CSC 121: Computer Science for Statistics

Sourced from: Radford M. Neal, University of Toronto, 2017

Data Frames Intro

Operations on Numeric Vectors that Produce One Number

R has several functions that take a numeric vector or matrix as their argument, and return a single number as their value, including:

`sum`	finds the sum of all elements.
`prod`	finds the product of all elements.
`max`	finds the largest of all elements.
`min`	finds the smallest of all elements.
`mean`	finds the mean (average) of all elements.

Features of R 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 — games need to respond quickly, 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 some:

- Specifying function arguments by name, with arguments often having default values — very useful for functions implementing statistical methods.
- Names for elements of vectors and lists, and for rows and columns of matrices and data frames — "age" is a better label for a column than the number 17.
- R's "data frames" for storing observations in a way that is convenient for statistical analysis.
- Special `NA` values to indicate where data is missing

We've talked about the first two, and will now talk about the last two.

Data Frames

A data frame is sort of like a list and sort of like a matrix. Each "row" of a data frame holds information on some individual, object, case, or whatever. The "columns" of a data frame correspond to variables whose values have been measured for each case. These variables can be numbers, logical (`TRUE/FALSE`) values, or character strings (but all values for one variable have the same type).

For example, here's how R prints a small data frame containing the heights and weights of three people:

```
> heights_and_weights
  name height weight
1 Fred     62    144
2 Mary     60    131
3  Joe     71    182
```

Getting Data Out of a Data Frame

You can get data from a data frame using subscripting operations similar to those for a matrix (by row and column index), or by operations similar to a list (using names of variables). For example:

```
> heights_and_weights           # The data frame from the last slide
  name height weight
1 Fred     62    144
2 Mary     60    131
3  Joe     71    182
> heights_and_weights$height    # All values of the "height" variable
[1] 62 60 71
> heights_and_weights[2,]       # All values for the 2nd person
  name height weight
2 Mary     60    131
> heights_and_weights[2,3]      # Value of 3rd variable for 2nd person
[1] 131
> heights_and_weights$weight[2] # ... and the same, by variable name
[1] 131
```

Creating a Data Frame

Using `as.data.frame`, you can create a data frame from a list (it just adds the `data.frame` class attribute) or from a matrix (it has to split it up into columns). If you don't provide variable names, R uses `V1`, `V2`, etc.

Examples: can also just use data.frame()

```
> as.data.frame (list (abc=c(1,3,2),
+                      pqr=c(TRUE,FALSE,FALSE),
+                      xyz=c("a","bb","c")))
  abc   pqr xyz
1   1  TRUE   a
2   3 FALSE  bb
3   2 FALSE   c
>
> as.data.frame (matrix (1:12, nrow=3, ncol=4))
  V1 V2 V3 V4
1  1  4  7 10
2  2  5  8 11
3  3  6  9 12
```

Reading Data Into a Data Frame

The `read.table` function creates a data frame using data it reads from a text file.

The file has to contain one line for each row of the data frame, containing a value (eg, a number, `TRUE/FALSE`, a string) for each variable for the case corresponding to that row.

If a `header=TRUE` argument is given to `read.table`, the names of the variables will be taken from the first line of the file.

Here's how we could read the heights and weights data frame from a file on the course web page:

```
heights_and_weights <-
  read.table ("http://www.cs.utoronto.ca/~radford/csc121/data7",
              header=TRUE)
```

The contents of the file read are as below:

```
name height weight
Fred 62 144
Mary 60 131
Joe 71 182
```

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.

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
```