R11

Vectors

Covered in R11

- Creating vectors
- Naming vectors
- Indexing vectors
- Vector arithmetic

1 Creating vectors

In the last chapter we briefly described the six types of objects (called classes) that R uses to store data. In this chapter we look at the most fundamental of these which is the vector.

A vector is a one-dimensional ordered collection of data of the *same type* (numeric, character, logical, complex or raw). Vectors are also called atomic vectors as there is no object more basic than this - even unassigned values (*eg* 5) are considered vectors with a single element.

You can find out whether an object is a vector by using is.vector(<object>). The dimension is called length and can be found by using the length(<vector object>) command.

The simplest way to create a vector is to use the c( ) function. The c stands for "concatenate" which means "combine" or "join together". Suppose we want to make a numeric vector, called v, containing the numbers 1 to 10. We could do this as follows:

```
v <- c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

Typing in the object name returns the following:

```
> v
 [1]  1  2  3  4  5  6  7  8  9 10
> 
```

Even though a vector is vertical, we can see that R displays vectors horizontally, which is why it measures the number of elements by length.

We can see that v is a vector of 10 elements:

```
> is.vector(v)
[1] TRUE
> length(v)
[1] 10
> 
```

But when we look at its class we obtain the following:

```
> class(v)
[1] "numeric"
> 
```

Rather than saying vector it gives the type of data it holds, as a vector is the default object and is defined by its elements.

Another way of obtaining all this information in one go is to use the str(<object>) command which displays the structure of an R object:

```
> str(v)
 num [1:10] 1 2 3 4 5 6 7 8 9 10
>
```

It says it is numeric data, gives the dimensions and the contents (in this case all of the contents, but for larger objects it will only give some of them).

We can even create vectors from other vectors. For example:

```
> w <- c(v, 0, v)
> w
 [1]  1  2  3  4  5  6  7  8  9 10  0  1  2  3  4  5  6  7  8  9 10
>
```

Note that because of the length of this vector, when we look at its structure it won't display all of its contents:

```
> str(w)
 num [1:21] 1 2 3 4 5 6 7 8 9 10 ...
>
```

Since a vector is a collection of data of the same type, if we try to create a vector of different data types R will try to coerce them all to the same type. For example:

```
> x <- c(3, TRUE, 5, FALSE)
> x
[1] 3 1 5 0
>
```

In this case it has converted the logical data into numeric data (TRUE becomes 1 and FALSE becomes 0) so that it is now a vector of numeric data .

Alternative methods of creating numeric vectors

If we want to create a numeric vector containing consecutive *integers* between a and b inclusive we could use the shorthand a:b. For example, either of the following would create a vector object containing the values 1, 2, 3, …., 10:

1:10

c(1:10)

If we want some other sequence of numbers we could use the sequence generation command:

seq(from, to, by, length)

- "from" is the value the sequence starts at. The default value is 1.
- "to" is the value it finishes at – if only one argument is given it will assume this is the "to" and the "from" argument will take the default value of 1
- "by" is an optional argument which gives the steps the sequence increases by, its default is ±1 unless the length option is used
- "length" is an optional argument that gives the required length of the sequence.

The following examples illustrate the use of this function:

```
> seq(10)
 [1]  1  2  3  4  5  6  7  8  9 10
> seq(5,10)
[1]  5  6  7  8  9 10
> seq(10,5)
[1] 10  9  8  7  6  5
> seq(1,10,2)
[1] 1 3 5 7 9
> seq(1,10,length=4)
[1]  1  4  7 10
>
```

Finally, we could use the many functions that produce simulations from common distributions such as runif(n) which returns n values from the $U(0,1)$ or rnorm(n) which returns n values from the $N(0,1)$ distribution:

```
> u <- runif(10)
> u
 [1] 0.03331517 0.05896850 0.81361991 0.57031271 0.33215259 0.50090018
 [7] 0.76868111 0.54902522 0.72675780 0.53099261
> n <- rnorm(10)
> n
 [1]  1.0038309 -1.1841480  1.1389673 -1.7390917  1.6105370 -0.2706324
 [7] -0.0143755  0.2465161  0.6192360  1.6310322
>
```

2 Naming vectors

It may be the case that we wish to name the elements in our vectors. For example suppose we want to create a vector, age, which contains the ages (34, 28, 47) of three policyholders (bob, larry and ginger). If wish to keep the names of the policyholders associated with their ages in the vector we could do this as follows:

```
> age <- c(bob=34,larry=28,ginger=47)
> age
   bob  larry ginger
    34     28     47
> 
```

We can see that when the vector is displayed it also includes the names. Similarly the names are given if we use the structure command:

```
> str(age)
 Named num [1:3] 34 28 47
 - attr(*, "names")= chr [1:3] "bob" "larry" "ginger"
> 
```

It says that it is a named numeric vector with 3 elements (34, 28, 47). Then it gives the attributes of the vector (*ie* its names attribute) which are the character data ("bob", "larry", "ginger").

An alternative way of performing this action would be to use the names function. Suppose we have another vector, claim.free, which gives the number of claim free years (3, 0, 8) for the three policyholders (bob, larry and ginger). We can assign the names to the vector claim.free as follows:

```
> claim.free <- c(3,0,8)
> names(claim.free) <- c("bob","larry","ginger")
> claim.free
   bob  larry ginger
     3      0      8
```

Note that you don't have to name every element in a vector, you could use "" for those elements to which you wish to give no name.

3 Indexing vectors

Sometimes we may be interested in only some of the elements in a vector. To do that we use indexing.

Recall that in a previous chapter we said that the [1] at the start of the output line referred to the index value of the first answer. The square brackets tell R it's an index and the number gives the position of the element.

Earlier we defined the vector v to contain the numbers 1, 2, 3, …., 10. So to select the third element we type v[3]:

```
> v[3]
[1] 3
>
```

Suppose I want to select the second and fifth elements of the vector v. If I try v[2,5] we get the following:

```
> v[2,5]
Error in v[2, 5] : incorrect number of dimensions
>
```

That's because with indexing it thinks we are referring to the second row and the fifth column. Since a vector only has one dimension it is very confused. So to specify both elements in just one dimension (*ie* both values are rows) we need to use the combine function as follows v[c(2,5)]:

```
> v[c(2,5)]
[1] 2 5
>
```

The following show clever ways of specifying the third to seventh elements and all the elements from the fourth value to the end:

```
> v[3:7]
[1] 3 4 5 6 7
> v[4:length(v)]
[1]  4  5  6  7  8  9 10
>
```


To specify all elements *except* the specified ones we use a negative in front of their positions:

```
> v[-3]
[1]  1  2  4  5  6  7  8  9 10
> v[-c(2,5)]
[1]  1  3  4  6  7  8  9 10
> v[-(3:7)]
[1]  1  2  8  9 10
> v[-(4:length(v))]
[1] 1 2 3
> 
```

We can use the results of logical tests to select elements. For example we could select all the values of v whose values are between 4 and 8 inclusive as follows:

```
> v[v>=4 & v<=8]
[1] 4 5 6 7 8
> 
```

We can see that all the elements for which the test is TRUE are selected.

Finally, if we have a named vector then we can select the elements using their names. For example, using our vector of policyholders' ages we could just select Larry's age as follows:

```
> age
   bob  larry ginger
    34     28     47
> age["larry"]
larry
   28
> 
```

4 Vector arithmetic

The standard arithmetic operations work on vectors but on an element by element basis:

```
> 2*v
 [1]  2  4  6  8 10 12 14 16 18 20
> v/2
 [1] 0.5 1.0 1.5 2.0 2.5 3.0 3.5 4.0 4.5 5.0
> v^2
 [1]   1   4   9  16  25  36  49  64  81 100
> log(v)
 [1] 0.0000000 0.6931472 1.0986123 1.3862944 1.6094379 1.7917595 1.9459101
 [8] 2.0794415 2.1972246 2.3025851
>
```

Whilst this is not quite the same as how vectors work in mathematics, it provides a powerful way of performing the same operation on many values at once.

Suppose we define two vectors v1 and v2 as follows:

```
> v1 <- 1:4
> v1
[1] 1 2 3 4
> v2 <- 4:1
> v2
[1] 4 3 2 1
>
```

We can see from the following examples that vectors operate on other vectors also on an element by element basis:

```
> v1+v2
[1] 5 5 5 5
> v1-v2
[1] -3 -1  1  3
> v1*v2
[1] 4 6 6 4
> v1/v2
[1] 0.2500000 0.6666667 1.5000000 4.0000000
> v1^v2
[1] 1 8 9 4
>
```

Let's have a look at what happens if vectors are of different lengths by first defining vectors v3 and v4 as the values (1,2) and (1, 2, 3), respectively:

```
> v3 <- 1:2
> v3
[1] 1 2
> v4 <- 1:3
> v4
[1] 1 2 3
>
```

Then when we add vectors v1 and v3 together we get the following:

```
> v1
[1] 1 2 3 4
> v3
[1] 1 2
> v1+v3
[1] 2 4 4 6
>
```

We can see that the shorter vector v3 has been extended by repetition to (1, 2, 1, 2). This is called *recycling*. This works fine as the length of v1 is a multiple of v3.

Let's look at what happens when this is not the case by adding vectors v1 and v4 together:

```
> v1
[1] 1 2 3 4
> v4
[1] 1 2 3
> v1+v4
[1] 2 4 6 5
Warning message:
In v1 + v4 :
  longer object length is not a multiple of shorter object length
>
```

We can see that the shorter vector v4 has been extended by repetition to (1, 2, 3, 1) and a warning message is displayed.

This process of recycling the shorter vector explains why v1 + 3 returns the following:

```
> v1
[1] 1 2 3 4
> v1+3
[1] 4 5 6 7
>
```

Recall that the default object is a vector. So 3 is treated as a vector of length 1. This is recycled to (3, 3, 3, 3) and added to vector v1.

5 Summary

Key terms

Vector	The default object type is a vector which is a one-dimensional ordered collection of data of the *same type* – it has one dimension which is called length
Length	The number of elements in a vector – called length as vectors are displayed horizontally even though they are actually vertical
Concatenate	Join together
Indexing	The process of selecting an element from a vector object
Recycling	The process of extending a smaller vector by repetition to make sense of operations with other vectors – returns a warning if the length of the smaller vector is not a multiple of the length of the larger vector

Key commands

is.vector(<object>)	Logical test of whether <object> is a vector Returns TRUE or FALSE
length(<vector object>)	Gives the number of elements in a vector
class(<object>)	Gives the class of an object. Since atomic vectors are defined by their data type it gives their data type instead
c(<element1>, …)	Combines <element1>, … together into a single object.
str(<object>)	Displays the structure of the <object>
a:b	Returns the integers a, a+1, a+2, …, b
seq(from, to, by, length)	Returns a sequence starting at “from”, finishing at “to”, either increasing in steps of “by” (default ±1) or equally spaced so that there are “length” length elements in the vector
runif(n)	Returns *n* simulated values from a U(0,1) distribution
rnorm(n)	Returns *n* simulated values from a N(0,1) distribution
names(<vector object>)	Names the elements in the <vector object>
<object>[<position>]	Returns the element from <object> at the given <position>

6 Have a go

You will only get proficient at R by practising.

1. Create a vector, *v*, containing the 10 numbers 21, 22, …, 30 using 3 methods:

   Using the c function

   Using :

   Using the seq function.

2. What type of vector (numeric, logical, character) would be formed from each of the following:

   c(3, 2, FALSE)

   c(TRUE, “bob”, FALSE)

   c(“larry”, 7, 2)

   c(5, TRUE, “ginger”).

   Check your answers using the is.numeric, is.logical, is.character functions.

3. Use the seq function to generate the following:

   70, 71, …., 85

   50, 47, 44, …., 14

   7 values equally spread between 1 and 31 inclusive.

4. Create a named vector of the temperatures 18, 20, 15 for the cities London, Paris and Stockholm using the c function.

   Create a named vector for the indices 6125.7, 17140.20 and 15323.10 for the FTSE 100, Dow Jones and Nikkei 225 using the names function.

5. Use indexing on the vector *v* from Q1 to display:

   3rd element

   4th and 7th elements

   6th-10th elements

   all but the 5th element

   all but the 2nd and 8th elements

   all but the 3rd to 6th elements

   all elements which are smaller than 27.

6. Create a vector *a* of (1, 2, 3, 4, 5, 6), a vector *b* of (0, 1) and a vector *c* of (5, 1, 3, 2).

   What will be the result of:

   b-1

   *b***c*

   a+*b*

   a^*b*

   a/*c*

7. Create a vector *n* containing 1,000 simulated $N(0,1)$ values.

   Use indexing to obtain all the values which are greater than 2 and store this in *m*.

   Use vectors *m* and *n* to obtain an empirical estimate of $P(Z > 2)$ (*ie* the probability that $Z > 2$).

   (Hint: use the lengths of the vectors).

