wrapr 1.6.2 is now up on CRAN. We have some neat new features for R users to try

The first is the %in\_block% alternate notation for let().

The wrapr let()-block allows easy replacement of names in name-capturing interfaces (such as transform()), as we show below.

library("wrapr")

column\_mapping <- qc(

AREA\_COL = Sepal.Area,

LENGTH\_COL = Sepal.Length,

WIDTH\_COL = Sepal.Width

)

# let-block notation

let(

alias = column\_mapping,

iris %.>%

transform(.,

AREA\_COL = (pi/4)\*LENGTH\_COL\*WIDTH\_COL) %.>%

subset(.,

select = qc(Species, AREA\_COL)) %.>%

head(.)

)

## Species Sepal.Area

## 1 setosa 14.01936

## 2 setosa 11.54535

## 3 setosa 11.81239

## 4 setosa 11.19978

## 5 setosa 14.13717

## 6 setosa 16.54049

The qc() notation allowed us to specify a named-vector without quotes. qc(a = b) is equivalent to c("a" = "b").

With the %in\_block% operator notation one writes the let()-block as an in-line operator supplying the mapping into a code block. The above example can now be re-written as the following.

# %in\_block% notation

column\_mapping %in\_block% {

iris %.>%

transform(.,

AREA\_COL = (pi/4)\*LENGTH\_COL\*WIDTH\_COL) %.>%

subset(.,

select = qc(Species, AREA\_COL)) %.>%

head(.)

}

## Species Sepal.Area

## 1 setosa 14.01936

## 2 setosa 11.54535

## 3 setosa 11.81239

## 4 setosa 11.19978

## 5 setosa 14.13717

## 6 setosa 16.54049

This notation can be handy for defining functions.

compute\_area <- function(

.data,

area\_col,

length\_col,

width\_col) c( # End of function argument definiton

AREA\_COL = area\_col,

LENGTH\_COL = length\_col,

WIDTH\_COL = width\_col

) %in\_block% { # End of argument mapping block

.data %.>%

transform(.,

AREA\_COL = (pi/4)\*LENGTH\_COL\*WIDTH\_COL)

} # End of function body block

iris %.>%

compute\_area(.,

'Sepal.Area', 'Sepal.Length', 'Sepal.Width') %.>%

compute\_area(.,

'Petal.Area', 'Petal.Length', 'Petal.Width') %.>%

subset(.,

select = c("Species", "Sepal.Area", "Petal.Area")) %.>%

head(.)

## Species Sepal.Area Petal.Area

## 1 setosa 14.01936 0.2199115

## 2 setosa 11.54535 0.2199115

## 3 setosa 11.81239 0.2042035

## 4 setosa 11.19978 0.2356194

## 5 setosa 14.13717 0.2199115

## 6 setosa 16.54049 0.5340708

We can think of the above function definition notation as having two blocks: the alias defining block (the portion before "%in\_block%") and the templated function body (the portion after "%in\_block%"). Notice how easy it is to use this notation to convert a non-standard (or name/code-capturing interface) into a value-oriented interface. The point is value-oriented interfaces are much more re-usable and easier to program over (use in for-loops, applies, and functions).

The second new feature is the orderv() function, a value-oriented adapter for base::order(). orderv() uses a vector of column names to compute an ordering permutation for a data.frame. We can use it as we show below.

library("wrapr")

sort\_columns <- qc(mpg, hp, gear)

ordering <- orderv(mtcars[ , sort\_columns, drop = FALSE],

decreasing = TRUE,

method = "radix")

head(mtcars[ordering, , drop = FALSE])

## mpg cyl disp hp drat wt qsec vs am gear carb

## Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1

## Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1

## Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2

## Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2

## Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1

## Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2

Of course we have also have all the steps wrapped in a convenient function: sortv().

mtcars %.>%

sortv(.,

sort\_columns,

decreasing = TRUE,

method = "radix") %.>%

head(.)

## mpg cyl disp hp drat wt qsec vs am gear carb

## Toyota Corolla 33.9 4 71.1 65 4.22 1.835 19.90 1 1 4 1

## Fiat 128 32.4 4 78.7 66 4.08 2.200 19.47 1 1 4 1

## Lotus Europa 30.4 4 95.1 113 3.77 1.513 16.90 1 1 5 2

## Honda Civic 30.4 4 75.7 52 4.93 1.615 18.52 1 1 4 2

## Fiat X1-9 27.3 4 79.0 66 4.08 1.935 18.90 1 1 4 1

## Porsche 914-2 26.0 4 120.3 91 4.43 2.140 16.70 0 1 5 2

A third new feature is mk\_formula(). mk\_formula() is used to build simple formulas for modeling tasks (which may have a large number of variables) without any string processing or parsing.

This remains good advice. However mk\_formula() is a more concise and more hygienic alternative. An example is given below.

# specifications of how to model,

# coming from somewhere else

outcome <- "mpg"

variables <- c("cyl", "disp", "hp", "carb")

# our modeling effort,

# fully parameterized!

f <- wrapr::mk\_formula(outcome, variables)

print(f)

## mpg ~ cyl + disp + hp + carb

model <- lm(f, data = mtcars)

print(model)

##

## Call:

## lm(formula = f, data = mtcars)

##

## Coefficients:

## (Intercept) cyl disp hp carb

## 34.021595 -1.048523 -0.026906 0.009349 -0.926863

The above notation is good for programming over modeling tasks.