

*7. R Data Structures

7.1 Vectors

Recall that vectors may have mode logical, numeric or character.

7.1.1 Subsets of Vectors

Recall (section 2.6.2) two common ways to extract subsets of vectors:

Specify the numbers of the elements that are to be extracted. One can use negative numbers to omit elements.

Specify a vector of logical values. The elements that are extracted are those for which the logical value is T. Thus suppose we want to extract values of **x** that are greater than 10.

The following demonstrates a third possibility, for vectors that have named elements:

```
> c(Andreas=178, John=185, Jeff=183)[c("John", "Jeff")]
  John Jeff
  185  183
```

A vector of names has been used to extract the elements.

7.1.2 Patterned Data

Use 5:15 to generate the numbers 5, 6, ..., 15. Entering 15:5 will generate the sequence in the reverse order.

To repeat the sequence (2, 3, 5) four times over, enter `rep(c(2,3,5), 4)` thus:

```
> rep(c(2,3,5),4)
[1] 2 3 5 2 3 5 2 3 5 2 3 5 2 3 5
```

If instead one wants four 2s, then four 3s, then four 5s, enter `rep(c(2,3,5), c(4,4,4))`.

```
> rep(c(2,3,5),c(4,4,4)) # An alternative is rep(c(2,3,5), each=4)
[1] 2 2 2 2 3 3 3 3 5 5 5 5
```

Note further that, in place of `c(4,4,4)` we could write `rep(4,3)`. So a further possibility is that in place of `rep(c(2,3,5), c(4,4,4))` we could enter `rep(c(2,3,5), rep(4,3))`.

In addition to the above, note that the function `rep()` has an argument `length.out`, meaning "keep on repeating the sequence until the length is `length.out`."

7.2 Missing Values

In R, the missing value symbol is **NA**. Any arithmetic operation or relation that involves **NA** generates an **NA**. This applies also to the relations `<`, `<=`, `>`, `>=`, `==`, `!=`. The first four compare magnitudes, `==` tests for equality, and `!=` tests for inequality. Users who do not carefully consider implications for expressions that include **Nas** may be puzzled by the results. Specifically, note that `x==NA` generates **NA**.

Be sure to use `is.na(x)` to test which values of **x** are **NA**. As `x==NA` gives a vector of **NAs**, you get no information at all about **x**. For example

```
> x <- c(1,6,2,NA)
> is.na(x) # TRUE for when NA appears, and otherwise FALSE
[1] FALSE FALSE FALSE TRUE
> x==NA # All elements are set to NA
[1] NA NA NA NA
> NA==NA
[1] NA
```

WARNING: This is chiefly for those who may move between R and S-PLUS. In important respects, R's behaviour with missing values is more intuitive than that of S-PLUS. Thus in R

```
y[x>2] <- x[x>2]
```

gives the result that the naive user might expect, i.e. replace elements of **y** with corresponding elements of **x** wherever `x>2`. Wherever `x>2` gives the result **NA**, no action is taken. In R, any **NA** in `x>2` yields a value of **NA** for `y[x>2]` on the left of the equation, and a value of **NA** for `x[x>2]` on the right of the equation.

In S-PLUS, the result on the right is the same, i.e. an **NA**. However, on the left, elements that have a subscript **NA** drop out. The vector on the left to which values will be assigned has, as a result, fewer elements than the vector on the right.

Thus the following has the effect in R that the naive user might expect, but not in S-PLUS:

```
x <- c(1,6,2,NA,10)
y <- c(1,4,2,3,0)
y[x>2] <- x[x>2]
y
```

In S-PLUS it is essential to specify, in the example just considered:

```
y[!is.na(x)&x>2] <- x[!is.na(x)&x>2]
```

Here is a further example of R's behaviour:

```
> x <- c(1,6,2,NA,10)
> x>2
[1] FALSE TRUE FALSE NA TRUE
> x[x>3] <- c(21,22) # Now, explain the result that follows
Warning message:
number of items to replace is not a multiple of replacement length
> x
[1] 1 21 2 NA 21
```

The safe way, in both S-PLUS and R, is to use `!is.na(x)` to limit the selection, on one or both sides as necessary, to those elements of **x** that are not **NAs**. We will have more to say on missing values in the section on data frames that now follows.

7.3 Data frames

The concept of a data frame is fundamental to the use of most of the R modelling and graphics functions. A data frame is a generalisation of a matrix, in which different columns may have different modes. All elements of any column must however have the same mode, i.e. all numeric or all factor, or all character.

Data frames where all columns hold numeric data have some, but not all, of the properties of matrices. There are important differences that arise because data frames are implemented as lists. To turn a data frame of numeric data into a matrix of numeric data, use `as.matrix()`.

Lists are discussed below, in section 7.6.

7.3.1 Extraction of Component Parts of Data frames

Consider the data frame `barley` that accompanies the lattice package:

```
> names(barley)
[1] "yield" "variety" "year" "site"
> levels(barley$site)
[1] "Grand Rapids" "DuLuth" "University Farm" "Morris"
[5] "Crookston" "Waseca"
```

We will extract the data for 1932, at the `DuLuth` site.

```
> DuLuth1932 <- barley[barley$year=="1932" & barley$site=="DuLuth",
+ c("variety","yield")]
  variety yield
66   Manchuria 22.56667
72   Glabron 25.86667
78   Svansota 22.23333
84   Velvet 22.46667
90   Trebi 30.60000
```


Printing the contents of the column with the name `country` gives the names, not the integer values. As in most operations with factors, R does the translation invisibly. There are though annoying exceptions that can make the use of factors tricky. To be sure of getting the country names, specify

```
as.character(islandcities$country)
```

To get the integer values, specify

```
unclass(islandcities$country)
```

By default, R sorts the level names in alphabetical order. If we form a table that has the number of times that each country appears, this is the order that is used:

```
> table(islandcities$country)
Australia Cuba Indonesia Japan Philippines Taiwan United Kingdom
      3      1         4      6             2      1             2
```

This order of the level names is purely a convenience. We might prefer countries to appear in order of latitude, from North to South. We can change the order of the level names to reflect this desired order:

```
> lev <- levels(islandcities$country)
> lev[c(7,4,6,2,5,3,1)]
[1] "United Kingdom" "Japan"          "Taiwan"          "Cuba"
[5] "Philippines"     "Indonesia"      "Australia"
> country <- factor(islandcities$country, levels=lev[c(7,4,6,2,5,3,1)])
> table(country)
United Kingdom Japan Taiwan Cuba Philippines Indonesia Australia
      2          6      1      1             2          4          3
```

In ordered factors, i.e. factors with ordered levels, there are inequalities that relate factor levels.

Factors have the potential to cause a few surprises, so be careful! Here are two points to note:

When a vector of character strings becomes a column of a data frame, R by default turns it into a factor. Enclose the vector of character strings in the wrapper function `I()` if it is to remain character.

There are some contexts in which factors become numeric vectors. To be sure of getting the vector of text strings, specify e.g. `as.character(country)`.

To extract the numeric levels 1, 2, 3, ..., specify `as.numeric(country)`.

7.6 Ordered Factors

Actually, it is their levels that are ordered. To create an ordered factor, or to turn a factor into an ordered factor, use the function `ordered()`. The levels of an ordered factor are assumed to specify positions on an ordinal scale. Try

```
> stress.level <- rep(c("low", "medium", "high"), 2)
> ord.f.stress <- ordered(stress.level, levels=c("low", "medium", "high"))
> ord.f.stress
[1] low medium high low medium high
Levels: low < medium < high
> ord.f.stress < "medium"
[1] TRUE FALSE FALSE TRUE FALSE FALSE
> ord.f.stress >= "medium"
[1] FALSE TRUE TRUE FALSE TRUE TRUE
```

Later we will meet the notion of inheritance. Ordered factors inherit the attributes of factors, and have a further ordering attribute. When you ask for the class of an object, you get details both of the class of the object, and of any classes from which it inherits. Thus:

```
> class(ord.f.stress)
[1] "ordered" "factor"
```

7.7 Lists

Lists make it possible to collect an arbitrary set of R objects together under a single name. You might for example collect together vectors of several different modes and lengths, scalars, matrices or more general arrays, functions, etc. Lists can be, and often are, a rag-tag of different objects. We will use for illustration the list object that R creates as output from an `lm` calculation.

For example, consider the linear model (`lm`) object `elastic.lm` (c. f. sections 1.1.4 and 2.1.4) created thus:

```
elastic.lm <- lm(distance~stretch, data=elasticband)
```

It is readily verified that `elastic.lm` consists of a variety of different kinds of objects, stored as a list. You can get the names of these objects by typing in

```
> names(elastic.lm)
[1] "coefficients" "residuals" "effects" "rank"
[5] "fitted.values" "assign" "qr" "df.residual"
[9] "xlevels" "call" "terms" "model"
```

The first list element is:

```
> elastic.lm$coefficients
(Intercept) stretch
-63.571429 4.553571
```

Alternative ways to extract this first list element are:

```
elastic.lm[["coefficients"]]
elastic.lm[[1]]
```

We can alternatively ask for the sublist whose only element is the vector `elastic.lm$coefficients`. For this, specify `elastic.lm[["coefficients"]]` or `elastic.lm[[1]]`. There is a subtle difference in the result that is printed out. The information is preceded by `$coefficients`, meaning "list element with name `coefficients`".

```
> elastic.lm[[1]]
$coefficients
(Intercept) stretch
-63.571429 4.553571
```

The second list element is a vector of length 7

```
> options(digits=3)
> elastic.lm$residuals
      1      2      3      4      5      6      7
2.107 -0.321 18.000 1.893 -27.786 13.321 -7.214
```

The tenth list element documents the function call:

```
> elastic.lm$call
lm(formula = distance ~ stretch, data = elasticband)
> mode(elastic.lm$call)
[1] "call"
```

*7.8 Matrices and Arrays

All elements of a matrix have the same mode, i.e. all numeric, or all character. Thus a matrix is a more restricted structure than a data frame. One reason for numeric matrices is that they allow a variety of mathematical operations that are not available for data frames. Matrices are likely to be important for those users who wish to implement new regression and multivariate methods. The `matrix` construct generalises to `array`, which may have more than two dimensions.

Note that matrices are stored columnwise. Thus consider

```
> xx <- matrix(1:6, ncol=3) # Equivalently, enter matrix(1:6, nrow=2)
> xx
      [,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6
```

If `xx` is any matrix, the assignment

```
x <- as.vector(xx)
```

places columns of `xx`, in order, into the vector `x`. In the example above, we get back the elements 1, 2, ..., 6.

Matrices have the attribute “dimension”. Thus

```
> dim(xx)
[1] 2 3
```

In fact a matrix *is* a vector (numeric or character) whose dimension attribute has length 2.

Now set

```
> x34 <- matrix(1:12,ncol=4)
> x34
     [,1] [,2] [,3] [,4]
[1,]    1    4    7   10
[2,]    2    5    8   11
[3,]    3    6    9   12
```

Here are examples of the extraction of columns or rows or submatrices

```
x34[2:3,c(1,4)] # Extract rows 2 & 3 & columns 1 & 4
x34[2,] # Extract the second row
x34[-2,] # Extract all rows except the second
x34[-2,-3] # Extract the matrix obtained by omitting row 2 & column 3
```

The `dimnames()` function assigns and/or extracts matrix row and column names. The `dimnames()` function gives a list, in which the first list element is the vector of row names, and the second list element is the vector of column names. This generalises in the obvious way for use with arrays, which we now discuss.

7.8.1 Arrays

The generalisation from a matrix (2 dimensions) to allow > 2 dimensions gives an array. A matrix is a 2-dimensional array.

Consider a numeric vector of length 24. So that we can easily keep track of the elements, we will make them 1, 2, ..., 24. Thus

```
x <- 1:24
```

Then

```
dim(x) <- c(2,12)
```

turns this into a 2 x 12 matrix.

```
> x
     [,1] [,2] [,3] [,4] [,5] [,6] [,7] [,8] [,9] [,10] [,11] [,12]
[1,]    1    3    5    7    9   11   13   15   17   19   21   23
[2,]    2    4    6    8   10   12   14   16   18   20   22   24
```

Now try

```
> dim(x) <-c(3,4,2)
> x
```

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

```
, , 2
     [,1] [,2] [,3] [,4]
[1,]   13   16   19   22
[2,]   14   17   20   23
```

```
[3,]   15   18   21   24
```

7.8.2 Conversion of Numeric Data frames into Matrices

There are various manipulations that are available for matrices, but not for data frames. Use `as.matrix()` to handle any conversion that may be necessary.

7.9 Exercises

Generate the numbers 101, 102, ..., 112, and store the result in the vector `x`.

Generate four repeats of the sequence of numbers (4, 6, 3).

Generate the sequence consisting of eight 4s, then seven 6s, and finally nine 3s. Store the numbers obtained, in order, in the columns of a 6 by 4 matrix.

Create a vector consisting of one 1, then two 2's, three 3's, etc., and ending with nine 9's.

For each of the following calculations, what you would expect? Check to see if you were right!

- a)

```
answer <- c(2, 7, 1, 5, 12, 3, 4)
for (j in 2:length(answer)){ answer[j] <- max(answer[j],answer[j-1])}
```
- b)

```
answer <- c(2, 7, 1, 5, 12, 3, 4)
for (j in 2:length(answer)){ answer[j] <- sum(answer[j],answer[j-1])}
```

In the built-in data frame `airquality` (`datasets` package): (a) Determine, for each of the columns of the data frame `airquality` (`datasets` package), the median, mean, upper and lower quartiles, and range; (b) Extract the row or rows for which `Ozone` has its maximum value; (c) extract the vector of values of `Wind` for values of `Ozone` that are above the upper quartile.

Refer to the Eurasian snow data that is given in Exercise 1.6. Find the mean of the snow cover (a) for the odd-numbered years and (b) for the even-numbered years.

Determine which columns of the data frame `Cars93` (`MASS` package) are factors. For each of these factor columns, print out the levels vector. Which of these are ordered factors?

Use `summary()` to get information about data in the data frames `attitude` (both in the `datasets` package), and `cpus` (`MASS` package). Write brief notes, for each of these data sets, on what this reveals.

From the data frame `mtcars` (`MASS` package) extract a data frame `mtcars6` that holds only the information for cars with 6 cylinders.

From the data frame `Cars93` (`MASS` package), extract a data frame which holds only information for small and sporty cars.