Win-Vector Blog

The Win-Vector LLC data science blog

R annoyances

March 20, 2010 John Mount Coding, Rants, Statistics Principle of Least Astonishment, R, R is not your friend, R programming annoyances

Readers returning to our blog will know that Win-Vector LLC is fairly "pro-R." You can take that to mean "in favor or R" or "professionally using R" (both statements are true). Some days we really don't feel that way.

Consider the following snippet of R code where we create a list with a single element named "x" that refers to a numeric vector. We start with a demonstration of the hard-coded method of pulling the x-value back out using the "$" operator.

```
> l <- list(x=c(1,2,3))
> l$x
[1] 1 2 3
```

But suppose we wanted to automate this; that is pass in the name of the value we want in a variable. We are after all using a computer, so automating a step seems like a reasonable desire. R supplies a notation for this using the "[]" operator. But something slightly different comes out under the "[]" operator than under the "$" operator:

```
> varName <- 'x'
> l[varName]
$x
[1] 1 2 3
```

Notice that the printed outputs are slightly different (one echoes "$x" and one does not). Let's use the "class()" method to see what is actually being returned in each case.

```
> class(l$x)
[1] "numeric"
> class(l['x'])
```

```
[1] "list"
```

Completely different return types are returned (in one case a numeric vector in the other a general list, not interchangeable types).

At this point you may think it is time to turn in our "pro" label and call ourselves "newb" (Internet slang for "newbie" or "idiot"). But let's slow down for a bit. When two views of the same situation disagree (such as the difference in opinion between the authors of R and myself whether the "[]" and "$" operators should return the same type) you at most know that at least one of those views is wrong. You don't really know if one view is right or even if one view is right which one it is. I can, however, bring in some additional argument to try and show the design of R is in fact wrong. The additional argument is "The Principle of Least Astonishment." This principle roughly says that it is a mistake to introduce unnecessary differences in outcomes (which to the unprepared user are unpleasant surprises). There may be some deep (yet obscure) reasons the two operators prefer to return different results. But the fact you would have to find a way to document and explain these differences really should make one think that this situation is really a mis-design and the "explanation" is really an attempt at a work around. Or to put it more rudely: there may be an explanation, but there is no excuse.

For another example consider creating a 3 by 3 matrix:

```
> m <- matrix(c(1,2,3,1,1,1,0,0,1),nrow=3,ncol=3)
> m
     [,1] [,2] [,3]
[1,]    1    1    0
[2,]    2    1    0
[3,]    3    1    1
```

Now select the last two rows of the matrix.

```
> m[c(FALSE,TRUE,TRUE),]
     [,1] [,2] [,3]
[1,]    2    1    0
[2,]    3    1    1
>
```

Now (for the punchline) try to select just the middle row of the matrix.

```
> m[c(FALSE,TRUE,FALSE),]
[1] 2 1 0
```

Notice that once again (and without warning) the result is subtly different. I admit that it seems paranoid to worry about such small differences- but when you are debugging a system that should work these are exactly the killing mistakes you are looking for. In this case the problem is pretty bad. See what happens if you tried to ask for the dimension of each of these differing returns:

```
> dim(m[c(FALSE,TRUE,TRUE),])
[1] 2 3
> dim(m[c(FALSE,TRUE,FALSE),])
NULL
```

The first case works fine (reports 2 rows and 3 columns). The second case returns "NULL" (instead of 1 row and 3 columns). In R NULL is sometimes used as an error-value (instead of throwing an exception) and this value will poison any further conditions or calculations it is involved in. The main way to deal with the arbitrary introduction of such NULLs is the incredibly tedious uncertain defensive coding practices that we argue against in Postel's Law: Not Sure Who To Be Angry With. Such code weakens both programs and programmers.

But what is going on in this example? Once again we use the "class()" method to inspect the subtly different results.

```
> class(m[c(FALSE,TRUE,TRUE),])
[1] "matrix"
> class(m[c(FALSE,TRUE,FALSE),])
[1] "numeric"
```

The result is disappointing. For a two-row select R returns a matrix (what we would expect). For a single-row select R does us the "favor" of converting the result into a vector. This is a disaster. A single row matrix is similar to a vector, but even R itself does not support the same set of operations and outcomes on vectors as it does on matrices (for example the failure of the "dim()" method). It is not safe to further calculate with these results (without by-hand converting the result back to a single row matrix which R can in fact represent). In my case this created crashing bugs deep in a long running analysis (and was hard to diagnose as the bug was in an "innocent operation" not in a "risky calculation").

All of this has to violate John Chambers' "Prime Directive" for data: "an obligation on all creators of software to program in such a way that the computations can be understood and trusted." Chambers' opinion being relevant as he is the author of the S language (of which R is an open source re-implementation). We continue to recommend R, but we also recommend being exceptionally careful when using it (which unfortunately adds time to projects).

March 20, 2010 John Mount Coding, Rants, Statistics Principle of Least Astonishment, R, R is not your friend, R programming annoyances

10 thoughts on “R annoyances”

Scott Locklin

March 20, 2010 at 11:20 am

One of the great ironies of R is it has a syntax very similar to that of ML. ML is a programming language that makes type errors of this kind completely impossible. Try it some time when you have nothing better to do; it’s frustrating in a different way (aka you wish you could turn off the strict type safety and have it do dumb R style typecasts), but it produces iron clad code which won’t ever faceplant you on a dumb type error.
As a design, R looks like a sort of sneeze. Probably stuff like you describe is a legacy of S+/ATT days (it would be interesting to check), but either way, I have come around to a reasonably good way of dealing with it. Pretend to be a dumb statistician that doesn’t know anything about programming, and that’s the “right way” to do stuff in R.

FWIIW, the Clojure dudes are trying to build an R style thing called “Incanter.” I’m a bit skeptical, but it would be hard to do worse as a design than R.
http://data-sorcery.org/

---

jmount

March 20, 2010 at 11:23 am

Scott, I also love statically type-checked languages (some of my “secret” projects are in Scala). People go on and on about how much they love dynamic and un-typed languages (where every function has all the power and pain of a macro)- but some day they will learn that 5 minutes of preparation (dealing with types and their declarations) can save a lot of debugging (and also produces better documented code).

---

tom

March 20, 2010 at 11:38 am

You should use [[

l[[varName]]
[1] 1 2 3

class(l[[varName]])
[1] “numeric”

---

tom

March 20, 2010 at 11:41 am

With respect to the matrix example: This should be:

m[c(FALSE,TRUE,FALSE), , drop = FALSE]
[,1] [,2] [,3]
[1,] 2 1 0

class(m[c(FALSE,TRUE,FALSE), , drop = FALSE])
[1] “matrix”

William Pietri

March 20, 2010 at 11:44 am

Wow, I'm glad it's not just me.

I needed to do some pretty basic statistical stuff on a couple hundred thousand points of data. The kind of thing that you can almost do in oocalc if you are willing to put up with a bunch of manual work. So I thought I'd try R and level up my stats skills a little.

I blew maybe a day between the command line and some of the graphical interfaces that purport to make things better, and got absolutely nowhere; one of the hurdles was types, and what could be applied to what. In the end it was easier just for me to code my own tools in Ruby. Scott's comment perfectly summed up my feelings: "As a design, R looks like a sort of sneeze."

---

jmount

March 22, 2010 at 8:26 am

A sincere thank you to Tom who posted very good solutions the the specific problems I discussed. It helps a lot (getting the right result out is much much safer than trying to repair a damaged result). However, it is still frustrating to have to distinguish between [] and [[]] (neither of which can be easily searched on in the R help system) and having to add a third drop=FALSE argument to all matrix selections. These requirements are an example of what one of my groups used to call "you forgot to set the 'do not lose' flag." This described when a function call failed because you forgot to set a useless legacy argument to the one value that allowed the function to work properly. Or in Alice in Wonderland terms: if [[]] and [rows,,drop=FALSE] are the natural operations then why don't they have more succinct names like [] and [rows,]? (The Alice in Wonderland analogy: if rule 42 is the "oldest rule in the book" then why isn't it called "rule 1?")

---

jmount

March 24, 2010 at 7:04 am

Been asking around and running into a lot of gotchas in productionizing R-models. Such as the rumored "Hadley Wickham says never to use factors" (the likely reason being it is next to impossible to guarantee you get the same transformation and indexing when loading new data to score). We hadn't run into this before because we

had be selling Java implementations of models generated in R (so we managed the data transformations by hand). What we had not expected was "lets keep things simple and let the client use R" would be so perilous.

Jeromy Anglim

March 29, 2010 at 4:54 pm

Interesting post. I agree that learning R can be frustrating at times.
As a social scientist, I came from a background using MS Windows, MS Word, SPSS, and a somewhat surface-level understanding of statistics.
R has a habit of pushing me towards thinking like a statistician.
The modelling notation, the requirement to create solutions from smaller tools, the way that R interfaces really well with LaTeX but poorly with MS Word, the use R makes of Unix-derived tools, these were all challenges at first. But ultimately they provide a far superior way of conducting data analysis.

I imagine a similar set of points could be made for R users coming from a computer science background. Learning R will probably encourage you to learn more about statistics and think more like an intelligent statistician.

The principle of least surprise depends on your expectations, which are in turn related to your prior training.
Returning a vector when subsetting a single row or column of a matrix is often useful behaviour.
When using a lapply or sapply function over the elements of a list, it's important to distinguish between "[]" and "[[]]".

---

jmount

March 30, 2010 at 7:21 am

@Jeromy Anglim Some good points and thanks for the comment. Just one amplification from my end. I agree some facility to return a single row as a vector is a useful feature (in principle). I deliberately used a cumbersome c(FALSE,TRUE,FALSE) notation (instead of just the index number: 2) to emphasize how the return-type changes due to mere changes of values of the input data (instead of in an orderly way due to the calling type/style).

Zhonghao Yu

April 19, 2010 at 7:51 am

Just try help('[') to find the help for indexing.

Comments are closed.
