Let’s discuss the [paper retracted in JAMA](https://jamanetwork.com/journals/jama/fullarticle/2752474) this week.

During analysis, the treatment variable coded [1,2] was recoded in error to [1,0]. The results of the analysis were therefore reversed. The [lung-disease self-management program](https://jamanetwork.com/journals/jama/fullarticle/2752467) actually resulted in more attendances at hospital, rather than fewer as had been [originally reported](https://jamanetwork.com/journals/jama/fullarticle/2714645).

**Recode check**

Checking of recoding is such an important part of data cleaning – we emphasise this a lot in HealthyR courses – but of course mistakes happen.

Our standard approach is this:

library(finalfit)

colon\_s %>%

mutate(

sex.factor2 = forcats::fct\_recode(sex.factor,

"F" = "Male",

"M" = "Female")

) %>%

count(sex.factor, sex.factor2)

# A tibble: 2 x 3

sex.factor sex.factor2 n

1 Female M 445

2 Male F 484

The miscode should be obvious.

**check\_recode()**

However, mistakes may still happen and be missed. So we’ve bashed out a useful function that can be applied to your whole dataset. This is not to replace careful checking, but may catch something that has been missed.

The function takes a data frame or tibble and fuzzy matches variable names. It produces crosstables similar to above for all matched variables.

So if you have coded something from sex to sex.factor it will be matched. The match is hungry so it is more likely to match unrelated variables than to miss similar variables. But if you recode death to mortality it won’t be matched.

Here’s a walk through.

# Install

devtools::install\_github('ewenharrison/finalfit')

library(finalfit)

library(dplyr)

# Recode example

colon\_s\_small = colon\_s %>%

select(-id, -rx, -rx.factor) %>%

mutate(

age.factor2 = forcats::fct\_collapse(age.factor,

"<60 years" = c("<40 years", "40-59 years")),

sex.factor2 = forcats::fct\_recode(sex.factor,

# Intentional miscode

"F" = "Male",

"M" = "Female")

)

# Check

colon\_s\_small %>%

check\_recode()

$index

# A tibble: 3 x 2

var1 var2

1 sex.factor sex.factor2

2 age.factor age.factor2

3 sex.factor2 age.factor2

$counts

$counts[[1]]

# A tibble: 2 x 3

sex.factor sex.factor2 n

1 Female M 445

2 Male F 484

$counts[[2]]

# A tibble: 3 x 3

age.factor age.factor2 n

1 <40 years <60 years 70

2 40-59 years <60 years 344

3 60+ years 60+ years 515

$counts[[3]]

# A tibble: 4 x 3

sex.factor2 age.factor2 n

1 M <60 years 204

2 M 60+ years 241

3 F <60 years 210

4 F 60+ years 274

As can be seen, the output takes the form of a list length 2. The first is an index of matched variables. The second is crosstables as tibbles for each variable combination. sex.factor2 can be seen as being miscoded. sex.factor2 and age.factor2 have been matched, but should be ignored.

Numerics are not included by default. To do so:

out = colon\_s\_small %>%

select(-extent, -extent.factor,-time, -time.years) %>% # choose to exclude variables

check\_recode(include\_numerics = TRUE)

out

# Output not printed for space

**Miscoding in survival::colon dataset?**

When doing this just today, we noticed something strange in our example dataset, survival::colon.

The variable node4 should be a binary recode of nodes greater than 4. But as can be seen, something is not right!

We’re interested in any explanations those working with this dataset might have.

# Select a tibble and expand

out$counts[[9]] %>%

print(n = Inf)

# Compressed output shown

# A tibble: 32 x 3

nodes node4 n

1 0 0 2

2 1 0 269

3 1 1 5

4 2 0 194

5 3 0 124

6 3 1 1

7 4 0 81

8 4 1 3

9 5 0 1

10 5 1 45

# … with 22 more rows

There we are then, a function that may be useful in detecting miscoding. So useful in fact, that we have immediately found probable miscoding in a standard R dataset.