a, "x")
```

```
## [1] "abcdef"
```

```r
attr(a, "y") <- 4:6
str(attributes(a))
```

```
## List of 2
##  $ x: chr "abcdef"
##  $ y: int [1:3] 4 5 6
```

```r
# Or equivalently:
a <- structure(1:3, x = "abcdef", y = 4:6)
str(attributes(a))
```

```
## List of 2
##  $ x: chr "abcdef"
##  $ y: int [1:3] 4 5 6
```

There are 2 attributes that is generally preserved. It is names (character vector giving each elements a name) and dim (short for dimension, used to turn vectors into matrices or arrays).

Names

We have a few way to creates name

```r
# Create name at the beginning
x <- c("a" = 1, "b" = 2, "c" = 3)
x
```

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

```r
# Create a names vector
y = 1:3
names(y) = c("a", "b", "c")
y
```

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

```
# Inline with setNames()
z <- setNames(1:3, c("a", "b", "c"))
z
```

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

Dimensions

```
# setting matrix
x <- matrix(1:6, nrow = 2, ncol = 3)
x
```

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

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

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

We can also modify the dim to set the object

```
z <- 1:6
dim(z) <- c(2, 3)
z
```

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

Vector, Matrix and Array have the functions as below.

A vector without a dim attribute is often thought as 1 dimension, but actually NULL dimension. We can also have matrix with one row or one column (row or column vector) with single dimension. We can reveal the difference in str() function

```
str(c(1:3))
```

```
##  int [1:3] 1 2 3
```

```
str(matrix(1:3, ncol = 1))
```

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

Vector	Matrix	Array
`names()`	`rownames()`, `colnames()`	`dimnames()`
`length()`	`nrow()`, `ncol()`	`dim()`
`c()`	`rbind()`, `cbind()`	`abind::abind()`
—	`t()`	`aperm()`
`is.null(dim(x))`	`is.matrix()`	`is.array()`

Figure 1: Formula

```
str(matrix(1:3, nrow = 1))
```

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

```
str(array(1:3, 3))
```

```
##  int [1:3(1d)] 1 2 3
```

Now I will view the source code of setNames and unname. I will use the function getAnywhere() to view that

```
getAnywhere(setNames)
```

```
## A single object matching 'setNames' was found
## It was found in the following places
##   package:stats
##   namespace:stats
##   namespace:methods
## with value
##
## function (object = nm, nm)
## {
##     names(object) <- nm
##     object
## }
## <bytecode: 0x7fbf4d905330>
## <environment: namespace:stats>
```

They simply use the function names() to set the name of the object

```
getAnywhere(unname)
```

```
## A single object matching 'unname' was found
## It was found in the following places
##   package:base
##   namespace:base
## with value
##
## function (obj, force = FALSE)
```

```
## {
##     if (!is.null(names(obj)))
##         names(obj) <- NULL
##     if (!is.null(dimnames(obj)) && (force || !is.data.frame(obj)))
##         dimnames(obj) <- NULL
##     obj
## }
## <bytecode: 0x7fbf4d99b600>
## <environment: namespace:base>
```

dim when applied to one dimensional vector will return NULL. nrow and ncol will applied for matrix.

What is the difference between these objects below?

```
x1 <- array(1:5, c(1, 1, 5))
x2 <- array(1:5, c(1, 5, 1))
x3 <- array(1:5, c(5, 1, 1))
```

x1 is the array of 5. x2 is the array of 1 with the matrix 1 * 5 x3 is the array of 1 with the matrix 5 * 1

Why we don't see the comment attribute of the object when we print?

```
structure(1:5, comment = "my attribute")
```

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

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

Usually dim and names are shown when printing out the attribute/ structure.

S3 atomic vectors

Factor

Factor is under the integer vector but has 2 attributes: class is factor and levels.

```
x <- factor(c("a", "b", "b", "a"))
typeof(x)
```

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

```
attributes(x)
```

```
## $levels
## [1] "a" "b"
##
## $class
## [1] "factor"
```

Another example of factor. Factor would make us know about other levels even it is not appeared in the observations.

```
sex_char <- c("m", "m", "m")
sex_factor <- factor(sex_char, levels = c("m", "f"))

table(sex_char)
```

```
## sex_char
## m
```

```
## 3
```

```
table(sex_factor)
```

```
## sex_factor
## m f
## 3 0
```

Order is a small variation of of factors. It might have meaning such as low, medium and high.

```
# Order
grade <- ordered(c("b", "b", "c", "a"), levels = c("c", "b", "a"))
grade
```

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

In base R, R automatically convert character vector to levels. However the author of the book suggest us if we know well about the data, we should use the parameter: stringAsFactor = FALSE and then convert them manually (in case the levels are not right or there are some unseen level in the data) ### Dates Date is built on top of double vector. They have class Date and no other attribute.

```
today <- Sys.Date()
typeof(today)
```

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

```
attributes(today)
```

```
## $class
## [1] "Date"
```

Datetimes

2 type of datetime information POSIXct and POSIXlt: ct stands for calendar time and lt stands for local time. I will focus on the POSIXct

```
now_ct <- as.POSIXct("2021-06-28 22:00", tz = "UTC")
typeof(now_ct)
```

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

```
attributes(now_ct)
```

```
## $class
## [1] "POSIXct" "POSIXt"
##
## $tzone
## [1] "UTC"
```

Difftime

```
one_week_1 <- as.difftime(1, unit = "weeks")
one_week_1
```

```
## Time difference of 1 weeks
```

Exercise

1. What sort of object the table return? What attributes does it have? How does the dimensionality change when you tabulate more variables?

table() will return the count of the observations if it is the character variable, but if it is factor, it will count all the level of that factor. #### 2. What happens to a factor if you revise its level?

```
f1 <- factor(letters)
levels(f1) <- rev(levels(f1))
```

This function reverses the level of the f1 object.

3. What is the differences between f1, f2 and f3

```
f2 <- rev(factor(letters))

f3 <- factor(letters, levels = rev(letters))
```

f1 is from a-z but level from z-a f2 is from z-a but level from a-z f3 is from a-z but level form z-a

List

List element are technically the same type and each element is a reference to another object, which can be any type. Because of referencing, the list size might be much smaller than we think it might be.

```
library(lobstr)
lobstr::obj_size(mtcars)
```

```
## 7,208 B
```

```
l2 <- list(mtcars, mtcars, mtcars)
obj_size(l2)
```

```
## 7,288 B
```

list can contain other list. like this

```
l3 <- list(list(list(1)))
```

Testing and coercion

We can test the list by using is.list(). coerce using as.list() and see type by using typeof()

```
typeof(l3)
```

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

```
is.list(l3)
```

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

```
as.list(1:3)
```

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

Matrices and arrays

list can be list matrix and list array by setting the attribute to them:

```
l <- list(1:3, "a", TRUE, 1.0)
dim(l) <- c(2, 2)
l
```

```
##      [,1]      [,2]
## [1,] Integer,3 TRUE
## [2,] "a"       1
```

```
l[[1, 1]]
```

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

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

Using unlist

```
typeof(unlist(l))
```

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

Using as.vector()

```
typeof(as.vector(l))
```

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

Dataframe and tibbles

Example of dataframe

```
df1 <- data.frame(x = 1:3, y = letters[1:3])
typeof(df1)
```

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

```
attributes(df1)
```

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

As we can see, dataframe is a list. Actually, dataframe is a named list of vectors. The attributes are names and row.names, class is data.frame. length of dataframe is the number of of columns.

Example of tibble

```
library(tibble)
df2 <- tibble(x = 1:3, y = letters[1:3])
typeof(df2)
```

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

```
attributes(df2)
```

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

```
df <- data.frame(
  x = 1:3,
  y = c("a", "b", "c")
)

str(df)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: Factor w/ 3 levels "a","b","c": 1 2 3
```

To convert str to factor, be aware to use StringAsFactors = FALSE

```
df_str <- data.frame(x = 1:3, y = c("a", "b", "c"), stringsAsFactors = FALSE)
str(df_str)
```

```
## 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: chr  "a" "b" "c"
```

Tibble is lazy because they never coerce their input. Let see the identical example with function tibble

```
df_str2 <- tibble(x = 1:3, y = c("a", "b", "c"))
str(df_str2)
```

```
## Classes 'tbl_df', 'tbl' and 'data.frame':    3 obs. of  2 variables:
##  $ x: int  1 2 3
##  $ y: chr  "a" "b" "c"
```

We can see that tibble do not convert them automatically str to factor.

```
names(data.frame('1' = 1))
```

```
## [1] "X1"
```

```
names(tibble('1' = 1))
```

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

Tibbe as we see do not change the name of non-syntactic names unless we use check.names = FALSE, they don't change name surrounded by ". Tibble allows you to refer to variable created during construction

```
x <- tibble(x = 1:3, y = x*2)
x
```

```
## # A tibble: 3 x 2
##       x     y
##   <int> <dbl>
## 1     1     2
## 2     2     4
## 3     3     6
```