I would like to introduce an exciting feature in the upcoming 1.9.6 version of the wrapr R package: value unpacking.

We will likely send this version to CRAN in a couple of weeks.

Here is an example of the unpack feature in use. First we set up some example data.

library(wrapr)

packageVersion('wrapr')

#> [1] '1.9.6'

# make some example data

d <- data.frame(

x = 1:2,

g = c('test', 'train'),

stringsAsFactors = FALSE)

Now we demonstrate the new feature: unpacking a named-list returned from a function. In this case we will demonstrate the effect using the function base::split(). base::split() splits data into a named list, with the names coming from the grouping vector. Our unpack feature will conveniently assign these sub-dataframes into our environment for us.

# unpack the data into our workspace

# notation is assignment-like: NEW\_VARAIBLE = NAMED\_ITEM

unpack[train\_set = train, test\_set = test] <- split(d, d$g)

In the above example base::split() built a named list of sub-dataframes from our original data frame d. We used unpack[] to assign these named items into our working environment as the new variables: train\_set and test\_set. The unpacking was triggered by assigning the split results into the special unpack[] notation. Notice the unpack specification itself also looks like assignments (or more precisely argument bindings) with new names (used to say where values will be written) on the left and old names (used to say where values are found) on the right.

We can confirm that the training data is in the train\_set variable, and the test data is in the test\_set variable:

# confirm we have the new variables

print(train\_set)

#> x g

#> 2 2 train

print(test\_set)

#> x g

#> 1 1 test

The unpacking notation, when used in this manner, doesn’t depend on the order of the values. This makes for very safe code that concisely documents intent.

There is a small side effect, due to R’s assignment rules using the []<- notation will write a valued named “unpack” into the working environment. If one wishes to avoid this they can use either a function notation:

unpack(split(d, d$g), train\_set = train, test\_set = test)

Or a “pipe into function” notation:

split(d, d$g) %.>% unpack(., train\_set = train, test\_set = test)

(Note: the above was the wrapr dot-pipe. Currently unpack() does not work with the margrittr pipe, as in that case the results appear to get written into a temporary intermediate environment created by magrittr, and then lost. This difference between pipes isn’t so much a problem with unpack, but the extent that the wrapr dot pipe is designed for user extension.)

A killer application of unpack is: replacing save(a, b, file = FNAME)/load(file = FNAME) with a much safer and more explicit saveRDS(list(a = a, b = b, ...), file = FNAME)/unpack(readRDS(file = FNAME), a, b, ...) pattern. Here we are using the convention that names alone such as “a” implicitly stand for “a = a“.

unpack also supports positional unpacking, as we see below.

list(x = 1:2, y = 3:4) -> unpack\_i[a, b]

print(a)

#> [1] 1 2

print(b)

#> [1] 3 4

Though we feel the named pattern is more compatible with existing R functions and style.

We are still working on choosing names for this function. Likely we will pick “unpack” for the functional form, and perhaps one of “to” or “into” for the array-bracket assignment form. Right now we implement all 3 names, each with all functionality.

If you don’t want to bring in all of wrapr with library(wrapr), you can bring in just a few bits as follows:

unpack <- wrapr::unpack

`%.>%` <- wrapr::`%.>%`

We are designing unpack to be very strict in its name checking before writing any values to the workspace. This is to avoid partial assignments where some fields are written and others are missing.f

We hope you check out the unpack feature and use it to your projects.

Related work includes:

* The zeallot::%<-% package already supplies excellent positional or ordered unpacking. But we feel that style may be more appropriate in the Python world where many functions return un-named tuples of results. Python functions tend to have positional tuple return values *because* the Python language has had positional tuple unpacking as a core language feature for a very long time (thus positional structures have become “Pythonic”). R has not emphasized positional unpacking, so R functions tend to return named lists or named structures. For named lists or named structures it may not be safe to rely on value positions. So I feel it is more “R-like” to use named unpacking.
* vadr::bind supplies named unpacking, but appears to use a “SOURCE = DESTINATION” notation. That is the reverse of a “DESTINATION = SOURCE” which is how both R assignments and argument binding are already written.
* base::attach. base::attach adds items to the search path with names controlled by the object being attached (instead of by the user).
* wrapr::let(). wrapr::let() re-maps names during code execution using a “TARGET = NEWNAME” target replacement scheme, where TARGET acts as if it had the name stored in NEWNAME for the duration of the let-block.