*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(Andrea=178, John=188, Jeff=185)[c("John","Jeff")]
 John  Jeff
  188   185
```

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
>
```

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(1,8,2,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 naïve 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 naïve user might expect, but not in S-PLUS:

```
x <- c(1,8,2,NA,10)
y <- c(1,4,2,3,5)
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(1,8,2,NA,10)
> x>2
[1] FALSE  TRUE FALSE    NA  TRUE
> x[x>2] <- 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 generalization 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 correct 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 23.56667
78          Svansota 22.25333
84            Velvet 22.46667
90             Trebi 30.65000
```

```
96              No. 457 22.70000
102             No. 462 22.50000
108            Peatland 31.36667
114             No. 475 27.36667
120 Wisconsin No.  38 29.33333
```

The first column holds the row labels, which in this case are the numbers of the rows that have been extracted. In place of c("variety","yield") we could have written, more simply, c(2,4).

7.3.2 Data Sets that Accompany R Packages

Type in data() to get a list of data sets (mostly data frames) associated with all packages that are in the current search path. To get information on the data sets that are included in the *datasets* package, specify

```
data(package="datasets")
```

and similarly for any other package.

In versions of R previous to 2.0.0, it is usually necessary to specifically bring any of these data frames into the working directory. (Ensure though that the relevant package is attached.) Thus to bring in the data set airquality (*datasets* package), type

```
data(airquality)
```

The default Windows distribution includes many commonly required packages. Other packages must be explicitly installed. For remaining sections of these notes, the *MASS* package, which comes with the default distribution, will be used from time to time.

The *base* package, and several other packages, are automatically attached at the beginning of the session. To attach any other installed package, use the library() command.

7.4 Data Entry

The function read.table() offers a ready means to read a rectangular array into an R data frame. Suppose that the file primates.dat contains:

```
"Potar monkey"    10 115
Gorilla          207 406
Human             62 1320
"Rhesus monkey" 6.8 179
Chimp          52.2 440
```

Then

```
primates <- read.table("a:/primates.txt")
```

will create the data frame primates, from a file on the a: drive. The text strings in the first column will become the first column in the data frame.

Suppose that primates is a data frame with three columns – species name, body weight, and brain weight. You can give the columns names by typing in:

```
names(primates)<-c("Species","Bodywt","Brainwt")
```

Here then are the contents of the data frame.

```
> primates
         Species Bodywt Brainwt
1   Potar monkey   10.0     115
2        Gorilla  207.0     406
3          Human   62.0    1320
4  Rhesus monkey    6.8     179
5          Chimp   52.2     440
```

Specify header=TRUE if there is an initial how of header information. If the number of headers is one less than the number of columns of data, then the first column will be used, providing entries are unique, for row labels.

7.4.1 Idiosyncrasies

The function read.table() is straightforward for reading in rectangular arrays of data that are entirely numeric. When, as in the above example, one of the columns contains text strings, the column is by default stored as a factor with as many different levels as there are unique text strings[35].

Problems may arise when small mistakes in the data cause R to interpret a column of supposedly numeric data as character strings, which are automatically turned into factors. For example there may be an O (oh) somewhere where there should be a 0 (zero), or an el (l) where there should be a one (1). If you use any missing value symbols other than the default (NA), you need to make this explicit see section 7.3.2 below. Otherwise any appearance of such symbols as *, period(.) and blank (in a case where the separator is something other than a space) will cause to whole column to be treated as character data.

Users who find this default behaviour of read.table() confusing may wish to use the parameter setting as.is = TRUE.[36] If the column is later required for use as a factor in a model or graphics formula, it may be necessary to make it into a factor at that time. Some functions do this conversion automatically.

7.4.2 Missing values when using read.table()

The function read.table() expects missing values to be coded as NA, unless you set na.strings to recognize other characters as missing value indicators. If you have a text file that has been output from SAS, you will probably want to set na.strings=c(".").

There may be multiple missing value indicators, e.g. na.strings=c("NA",".","*",""). The "" will ensure that empty cells are entered as NAs.

7.4.3 Separators when using read.table()

With data from spreadsheets[37], it is sometimes necessary to use tab ("\t") or comma as the separator. The default separator is white space. To set tab as the separator, specify sep="\t".

7.5 Factors and Ordered Factors

We discussed factors in section 2.6.4. They provide an economical way to store vectors of character strings in which there are many multiple occurrences of the same strings. More crucially, they have a central role in the incorporation of qualitative effects into model and graphics formulae.

Factors have a dual identity. They are stored as integer vectors, with each of the values interpreted according to the information that is in the table of levels[38].

The data frame islandcities that accompanies these notes holds the populations of the 19 island nation cities with a 1995 urban centre population of 1.4 million or more. The row names are the city names, the first column (country) has the name of the country, and the second column (population) has the urban centre population, in millions. Here is a table that gives the number of times each country occurs

Australia	Cuba	Indonesia	Japan	Philippines	Taiwan	United Kingdom
3	1	4	6	2	1	2

[There are 19 cities in all.]

---

35 Storage of columns of character strings as factors is efficient when a small number of distinct strings that are of modest length are each repeated a large number of times.

36 Specifying as.is = T prevents columns of (intended or unintended) character strings from being converted into factors.

37 One way to get mixed text and numeric data across from Excel is to save the worksheet in a .csv text file with comma as the separator. If for example file name is myfile.csv and is on drive a:, use read.table("a:/myfile.csv", sep=",") to read the data into R. This copes with any spaces which may appear in text strings. [But watch that none of the cell entries include commas.]

38 Factors are vectors which have mode numeric and class "factor". They have an attribute levels that holds the level names.

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 
             3              1              4              6              2 
        Taiwan United Kingdom 
             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)
country
 United Kingdom           Japan          Taiwan            Cuba     Philippines 
              2              6              1              1              2 
      Indonesia       Australia 
              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)
> ordf.stress<-ordered(stress.level, levels=c("low","medium","high"))
> ordf.stress
[1] low    medium high   low    medium high  
Levels: low < medium < high
> ordf.stress<"medium"
[1]  TRUE FALSE FALSE  TRUE FALSE FALSE
> ordf.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(ordf.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 1.1.6 and 2.1.4) created thus:

```
elastic.lm <- lm(distance~stretch, data=elastic1)
```

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 
  -65.570429    4.555571 
```

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 
  -65.570429    4.555571 
```

The second list element is a vector of length 7

```
> options(digits=5)
> elastic.lm$residuals
      1       2       3       4       5       6       7 
  2.187  -0.332  18.000   1.893 -27.788  13.321  -7.214 
```

The tenth list element documents the function call:

```
> elastic.lm$call
lm(formula = distance ~ stretch, data = elastic1)
> mode(elastic.lm$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 some regression and multivariate methods. The `matrix` construct generalizes 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
```