Roith

R Tutorial: Creating Data Objects in R

1. **Creating Data in R:**
R is a *functional* language as most of the operations in R are carried out through the use of *functions*: either internal R functions or user-constructed functions. The function c() is an example of a simple R function. It is used to create *vector* functions as follows:

```
> y <- c(5.0, 7.1, 13.0)
> y
[1]  5.0  7.1 13.0
> regions <- c("Northeast", "Midwest", "Southeast", "Northwest", "Southwest")
> regions
[1] "Northeast" "Midwest"   "Southeast" "Northwest" "Southwest"
```

When the number of values gets larger, the scan() function is more useful. You may input data values separated by blanks at the keyboard using scan(). In this form, scan() prompts the user to enter data with a varying number of values to a line until an entire blank line is entered to signal the end of data entry:

```
> z <- scan()
1: 3.7 6.5 1.9 8.4 10
6: 6.2 15 11.4
9: 13.45
10:
Read 9 items
> z
[1]  3.70  6.50  1.90  8.40 10.00  6.20 15.00 11.40 13.45
```

In the following example, character strings with embedded spaces are to be entered as data values, and hence a *separator* different from a blank space is required.

```
> actors <- scan(what="", sep="#")
1: Jimmy Stewart#Robert Redford#Matt Damon
4: Clark Gable#John Wayne
6: Marlon Brando#Jack Nicholson
8:
> actors
[1] "Jimmy Stewart"  "Robert Redford" "Matt Damon"     "Clark Gable"
[5] "John Wayne"     "Marlon Brando"  "Jack Nicholson"
```

Note that the value “”, specified as the value of the argument `what=`, indicates that these values are to be read as character strings. Alternatively, the *type* may be specified as `what="character"`. By default, the *type* of data values is assumed to be numeric (stored as a double). Finally, `sep="#"` specifies that the alternative *separator* to be used is the # symbol.

2. **Reading Data from Text Files:**
Most text data files are in "ASCII Format", and are usually produced by a text editor such as Notepad or Emacs.

Since the text file *blood.txt* is in the current working directory, it can be accessed within R simply as shown in the example below. Otherwise, it is necessary to enter the *complete pathname* of the location of the text file within the double quotes. Additionally, when entering the complete pathname you will need to change the folder separators from a single backslash, \ to either a double backslash, \\ or a forward slash, /. scan() reads the data from the text file row-wise and creates a vector object.

```
> blood0<-scan("U:\\Math 2994\\blood.txt")
Read 20 items
> blood0
 [1]  1 62  1 60  1 63  1 59  2 63  2 67  2 71  2 64  2 65  2 66
> blood1 <- scan("blood.txt")
Read 20 items
> blood1
 [1]  1 62  1 60  1 63  1 59  2 63  2 67  2 71  2 64  2 65  2 66
```

This data set consists of values for two *diets*, 1 and 2, randomly assigned to 10 mice, and *blood coagulation times* measured for each animal. Two separate vector objects can be created to represent them using the scan() function in the following way:

```
> blood2 <- scan("blood.txt", what=list(diet=0, time=0))
Read 10 records
> blood2
$diet
 [1] 1 1 1 1 2 2 2 2 2 2
$time
 [1] 62 60 63 59 63 67 71 64 65 66
```

The `what=list(diet=0, time=0)` argument to scan() specifies that two vectors named `diet` and `time` will be created as components of a *list* object. `diet=0, time=0` indicates that values for these vectors are to be read as numeric values. It is important that the data be alternating between diet and time in the original file for this to work. `blood2` is an example of a *list* object that will be discussed in detail later. The two vectors named `diet` and `time` are components of the list `blood2` and may be referenced as `blood2$diet` and `blood2$time`, thus:

```
> blood2$diet
```

```
 [1] 1 1 1 1 2 2 2 2 2 2
> blood2$time
 [1] 62 60 63 59 63 67 71 64 65 66
```

Typically, scan() is used to feed data into another function to create an object that has a desired structure. For example, the data in the text named *insulin.txt* available in the current working directory can be used to create a *matrix* object. In the following example, the scan() function is used directly as an argument to the matrix() function, to create the matrix object named insulin1.

```
> insulin1 <- matrix(scan("insulin.txt"), ncol=3, byrow=T)
Read 24 items
> insulin1
     [,1] [,2] [,3]
[1,] 2.02 1.49 0.33
[2,] 3.83 2.67 1.67
[3,] 6.67 4.62 4.67
[4,] 5.38 4.18 2.45
[5,] 5.49 2.78 2.29
[6,] 3.50 2.56 1.95
[7,] 5.90 4.46 0.49
[8,] 4.89 3.79 1.81
```

A *data frame* is an R object very similar in appearance to a matrix object except that columns in a data frame can be of different types: *numeric* or *character*. This allows standard rectangular data sets to be represented as a data frame. Data frames are a required form of input data for many statistical functions in R used for data analysis and modeling. The read.table() function allows us to convert data available in an external text file directly into a data frame. In the following example, we use the text file *insulin.txt* to create a data frame in R.

```
> insulin2 <- read.table("insulin.txt")
> insulin2
    V1   V2   V3
1 2.02 1.49 0.33
2 3.83 2.67 1.67
3 6.67 4.62 4.67
4 5.38 4.18 2.45
5 5.49 2.78 2.29
6 3.50 2.56 1.95
7 5.90 4.46 0.49
8 4.89 3.79 1.81
```

An object *type* can be checked with the class() function. An object can be changed typically with the `as.####` prefix. For example, a matrix can be converted to a data frame with as.data.frame() and vice versa with as.matrix().

```
> class(insulin1)
[1] "matrix"
> class(insulin2)
[1] "data.frame"
> insulin3 <- as.data.frame(insulin2)
> class(insulin3)
[1] "data.frame"
```

In the absence of user-specified names for the columns (variables) and the rows (observations), by default R assigns the names V1, V2, etc. for the columns and the names 1, 2, etc. for the rows. The user can assign names by including them in the text file along with the data itself or by using the keyword parameters `col.names` and/or `row.names` when using the `read.table` command, as illustrated below:

```
> insulin2 <- read.table("insulin.txt", col.names=c("Week1", "Week2", "Week3"))
> insulin2
  Week1 Week2 Week3
1  2.02  1.49  0.33
2  3.83  2.67  1.67
3  6.67  4.62  4.67
4  5.38  4.18  2.45
5  5.49  2.78  2.29
6  3.50  2.56  1.95
7  5.90  4.46  0.49
8  4.89  3.79  1.81
```

If the data values were separated by another character, such as a comma, then a different command such as

```
> insulin2 <- read.table("insulin.txt”, sep=”,”)
```

can be used.

As observed before, this is an option if one or more of the variables were of character type that included spaces, because blanks cannot be used as separators in this case.

A number of built-in datasets are supplied with R. Sample datasets are also available with other packages that may be already installed in R. To see a list of the datasets available with the base system, use the command data(). To load one of the datasets listed into R, use the same function. For example, data(cars) or data(DDT, package=”MASS”):

```
> data(cars)
> cars
  speed dist
1     4    2
2     4   10
3     7    4
4     7   22
…
> data(DDT, package="MASS")
> DDT
 [1] 2.79 2.93 3.22 3.78 3.22 3.38 3.18 3.33 3.34 3.06 3.07 3.56 3.08 4.64 3.34
```

cars is an R data frame object. Attaching the data frame using the attach(cars) command, allows the columns in the data frame to be accessed by simply giving their names. Thus,

```
> attach(cars)
> speed
 [1]  4  4  7  7  8  9 10 10 10 11 11 12 12 12 12 13 13 13 13 14 14 14 14 15 15
[26] 15 16 16 17 17 17 18 18 18 18 19 19 19 20 20 20 20 20 22 23 24 24 24 24 25
```

3. Generating Data in R:

The simplest operator for generating data vectors in R is the colon : called the sequence operator. For example,

```
> index <- 1:10
> index
 [1]  1  2  3  4  5  6  7  8  9 10
> j <- 4:-5
> j
 [1]  4  3  2  1  0 -1 -2 -3 -4 -5
```

This operator generates vectors of numbers incremented by either 1 or -1. You may use the seq() function to generate sequences incremented by specific constants:

```
> seq(0, 1, by=0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(0, 1, 0.1)
 [1] 0.0 0.1 0.2 0.3 0.4 0.5 0.6 0.7 0.8 0.9 1.0
> seq(0, 1, length = 10)
 [1] 0.0000000 0.1111111 0.2222222 0.3333333 0.4444444 0.5555556 0.6666667
 [8] 0.7777778 0.8888889 1.0000000
```

The arguments to an R function may be specified as *named* arguments i.e., in the form `name=value` or just by specifying the values if they are provided in the same order as given in the function specifications. For seq() function, it is recommended that the third parameter always be named. The rep() function is another function for generating data in R. In the simplest use, this function repeats values in the first argument a number of times equal to the second argument. Some examples are:

```
> rep(3,5)
[1] 3 3 3 3 3
> rep(1:5,3)
 [1] 1 2 3 4 5 1 2 3 4 5 1 2 3 4 5
> rep(1:5, each=2)
 [1] 1 1 2 2 3 3 4 4 5 5
> rep(1:5, rep(2,5))
 [1] 1 1 2 2 3 3 4 4 5 5
> rep(1:5, each=2, times=3)
 [1] 1 1 2 2 3 3 4 4 5 5 1 1 2 2 3 3 4 4 5 5 1 1 2 2 3 3 4 4 5 5
```

4. **Generating Random Data in R:**
Random samples from standard distributions are used in statistics for large scale simulations using Monte Carlo methods. They can be generated using the functions such as runif(), rnorm(), rgamma(), rbinom(), rexp(), and rt(). As you may observe, the function name is made by combining the letter “r” with a mnemonic name identifying the target distribution.

```
> runif(10)
 [1] 0.5062525 0.9476340 0.5244933 0.7257768 0.1029997 0.5964180 0.5610174
 [8] 0.0187926 0.6000028 0.1837264
> rnorm(5, mean=0, sd=1)
[1]  1.022534164 -0.019355635 -0.002194038  1.456263200 -0.115772189
> rnorm(5, mean=5, sd=2)
[1] 8.075794 5.989680 5.403330 2.847633 2.149664
> rt(8, 10)
[1]  0.1915325 -0.4911174  0.5481695 -0.9318543  0.6760705 -0.5001691
[7]  0.2293260 -1.5575078
> rbinom(20, 5, 0.5)
 [1] 3 5 2 4 3 1 3 4 3 3 3 2 4 4 2 3 0 2 3 1
> stem(rnorm(100,5,2))

  The decimal point is at the |

  -0 | 0
   0 | 0355682259
   2 | 3555667888902333348888899
   4 | 00122223566677778990112223457788899
   6 | 000114445580111233378
   8 | 35567068
  10 | 2
```