Data Types and Structures

Overview

Teaching: 45 min
Exercises: 0 min

Questions

- What are the different data types in R?
- What are the different data structures in R?
- How do I access data within the various data structures?

Objectives

- Expose learners to the different data types in R and show how these data types are used in data structures.
- Learn how to create vectors of different types.
- Be able to check the type of vector.
- Learn about missing data and other special values.
- Get familiar with the different data structures (lists, matrices, data frames).

Understanding Basic Data Types and Data Structures in R

To make the best of the R language, you'll need a strong understanding of the basic data types and data structures and how to operate on them.

Data structures are very important to understand because these are the objects you will manipulate on a day-to-day basis in R. Dealing with object conversions is one of the most common sources of frustration for beginners.

Everything in R is an object.

R has 6 basic data types. (In addition to the five listed below, there is also *raw* which will not be discussed in this workshop.)

- character
- numeric (real or decimal)
- integer
- logical
- complex

Elements of these data types may be combined to form data structures, such as atomic vectors. When we call a vector *atomic*, we mean that the vector only holds data of a single data type. Below are examples of atomic character vectors, numeric vectors, integer vectors, etc.

- **character**: `"a"` , `"swc"`
- **numeric**: `2` , `15.5`
- **integer**: `2L` (the `L` tells R to store this as an integer)
- **logical**: `TRUE` , `FALSE`
- **complex**: `1+4i` (complex numbers with real and imaginary parts)

R provides many functions to examine features of vectors and other objects, for example

- `class()` - what kind of object is it (high-level)?
- `typeof()` - what is the object's data type (low-level)?
- `length()` - how long is it? What about two dimensional objects?
- `attributes()` - does it have any metadata?

R

```
# Example
x <- "dataset"
typeof(x)
```

Output

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

R

```
attributes(x)
```

Output

```
NULL
```

R

```
y <- 1:10
y
```

Output

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

R

```
typeof(y)
```

Output

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

R

```
length(y)
```

Output

```
[1] 10
```

R

```
z <- as.numeric(y)
z
```

Output

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

R

```
typeof(z)
```

Output

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

R has many **data structures**. These include

- atomic vector
- list
- matrix
- data frame
- factors

Vectors

A vector is the most common and basic data structure in R and is pretty much the workhorse of R. Technically, vectors can be one of two types:

- atomic vectors
- lists

although the term "vector" most commonly refers to the atomic types not to lists.

The Different Vector Modes

A vector is a collection of elements that are most commonly of mode `character`, `logical`, `integer` or `numeric`.

You can create an empty vector with `vector()`. (By default the mode is `logical`. You can be more explicit as shown in the examples below.) It is more common to use direct constructors such as `character()`, `numeric()`, etc.

R

```
vector() # an empty 'logical' (the default) vector
```

Output

```
logical(0)
```

R

```
vector("character", length = 5) # a vector of mode 'character' with 5 elements
```

Output

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

R

```
character(5) # the same thing, but using the constructor directly
```

Output

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

R

```
numeric(5)   # a numeric vector with 5 elements
```

Output

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

R

```
logical(5)   # a logical vector with 5 elements
```

Output

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

You can also create vectors by directly specifying their content. R will then guess the appropriate mode of storage for the vector. For instance:

R

```
x <- c(1, 2, 3)
```

will create a vector `x` of mode `numeric`. These are the most common kind, and are treated as double precision real numbers. If you wanted to explicitly create integers, you need to add an `L` to each element (or *coerce* to the integer type using `as.integer()`).

R

```
x1 <- c(1L, 2L, 3L)
```

Using `TRUE` and `FALSE` will create a vector of mode `logical`:

R

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

While using quoted text will create a vector of mode `character`:

R

```
z <- c("Sarah", "Tracy", "Jon")
```

Examining Vectors

The functions `typeof()`, `length()`, `class()` and `str()` provide useful information about your vectors and R objects in general.

R

```
typeof(z)
```

Output

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

R

```
length(z)
```

Output

```
[1] 3
```

R

```
class(z)
```

Output

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

R

```
str(z)
```

Output

```
 chr [1:3] "Sarah" "Tracy" "Jon"
```

Adding Elements

The function `c()` (for combine) can also be used to add elements to a vector.

R

```
z <- c(z, "Annette")
z
```

Output

```
[1] "Sarah"   "Tracy"   "Jon"     "Annette"
```

R

```
z <- c("Greg", z)
z
```

Output

```
[1] "Greg"    "Sarah"   "Tracy"   "Jon"     "Annette"
```

Vectors from a Sequence of Numbers

You can create vectors as a sequence of numbers.

R

```
series <- 1:10
seq(10)
```

Output

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

R

```
seq(from = 1, to = 10, by = 0.1)
```

Output

```
 [1]  1.0  1.1  1.2  1.3  1.4  1.5  1.6  1.7  1.8  1.9  2.0  2.1  2.2  2.3  2.4
[16]  2.5  2.6  2.7  2.8  2.9  3.0  3.1  3.2  3.3  3.4  3.5  3.6  3.7  3.8  3.9
[31]  4.0  4.1  4.2  4.3  4.4  4.5  4.6  4.7  4.8  4.9  5.0  5.1  5.2  5.3  5.4
[46]  5.5  5.6  5.7  5.8  5.9  6.0  6.1  6.2  6.3  6.4  6.5  6.6  6.7  6.8  6.9
[61]  7.0  7.1  7.2  7.3  7.4  7.5  7.6  7.7  7.8  7.9  8.0  8.1  8.2  8.3  8.4
[76]  8.5  8.6  8.7  8.8  8.9  9.0  9.1  9.2  9.3  9.4  9.5  9.6  9.7  9.8  9.9
[91] 10.0
```

Missing Data

R supports missing data in vectors. They are represented as `NA` (Not Available) and can be used for all the vector types covered in this lesson:

R

```
x <- c(0.5, NA, 0.7)
x <- c(TRUE, FALSE, NA)
x <- c("a", NA, "c", "d", "e")
x <- c(1+5i, 2-3i, NA)
```

The function `is.na()` indicates the elements of the vectors that represent missing data, and the function `anyNA()` returns `TRUE` if the vector contains any missing values:

R

```
x <- c("a", NA, "c", "d", NA)
y <- c("a", "b", "c", "d", "e")
is.na(x)
```

Output

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

R

```
is.na(y)
```

Output

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

R

```
anyNA(x)
```

Output

```
[1] TRUE
```

R

```
anyNA(y)
```

Output

```
[1] FALSE
```

Other Special Values

`Inf` is infinity. You can have either positive or negative infinity.

R

```
1/0
```

Output

```
[1] Inf
```

`NaN` means Not a Number. It's an undefined value.

R

```
0/0
```

Output

```
[1] NaN
```

What Happens When You Mix Types Inside a Vector?

R will create a resulting vector with a mode that can most easily accommodate all the elements it contains. This conversion between modes of storage is called "coercion". When R converts the mode of storage based on its content, it is referred to as "implicit coercion". For instance, can you guess what the following do (without running them first)?

R

```
xx <- c(1.7, "a")
xx <- c(TRUE, 2)
xx <- c("a", TRUE)
```

You can also control how vectors are coerced explicitly using the `as.<class_name>()` functions:

R

```
as.numeric("1")
```

Output

```
[1] 1
```

R

```
as.character(1:2)
```

Output

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

Finding Commonalities

Do you see a property that's common to all these vectors above?

Solution

Objects Attributes

Objects can have **attributes**. Attributes are part of the object. These include:

- names
- dimnames
- dim
- class
- attributes (contain metadata)

You can also glean other attribute-like information such as length (works on vectors and lists) or number of characters (for character strings).

R

```
length(1:10)
```

Output

```
[1] 10
```

R

```
nchar("Software Carpentry")
```

Output

```
[1] 18
```

Matrix

In R matrices are an extension of the numeric or character vectors. They are not a separate type of object but simply an atomic vector with dimensions; the number of rows and columns. As with atomic vectors, the elements of a matrix must be of the same data type.

R

```
m <- matrix(nrow = 2, ncol = 2)
m
```

Output

```
     [,1] [,2]
[1,]   NA   NA
[2,]   NA   NA
```

R

```
dim(m)
```

Output

```
[1] 2 2
```

You can check that matrices are vectors with a class attribute of `matrix` by using `class()` and `typeof()`.

R

```
m <- matrix(c(1:3))
class(m)
```

Output

```
[1] "matrix" "array"
```

R

```
typeof(m)
```

Output

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

While `class()` shows that m is a matrix, `typeof()` shows that fundamentally the matrix is an integer vector.

Data types of matrix elements

Consider the following matrix:

R

```
FOURS <- matrix(
  c(4, 4, 4, 4),
  nrow = 2,
  ncol = 2)
```

Given that `typeof(FOURS[1])` returns `"double"`, what would you expect `typeof(FOURS)` to return? How do you know this is the case even without running this code?

Hint Can matrices be composed of elements of different data types?

Solution

We know that `typeof(FOURS)` will also return `"double"` since matrices are made of elements of the same data type. Note that you could do something like `as.character(FOURS)` if you needed the elements of `FOURS` *as characters*.

Matrices in R are filled column-wise.

R

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

Other ways to construct a matrix

R

```
m      <- 1:10
dim(m) <- c(2, 5)
```

This takes a vector and transforms it into a matrix with 2 rows and 5 columns.

Another way is to bind columns or rows using `rbind()` and `cbind()` ("row bind" and "column bind", respectively).

R

```
x <- 1:3
y <- 10:12
cbind(x, y)
```

Output

```
     x  y
[1,] 1 10
[2,] 2 11
[3,] 3 12
```

R

```
rbind(x, y)
```

Output

```
  [,1] [,2] [,3]
x    1    2    3
y   10   11   12
```

You can also use the `byrow` argument to specify how the matrix is filled. From R's own documentation:

R

```
mdat <- matrix(c(1, 2, 3, 11, 12, 13),
               nrow = 2,
               ncol = 3,
               byrow = TRUE)
mdat
```

Output

```
     [,1] [,2] [,3]
[1,]    1    2    3
[2,]   11   12   13
```

Elements of a matrix can be referenced by specifying the index along each dimension (e.g. "row" and "column") in single square brackets.

R

```
mdat[2, 3]
```

Output

```
[1] 13
```

List

In R lists act as containers. Unlike atomic vectors, the contents of a list are not restricted to a single mode and can encompass any mixture of data types. Lists are sometimes called generic vectors, because the elements of a list can by of any type of R object, even lists containing further lists. This property makes them fundamentally different from atomic vectors.

A list is a special type of vector. Each element can be a different type.

Create lists using `list()` or coerce other objects using `as.list()` . An empty list of the required length can be created using `vector()`

R

```
x <- list(1, "a", TRUE, 1+4i)
x
```

Output

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

[[2]]
[1] "a"

[[3]]
[1] TRUE

[[4]]
[1] 1+4i
```

R

```
x <- vector("list", length = 5) # empty list
length(x)
```

Output

```
[1] 5
```

The content of elements of a list can be retrieved by using double square brackets.

R

```
x[[1]]
```

Output

```
NULL
```

Vectors can be coerced to lists as follows:

R

```
x <- 1:10
x <- as.list(x)
length(x)
```

Output

```
[1] 10
```

Examining Lists

1. What is the class of `x[1]` ?
2. What is the class of `x[[1]]` ?

Solution

Elements of a list can be named (i.e. lists can have the `names` attribute)

R

```
xlist <- list(a = "Karthik Ram", b = 1:10, data = head(iris))
xlist
```

Output

```
$a
[1] "Karthik Ram"

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

$data
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
6          5.4         3.9          1.7         0.4  setosa
```

R

```
names(xlist)
```

Output

```
[1] "a"    "b"    "data"
```

Examining Named Lists

1. What is the length of this object?
2. What is its structure?

Solution

Lists can be extremely useful inside functions. Because the functions in R are able to return only a single object, you can “staple” together lots of different kinds of results into a single object that a function can return.

A list does not print to the console like a vector. Instead, each element of the list starts on a new line.

Elements are indexed by double brackets. Single brackets will still return a(nother) list. If the elements of a list are named, they can be referenced by the `$` notation (i.e. `xlist$data`).

Data Frame

A data frame is a very important data type in R. It’s pretty much the *de facto* data structure for most tabular data and what we use for statistics.

A data frame is a *special type of list* where every element of the list has same length (i.e. data frame is a “rectangular” list).

Data frames can have additional attributes such as `rownames()`, which can be useful for annotating data, like `subject_id` or `sample_id`. But most of the time they are not used.

Some additional information on data frames:

- Usually created by `read.csv()` and `read.table()`, i.e. when importing the data into R.
- Assuming all columns in a data frame are of same type, data frame can be converted to a matrix with data.matrix() (preferred) or as.matrix(). Otherwise type coercion will be enforced and the results may not always be what you expect.
- Can also create a new data frame with `data.frame()` function.
- Find the number of rows and columns with `nrow(dat)` and `ncol(dat)`, respectively.
- Rownames are often automatically generated and look like 1, 2, …, n. Consistency in numbering of rownames may not be honored when rows are reshuffled or subset.

Creating Data Frames by Hand

To create data frames by hand:

R

```
dat <- data.frame(id = letters[1:10], x = 1:10, y = 11:20)
dat
```

Output

```
   id  x  y
1   a  1 11
2   b  2 12
3   c  3 13
4   d  4 14
5   e  5 15
6   f  6 16
7   g  7 17
8   h  8 18
9   i  9 19
10  j 10 20
```

Useful Data Frame Functions

- `head()` - shows first 6 rows
- `tail()` - shows last 6 rows
- `dim()` - returns the dimensions of data frame (i.e. number of rows and number of columns)
- `nrow()` - number of rows
- `ncol()` - number of columns
- `str()` - structure of data frame - name, type and preview of data in each column
- `names()` or `colnames()` - both show the `names` attribute for a data frame
- `sapply(dataframe, class)` - shows the class of each column in the data frame

See that it is actually a special list:

R

```
is.list(dat)
```

Output

```
[1] TRUE
```

R

```
class(dat)
```

Output

```
[1] "data.frame"
```

Because data frames are rectangular, elements of data frame can be referenced by specifying the row and the column index in single square brackets (similar to matrix).

R

```
dat[1, 3]
```

Output

```
[1] 11
```

As data frames are also lists, it is possible to refer to columns (which are elements of such list) using the list notation, i.e. either double square brackets or a `$` .

R

```
dat[["y"]]
```

Output

```
 [1] 11 12 13 14 15 16 17 18 19 20
```

R

```
dat$y
```

Output

```
 [1] 11 12 13 14 15 16 17 18 19 20
```

The following table summarizes the one-dimensional and two-dimensional data structures in R in relation to diversity of data types they can contain.

Dimensions	Homogenous	Heterogeneous
1-D	atomic vector	list
2-D	matrix	data frame

Lists can contain elements that are themselves muti-dimensional (e.g. a lists can contain data frames or another type of objects). Lists can also contain elements of any length, therefore list do not necessarily have to be “rectangular”. However in order for the list to qualify as a data frame, the length of each element has to be the same.

Column Types in Data Frames

Knowing that data frames are lists, can columns be of different type?

What type of structure do you expect to see when you explore the structure of the `iris` data frame? Hint: Use `str()` .

Solution

Key Points

- R's basic data types are character, numeric, integer, complex, and logical.
- R's basic data structures include the vector, list, matrix, data frame, and factors. Some of these structures require that all members be of the same data type (e.g. vectors, matrices) while others permit multiple data types (e.g. lists, data frames).
- Objects may have attributes, such as name, dimension, and class.

‹ (../12-supp-factors/index.html)

› (../14-supp-call-stack/