Updating Formulas

Peter DeWitt
https://www.github.com/dewittpe/drug-20160330

Denver R Users Group

30 March 2016

The update function

- Extremely useful for making quick changes to objects.
- The `update` function is an S3 method found in the `stats` package.
- Other `update` methods are found in other packages.

```
# Example
data("diamonds", package = "ggplot2")
fit <- lm(price ~ cut + color + carat, data = diamonds)
fit

##
## Call:
## lm(formula = price ~ cut + color + carat, data = diamonds)
##
## Coefficients:
## (Intercept)        cut.L        cut.Q        cut.C        cut^4
##    -3149.82      1243.35      -531.75       372.06        76.15
##     color.L      color.Q      color.C      color^4      color^5
##    -1579.17      -732.85      -107.41        81.63      -138.64
##     color^6        carat
##     -161.09      8183.74
```

Example: update formula in the linear model

```
# the dot means "keep all of this side"
# add clarity to the rhs
update(fit, formula = . ~ . + clarity)

##
## Call:
## lm(formula = price ~ cut + color + carat + clarity, data = diamonds)
##
## Coefficients:
## (Intercept)        cut.L        cut.Q        cut.C        cut^4
##   -3710.603      698.907     -327.686      180.565       -1.207
##     color.L      color.Q      color.C      color^4      color^5
##   -1910.288     -627.954     -171.960       21.678      -85.943
##     color^6        carat    clarity.L    clarity.Q    clarity.C
##     -49.986     8886.129     4217.535    -1832.406      923.273
##   clarity^4    clarity^5    clarity^6    clarity^7
##    -361.995      216.616        2.105      110.340
```

Example: update formula in the linear model

```
# the dot means "keep all of this side"
# remove cut from the rhs
update(fit, formula = . ~ . - cut)

##
## Call:
## lm(formula = price ~ color + carat, data = diamonds)
##
## Coefficients:
## (Intercept)      color.L      color.Q      color.C      color^4
##    -2702.23     -1572.20      -741.14      -122.70        78.77
##     color^5      color^6        carat
##     -144.74      -180.75      8066.62
```

Example: update data set in the linear model

```
# remove the ordering form diamonds$cut
update(fit,
       data = dplyr::mutate(diamonds, cut = factor(cut, ordered = FALSE)))

##
## Call:
## lm(formula = price ~ cut + color + carat, data = dplyr::mutate(diamonds,
##     cut = factor(cut, ordered = FALSE)))
##
## Coefficients:
##  (Intercept)        cutGood  cutVery Good    cutPremium      cutIdeal
##     -4328.97        1126.98       1518.00       1442.73       1808.04
##      color.L        color.Q       color.C       color^4       color^5
##     -1579.17        -732.85       -107.41         81.63       -138.64
##      color^6          carat
##      -161.09        8183.74
```

Example: update multiple parts of an object

```
update(fit,
       formula = . ~ . + clarity,
       data = dplyr::mutate(diamonds, cut = factor(cut, ordered = FALSE)))

##
## Call:
## lm(formula = price ~ cut + color + carat + clarity, data = dplyr::mutate(dia
##     cut = factor(cut, ordered = FALSE)))
##
## Coefficients:
##  (Intercept)        cutGood  cutVery Good     cutPremium       cutIdeal
##    -4385.030        655.767       848.717        869.396        998.254
##      color.L        color.Q       color.C        color^4        color^5
##    -1910.288       -627.954      -171.960         21.678        -85.943
##      color^6          carat     clarity.L      clarity.Q      clarity.C
##      -49.986       8886.129      4217.535      -1832.406        923.273
##    clarity^4      clarity^5     clarity^6      clarity^7
##     -361.995        216.616         2.105        110.340
```

Example: Discretization of carat

```
# Carat is on the rhs as a continuous and categorical variable
update(fit,
       formula = . ~ . + cut(carat, breaks = c(0, 0.5, 1.0, 2.0, 5.0))
       )$call

## lm(formula = price ~ cut + color + carat + cut(carat, breaks = c(0,
##     0.5, 1, 2, 5)), data = diamonds)

# cut and color are missing from the rhs
update(fit,
       formula = . ~ cut(carat, breaks = c(0, 0.5, 1.0, 2.0, 5.0))
       )$call

## lm(formula = price ~ cut(carat, breaks = c(0, 0.5, 1, 2, 5)),
##     data = diamonds)
```

Example: Discretization of carat

```
# correct specification
fit <- update(fit,
              formula = . ~ . - carat +
                            cut(carat, breaks = c(0, 0.5, 1.0, 2.0, 5.0)))
fit$call

## lm(formula = price ~ cut + color + cut(carat, breaks = c(0, 0.5,
##     1, 2, 5)), data = diamonds)
```

Example: Change the arguments passed to `cut`

```
# this does not work.  breaks needs to be passed to cut
update(fit, breaks = c(0, 2, 5))

## Warning in lm.fit(x, y, offset = offset, singular.ok = singular.ok,
...):  extra argument 'breaks' is disregarded.

##
## Call:
## lm(formula = price ~ cut + color + cut(carat, breaks = c(0, 0.5,
##     1, 2, 5)), data = diamonds, breaks = c(0, 2, 5))
##
## Coefficients:
##                                    (Intercept)
##                                        483.250
##                                          cut.L
##                                        502.312
##                                          cut.Q
##                                       -318.269
##                                          cut.C
##                                        159.146
##                                          cut^4
##                                          2.116
##                                        color.L
##                                      -1044.815
##                                        color.Q
```

Want: Update the breaks in the `formula` passed to `lm`

```r
# Define a function new_breaks to update the breaks argument in cut
# within a formula.
new_breaks <- function(form, brks) {
  rr <- function(x, brks) {
      if(is.call(x) && grepl("cut", deparse(x[[1]]))) {
          x$breaks <- brks
          x
      } else if (is.recursive(x)) {
          as.call(lapply(as.list(x), rr, brks))
      } else {
          x
      }
  }

  z <- lapply(as.list(form), rr, brks)
  z <- eval(as.call(z))
  environment(z) <- environment(form)
  z
}
```

Old, Update, and Updated `calls`

```
# original call
fit$call

## lm(formula = price ~ cut + color + cut(carat, breaks = c(0, 0.5,
##     1, 2, 5)), data = diamonds)

# update the call
fit <- update(fit,
              formula = new_breaks(formula(fit), brks = c(0, 2.0, 5.0)))

# view updated call
fit$call

## lm(formula = price ~ cut + color + cut(carat, breaks = c(0, 2,
##     5)), data = diamonds)
```

Wrap Up

- Functions like `new_breaks` are very useful when running automated tasks, perhaps searching for an 'optimum' binning of `carat`?
- A great exercise for dealing with
  - Non-standard evaluation (see "Advanced R" by Wickham `http://adv-r.had.co.nz/Computing-on-the-language.html`)
  - Recursion
  - Working with `environments`