2 Vectors

(AST230) R for Data Science

Md Rasel Biswas

Creating vectors in R

- Up until now we've been creating simple objects by directly assigning a single value to an object.
- It's very likely that you'll soon want to progress to creating more complicated objects. Happily, R has a multitude of functions to help you do this
- The first function we will learn about is the `c()` function.
- The `c()` function is short for concatenate and we use it to join together a series of values and store them in a data structure called a **vector** or **"atomic vector"**

```r
my_vec <- c(2, 3, 1, 6, 4, 3, 3, 7)
my_vec
```

```
[1] 2 3 1 6 4 3 3 7
```

Creating vectors in R

- Now that we've created a vector. we can use other functions to do useful stuff with this object
- For example, we can calculate the mean, variance, standard deviation and number of elements in our vector by using the `mean()`, `var()`, `sd()` and `length()` functions

```
mean(my_vec)    # returns the mean of my_vec
```

```
[1] 3.625
```

```
var(my_vec)     # returns the variance of my_vec
```

```
[1] 3.982143
```

```
sd(my_vec)      # returns the standard deviation of my_vec
```

```
[1] 1.995531
```

```
length(my_vec)  # returns the number of elements in my_vec
```

```
[1] 8
```

Creating vectors in R

Important

Scalar is a vector of length one

- If we wanted to use any of these values later on in our analysis we can just assign the resulting value to another object.

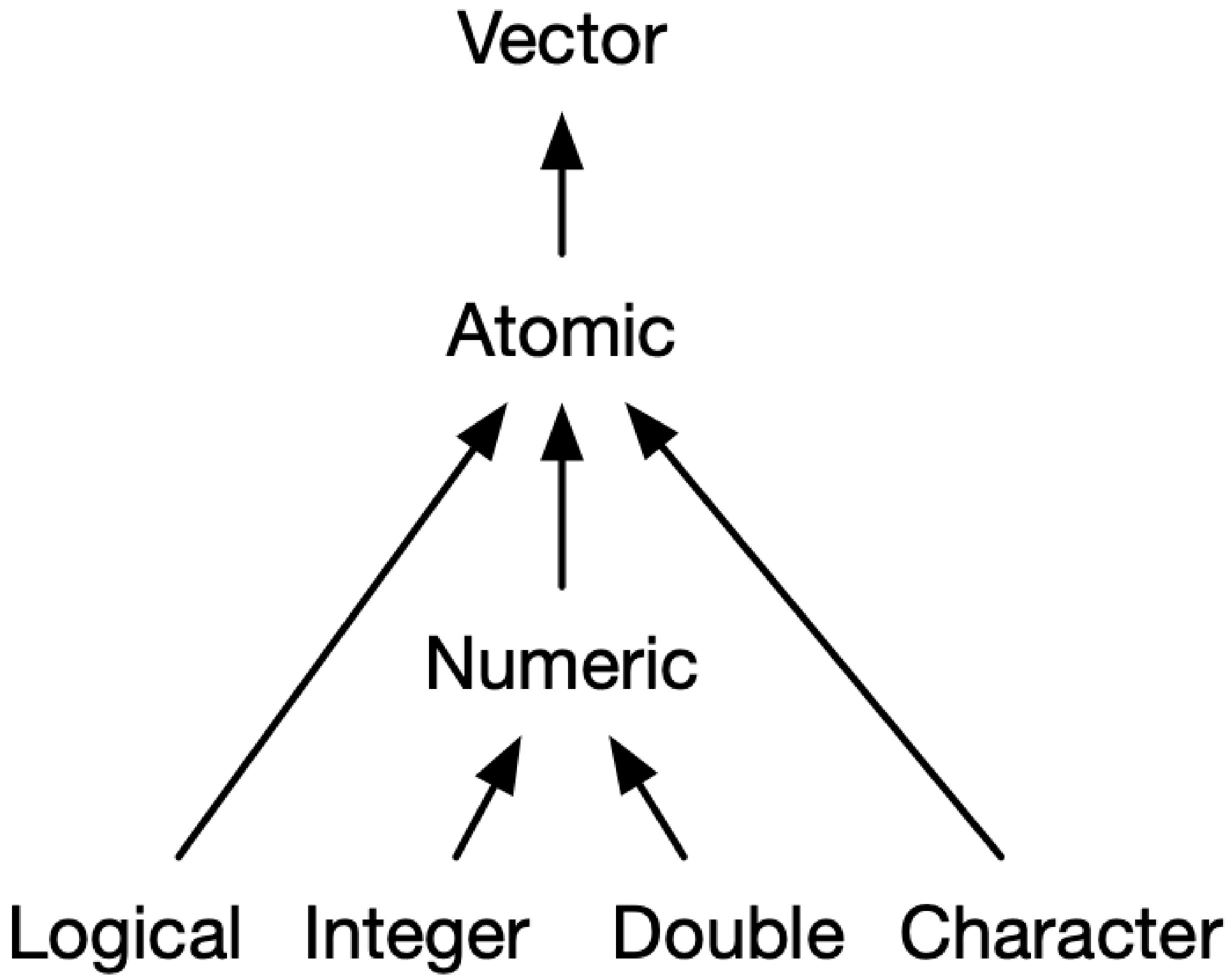
```r
vec_mean <- mean(my_vec)    # returns the mean of my_vec
vec_mean
```

```
[1] 3.625
```

Vector basics

- Atomic vectors are of six types:
  - `logical`
  - `integer`
  - `double`
  - `character`
  - ~~`complex`~~
  - ~~`raw`~~
- Integer and double vectors are collectively known as numeric vectors.

Vector basics



Vector basics

- Every vector has two key properties:

  1. Its **type**, which you can determine with `typeof()`

```r
my_vec <- c(2, 3, 1, 6, 4, 3, 3, 7)
typeof(my_vec)
```

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

2. Its **length**, which you can determine with `length()`.

```r
length(my_vec)
```

```
[1] 8
```

- Vectors can also contain arbitrary additional metadata in the form of attributes (More on this later)

Logical vectors

Logical vectors are the simplest type of atomic vector because they can take only two possible values: `FALSE` and `TRUE`

```
x_l <- c(TRUE, FALSE, TRUE)
x_l
```

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

```
typeof(x_l)
```

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

```
is.logical(x_l)
```

```
[1] TRUE
```

Logical vectors

logical operator	symbol in R
equal to	`==`
greater or greater equal	`>`,`>=`
less or less equal	`<`,`<=`
not equal	`!=`

```
10 == 15
```

```
[1] FALSE
```

```
10 != 15
```

```
[1] TRUE
```

```
10 > 15
```

```
[1] FALSE
```

```
10 < 15
```

```
[1] TRUE
```

Numeric vectors

- Integer and double vectors are known collectively as numeric vectors
- In R, numbers are doubles by default. To make an integer, place an `L` after the number:

```
x_d <- c(1., 5.5, 20.134, .32)
x_d
```

```
[1]  1.000  5.500 20.134  0.320
```

```
typeof(x_d); is.double(x_d)
```

```
[1] "double"
[1] TRUE
```

```
(x_i <- c(1L, 50L, 20L, 32L))
```

```
[1]  1 50 20 32
```

```
typeof(x_i); is.integer(x_i)
```

```
[1] "integer"
[1] TRUE
```

Character vectors

- Character vectors are used to represent string values. You can think of character strings as something like a word (or multiple words).
- It is represented by a collection of characters between double quotes (`"`)

```r
x_c <- c("boy", "boy", "girl")
x_c
```

```
[1] "boy"  "boy"  "girl"
```

```r
typeof(x_c)
```

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

```r
is.character(x_c)
```

```
[1] TRUE
```

Missing values

- NULL is often used to represent the absence of a vector
  - NULL typically behaves like a vector of length 0

```
my_vec1 <- NULL
my_vec1 <- c(my_vec1, 10)
my_vec1
```

```
[1] 10
```

- NA is used to represent the absence of a value in a vector.

```
my_vec2 <- c(18, 21, NA, 22)
my_vec2
```

```
[1] 18 21 NA 22
```

- `interger` and `double` $\rightarrow$ quantitative data
- `character` $\rightarrow$ qualitative data
- `logical` $\rightarrow$ binary data

Sequence of numbers

- Sometimes it can be useful to create a vector that contains a regular sequence of values in steps of one.
- Here we can make use of a shortcut using the `:` (colon) symbol.

```
my_seq <- 1:10      # create regular sequence
my_seq
```

```
[1]  1  2  3  4  5  6  7  8  9 10
```

```
my_seq2 <- 10:1     # in decending order
my_seq2
```

```
[1] 10  9  8  7  6  5  4  3  2  1
```

```
-5:4
```

```
[1] -5 -4 -3 -2 -1  0  1  2  3  4
```

Sequence of numbers: `seq()`

- Other useful functions for generating vectors of sequences include the `seq()` and `rep()` functions.
- For example, to generate a sequence from `1` to `5` in steps of `0.5`

```r
seq(from = 1, to = 5, by = 0.5)
```

```
[1] 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
```

```r
seq(from = 1, to = 5, length.out = 8)
```

```
[1] 1.000000 1.571429 2.142857 2.714286 3.285714 3.857143 4.428571 5.000000
```

- Here we've used the arguments `from =` and `to = to` define the limits of the sequence and the `by =` argument to specify the increment of the sequence.
- Play around with other values for these arguments to see their effect

Repeat vectors using `rep()`

- The `rep()` function allows you to replicate (repeat) values a specified number of times. To repeat the value 2, 10 times

```
rep(2, times = 10)
```

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

- The arguments `times`, `each` and `length.out` are used in `rep()` to obtain different vectors
- We can also repeat non-numeric values. e.g.

```
rep("boy", times = 3)
```

```
[1] "boy" "boy" "boy"
```

Repeat vectors using rep()

```r
rep(c("boy", "girl"), each = 3)
```

```
[1] "boy"  "boy"  "boy"  "girl" "girl" "girl"
```

```r
rep(c("boy", "girl"), times = 3, each = 2)
```

```
 [1] "boy"  "boy"  "girl" "girl" "boy"  "boy"  "girl" "girl" "boy"  "boy" 
[11] "girl" "girl"
```

```r
rep(c("boy", "girl"), length.out = 6)
```

```
[1] "boy"  "girl" "boy"  "girl" "boy"  "girl"
```

Exercise 2

1. Create the vector (1, 2, 3, 4, 5, 6, 7, 8, 9, 10) in three ways:

- using `c()`, `:`, and `seq()` 2. Create the vector (2.1, 4.1, 6.1, 8.1) in two ways:

- using `c()` and once `seq()` 3. Create the vector (0, 5, 10, 15) in 3 ways:

- using `c()`, `seq()` with a `by` argument, and `seq()` with a `length.out` argument.

4. Generate the following sequences using `rep()` or `seq()` functions.

- (1, 1, 2, 2, 3, 3, 4, 4)
- (1, 2, 2, 3, 3, 3, 4, 4, 4, 4)
- (-0.50, -0.25, 0, 0.25, 0.5, 0.75, 1)

Exercise 2

5. Create the vector (101, 102, 103, 200, 205, 210, 1000, 1100, 1200) using a combination of the `c()` and `seq()` functions
6. Create a vector that repeats the integers from 1 to 5, 10 times, i.e. (1, 2, 3, 4, 5, 1, 2, 3, 4, 5, . . .), and the length of the vector should be 50!
7. Create the same vector as before, but this time repeat 1, 10 times, then 2, 10 times, etc., i.e. (1, 1, 1, . . ., 2, 2, 2, . . ., . . ., 5, 5, 5) and the length of the vector should also be 50