

1 Wrangling categorical data in R

2 Amelia McNamara¹

3 ¹Smith College

4 ABSTRACT

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

8 Keywords:

9 INTRODUCTION

10 Factors are the data type that R uses for categorical data. For example, a gender variable might include
11 the categories `male`, `female` and `gender non-conforming`. Storing this information as a factor
12 is the alternative to storing it as a series of character strings.

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15 to use hashed versions of all character strings, so the storage issue is no longer valid (Peng, 2015).

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

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

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

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

28 LOADING THE DATA

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

```
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 2014 ...
## $ Respondent.id.number : num 1 2 3 4 5 6 7 8 9 10 ...
## $ Labor.force.status : chr "Working fulltime" "Working fulltime" ...
## $ Rs.occupational.prestige.score...1970. : num 0 0 0 0 0 0 0 0 0 0 ...
## $ Marital.status : chr "Divorced" "Married" "Divorced" "Married" ...
## $ 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 average" ...
## $ 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 republican" ...
## $ Opinion.of.family.income : chr "Above average" "Above average" ...
## $ Sexual.orientation : chr "Heterosexual or straight" "Heterosexual or straight" ...
```

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`) %>%
  rename(PolParty = `Political party affiliation`) %>%
  rename(Age = `Age of respondent`)

## Error: Unknown variables: Labor force status.
```

43 CONSIDERING SOME FACTOR VARIABLES

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

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

## Error in eval(expr, envir, enclos): object 'LaborStatus' not found

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

## Error: Unknown column 'LaborStatus'

levels(GSS$PolParty)

## Error: Unknown column 'PolParty'
```

46 CHANGING THE LABELS OF FACTORS (BASE R)

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

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

```
summary(GSS$LaborStatus)

## Error: Unknown column 'LaborStatus'

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

## Error: Unknown column 'LaborStatus'

summary(GSS$LaborStatus)

## Error: Unknown column 'LaborStatus'
```

51 CHANGING THE LABELS OF FACTORS (DPLYR)

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

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

## Error in eval(expr, envir, enclos): object 'PolParty' not found
```

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")

## Error: Unknown column 'LaborStatus'

summary(GSS$LaborStatus)

## Error: Unknown column 'LaborStatus'
```

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))

## Error in eval(expr, envir, enclos): object 'Age' not found

summary(GSS$Age)

## Error: Unknown column 'Age'
```

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")

## Error: Unknown column 'Age'

GSS$Age <- as.numeric(as.character(GSS$Age))

## Error: Unknown column 'Age'

summary(GSS$Age)

## Error: Unknown column 'Age'
```

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")) %>%
  mutate(Age = factor(Age))

## Error in eval(expr, envir, enclos): object 'Age' not found

summary(GSS$Age)

## Error: Unknown column 'Age'
```

73 Another way to do this:

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

74 ACKNOWLEDGEMENTS

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

77 IDEAS FROM NICK

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

80 REFERENCES

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