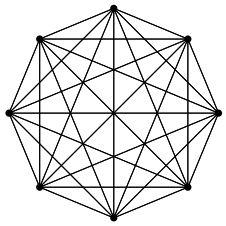
Chapter 14

Coercion



The topic of coercion was first encountered in the context of automatic coercion in Chapter 11 when numeric values were automatically coerced to character strings. Subsequently, we have seen logical values coerced to zeros and ones when certain functions, such as `mean` or `sum`, were applied to an object whose elements were logical values. This chapter could be titled "explicit coercion" in order to emphasize that explicit steps can be taken to coerce the elements of an object. The two topics considered in this chapter are (a) the `is` family of functions, and (b) the `as` family of functions.

14.1 The `is` family of functions

Sometimes the first step in the explicit coercion of the elements of an object is to determine the data type of the elements currently comprising the object. This section surveys the `is` family of functions, which is used to determine whether the elements of an object are of a particular data type. To review, the six data types that we have encountered so far in increasing order by the amount of information that they contain are:

- `NULL`
- `logical`
- `integer`
- `numeric`
- `complex`
- `character`

The `numeric` data type is also known as `double`. A function name associated with a member of the `is` family of functions typically consists of the concatenation of (a) the word `is`, (b) a period, and (c) the inquiry concerning the data type of the object. The first illustration asks whether 5 is numeric.

```
> is.numeric(5)              # is 5 numeric?
[1] TRUE
```

It is. Now for a trick question. Is 5 an integer?

```
> is.integer(5)              # is 5 an integer?
[1] FALSE
```

The answer is no. If you want 5 to be of the `integer` data type, it must be input as `5L`.

```
> is.integer(5L)             # is 5L an integer?
[1] TRUE
```

The number 5 as a numeric value and the number 5 as an integer look identical when displayed, but these two values are stored in a different fashion internally in R. The previous three commands could have been executed using the `is` function alone with the commands

```
> is(5, "numeric")           # is 5 numeric?
[1] TRUE
> is(5, "integer")           # is 5 an integer?
[1] FALSE
> is(5L, "integer")          # is 5L an integer?
[1] TRUE
```

which yield the same results.

The `is` family of functions includes those listed in the table below. As an illustration, to determine whether $5 + 12i$ is a complex number, use the `is.complex` function.

```
> is.complex(5 + 12i)        # is 5 + 12i complex?
[1] TRUE
```

It is. Next, check to see whether 5 is a character.

```
> is.character(5)            # is 5 character?
[1] FALSE
```

It is not. Although the functions in the `is` family of functions have been applied thus far to vectors of length one, they can all be applied to vectors, matrices, or arrays. The three data structures that have been introduced thus far—vectors, matrices, and arrays—are required to have elements of the same data type. Objects of this nature are known as *atomic*. The `is.atomic` function tests to see whether an object is indeed atomic.

function	description
`is.null`	is an object `NULL`?
`is.logical`	are the elements of an object of data type `logical`?
`is.integer`	are the elements of an object of data type `integer`?
`is.numeric`	are the elements of an object of data type `numeric`?
`is.double`	are the elements of an object of data type `double`?
`is.complex`	are the elements of an object of data type `complex`?
`is.character`	are the elements of an object of data type `character`?
`is.factor`	are the elements of an object of data type `factor`?
`is.na`	are the elements of an object of data type `NA`?
`is.function`	is an object a function?
`is.atomic`	is an object `atomic`?