We want to fit a linear model where the names of the data columns carrying the outcome to predict (y), the explanatory variables (x1, x2), and per-example row weights (wt) are given to us as strings.

Lets start with our example data and parameters. The point is: we are assuming the data and parameters come to us as arguments and are not known at the time of writing the script or program.

# our inputs

d <- data.frame(

x1 = c(1, 2, 3, 4),

x2 = c(0, 0, 1, 1),

y = c(3, 3, 4, -5),

wt = c(1, 2, 1, 1))

knitr::kable(d)

| **x1** | **x2** | **y** | **wt** |
| --- | --- | --- | --- |
| 1 | 0 | 3 | 1 |
| 2 | 0 | 3 | 2 |
| 3 | 1 | 4 | 1 |
| 4 | 1 | -5 | 1 |

outcome\_name <- "y"

explanatory\_vars <- c("x1", "x2")

name\_for\_weight\_column <- "wt"

For everything except the weights this is easy, as the linear regression function lm() is willing to take strings in its first argument “formula” (and also, there are tools for building up formula objects).

# start our generic solution

formula\_str <- paste(

outcome\_name,

"~",

paste(explanatory\_vars, collapse = " + "))

print(formula\_str)

## [1] "y ~ x1 + x2"

model <- lm(formula\_str,

data = d)

print(model)

##

## Call:

## lm(formula = formula\_str, data = d)

##

## Coefficients:

## (Intercept) x1 x2

## 9.75 -4.50 5.50

format(model$terms)

## [1] "y ~ x1 + x2"

However, once we try to add weights we have problems.

lm(formula\_str,

data = d,

weights = name\_for\_weight\_column)

## Error in model.frame.default(formula = formula\_str, data = d, weights = name\_for\_weight\_column, : variable lengths differ (found for '(weights)')

This is a bit disappointing, as much of the point of working in R is being able to write parameterized scripts and programs over the R functions. So we really want to be able to take names of columns from an external source.

The reason is the following (taken from help(lm)):

All of weights, subset and offset are evaluated in the same way as variables in formula, that is first in data and then in the environment of formula.

This means the weights argument is not treated as a value, but instead the name typed in is captured through [“non standard evaluation” (NSE)](https://developer.r-project.org/nonstandard-eval.pdf). The data frame environment, and formula environment are searched for a column or value named “name\_for\_weight\_column”, and not for one named “wt”.

This is a big hindrance to using lm() programmatically. The NSE notation style can be convenient when working interactively, but is often a burden when programming.

However, R has some ways out of the problem.

The first solution is [bquote()](https://stat.ethz.ch/R-manual/R-devel/library/base/html/bquote.html) which is part of [R](https://www.r-project.org/) itself (in the base package).

eval(bquote(

lm(

formula\_str,

data = d,

weights = .([as.name](http://as.name)(name\_for\_weight\_column)))

))

##

## Call:

## lm(formula = formula\_str, data = d, weights = wt)

##

## Coefficients:

## (Intercept) x1 x2

## 9.429 -3.857 3.571

In the above, the .() notation indicates to replace the .()-expression with its evaluated value before evaluating the rest of the expression. This is a substitution principle based on escaping notation (also called quasiquoting).

Another solution is the [let()](https://winvector.github.io/wrapr/reference/let.html) function from the [wrapr](https://winvector.github.io/wrapr/) package (a user extension package, not part of R itself).

wrapr::let(

c(NAME\_FOR\_WEIGHT\_COLUMN = name\_for\_weight\_column),

lm(

formula\_str,

data = d,

weights = NAME\_FOR\_WEIGHT\_COLUMN)

)

##

## Call:

## lm(formula = formula\_str, data = d, weights = wt)

##

## Coefficients:

## (Intercept) x1 x2

## 9.429 -3.857 3.571

In the above, the left-hand sides of the named vector are symbols to be replaced and the right had sides refer to values to replace them with. The specification c(NAME\_FOR\_WEIGHT\_COLUMN = name\_for\_weight\_column) means to replace NAME\_FOR\_WEIGHT\_COLUMN with wt (the value referred to by name\_for\_weight\_column). This is a substitution principle based on named substitution targets.

Another solution is the !! notation from the [rlang](https://rlang.r-lib.org/) package (a user extension package, not part of R itself).

rlang::eval\_tidy(rlang::quo(

lm(

formula\_str,

data = d,

weights = !![as.name](http://as.name)(name\_for\_weight\_column))

))

##

## Call:

## lm(formula = formula\_str, data = d, weights = wt)

##

## Coefficients:

## (Intercept) x1 x2

## 9.429 -3.857 3.571

In the above, the !! notation indicates to replace the !!-expression with its evaluated value before evaluating the rest of the expression. This is a substitution principle based on escaping notation (also called quasiquoting).

Note the argument types and/or the internals of lm() do not currently appear to allow the use of the newer rlang double curly brace notation (a notation that [can replace !!rlang::enquo()](https://www.tidyverse.org/articles/2019/06/rlang-0-4-0/), and possibly other forms).

rlang::eval\_tidy(rlang::quo(

lm(

formula\_str,

data = d,

weights = {{name\_for\_weight\_column}})

))

## Error in model.frame.default(formula = formula\_str, data = d, weights = ~"wt", : invalid type (language) for variable '(weights)'

We only mention the “{{}}” notation as rlang familiar readers are likely to wonder about using it.

All of the above solutions are using [meta-programming](https://en.wikipedia.org/wiki/Metaprogramming) tools. These are tools that let programs treat programs as data. As such they are very powerful. In fact they are big hammers for such a simple problem as specifying a column name that is already stored as data. However, the above tools should not be blamed for the awkwardness of having a need for them.