

## 2.4 Logical vectors

As well as numerical vectors, R allows manipulation of logical quantities. The elements of a logical vector can have the values TRUE, FALSE, and NA (for “not available”, see below). The first two are often abbreviated as T and F, respectively. Note however that T and F are just variables which are set to TRUE and FALSE by default, but are not reserved words and hence can be overwritten by the user. Hence, you should always use TRUE and FALSE.

Logical vectors are generated by *conditions*. For example

```
> temp <- x > 13
```

sets `temp` as a vector of the same length as `x` with values FALSE corresponding to elements of `x` where the condition is *not* met and TRUE where it is.

The logical operators are `<`, `<=`, `>`, `>=`, `==` for exact equality and `!=` for inequality. In addition if `c1` and `c2` are logical expressions, then `c1 & c2` is their intersection (“*and*”), `c1 | c2` is their union (“*or*”), and `!c1` is the negation of `c1`.

Logical vectors may be used in ordinary arithmetic, in which case they are *coerced* into numeric vectors, FALSE becoming 0 and TRUE becoming 1. However there are situations where logical vectors and their coerced numeric counterparts are not equivalent, for example see the next subsection.

## 2.5 Missing values

In some cases the components of a vector may not be completely known. When an element or value is “not available” or a “missing value” in the statistical sense, a place within a vector may be reserved for it by assigning it the special value NA. In general any operation on an NA becomes an NA. The motivation for this rule is simply that if the specification of an operation is incomplete, the result cannot be known and hence is not available.

The function `is.na(x)` gives a logical vector of the same size as `x` with value TRUE if and only if the corresponding element in `x` is NA.

```
> z <- c(1:3,NA); ind <- is.na(z)
```

Notice that the logical expression `x == NA` is quite different from `is.na(x)` since NA is not really a value but a marker for a quantity that is not available. Thus `x == NA` is a vector of the same length as `x` *all* of whose values are NA as the logical expression itself is incomplete and hence undecidable.

Note that there is a second kind of “missing” values which are produced by numerical computation, the so-called *Not a Number*, NaN, values. Examples are

```
> 0/0
```

or

```
> Inf - Inf
```

which both give NaN since the result cannot be defined sensibly.

In summary, `is.na(xx)` is TRUE *both* for NA and NaN values. To differentiate these, `is.nan(xx)` is only TRUE for NaNs.

Missing values are sometimes printed as <NA> when character vectors are printed without quotes.

## 2.6 Character vectors

Character quantities and character vectors are used frequently in R, for example as plot labels. Where needed they are denoted by a sequence of characters delimited by the double quote character, e.g., "x-values", "New iteration results".

Character strings are entered using either matching double (") or single (') quotes, but are printed using double quotes (or sometimes without quotes). They use C-style escape sequences, using \ as the escape character, so \\ is entered and printed as \\, and inside double quotes " is entered as \". Other useful escape sequences are \n, newline, \t, tab and \b, backspace—see ?Quotes for a full list.

Character vectors may be concatenated into a vector by the `c()` function; examples of their use will emerge frequently.

The `paste()` function takes an arbitrary number of arguments and concatenates them one by one into character strings. Any numbers given among the arguments are coerced into character strings in the evident way, that is, in the same way they would be if they were printed. The arguments are by default separated in the result by a single blank character, but this can be changed by the named parameter, `sep=string`, which changes it to *string*, possibly empty.

For example

```
> labs <- paste(c("X","Y"), 1:10, sep="")
```

makes `labs` into the character vector

```
c("X1", "Y2", "X3", "Y4", "X5", "Y6", "X7", "Y8", "X9", "Y10")
```

Note particularly that recycling of short lists takes place here too; thus `c("X", "Y")` is repeated 5 times to match the sequence `1:10`.<sup>3</sup>

## 2.7 Index vectors; selecting and modifying subsets of a data set

Subsets of the elements of a vector may be selected by appending to the name of the vector an *index vector* in square brackets. More generally any expression that evaluates to a vector may have subsets of its elements similarly selected by appending an index vector in square brackets immediately after the expression.

Such index vectors can be any of four distinct types.

1. **A logical vector.** In this case the index vector must be of the same length as the vector from which elements are to be selected. Values corresponding to TRUE in the index vector are selected and those corresponding to FALSE are omitted. For example

```
> y <- x[!is.na(x)]
```

creates (or re-creates) an object `y` which will contain the non-missing values of `x`, in the same order. Note that if `x` has missing values, `y` will be shorter than `x`. Also

```
> (x+1)[(!is.na(x)) & x>0] -> z
```

creates an object `z` and places in it the values of the vector `x+1` for which the corresponding value in `x` was both non-missing and positive.

<sup>3</sup> `paste(..., collapse=ss)` joins the arguments into a single character string putting `ss` in between. There are more tools for character manipulation, see the help for `sub` and `substring`.

### 5.3 Index matrices

As well as an index vector in any subscript position, a matrix may be used with a single *index matrix* in order either to assign a vector of quantities to an irregular collection of elements in the array, or to extract an irregular collection as a vector.

A matrix example makes the process clear. In the case of a doubly indexed array, an index matrix may be given consisting of two columns and as many rows as desired. The entries in the index matrix are the row and column indices for the doubly indexed array. Suppose for example we have a 4 by 5 array *X* and we wish to do the following:

- Extract elements *X*[1,3], *X*[2,2] and *X*[3,1] as a vector structure, and
- Replace these entries in the array *X* by zeroes.

In this case we need a 3 by 2 subscript array, as in the following example.

```
> x <- array(1:20, dim=c(4,5)) # Generate a 4 by 5 array.
> x
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    9   13   17
[2,]    2    6   10   14   18
[3,]    3    7   11   15   19
[4,]    4    8   12   16   20
> i <- array(c(1:3,3:1), dim=c(3,2))
> i                                     # i is a 3 by 2 index array.
      [,1] [,2]
[1,]    1    3
[2,]    2    2
[3,]    3    1
> x[i]                                  # Extract those elements
[1] 9 6 3
> x[i] <- 0                              # Replace those elements by zeros.
> x
      [,1] [,2] [,3] [,4] [,5]
[1,]    1    5    0   13   17
[2,]    2    0   10   14   18
[3,]    0    7   11   15   19
[4,]    4    8   12   16   20
>
```

Negative indices are not allowed in index matrices. NA and zero values are allowed: rows in the index matrix containing a zero are ignored, and rows containing an NA produce an NA in the result.

As a less trivial example, suppose we wish to generate an (unreduced) design matrix for a block design defined by factors *blocks* (*b* levels) and *varieties* (*v* levels). Further suppose there are *n* plots in the experiment. We could proceed as follows:

```
> Xb <- matrix(0, n, b)
> Xv <- matrix(0, n, v)
> ib <- cbind(1:n, blocks)
> iv <- cbind(1:n, varieties)
> Xb[ib] <- 1
> Xv[iv] <- 1
> X <- cbind(Xb, Xv)
```

To construct the incidence matrix, *N* say, we could use

```
> N <- crossprod(Xb, Xv)
```