Ch_8.3_Robert_Ramo

Robert RÃ¤mÃ¶

13 november 2018

8.3 Devils

The most devilish problem is getting data from a file into R correctly.

8.3.1 read.table

The `read.table` function is the most common way of getting data into R. Reading its help file three times is going to be very efficient time management if you ever use `read.table`. In particular the `header` and `row.names` arguments control what (if anything) in the file should be used as column and row names.
Another great time management tool is to inspect the result of the data you have read before attempting to use it.

8.3.2 read a table

The `read.table` function does not create a table—it creates a data frame. You don't become a book just because you read a book. The `table` function returns a table.
The idea of `read.table` and relatives is that they read data that are in a rectangular format.

8.3.3 the missing, the whole missing and nothing but the missing

Misreading missing values is an efficacious way of producing garbage. Missing values can become non-missing, non-missing values can become missing, logically numeric columns can become factors. The `na.strings` argument to `read.table` needs to be set properly. An example might be:

`na.strings=c('.', '-999')`

8.3.4 misquoting

A quite common file format is to have a column of names followed by some number of columns of data. If there are any apostrophes in those names, then you are likely to get an error reading the file unless you have set the quote argument to `read.table`. A likely value for `quote` is: `quote="` This sounds like easy advise, but almost surely it is not going to be apparent that quotes are the problem. You may get an error that says there was the wrong number of items in a line. When you get such an error, it is often a good idea to use count.fields to get a sense of what R thinks about your file. Something along the lines of: `foo.cf <- count.fields('foo.txt', sep='\t') table(foo.cf)`

8.3.5 thymine is TRUE, female is FALSE

You are reading in DNA bases identified as A, T, G and C. The columns are read as factors. Except for the column that is all T—that column is logical. Similarly, a column for gender that is all F for female will be logical. The solution is to use the read.table argument: colClasses='character'

or

colClasses='factor'

as you like. If there are columns of other sorts of data, then you need to give colClasses a vector of appropriate types for the columns in the file. Using colClasses can also make the call much more efficient.

Figure 8.3: The treacherous to country and the treacherous to guests and hosts by Sandro Botticelli.

8.3.6 whitespace is white

Whitespace is invisible, and we have a predilection to believe that invisible means non-existent. factor(c('A ', 'A', 'B')) [1] A A B Levels: A A B It is extraordinarily easy to get factors like this when reading in data. Setting the strip.white argument of read.table to TRUE can prevent this.

8.3.7 extraneous fields

When a file has been created in a spreadsheet, there are sometimes extraneous empty fields in some of the lines of the file. In such a case you might get an error similar to: mydat <- read.table('myfile', header=TRUE, sep='̂') Error in scan(file, what, nmax, sep, dec, quote, skip, : line 10 did not have 55 elements This, of course, is a perfect instance to use count.fields to see what is going on. If extraneous empty fields do seem to be the problem, then one solution is: mydat <- read.table('myfile', header=TRUE, sep='̂', + fill=TRUE) mydat <- mydat[, 1:53] At this point, it is wiser than usual to carefully inspect the results to see that the data are properly read and aligned. 8.3.8 fill and extraneous fields When the fill argument is TRUE (which is the default for read.csv and read.delim but not for read.table), there can be trouble if there is a variable number of fields in the file. writeLines(c("A,B,C,D", + "1,a,b,c", + "2,d,e,f", + "3,a,i,j", + "4,a,b,c", + "5,d,e,f", + "6,g,h,i,j,k,l,m,n"), + con=file("test.csv")) read.csv("test.csv") read.csv("test.csv", fill=FALSE)

The first 5 lines of the file are checked for consistency of the number of fields. Use count.fields to check the whole file. 8.3.9 reading messy files read.table and its relatives are designed for files that are arranged in a tabular form. Not all files are in tabular form. Trying to use read.table or a relative on a file that is not tabular is folly—you can end up with mangled data. Two functions used to read files with a more general layout are scan and readLines. 8.3.10 imperfection of writing then reading Do not expect to write data to a file (such as with write.table), read the data back into R and have that be precisely the same as the original. That is doing two translations, and there is often something lost in translation. You do have some choices to get the behavior that you want: • Use save to store the object and use attach or load to use it. This works with multiple objects. • Use dput to write an ASCII representation of the object and use dget to bring it back into R. • Use serialize to write and unserialize to read it back. (But the help file warns that the format is subject to change.) 8.3.11 non-vectorized function in integrate The integrate function expects a vectorized function. When it gives an argument of length 127, it expects to get an answer that is of length 127. It shares its displeasure if that is not what it gets:

fun1 <- function(x) sin(x) + sin(x-1) + sin(x-2) + sin(x-3) 105 8.3. DEVILS CIRCLE 8. BELIEVING IT DOES AS INTENDED integrate(fun1, 0, 2) -1.530295 with absolute error < 2.2e-14 fun2 <- function(x) sum(sin(x - 0:3)) integrate(fun2, 0, 2) Error in integrate(fun2, 0, 2) : evaluation of function gave a result of wrong length In addition: Warning message: longer object length is not a multiple of shorter object length in: x - 0:3 fun3 <- function(x) rowSums(sin(outer(x, 0:3, '-'))) integrate(fun3, 0, 2) -1.530295 with absolute error < 2.2e-14 fun1 is a straightforward implementation of what was wanted, but not easy to generalize. fun2 is an ill-conceived attempt at mimicking fun1. fun3 is a proper implementation of the function using outer as a step in getting the vectorization correct.