Wrangling categorical data in R

Amelia McNamara[1]

[1]**Smith College**

ABSTRACT

Working with categorical data in R (known as factor variables) can be particularly tricky. This paper presents a few approaches to wrangling this type of data, using the base R package as well as dplyr and mosaic.

Keywords:

INTRODUCTION

Categorical data is important in data science, particularly because so much survey data is categorical. While defensive programming is always important when working with data, categorical data presents some particular problems that can slip in without the analyst noticing.

In this paper, we discuss some places where analysts get tripped up when working with factor variables (R's data type for categorical data).

HISTORY OF FACTORS IN R

Consider a gender variable including the categories `male`, `female` and `gender non-conforming`. In R, there are two ways to store this information. One is to use a series of character strings, and the other is to store it as a factor.

Historically, storing categorical data as a factor variable was more efficient than storing the same data as strings, because factor variables only store the factor labels once (Peng, 2015). However, R has changed to use hashed versions of all character strings, so the storage issue is no longer valid (Peng, 2015).

Factors can be very tricky to deal with, which has led to the online `stringsAsFactors = HELLNO` movement. This refers to the default behavior of many of R's data import functions to take any variable composed as strings and automatically convert the variable to a factor. The R community has been moving away from this default behavior, with functions from Hadley Wickham's **readr** package defaulting to leaving strings as-is.

However, factor variables are important when it comes to modeling. When you pass a factor variable into **lm** or **glm**, R automatically creates dummy variables for each of the levels and picks one as a reference group. This behavior is lost if the variable is stored as a character vector.

So, factors are important. But, they can often be hard to deal with. Because of the way the group numbers are stored separately from the factor labels, it can be easy to overwrite data in such a way that the original data is lost. In this paper, we will consider the best practices for working with factor data.

To do this, we will consider data from the General Social Survey.

LOADING THE DATA

We have several options for how to get this data. We could download it in SPSS or Stata formats and use the foreign package to read it in. The GSS download even provides an R file to do the translation for you. Here is the result of that:

```r
source('../data/GSS.r')
```

```
str(GSS)

## 'data.frame': 2538 obs. of  17 variables:
##  $ YEAR    : int  2014 2014 2014 2014 2014 2014 2014 2014 2014 2014 ...
##  $ ID_     : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ WRKSTAT : int  1 1 4 2 5 1 9 1 8 1 ...
##  $ PRESTIGE: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ MARITAL : int  3 1 3 1 1 1 1 1 5 1 ...
##  $ CHILDS  : int  0 0 1 2 3 1 2 2 4 3 ...
##  $ AGE     : int  53 26 59 56 74 56 63 34 37 30 ...
##  $ EDUC    : int  16 16 13 16 17 17 12 17 10 15 ...
##  $ SEX     : int  1 2 1 2 2 2 1 1 2 2 ...
##  $ RACE    : int  1 1 1 1 1 1 1 1 1 3 ...
##  $ INCOM16 : int  2 3 2 2 4 4 2 3 3 1 ...
##  $ INCOME  : int  12 12 12 12 13 12 13 12 10 12 ...
##  $ RINCOME : int  12 12 0 9 0 12 13 12 0 12 ...
##  $ INCOME72: int  0 0 0 0 0 0 0 0 0 0 ...
##  $ PARTYID : int  5 5 6 5 3 6 6 8 3 3 ...
##  $ FINRELA : int  4 4 2 4 3 4 9 3 2 3 ...
##  $ SEXORNT : int  3 3 3 3 3 9 0 0 3 3 ...
##  - attr(*, "col.label")= chr  "Gss year for this respondent                       " "Respondent
```

Obviously, this is less than ideal. Now, all the factor variables are encoded as integers, but their level labels have been lost. We have to look at a codebook to determine if `SEX == 1` indicates male or female. We would rather preserve the integrated level labels. In order to do this, our best option is to download the data as an Excel file and use the **readxl** package to load it.

```
library(readxl)
GSS <- read_excel("../data/GSS.xls")
names(GSS) <- make.names(names(GSS), unique=TRUE)
str(GSS)

## Classes 'tbl_df', 'tbl' and 'data.frame': 2540 obs. of  17 variables:
##  $ Gss.year.for.this.respondent.......................: num  2014 2014 2014 2014 2014 ...
##  $ Respondent.id.number                               : num  1 2 3 4 5 6 7 8 9 10 ...
##  $ Labor.force.status                                 : chr  "Working fulltime" "Working fulltin
##  $ Rs.occupational.prestige.score...1970.             : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Marital.status                                     : chr  "Divorced" "Married" "Divorced" "Ma
##  $ Number.of.children                                 : num  0 0 1 2 3 1 2 2 4 3 ...
##  $ Age.of.respondent                                  : chr  "53.000000" "26.000000" "59.000000"
##  $ Highest.year.of.school.completed                   : num  16 16 13 16 17 17 12 17 10 15 ...
##  $ Respondents.sex                                    : chr  "Male" "Female" "Male" "Female" ..
##  $ Race.of.respondent                                 : chr  "White" "White" "White" "White" ..
##  $ Rs.family.income.when.16.yrs.old                   : chr  "Below average" "Average" "Below av
##  $ Total.family.income                                : chr  "$25000 or more" "$25000 or more" "
##  $ Respondents.income                                 : chr  "$25000 or more" "$25000 or more" "
##  $ Total.family.income.1                              : chr  "Not applicable" "Not applicable" "
##  $ Political.party.affiliation                        : chr  "Not str republican" "Not str repub
##  $ Opinion.of.family.income                           : chr  "Above average" "Above average" "Be
##  $ Sexual.orientation                                 : chr  "Heterosexual or straight" "Heteros
```

That's a little better. Now we have preserved the character strings. But, the data is not yet useable in an analysis.

RENAMING THE VARIABLES

One problem is that the variable names (while human readable) are full of spaces, so are hard to use. But, we can rename them.

There is a fragile way to do this in **base** R, but we'll use the more robust `rename()` function from the **dplyr** package. `rename()`

```
library(dplyr)
```

```
GSS <- GSS %>%
  rename(LaborStatus = Labor.force.status,
         PolParty = Political.party.affiliation,
         Age = Age.of.respondent)
```

```
sessionInfo()

## R version 3.3.0 (2016-05-03)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X 10.11.5 (El Capitan)
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
## [1] dplyr_0.4.3.9001 readxl_0.1.1     foreign_0.8-66   knitr_1.13
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.5     packrat_0.4.7-1 assertthat_0.1  R6_2.1.2
##  [5] DBI_0.4-1       formatR_1.2.1   magrittr_1.5    evaluate_0.9
##  [9] highr_0.5.1     stringi_1.1.1   lazyeval_0.2.0  tools_3.3.0
## [13] stringr_1.0.0   tibble_1.0
```

CONSIDERING SOME FACTOR VARIABLES

Once we have variable names that are easier to work with, we can begin to think about how the data should be cleaned.

```
GSS <- GSS %>%
  mutate(LaborStatus = factor(LaborStatus),
         PolParty = factor(PolParty))

levels(GSS$LaborStatus) # I wish I had a piece of dplyr code for this

## [1] "Keeping house"    "No answer"        "Other"
## [4] "Retired"          "School"           "Temp not working"
## [7] "Unempl, laid off" "Working fulltime" "Working parttime"

levels(GSS$PolParty)

##  [1] "Don't know"         "Ind,near dem"       "Ind,near rep"
##  [4] "Independent"        "No answer"          "Not str democrat"
##  [7] "Not str republican" "Other party"        "Strong democrat"
## [10] "Strong republican"
```

CHANGING THE LABELS OF FACTORS (BASE R)

One action you might want to take is just to change the text of one (or more) of the factor labels, so it appears more nicely formatted in a **ggplot2** plot, for example.

Here is how I do that in base R. Typically, I end up ruining something in the process of doing this, so I *always* start with a summary call, to check after I have done my attempt.

```
summary(GSS$LaborStatus)

##    Keeping house        No answer            Other          Retired
##              263                2               76              460
##           School Temp not working Unempl, laid off Working fulltime
##               90               40              104             1230
## Working parttime             NA's
##              273                2
```

```
levels(GSS$LaborStatus) <- c(levels(GSS$LaborStatus)[1:5],
                             "Temporarily not working",
                             "Unemployed, laid off",
                             "Working full time",
                             "Working part time")
summary(GSS$LaborStatus)

##           Keeping house               No answer                   Other
##                     263                       2                      76
##                 Retired                  School Temporarily not working
##                     460                      90                      40
##    Unemployed, laid off       Working full time       Working part time
##                     104                    1230                     273
##                    NA's
##                       2
```

This method is less than ideal, because it depends on the data coming in with the factor levels ordered in a particular way. If the data gets changed outside of R, this code will not perform as expected. Additionally, if the code gets run more than once, it can also lead to unexpected behavior.

CHANGING THE LABELS OF FACTORS (DPLYR)

In **dplyr**, you can use the `recode` function to do the same thing. There are a few things to remember with `recode`. The first is that it is a vector function, which means it must be used within a `mutate` call or with a variable pulled out using `$`. The second is that you need to tell it which variable to recode, even if you are overwriting an existing variable.

```
GSS <- GSS %>%
  mutate(PolParty =  recode(PolParty, `Not str republican` = "Not a strong republican"))
```

COMBINING SEVERAL LEVELS INTO ONE

This is another common task. Maybe you want fewer coefficients to interpret in your model, or the process that generated the data makes a finer distinction between categories than your research. For whatever the reason, you want to group together levels that are currently separate.

How I do this in base R:

```
levels(GSS$LaborStatus) <- c("Not employed", "No answer",
                             "Other", "Not employed",
                             "Not employed", "Not employed",
                             "Not employed", "Employed", "Employed")
summary(GSS$LaborStatus)

## Not employed    No answer        Other     Employed         NA's
##          957            2           76         1503            2
```

1 MOSAIC COMBINING LEVELS

```
library(mosaic)
data(Births78)
Births78 <- Births78 %>%
  mutate(weekend = derivedFactor(weekend = wday== "Sun" | wday == "Sat", .default="weekday"))
```

COMBINING MANY CATEGORIES INTO ONE

In this data, age is provided as an integer for respondents 18-88, but then also includes the possible answer ”89 or older” as well as a possible ”No answer” and NA values.

```
GSS <- GSS %>%
  mutate(Age = factor(Age))
summary(GSS$Age)

##   18.000000   19.000000   20.000000   21.000000   22.000000   23.000000
##           6          25          26          24          28          30
##   24.000000   25.000000   26.000000   27.000000   28.000000   29.000000
##          31          48          47          41          31          51
##   30.000000   31.000000   32.000000   33.000000   34.000000   35.000000
##          57          49          55          47          46          40
##   36.000000   37.000000   38.000000   39.000000   40.000000   41.000000
##          40          54          47          52          46          54
##   42.000000   43.000000   44.000000   45.000000   46.000000   47.000000
##          35          54          39          41          34          43
##   48.000000   49.000000   50.000000   51.000000   52.000000   53.000000
##          32          39          54          45          37          60
##   54.000000   55.000000   56.000000   57.000000   58.000000   59.000000
##          53          52          60          43          60          47
##   60.000000   61.000000   62.000000   63.000000   64.000000   65.000000
##          46          38          44          42          38          40
##   66.000000   67.000000   68.000000   69.000000   70.000000   71.000000
##          35          41          21          23          32          28
##   72.000000   73.000000   74.000000   75.000000   76.000000   77.000000
##          20          22          25          21          24          17
##   78.000000   79.000000   80.000000   81.000000   82.000000   83.000000
##          28          26          16          14           8          11
##   84.000000   85.000000   86.000000   87.000000   88.000000 89 or older
##          13           6           9           8          11          19
##   No answer        NA's
##           9           2
```

We might want to turn this into a factor variable with two levels: 18-65, and over 65. In this case, it would be much easier to deal with a conditional statement about the numeric values, rather than writing out each of the numbers as a character vector.

But, in order to do that we need to make it numeric.

```
# GSS$Age[GSS$Age == "No answer"] <- NA # Do I really need this? Nope!
levels(GSS$Age) <- c(levels(GSS$Age)[1:71], "89", "No answer")
GSS$Age <- as.numeric(as.character(GSS$Age))
summary(GSS$Age)

##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   18.00   34.00   49.00   49.01   62.00   89.00      11
```

Of course, we're cheating a little bit here– if we were going to use this as a numeric variable in an analysis, we wouldn't necessarily want to turn all the ”89 or older” cases into the number ”89”. But, we're just on our way to a two-category factor, so those cases would have gone to the ”65 and up” category one way or the other.

```
GSS <- GSS %>%
  mutate(Age = if_else(Age<65, "18-65", "65 and up"),
         Age = factor(Age))
summary(GSS$Age)

##     18-65 65 and up      NA's
##      2011       518        11
```

Another way to do this:

```
# young <- as.character(18:64)
# derivedVariable(Age %in% young = "18-65", Age )
```

ACKNOWLEDGEMENTS

Thanks to my students Kelcie Grenier, Kat Kyuchukov, and Emily Ruppel, whose spring 2016 project in my SDS 291 class formed the inspiration for this paper.

IDEAS FROM NICK

Two ways to do each thing (as long as one isn't totally stupid) Why is this hard? Why is this error-prone? Missing values A few exercises for summer students Appendices for less interesting examples?

REFERENCES

Peng, R. D. (2015). stringsAsFactors: An unauthorized biography. http://simplystatistics.org/2015/07/24/stringsasfactors-an-unauthorized-biography/.