**Introduction**

The idea of turning base R’s get and set operators ([, [[, [<-, [[<-) into human readable and pipeable functions. It was kindly pointed out in the comments that the get() and set() functions I defined in that blog post are actually exported in the magrittr package as magrittr::extract2() ([[) and magrittr::inset2() ([[<-). In fact, there are a whole host of “alias” functions exported by magrittr, see ?magrittr::extract2 for more. However if we are developing a package, we may not necessarily want to Import: magrittr, we may only want to Suggest it as a package that complements our package. This is especially true when the functions we will be importing are simple aliases of other functions that we can just as easily create ourselves. Now sure, a lot of people already have and use magrittr, in which case they can use it, but not everyone wants it or uses it, so we shouldn’t enforce that dependency on users.

Take for example if we were to create a package that recreates dplyr’s main verbs, select(); filter(); mutate(); and arrange(), using base R only. Think of it as a “poor man’s” dplyr, of course I jest – base is awesome. Oftentimes the main complaint I hear about dplyr is the sheer number of dependencies it has and the installation times that come with that; not to mention APIs have changed a few times over the years. base on the other hand already comes pre-installed with R and the API is extremely stable. The reason people like dplyr, however, is because the API on offer is extremely flexible and easy to understand. This blog post will show how we can recreate these verbs using base R and aliases to R’s operator functions and use them in conjunction with magrittr.

**Select**

dplyr::select() allows the user to subset the columns of a data.frame and always return a data.frame. Thus to recreate this function we will need the operator for subsetting columns of a data.frame which is [, or more specifically, [.data.frame. Let’s take a look at the arguments for this function.

args(`[.data.frame`)

# function (x, i, j, drop = if (missing(i)) TRUE else length(cols) ==

# 1)

# NULL

We see that it takes:

* x – the data.frame
* i – the rows to subset
* j – the columns to subset
* drop – whether to return a vector if only one column is left

We will define our wrapper for the [ function in the same way that magrittr does.

extract <- `[`

As this is an S3 generic, R will know to dispatch to [.data.frame when it is passed a data.frame. Hence, we can now define a select() function which is similar in functionality to that of dplyr::select(). Note that we tell R that we wish to subset all of the rows in the i position by leaving the argument blank.

select <- function(.data, ...) {

cols <- vapply(substitute(...()), deparse, NA\_character\_)

extract(.data, , cols, drop = FALSE)

}

This function uses a couple of tricks here, so I’ll break them down. To use non-standard evaluation in the same way that dplyr does, that is to pass non-quoted column names, we must deparse them. We loop over the columns passed via ... using a vapply(). The substitute(...()) gives us a list-like object of all the symbols we pass which we can loop over. Using this function, we can now select a single column.

mtcars %>% select(mpg)

# mpg

# Mazda RX4 21.0

# Mazda RX4 Wag 21.0

# Datsun 710 22.8

# Hornet 4 Drive 21.4

# Hornet Sportabout 18.7

# Valiant 18.1

# Duster 360 14.3

# Merc 240D 24.4

# Merc 230 22.8

# ... 23 rows omitted

Or multiple columns by passing a vector.

mtcars %>% select(mpg, cyl)

# mpg cyl

# Mazda RX4 21.0 6

# Mazda RX4 Wag 21.0 6

# Datsun 710 22.8 4

# Hornet 4 Drive 21.4 6

# Hornet Sportabout 18.7 8

# Valiant 18.1 6

# Duster 360 14.3 8

# Merc 240D 24.4 4

# Merc 230 22.8 4

# ... 23 rows omitted

And of course, this function works without magrittr.

select(mtcars, mpg, cyl)

# mpg cyl

# Mazda RX4 21.0 6

# Mazda RX4 Wag 21.0 6

# Datsun 710 22.8 4

# Hornet 4 Drive 21.4 6

# Hornet Sportabout 18.7 8

# Valiant 18.1 6

# Duster 360 14.3 8

# Merc 240D 24.4 4

# Merc 230 22.8 4

# ... 23 rows omitted

For bonus points, we can write an equivalent of dplyr::pull() by setting the drop = TRUE argument and removing the cols parameter since we are only dealing with one column.

pull <- function(.data, var) {

var <- deparse(substitute(var))

stopifnot(length(var) == 1)

extract(.data, , var, drop = TRUE)

}

mtcars %>% pull(mpg)

# [1] 21.0 21.0 22.8 21.4 18.7 18.1 14.3 24.4 22.8 19.2 17.8 16.4 17.3 15.2 10.4 10.4 14.7

# [18] 32.4 30.4 33.9 21.5 15.5 15.2 13.3 19.2 27.3 26.0 30.4 15.8 19.7 15.0 21.4

**Filter**

As we saw in the previous section, [.data.frame takes i as an argument which represents the rows to filter. Thus we can use a similar method to that used for select() only in this case, we must build the expressions by which to filter and separate them with an ampersand from which we can parse and evaluate.

filter <- function(.data, ...) {

conditions <- paste(vapply(substitute(...()), deparse, NA\_character\_), collapse = " & ")

extract(.data, with(.data, eval(parse(text = conditions))), )

}

mtcars %>% filter(cyl == 4)

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

# Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1

# Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2

# Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2

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

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

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

# Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1

# 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

# ... 2 rows omitted

mtcars %>% filter(cyl <= 5 & am > 0)

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

# Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1

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

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

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

# 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

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

# Volvo 142E 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2

mtcars %>% filter(cyl == 4 | cyl == 8)

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

# Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1

# Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2

# Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4

# Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2

# Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2

# Merc 450SE 16.4 8 275.8 180 3.07 4.070 17.40 0 0 3 3

# Merc 450SL 17.3 8 275.8 180 3.07 3.730 17.60 0 0 3 3

# Merc 450SLC 15.2 8 275.8 180 3.07 3.780 18.00 0 0 3 3

# Cadillac Fleetwood 10.4 8 472.0 205 2.93 5.250 17.98 0 0 3 4

# ... 16 rows omitted

mtcars %>% filter(!(cyl %in% c(4, 6)), am != 0)

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

# Ford Pantera L 15.8 8 351 264 4.22 3.17 14.5 0 1 5 4

# Maserati Bora 15.0 8 301 335 3.54 3.57 14.6 0 1 5 8

We can also get a copy of dplyr::slice() really cheaply.

slice <- function(.data, ...) {

stopifnot(is.numeric(...) || is.integer(...))

extract(.data, ..., )

}

mtcars %>% slice(1:3)

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

# Mazda RX4 21.0 6 160 110 3.90 2.620 16.46 0 1 4 4

# Mazda RX4 Wag 21.0 6 160 110 3.90 2.875 17.02 0 1 4 4

# Datsun 710 22.8 4 108 93 3.85 2.320 18.61 1 1 4 1

**Arrange**

The final function using the extract() alias that I want to highlight is arrange(). I want to highlight this function because of the required trick with eval.parent()

arrange <- function(.data, ...) {

rows <- eval.parent(substitute(with(.data, order(...))))

extract(.data, rows, , drop = FALSE)

}

We use eval.parent() instead of eval(),The eval.parent() trick has been [proposed by @MoodyMudskipper](https://stackoverflow.com/a/58239679/300187) as a way to address this problem and allows us to seamlessly use arrange() inside other functions, including magrittr pipes:

mtcars %>% arrange(mpg)

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

# Cadillac Fleetwood 10.4 8 472.0 205 2.93 5.250 17.98 0 0 3 4

# Lincoln Continental 10.4 8 460.0 215 3.00 5.424 17.82 0 0 3 4

# Camaro Z28 13.3 8 350.0 245 3.73 3.840 15.41 0 0 3 4

# Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4

# Chrysler Imperial 14.7 8 440.0 230 3.23 5.345 17.42 0 0 3 4

# Maserati Bora 15.0 8 301.0 335 3.54 3.570 14.60 0 1 5 8

# Merc 450SLC 15.2 8 275.8 180 3.07 3.780 18.00 0 0 3 3

# AMC Javelin 15.2 8 304.0 150 3.15 3.435 17.30 0 0 3 2

# Dodge Challenger 15.5 8 318.0 150 2.76 3.520 16.87 0 0 3 2

# ... 23 rows omitted

mtcars %>% arrange(cyl, mpg)

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

# Volvo 142E 21.4 4 121.0 109 4.11 2.780 18.60 1 1 4 2

# Toyota Corona 21.5 4 120.1 97 3.70 2.465 20.01 1 0 3 1

# Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1

# Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2

# Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2

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

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

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

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

# ... 23 rows omitted

**Mutate**

If we wish to create new columns in our dataset, particularly columns created using existing columns in the data, we must use the [<- operator, specifically, [<-.data.frame.

args(`[<-.data.frame`)

# function (x, i, j, value)

# NULL

[<-.data.frame takes the arguments:

* x – the data
* i – the rows to create
* j – the columns to create
* value – the value to give to the rows/columns

We will assign this operator to inset – the same as magrittr does.

inset <- `[<-`

Here we lapply() over each of the conditions to return a list of vectored results of our expressions. We then use the inset() function to add these vectors as new columns to the data.frame.

mutate <- function(.data, ...) {

conditions <- vapply(substitute(...()), deparse, NA\_character\_)

new\_data <- lapply(

conditions,

function(x, .data) with(.data, eval(parse(text = x))),

.data

)

inset(.data, , names(conditions), new\_data)

}

mtcars %>% mutate(mpg2 = mpg \* 2)

# mpg cyl disp hp drat wt qsec vs am gear carb mpg2

# Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4 42.0

# Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4 42.0

# Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1 45.6

# Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1 42.8

# Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2 37.4

# Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1 36.2

# Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4 28.6

# Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2 48.8

# Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2 45.6

# ... 23 rows omitted

mtcars %>% mutate(mpg2 = mpg \* 2, cyl2 = cyl \* 2)

# mpg cyl disp hp drat wt qsec vs am gear carb mpg2 cyl2

# Mazda RX4 21.0 6 160.0 110 3.90 2.620 16.46 0 1 4 4 42.0 12

# Mazda RX4 Wag 21.0 6 160.0 110 3.90 2.875 17.02 0 1 4 4 42.0 12

# Datsun 710 22.8 4 108.0 93 3.85 2.320 18.61 1 1 4 1 45.6 8

# Hornet 4 Drive 21.4 6 258.0 110 3.08 3.215 19.44 1 0 3 1 42.8 12

# Hornet Sportabout 18.7 8 360.0 175 3.15 3.440 17.02 0 0 3 2 37.4 16

# Valiant 18.1 6 225.0 105 2.76 3.460 20.22 1 0 3 1 36.2 12

# Duster 360 14.3 8 360.0 245 3.21 3.570 15.84 0 0 3 4 28.6 16

# Merc 240D 24.4 4 146.7 62 3.69 3.190 20.00 1 0 4 2 48.8 8

# Merc 230 22.8 4 140.8 95 3.92 3.150 22.90 1 0 4 2 45.6 8

# ... 23 rows omitted

Note that unlike dplyr::mutate(), we cannot create columns based on expressions we pass, for example, the following would not work.

mtcars %>% mutate(mpg2 = mpg \* 2, mpg3 = mpg2 \* 3)

# Error in eval(parse(text = x)): object 'mpg2' not found

As a bonus, we can combine our mutate() function with extract() to create a copy of dplyr::transmute().

transmute <- function(.data, ...) {

conditions <- vapply(substitute(...()), deparse, NA\_character\_)

mutated <- mutate(.data, ...)

extract(mutated, names(conditions))

}

mtcars %>% transmute(mpg2 = mpg \* 2, cyl2 = cyl \* 2)

# mpg2 cyl2

# Mazda RX4 42.0 12

# Mazda RX4 Wag 42.0 12

# Datsun 710 45.6 8

# Hornet 4 Drive 42.8 12

# Hornet Sportabout 37.4 16

# Valiant 36.2 12

# Duster 360 28.6 16

# Merc 240D 48.8 8

# Merc 230 45.6 8

# ... 23 rows omitted

**Chaining**

As a final note, it should be clear that due to the nature of magrittr, your standard chaining of functions will still work.

mtcars %>%

filter(cyl == 4) %>%

select(mpg, cyl, wt, disp)

# mpg cyl wt disp

# Datsun 710 22.8 4 2.320 108.0

# Merc 240D 24.4 4 3.190 146.7

# Merc 230 22.8 4 3.150 140.8

# Fiat 128 32.4 4 2.200 78.7

# Honda Civic 30.4 4 1.615 75.7

# Toyota Corolla 33.9 4 1.835 71.1

# Toyota Corona 21.5 4 2.465 120.1

# Fiat X1-9 27.3 4 1.935 79.0

# Porsche 914-2 26.0 4 2.140 120.3

# ... 2 rows omitted

**Conclusion**

The idea behind this blog post was to highlight how we can use more human readable versions of R’s primitive operators to aid in pipeable data manipulation functions. Of course the solutions provided in this blog post are over-engineered and you would probably write them in a different way if you were seriously thinking about releasing them as a package. Also, whilst these functions are available via an import of magrittr, you may not wish to force the user to import magrittr and may wish to keep it as a suggestion instead. This reduces the number of dependencies on your package.