2. Basic Data Types

Contents

We look at some of the ways that R can store and organize data. This is a basic introduction to a small subset of the different data types recognized by R and is not comprehensive in any sense. The main goal is to demonstrate the different kinds of information R can handle. It is assumed that you know how to enter data or read data files which is covered in the first chapter.

2.1. Variable Types

2.1.1. Numbers

The way to work with real numbers has already been covered in the first chapter and is briefly discussed here. The most basic way to store a number is to make an assignment of a single number:

```
> a <- 3
>
```

The "<-" tells R to take the number to the right of the symbol and store it in a variable whose name is given on the left. You can also use the "=" symbol. When you make an assignment R does not print out any information. If you want to see what value a variable has just type the name of the variable on a line and press the enter key:

```
> a
[1] 3
```

This allows you to do all sorts of basic operations and save the numbers:

```
> b <- sqrt(a*a+3)
> b
[1] 3.464102
```

If you want to get a list of the variables that you have defined in a particular session you can list them all using the ls command:

```
> ls()
[1] "a" "b"
```

You are not limited to just saving a single number. You can create a list (also called a "vector") using the c command:

```
> a <- c(1,2,3,4,5)
> a
[1] 1 2 3 4 5
> a+1
[1] 2 3 4 5 6
> mean(a)
[1] 3
> var(a)
[1] 2.5
```

You can get access to particular entries in the vector in the following manner:

```
> a <- c(1,2,3,4,5)
> a[1]
[1] 1
> a[2]
[1] 2
> a[0]
numeric(0)
> a[5]
[1] 5
> a[6]
[1] NA
```

Note that the zero entry is used to indicate how the data is stored. The first entry in the vector is the first number, and if you try to get a number past the last number you get "NA."

Examples of the sort of operations you can do on vectors is given in a next chapter.

To initialize a list of numbers the *numeric* command can be used. For example, to create a list of 10 numbers, initialized to zero, use the following command:

```
> a <- numeric(10)
> a
[1] 0 0 0 0 0 0 0 0 0 0
```

If you wish to determine the data type used for a variable the *type* command:

```
> typeof(a)
[1] "double"
```

2.1.2. Strings

You are not limited to just storing numbers. You can also store strings. A string is specified by using quotes. Both single and double quotes will work:

```
> a <- "hello"
> a
[1] "hello"
> b <- c("hello","there")
> b
[1] "hello" "there"
> b[1]
[1] "hello"
```

The name of the type given to strings is *character*,

```
> typeof(a)
[1] "character"
> a = character(20)
> a
[1] "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" "" ""
"" "" ""
```

2.1.3. Factors

Another important way R can store data is as a factor. Often times an experiment includes trials for different levels of some explanatory variable. For example, when looking at the impact of carbon dioxide on the growth rate of a tree you might try to observe how different trees grow when exposed to different preset concentrations of carbon dioxide. The different levels are also called factors.

Assuming you know how to read in a file, we will look at the data file given in the first chapter. Several of the variables in the file are factors:

```
> summary(tree$CHBR)
A1  A2  A3  A4  A5  A6  A7  B1  B2  B3  B4  B5  B6  B7
C1  C2  C3  C4  C5  C6
3   1   1   3   1   3   1   1   3   3   3   3   3   3
1   3   1   3   1   1
C7 CL6 CL7  D1  D2  D3  D4  D5  D6  D7
1   1   1   1   1   3   1   1   1   1
```

Because the set of options given in the data file corresponding to the “CHBR” column are not all numbers R automatically assumes that it is a factor. When you use summary on a factor it does not print out the five point summary, rather it prints out the possible values and the frequency that they occur.

In this data set several of the columns are factors, but the researchers used numbers to indicate the different levels. For example, the first column, labeled “C,” is a factor. Each trees was grown in an environment with one of four different possible levels of carbon dioxide. The researchers quite sensibly labeled these four environments as 1, 2, 3, and 4. Unfortunately, R cannot determine that these are factors and must assume that they are regular numbers.

This is a common problem and there is a way to tell R to treat the "C" column as a set of factors. You specify that a variable is a factor using the factor command. In the following example we convert tree$C into a factor:

```
> tree$C
[1]  1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
[39] 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4
> summary(tree$C)
   Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
  1.000   2.000   2.000   2.519   3.000   4.000
> tree$C <- factor(tree$C)
> tree$C
[1]  1 1 1 1 1 1 1 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 3 3 3 3 3 3
[39] 3 3 3 4 4 4 4 4 4 4 4 4 4 4 4 4
Levels: 1 2 3 4
> summary(tree$C)
1  2  3  4
8 23 10 13
> levels(tree$C)
[1] "1" "2" "3" "4"
```

Once a vector is converted into a set of factors then R treats it differently. A set of factors have a discrete set of possible values, and it does not make sense to try to find averages or other numerical descriptions. One thing that is important is the number of times that each factor appears, called their "frequencies," which is printed using the summary command.

2.1.4. Data Frames

Another way that information is stored is in data frames. This is a way to take many vectors of different types and store them in the same variable. The vectors can be of all different types. For example, a data frame may contain many lists, and each list might be a list of factors, strings, or numbers.

There are different ways to create and manipulate data frames. Most are beyond the scope of this introduction. They are only mentioned here to offer a more complete description. Please see the first chapter for more information on data frames.

One example of how to create a data frame is given below:

```
> a <- c(1,2,3,4)
> b <- c(2,4,6,8)
> levels <- factor(c("A","B","A","B"))
> bubba <- data.frame(first=a,
                      second=b,
                      f=levels)
> bubba
first second f
1     1      2 A
2     2      4 B
3     3      6 A
4     4      8 B
> summary(bubba)
    first          second    f
Min.   :1.00   Min.   :2.0   A:2
1st Qu.:1.75   1st Qu.:3.5   B:2
Median :2.50   Median :5.0
Mean   :2.50   Mean   :5.0
3rd Qu.:3.25   3rd Qu.:6.5
Max.   :4.00   Max.   :8.0
> bubba$first
[1] 1 2 3 4
> bubba$second
[1] 2 4 6 8
> bubba$f
[1] A B A B
Levels: A B
```

2.1.5. Logical

Another important data type is the logical type. There are two predefined variables, *TRUE* and *FALSE*:

```
> a = TRUE
> typeof(a)
[1] "logical"
> b = FALSE
> typeof(b)
[1] "logical"
```

The standard logical operators can be used:

<	less than
>	great than
<=	less than or equal
>=	greater than or equal

==	equal to
!=	not equal to
|	entry wise or
||	or
!	not
&	entry wise and
&&	and
xor(a,b)	exclusive or

Note that there is a difference between operators that act on entries within a vector and the whole vector:

```
> a = c(TRUE,FALSE)
> b = c(FALSE,FALSE)
> a|b
[1]  TRUE FALSE
> a||b
[1] TRUE
> xor(a,b)
[1]  TRUE FALSE
```

There are a large number of functions that test to determine the type of a variable. For example the *is.numeric* function can determine if a variable is numeric:

```
> a = c(1,2,3)
> is.numeric(a)
[1] TRUE
> is.factor(a)
[1] FALSE
```

2.2. Tables

Another common way to store information is in a table. Here we look at how to define both one way and two way tables. We only look at how to create and define tables; the functions used in the analysis of proportions are examined in another chapter.

2.2.1. One Way Tables

The first example is for a one way table. One way tables are not the most interesting example, but it is a good place to start. One way to create a table is using the table command. The arguments it takes is a vector of factors, and it calculates the frequency that each factor occurs. Here is an example of how to create a one way table:

```
> a <- factor(c("A","A","B","A","B","B","C","A","C"))
> results <- table(a)
> results
a
A B C
4 3 2
> attributes(results)
$dim
[1] 3

$dimnames
$dimnames$a
[1] "A" "B" "C"


$class
[1] "table"

> summary(results)
Number of cases in table: 9
Number of factors: 1
```

If you know the number of occurrences for each factor then it is possible to create the table directly, but the process is, unfortunately, a bit more convoluted. There is an easier way to define one-way tables (a table with one row), but it does not extend easily to two-way tables (tables with more than one row). You must first create a matrix of numbers. A matrix is like a vector in that it is a list of numbers, but it is different in that you can have both rows and columns of numbers. For example, in our example above the number of occurrences of “A” is 4, the number of occurrences of “B” is 3, and the number of occurrences of “C” is 2. We will create one row of numbers. The first column contains a 4, the second column contains a 3, and the third column contains a 2:

```
> occur <- matrix(c(4,3,2),ncol=3,byrow=TRUE)
> occur
     [,1] [,2] [,3]
[1,]    4    3    2
```

At this point the variable “occur” is a matrix with one row and three columns of numbers. To dress it up and use it as a table we would like to give it labels for each columns just like in the previous example. Once that is done we convert the matrix to a table using the as.table command:

```
> colnames(occur) <- c("A","B","C")
> occur
     A B C
[1,] 4 3 2
> occur <- as.table(occur)
> occur
  A B C
A 4 3 2
> attributes(occur)
$dim
[1] 1 3

$dimnames
$dimnames[[1]]
[1] "A"

$dimnames[[2]]
[1] "A" "B" "C"


$class
[1] "table"
```

2.2.2. Two Way Tables

If you want to add rows to your table just add another vector to the argument of the table command. In the example below we have two questions. In the first question the responses are labeled “Never,” “Sometimes,” or “Always.” In the second question the responses are labeled “Yes,” “No,” or “Maybe.” The set of vectors “a,” and “b,” contain the response for each measurement. The third item in “a” is how the third person responded to the first question, and the third item in “b” is how the third person responded to the second question.

```
> a <-
c("Sometimes","Sometimes","Never","Always","Always","Som

> b <-
c("Maybe","Maybe","Yes","Maybe","Maybe","No","Yes","No")

> results <- table(a,b)
> results
           b
a           Maybe No Yes
  Always        2  0   0
  Never         0  1   1
  Sometimes     2  1   1
```

The table command allows us to do a very quick calculation, and we can immediately see that two people who said “Maybe” to the first question also said “Sometimes” to the second question.

Just as in the case with one-way tables it is possible to manually enter two way tables. The procedure is exactly the same as above except that we now have more than one row. We give a brief example below to demonstrate how to enter a two-way table that includes breakdown of a group of people by both their gender and whether or not they smoke. You enter all of the data as one long list but tell R to break it up into some number of columns:

```
> sexsmoke<-matrix(c(70,120,65,140),ncol=2,byrow=TRUE)
> rownames(sexsmoke)<-c("male","female")
> colnames(sexsmoke)<-c("smoke","nosmoke")
> sexsmoke <- as.table(sexsmoke)
> sexsmoke
       smoke nosmoke
male      70     120
female    65     140
```

The matrix command creates a two by two matrix. The *byrow=TRUE* option indicates that the numbers are filled in across the rows first, and the *ncols=2* indicates that there are two columns.

Previous Next