refactor

Henrik Renlund

April 1, 2015

Contents

1 Purpose

The function[1] `refactor` was build mainly to be able to easily rename, reorder, and merge the levels of a factor. It can also create and exclude levels, and set `NA` to be a level.

Motivating example:

```
lev <- c(" ", "never", "some", "a lot")
smoking <- factor(sample(x=lev, 100, TRUE), levels=lev)
table(smoking)

## smoking
##       never  some a lot
##    28    23    27    22
```

Suppose you would want to clean up the variable `smoking` to have only levels ”no” (previously ’never’), ”yes” (previously ’some’ and ’a lot’) and ”unknown” (previously ’ ’) - and in that order.

[1]The name `relevel` was taken!

```
L <- list(never="no", some=c("yes", "a lot"), " "="unknown")
table(refactor(smoking, L))

##
##      no     yes unknown
##      23      49      28
```

2 Basic functionality

2.1 Renaming and reordering levels

Levels can be renamed by providing a list with entries of the old level name that equals a new name.

```
(x <- factor(LETTERS[1:2]))

## [1] A B
## Levels: A B

L <- list(A="newA", B="newB")
refactor(x,L)

## [1] newA newB
## Levels: newA newB
```

The order of the list determines the order of the levels created.

```
(x <- factor(LETTERS[1:2], levels=LETTERS[2:1]))

## [1] A B
## Levels: B A

L <- list(A="newA", B="newB")
refactor(x,L)

## [1] newA newB
## Levels: newA newB
```

If you simply want to reorder you must provide a `NULL` new name.

```
(x <- factor(LETTERS[1:3]))

## [1] A B C
## Levels: A B C

L <- list(B=NULL, A=NULL, C=NULL)
refactor(x,L)
```

```
## [1] A B C
## Levels: B A C
```

The list L need not cover all existing levels. If not, the remaining levels will be added in the same order as they appeared in the original.

```
(x <- factor(LETTERS[1:5]))

## [1] A B C D E
## Levels: A B C D E

L <- list(D=NULL, B="newB")
refactor(x,L)

## [1] A    newB C    D    E
## Levels: D newB A C E
```

2.2 Adding and merging levels

If L includes a non-existing level, this level will be added.

```
(x <- factor(LETTERS[1:2]))

## [1] A B
## Levels: A B

L <- list(Foo=NULL)
refactor(x,L)

## [1] A B
## Levels: Foo A B
```

There is an option to put these new levels at the end of the levels created.

```
(x <- factor(LETTERS[1:2]))

## [1] A B
## Levels: A B

L <- list(Foo=NULL, Bar=NULL)
refactor(x,L, new.last=TRUE)

## [1] A B
## Levels: A B Foo Bar
```

If you want to merge two levels this can be acheived by including one in the other, as is best shown by example:

```
(x <- factor(LETTERS[1:3]))

## [1] A B C
## Levels: A B C

L <- list(A="B") # level B to be absorbed by level A
refactor(x,L)

## [1] A A C
## Levels: A C
```

Note that the above did not rename `A` as `B`. The entry `A` in list `L` can contain a new name as well as levels to be absorbed. Thus it cannot be renamed as an existing level.

```
(x <- factor(LETTERS[1:3]))

## [1] A B C
## Levels: A B C

L <- list(A=c("merge_AB", "B"))
refactor(x,L)

## [1] merge_AB merge_AB C
## Levels: merge_AB C
```

The same thing can be acheived by

```
(x <- factor(LETTERS[1:3]))

## [1] A B C
## Levels: A B C

L <- list(B=c("merge_AB", "A"))
refactor(x,L)

## [1] merge_AB merge_AB C
## Levels: merge_AB C
```

The order of entry `B` is unimportant, but the first non-level name will be used as a new name.

```
(x <- factor(LETTERS[1:3]))

## [1] A B C
## Levels: A B C
```

```r
L <- list(B=c("A", "merger", "new_name"))
refactor(x,L)

## Warning in refactor(x, L): [refactor] Multiple new names for level
'B'. Only the first one will be used.

## [1] merger merger C
## Levels: merger C
```

One cannot merge an existing level into more than one place, the first encountered will be used.

```r
(x <- factor(LETTERS[1:3]))

## [1] A B C
## Levels: A B C

L <- list(B=c("merge_AB","A"), C=c("new_C", "A"))
refactor(x,L)

## Warning in refactor(x, L): [refactor] Key 'C' has entries used previously.
These will be skipped

## [1] merge_AB merge_AB new_C
## Levels: merge_AB new_C
```

2.3 The empty string

One cannot specify the element name in a list explicitly to be the empty string

```r
# list(a_name="value", ""="another_value") # not possible
list(a_name="value", "another_value") # possible

## $a_name
## [1] "value"
##
## [[2]]
## [1] "another_value"
```

Therefore, if you want to recode a empty string value can do

```r
x <- c("A", "", "B")
L <- list("A"="New A", "not_so_empty_now", "B"="New B")
refactor(x, L)
```

```
## [1] New A             not_so_empty_now New B
## Levels: New A not_so_empty_now New B
```

2.4 Excluding and adding `NA` as level

You can delete levels, i.e. turning it into `NA`.

```
(x <- factor(LETTERS[1:4]))

## [1] A B C D
## Levels: A B C D

refactor(x,exclude=c("A","C"))

## [1] <NA> B    <NA> D
## Levels: B D
```

You can make `NA` into a level `missing` by setting `na.level` to any non null value

```
(x <- factor(c(LETTERS[1:2],NA)))

## [1] A    B    <NA>
## Levels: A B

refactor(x,na.level=TRUE)

## [1] A       B       missing
## Levels: A B missing
```

or specify any name by setting `na.level` to a string

```
(x <- factor(c(LETTERS[1:2],NA)))

## [1] A    B    <NA>
## Levels: A B

refactor(x,na.level="saknas")

## [1] A      B      saknas
## Levels: A B saknas
```

3 Order of operations

The function operates in the following order:

- if wanted, turn `NA` into a level
- if wanted, turn certain levels into `NA`
- if wanted, rename and reorder levels

Thus you can delete levels whilst making original `NA` into a new level.

```
(x <- factor(c(LETTERS[1:2],NA)))

## [1] A    B    <NA>
## Levels: A B

refactor(x, exclude="A", na.level=TRUE)

## [1] <NA>    B       missing
## Levels: B missing
```

But would you want to consider both `A` and `NA` as `missing` you would have to 'rename' `NA` and let that name be absorbed by `A` which takes the new name 'missing'.

```
(x <- factor(c(LETTERS[1:2],NA)))

## [1] A    B    <NA>
## Levels: A B

L <- list(A=c("missing", "NA_name"))
refactor(x, L, na.level="NA_name")

## [1] missing B       missing
## Levels: missing B
```