We have some really nifty upcoming enhancements to wrapr unpack/to.

One of the new notations is the use of := as an alternate assignment operator for unpack/to.

This lets us write code like the following.

First let’s attach our package and set up some example data.

library(wrapr) # attach package

packageVersion("wrapr") # confirm we have at least version 2.0.0

#> [1] ‘2.0.0’

# example data

d <- data.frame(

x = 1:9,

group = c('train', 'calibrate', 'test'),

stringsAsFactors = FALSE)

base::split() is a very handy function for splitting a data frame into smaller data frames by group. For example:

print(split(d, d$group))

#> $calibrate

#> x group

#> 2 2 calibrate

#> 5 5 calibrate

#> 8 8 calibrate

#>

#> $test

#> x group

#> 3 3 test

#> 6 6 test

#> 9 9 test

#>

#> $train

#> x group

#> 1 1 train

#> 4 4 train

#> 7 7 train

Often we want these split data frame to be in our working environment, instead of trapped in a list. The usual way to achieve this would be to store the split list into a temporary variable and then assign elements of the list into our environment one at a time. This isn’t a problem, but it also isn’t as elegant as the following.

# assign split data into environment

unpack[

traind = train,

testd = test,

cald = calibrate

] := split(d, d$group)

After this step our environment has the three split data frames, using names of our choosing. For example we have:

knitr::kable(traind)

|  | **x** | **group** |
| --- | --- | --- |
| 1 | 1 | train |
| 4 | 4 | train |
| 7 | 7 | train |

Notice we didn’t need to introduce a temporary variable to hold the list of splits. This is not a huge thing, but it more neatly documents intent. It is a small thing, but being elegant in the small things can help us achieve elegance in large projects.

unpack and to has been designed to have very regular and versatile notation. If we prefer we can use arrows to specify the assignments.

# assign split data into environment

unpack[

traind <- train,

testd <- test,

cald <- calibrate

] := split(d, d$group)

Or we can use a pipe to assign to the right.

split(d, d$group) %.>%

unpack[

traind <- train,

testd <- test,

cald <- calibrate

]

And unpack can be also used in a more traditional non-operator notation as follows.

unpack(

split(d, d$group),

traind <- train,

testd <- test,

cald <- calibrate

)

An interesting side-note is how similar the above form is to the following.

with(

split(d, d$group),

{

traind <<- train

testd <<- test

cald <<- calibrate

}

)