CSC 120: Computer Science for the Sciences (R section)

Radford M. Neal, University of Toronto, 2015

http://www.cs.utoronto.ca/∼radford/csc120/

Week 8

Names and NA — Two R Features for Statistics

R is a general purpose programming language. You can write all sorts of programs in R — video games, accounting packages, word processors, programs for navigating rocket ships to Mars, ...

But R is more appropriate for some of these tasks than others. It's probably not the best choice for video game programming — a game may need to respond very quickly to user input, but speed is not R's strong point. On the other hand, some features of R that are not common in other languages are especially useful for statistical applications.

Here are two:

- Special `NA` values to indicate where data is missing
- Names for elements of vectors and lists, and for rows and columns of matrices and data frames.

Indicating Missing Values with NA

It is very common for data collected to have some missing values — where the subject declined to answer one of the survey questions, or the interviewer forgot to fill out one page of the form, or where the machine taking the readings was broken that day.

Sometimes these values are indicated by some special number like −999. But this is very unreliable. The person analysing the data may not realize that this is what −999 is supposed to mean, leading to drastically incorrect averages. Or there may be an actual, non-missing, value of −999!

R supports representation of missing data by a special NA value. NA can be the value of an element in a vector, matrix, or data frame. For example:

```
> c(5,1,NA,8,NA)
[1]  5  1 NA  8 NA
```

Arithmetic on NA values

Arithmetic operations where one or both operands are NA produce NA as the result:

```
> a <- c(5,1,NA,8,NA)
> a+100
[1] 105 101  NA 108  NA
> b <- c(10,NA,20,NA,NA)
> a*b
[1] 50 NA NA NA NA
```

Comparisons with NA also produce NA, rather than TRUE or FALSE. Trying to use NA as an `if` or `while` condition gives an error:

```
> a == 1
[1] FALSE  TRUE    NA FALSE    NA
> if (a[3]==1) cat("true\n") else cat("false\n")
Error in if (a[3] == 1) cat("true\n") else cat("false\n") :
  missing value where TRUE/FALSE needed
```

Checking For NA

Sometimes you need to check whether a value is NA. But you *can't* do this with something like `if (a == NA) ...` — that will always give an error!

Instead, you can use the `is.na` function. It can be applied to a single value, giving TRUE or FALSE, or a vector of values, giving a logical vector.

For example, the following statements create a modified version of the `airquality` demonstration dataset in which missing values for solar radiation are replaced by the average of all the non-missing measurements (found with `mean` using the `na.rm` option):

```
ave_solar <- mean (airquality$Solar.R, na.rm=TRUE)
mod_airquality <- airquality
for (i in 1:nrow(mod_airquality))
    if (is.na(mod_airquality$Solar.R[i]))
        mod_airquality$Solar.R[i] <- ave_solar
```

(We'll see later how one can do this more easily using logical indexes.)

NA and NaN

A value will also be "missing" if it is the result of an undefined mathematical operation. R prints such values as NaN, not NA, but `is.na` will be TRUE for them. Operations on NaN produce NaN as a result. Here are some examples:

```
> 0/0
[1] NaN
> sqrt(-1)
[1] NaN
Warning message:
In sqrt(-1) : NaNs produced
> x <- 0/0
> 10*x
[1] NaN
> v <- asin((-2):2)
Warning message:
In asin((-2):2) : NaNs produced
> v
[1]       NaN -1.570796  0.000000  1.570796       NaN
> v / 0
[1]  NaN -Inf  NaN  Inf  NaN
```

Names for Elements of Vectors

When doing statistical analysis, referring something by name – for example, "age" — is much more reliable than referring to it by some number.

We've already seen that names can be attached to list elements. You can also give names to elements in a vector, either when first creating it, or later. For example:

```
> u <- c (abc=9, def=10, xyz=3)
> u
abc def xyz
  9  10   3
> v <- c(9,10,3)
> names(v) <- c("abc","def","xyz")
> v
abc def xyz
  9  10   3
```

Unfortunately, unlike for lists, you can't get at elements in vectors by name using `$`. But you can use the name as an index:

```
> v["def"]
def
 10
```

Names for Rows and Columns of Matrices and Data Frames

You can also give names to rows and columns of a matrix:

```
> M <- matrix (1:12, nrow=3, ncol=4)
> rownames(M) <- c("ab","cd","ef")
> colnames(M) <- c("w","x","y","z")
> M
   w x y  z
ab 1 4 7 10
cd 2 5 8 11
ef 3 6 9 12
> M["cd","y"]
[1] 8
```

If you convert the matrix to a data frame, its row and column names become those of the data frame:

```
> df <- data.frame(M)
> df$x
[1] 4 5 6
```

Using Vectors of Indexes

A subscript used with `[ ]` can be a vector of indexes, rather than just one index, yielding a subset of elements having those indexes, not just one element. Some examples, using some variables defined earlier (note how names are carried on):

```
> v
abc def xyz
  9  10   3
> v[c(1,3)]
abc xyz
  9   3
> M
   w x y  z
ab 1 4 7 10
cd 2 5 8 11
ef 3 6 9 12
> M[c(3,1),c(2,4,4)] # Indexes needn't be in order, can be duplicates
   x  z  z
ef 6 12 12
ab 4 10 10
```