We recently encountered a situation where We wanted to run several linear models, but where the response variables would depend on previous steps in the data analysis pipeline. Let me illustrate using the mtcars dataset:

data(mtcars)

head(mtcars)

#> 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

#> Hornet 4 Drive 21.4 6 258 110 3.08 3.215 19.44 1 0 3 1

#> Hornet Sportabout 18.7 8 360 175 3.15 3.440 17.02 0 0 3 2

#> Valiant 18.1 6 225 105 2.76 3.460 20.22 1 0 3 1

Let’s say I wanted to fit a linear model of mpg vs. hp and get the coefficients. This is easy:

lm(mpg ~ hp, data = mtcars)$coefficients

#> (Intercept) hp

#> 30.09886054 -0.06822828

But what if I wanted to fit a linear model of y vs. hp, where y is a response variable that I won’t know until runtime? Or what if I want to fit 3 linear models: each of mpg, disp, drat vs. hp? Or what if I want to fit 300 such models? There has to be a way to do this programmatically.

It turns out that there are at least 4 different ways to achieve this in R. For all these methods, let’s assume that the responses we want to fit models for are in a character vector:

response\_list <- c("mpg", "disp", "drat")

Here are the 4 ways I know (in decreasing order of preference):

**1. as.formula()**

as.formula() converts a string to a formula object. Hence, we can programmatically create the formula we want as a string, then pass that string to as.formula():

for (y in response\_list) {

lmfit <- lm(as.formula(paste(y, "~ hp")), data = mtcars)

print(lmfit$coefficients)

}

#> (Intercept) hp

#> 30.09886054 -0.06822828

#> (Intercept) hp

#> 20.99248 1.42977

#> (Intercept) hp

#> 4.10990867 -0.00349959

**2. Don’t specify the data option**

Passing the data = mtcars option to lm() gives us more succinct and readable code. However, lm() also accepts the response vector and data matrix themselves:

for (y in response\_list) {

lmfit <- lm(mtcars[[y]] ~ mtcars$hp)

print(lmfit$coefficients)

}

#> (Intercept) hp

#> 30.09886054 -0.06822828

#> (Intercept) hp

#> 20.99248 1.42977

#> (Intercept) hp

#> 4.10990867 -0.00349959

**3. get()**

get() searches for an R object by name and returns that object if it exists.

for (y in response\_list) {

lmfit <- lm(get(y) ~ hp, data = mtcars)

print(lmfit$coefficients)

}

#> (Intercept) hp

#> 30.09886054 -0.06822828

#> (Intercept) hp

#> 20.99248 1.42977

#> (Intercept) hp

#> 4.10990867 -0.00349959

**4. eval(parse())**

This one is a little complicated. parse() returns the parsed but unevaluated expressions, while eval() evaluates those expressions (in a specified environment).

for (y in response\_list) {

lmfit <- lm(eval(parse(text = y)) ~ hp, data = mtcars)

print(lmfit$coefficients)

}

#> (Intercept) hp

#> 30.09886054 -0.06822828

#> (Intercept) hp

#> 20.99248 1.42977

#> (Intercept) hp

#> 4.10990867 -0.00349959

Of course, for any of these methods, we could replace the outer loop with apply() or purrr::map().

References:

1. johnramey. [Converting a String to a Variable Name On-The-Fly and Vice-versa in R](http://johnramey.net/blog/2010/12/28/converting-a-string-to-a-variable-name-on-the-fly-and-vice-versa-in-r/).