Wrangling categorical data in R

Amelia McNamara[1] and Nicholas J Horton[2]

[1]**Smith College**
[2]**Amherst College**

ABSTRACT

Wrangling of categorical data is an important part of the analysis cycle. Many aspects of these operations can be tricky, particularly for complex transformations. This paper discusses aspects of transformation of categorical variables in R. We suggest defensive coding strategies and principles for data wrangling to ensure data quality and sound analysis.

Keywords:

INTRODUCTION

The wrangling of categorical data is an important component in data science because so many variables are categorical. Gender, income bracket, and state are all examples of categorical data. While defensive coding is important for any analysis, categorical data presents particular problems that can slip in without the analyst noticing.

In this paper, we consider a number of common idioms that often arise in data cleaning and preparation, propose some guidelines for defensive coding, and discuss some settings where analysts often get tripped up when working with categorical variables and factors (R's data type for categorical data).

In particular, we are considering how categorical data is treated in **base** R versus the so-called tidyverse (Wickham, 2014). Tools from the tidyverse are discussed in another paper in this special issue, but briefly they aim to make analysis more pure, predictable, and pipeable. One canonical thought exercise is whether, after the analysis, a new version of the data could be supplied in the code and have parallel results come out the other end (Broman, 2015). Again, categorical data can make this task even more complex.

THE IMPORTANCE OF TOOLING

This is where I think we will make the case about how base R is both fragile and often does the wrong thing (like the canonical first example) while the tidyverse is better. Better? More sophisticated? Tidy? Maybe the word ”affordances” is warranted?

These problems are hard, common, and important.

Spreadsheets can lead to many problems, particularly with categorical data.

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 a consideration (Peng, 2015).

Factors can be very tricky to deal with, since many operations applied to them return different values than when applied to character vectors. As an example, consider a set of decades,

```
x1 <- c(10, 10, 20, 20, 40)
```

```
x1f <- factor(x1)
ds <- data.frame(x1, x1f)
library(dplyr)
ds <- mutate(ds, x1recover = as.numeric(x1f))
ds

##   x1 x1f x1recover
## 1 10  10         1
## 2 10  10         1
## 3 20  20         2
## 4 20  20         2
## 5 40  40         3
```

This is unexpected because `as.numeric()` feels like the way to recover numeric information in the **base** R paradigm. Compare the following:

```
as.numeric(c("hello"))

## [1] NA

as.numeric(factor(c("hello")))

## [1] 1
```

This behavior has led to an online movement against 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 tidyverse moves away from this default behavior, with functions from the **readr** package defaulting to leaving strings as-is. (Others have chosen to add `options(stringAsFactors=FALSE)` into their startup commands.)

Although the storage issues have been solved, and there are problems with defaulting strings to factors, factors are still necessary for some data analytic tasks. The most salient case is in 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. Factor variables also allow for the possibility of ordering between classes. Text strings `low, medium, high` would not preserve the ordering inherent in the groups. Again, this can be important for modeling when doing ordinal logistic regression and multinomial logistic regression. 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. There are some import issues inherent to the data which are not particular to categorical data, so that processing is in Appendices A, B. We'll work with the data that has cleaned variable names.

```
GSS <- read.csv("../data/GSScleaned.csv")
str(GSS)

## 'data.frame': 2540 obs. of  17 variables:
##  $ Gss.year.for.this.respondent.......................: int  2014 2014 2014 2014 2014 2014 2014
##  $ Respondent.id.number                               : int  1 2 3 4 5 6 7 8 9 10 ...
##  $ LaborStatus                                        : Factor w/ 9 levels "Keeping house",..: 
##  $ Rs.occupational.prestige.score...1970.             : int  0 0 0 0 0 0 0 0 0 0 ...
##  $ Marital.status                                     : Factor w/ 6 levels "Divorced","Married",
##  $ Number.of.children                                 : int  0 0 1 2 3 1 2 2 4 3 ...
##  $ Age                                                : Factor w/ 73 levels "18.000000","19.000
##  $ Highest.year.of.school.completed                   : int  16 16 13 16 17 17 12 17 10 15 ...
##  $ Respondents.sex                                    : Factor w/ 2 levels "Female","Male": 2 1
##  $ Race.of.respondent                                 : Factor w/ 3 levels "Black","Other",..: 
##  $ Rs.family.income.when.16.yrs.old                   : Factor w/ 7 levels "Above average",..: 
##  $ Total.family.income                                : Factor w/ 14 levels "$1000 to 2999",..:
##  $ Respondents.income                                 : Factor w/ 15 levels "$1000 to 2999",..:
##  $ Total.family.income.1                              : Factor w/ 1 level "Not applicable": 1 1
##  $ PolParty                                           : Factor w/ 10 levels "Don't know","Ind,ne
##  $ Opinion.of.family.income                           : Factor w/ 7 levels "Above average",..: 
##  $ Sexual.orientation                                 : Factor w/ 6 levels "Bisexual","Dont know
```

The rest of this paper is arranged around case studies:

1. Changing the labels of factor levels
2. Reordering factor levels
3. Combining several levels into one. Both string-like labels and numeric, probably go together.
4. Making derived factor variables

CHANGING THE LABELS OF FACTOR LEVELS

For this example, we will be considering the labor status variable. It has 9 factor levels. Most of the labels are spelled out fully, but a few are strangely formatted. We want to change this.

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.

There are two typical approaches in base R. One is more compact, but depends on the levels of the factor not changing in the data being fed in, and the other is more robust, but extremely verbose. In contrast, the **dplyr** package offers a method that is much more human readable, while also supporting reproducibility.

Compact but fragile (base R)

```
levels(GSS$LaborStatus)

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

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

with(GSS, summary(LaborStatus))   # I prefer this to the $

##    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, for example so that ”Working full time” becomes the first level, the code will silently fail. There is also the problem of additional factor levels being added after the fact. In our experience, both with students and scientific collaborators, spreadsheet data can be easily changed in these ways.

Robust but verbose (base R)

The more robust method in **base** R is to use subsetting to overwrite particular values in the data.

```
summary(GSS$PolParty)

##         Don't know        Ind,near dem        Ind,near rep
##                  1                 337                 249
##        Independent           No answer   Not str democrat
##                502                  25                 406
## Not str republican         Other party     Strong democrat
##                292                  62                 419
##  Strong republican                NA's
##                245                   2

GSS$PolParty <- as.character(GSS$PolParty)
GSS$PolParty[GSS$PolParty=="Ind,near dem"] <- "Independent, near democrat"
GSS$PolParty[GSS$PolParty == "Ind,near rep"] <- "Independent, near republican"
GSS$PolParty[GSS$PolParty == "Not str democrat"] <- "Not strong democrat"
GSS$PolParty <- factor(GSS$PolParty)
summary(GSS$PolParty)

##                   Don't know                  Independent
##                            1                          502
##   Independent, near democrat Independent, near republican
##                          337                          249
##                    No answer           Not str republican
##                           25                          292
##          Not strong democrat                  Other party
##                          406                           62
##              Strong democrat            Strong republican
##                          419                          245
##                         NA's
##                            2
```

Obviously, this can get tedious, and it is possible to miss cases.

Direct and robust (dplyr)

In the **dplyr** package, you can use the `recode` function to recode factor levels the same thing. `recode` is a vector function, which means it must be used within a `mutate` call or with a variable pulled out using `$`. Somewhat differently from other **dplyr** functions, you must specify 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"))
```

Aside – Editing whitespace out of levels

Whitespace can be dealt with when data is read, or later using string manipulations. The easiest way to do this is using **base** R.

```
gender <-factor(c("male ", "male  ", "male    ", "male"))
levels(gender)

## [1] "male"     "male "    "male  "   "male    "

gender <- factor(trimws(gender))
levels(gender)

## [1] "male"
```

REORDERING FACTOR LEVELS

Often, factor levels have a natural ordering to them. However, the default in **base** R is to order levels alphabetically. Again, there is a fragile way to reorder the factor levels in base R, and a more robust method in the tidyverse.

Fragile method (base R)

```
summary(GSS$Opinion.of.family.income)

##     Above average           Average     Below average        Don't know
##               483              1118               666                21
## Far above average Far below average         No answer              NA's
##                65               179                 6                 2

levels(GSS$Opinion.of.family.income)

## [1] "Above average"     "Average"           "Below average"
## [4] "Don't know"        "Far above average" "Far below average"
## [7] "No answer"

levels(GSS$Opinion.of.family.income) <- levels(GSS$Opinion.of.family.income)[c(5,1:3,6,4,7)]
levels(GSS$Opinion.of.family.income)

## [1] "Far above average" "Above average"     "Average"
## [4] "Below average"     "Far below average" "Don't know"
## [7] "No answer"
```

This is both verbose and depends on the number and order of the levels staying the same. Luckily, if another factor level is added, the above code will throw an error because the `number of levels differs`.

However, if the code gets run more than once, the order will be broken. Particularly when working dynamically, this is all too easy to do.

```
levels(GSS$Opinion.of.family.income) <- levels(GSS$Opinion.of.family.income)[c(5,1:3,6,4,7)]
levels(GSS$Opinion.of.family.income)

## [1] "Far below average" "Far above average" "Above average"
## [4] "Average"           "Don't know"        "Below average"
## [7] "No answer"
```

The more times the code is run, the worse it gets.

Not sure if I should include this.... it gets worse! It is soooo tempting to write this code, which ruins your data completely.

```
test <- GSS$Opinion.of.family.income
summary(test)

## Far below average Far above average     Above average           Average
##               483              1118               666                21
##        Don't know     Below average         No answer              NA's
##                65               179                 6                 2

levels(test)  <- c("Far above average", "Above average", "Average", "Below Average", "Far below ave
summary(test)

## Far above average     Above average           Average     Below Average
##               483              1118               666                21
## Far below average        Don't know         No answer              NA's
##                65               179                 6                 2
```

Robust method

??

```
# library(devtools)
# install_github("hadley/forcats")
library(forcats)
# I was expecting this to be obvious, now I'm not sure.
```

COMBINING SEVERAL LEVELS INTO ONE

Combining discrete levels

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.

Fragile method (base R)

This method overwrites the labels of factor levels with repeated labels in order to group levels together.

```
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
```

As before, this is fragile because it depends on the order of the factor levels not changing, and on a human accurately counting the indices of all the levels they wish to change.

Robust method

```
levels(GSS$Race.of.respondent)

## [1] "Black" "Other" "White"

GSS <- GSS %>%
  mutate(Race.of.respondent =  recode(Race.of.respondent, `Black` = "Nonwhite", `Other` = "Nonwhite
levels(GSS$Race.of.respondent)

## [1] "Nonwhite" "White"
```

COMBINING NUMERIC-TYPE LEVELS

This is something that is often a problem with data even when `stringsasfactors=FALSE`. Often variables like age or income are right censored, so there is a final category containing the lumped remainder of people. This means the data is necessarily at least a character string if not a factor. But, it feels natural to work with numeric expressions when recoding this data.

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.

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.

Fragile method (base R)

In order to break this data apart as simply as possible, we need to make it numeric. To start, we recode the label for ”89 or older” to read ”89”. Already, we are doing something fragile.

```
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

# Need to flesh out a base R approach to this. It's going to need some conditional logic.
```

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.

Robust method

```
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
```

CREATING DERIVED CATEGORICAL VARIABLE

XX THESE ARE STILL WORDED AS TASKS

The National Institutes of Alcohol Abuse and Alcoholism have published guidelines for moderate drinking. These state that women, or men aged 65 or older should drink no more than one drink per day on average and no more than three drinks at a sitting.

The `HELPmiss` dataset from the **mosaicData** package includes baseline data from a randomized clinical trial (Health Evaluation and Linkage to Primary Care).

variable	description
sex	gender of subject `female` or `male`
i1	average number of drinks per day (in last 30 days)
i2	maximum number of drinks per day (in past 30 days)
age	age (in years)

Use these guidelines and the `HELPsmall` dataset to create a new variable called `abstinent` for those that reported no drinking based on the value of their `i1` variable and `moderate` for those that do not exceed the NIAAA guidelines. All other non-missing values should be coded as `highrisk`.

```
library(dplyr); library(mosaic); library(readr)
```

```
data(HELPmiss)
HELPsmall <- HELPmiss %>%
  mutate(i1 = ifelse(id==1, NA, i1)) %>%  # make one value missing
  select(sex, i1, i2, age)
```

Fragile method (base R)

to be fleshed out

Robust method (dplyr)

```
glimpse(HELPsmall)
```

```
## Observations: 470
## Variables: 4
## $ sex <fctr> male, male, male, female, male, female, female, male, fem...
## $ i1  <int> NA, 56, 0, 5, 10, 4, 13, 12, 71, 20, 0, 13, 20, 13, 51, 0,...
## $ i2  <int> 26, 62, 0, 5, 13, 4, 20, 24, 129, 27, 0, 13, 31, 20, 51, 0...
## $ age <int> 37, 37, 26, 39, 32, 47, 49, 28, 50, 39, 34, 58, 58, 60, 36...

# I definitely want to remove these ASAP
#attach(HELPsmall)

HELPsmall <- with(HELPsmall,  # this won't work unless HELPsmall is made accessible
  mutate(HELPsmall,
    drink_stat = case_when(
      i1 == 0 ~ "abstinent",
      i1 <= 1 & i2 <= 3 & sex=='female' ~ "moderate",
      i1 <= 1 & i2 <= 3 & sex=='male' & age >= 65 ~ "moderate",
      i1 <= 2 & i2 <= 4 & sex=='male' ~ "moderate",
      TRUE ~ "highrisk"
)))
tally( ~ drink_stat, data = HELPsmall)

##
## abstinent  highrisk  moderate      <NA>
##        69       372        28         1
```

DEFENSIVE CODING

It would be good practice to write conditional testing statements into code using factors. Here is some code that doesn't work:

```
expect_equivalent(levels(GSS$Respondents.sex), c("Male", "Female"))
```

ACKNOWLEDGEMENTS

IDEAS

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 Appendices for less interesting examples?

Thisiswhereappendicesshouldstart!

A 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:

```
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 usable in an analysis.

B 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)
write_csv(GSS, path="../data/GSScleaned.csv")
```

C CLASS EXERCISE

As we see this paper being helpful at the undergraduate level, we have included the following as a possible exercise for students once they have mastered the basics of robust factor manipulation.

Subjects in the HELP study were categorized into categories of drug and alcohol involvement, as displayed in the following table.

```
HELPbase <- HELPfull %>%
  filter(TIME==0)
tally( ~ PRIM_SUB + SECD_SUB, data=HELPbase)

##          SECD_SUB
## PRIM_SUB  0  1  2  3  4  5  6  7  8
##        1 99  0 57 13  1  3 11  0  1
##        2 51 84  0  6  0  0 15  0  0
##        3 57 28 29  0  0  6  5  1  2
##        6  0  1  0  0  0  0  0  0  0
```

Note that the following codings of substance use involvement were used:

value	description
0	None
1	Alcohol
2	Cocaine
3	Heroin
4	Barbituates
5	Benzos
6	Marijuana
7	Methadone
8	Opiates

Create a new variable called ‘primsub‘ that combines the primary and secondary substances into a categorical variable with values corresponding to primary and secondary substances of the form: `alcohol only`, `cocaine only`, ‘heroin only‘, ‘alcohol-cocaine‘, ‘cocaine-alcohol‘, or ‘other‘. Code any group with fewer than 5 entries as ‘alcohol-other‘, ‘cocaine-other‘, or ‘heroin-other‘. If ‘PRIM_SUB==6‘ make the ‘primsub‘ variable missing.

How many subjects are there in the ‘alcohol-none‘ group? How many subjects are there in the ‘alcohol-other‘ group? What are the three most common groups?

```
HELPbase <- with(HELPbase,
```

```
  mutate(HELPbase,
    primary= recode(PRIM_SUB,
      `1`="alcohol",
      `2`="cocaine",
      `3`="heroin",
      `4`="barbituates",
      `5`="benzos",
      `6`="marijuana",
      `7`="methadone",
      `8`="opiates"),
    second=recode(SECD_SUB,
      `0`="none",
      `1`="alcohol",
      `2`="cocaine",
      `3`="heroin",
      `4`="barbituates",
      `5`="benzos",
      `6`="marijuana",
      `7`="methadone",
      `8`="opiates"),
    title=paste0(primary, "-", second)
))
```

```
tally(~ primary, data=HELPbase)

##
##    alcohol    cocaine     heroin marijuana
##        185        156        128          1

tally(~ second, data=HELPbase)

##
##      alcohol barbituates      benzos      cocaine       heroin   marijuana
##          113           1           9           86           19          31
##    methadone        none     opiates
##            1         207           3

counts <- HELPbase %>%
  group_by(primary, second) %>%
  summarise(observed=n())

merged <- left_join(HELPbase, counts, by=c("primary", "second"))
```

```
merged <- with(merged,
```

```
  mutate(merged,
    title =
      case_when(
        observed < 5 & primary=="alcohol" ~ "alcohol-other",
        observed < 5 & primary=="cocaine" ~ "cocaine-other",
        observed < 5 & primary=="heroin" ~ "heroin-other",
        TRUE ~ title),
    title = ifelse(primary=="marijuana", NA, title)))
tally(~ title + observed, data=merged)

##                      observed
## title                 1  2  3  5  6 11 13 15 28 29 51 57 84 99
##   alcohol-cocaine     0  0  0  0  0  0  0  0  0  0  0 57  0  0
##   alcohol-heroin      0  0  0  0  0  0 13  0  0  0  0  0  0  0
##   alcohol-marijuana   0  0  0  0  0 11  0  0  0  0  0  0  0  0
##   alcohol-none        0  0  0  0  0  0  0  0  0  0  0  0  0 99
##   alcohol-other       2  0  3  0  0  0  0  0  0  0  0  0  0  0
##   cocaine-alcohol     0  0  0  0  0  0  0  0  0  0  0  0 84  0
##   cocaine-heroin      0  0  0  0  6  0  0  0  0  0  0  0  0  0
##   cocaine-marijuana   0  0  0  0  0  0  0 15  0  0  0  0  0  0
##   cocaine-none        0  0  0  0  0  0  0  0  0  0 51  0  0  0
##   heroin-alcohol      0  0  0  0  0  0  0  0 28  0  0  0  0  0
##   heroin-benzos       0  0  0  0  6  0  0  0  0  0  0  0  0  0
##   heroin-cocaine      0  0  0  0  0  0  0  0  0 29  0  0  0  0
##   heroin-marijuana    0  0  0  5  0  0  0  0  0  0  0  0  0  0
##   heroin-none         0  0  0  0  0  0  0  0  0  0  0 57  0  0
##   heroin-other        1  2  0  0  0  0  0  0  0  0  0  0  0  0
##   <NA>                1  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Answers:

```
tally(~ title=="alcohol-none", data=merged)

##
##  TRUE FALSE  <NA>
##    99   370     1

tally(~ title=="alcohol-other", data=merged)

##
##  TRUE FALSE  <NA>
##     5   464     1

sort(tally(~ title, data=merged), decreasing=TRUE)[1:3]

##
##    alcohol-none cocaine-alcohol alcohol-cocaine
##              99              84              57
```

REFERENCES

Broman, K. (2015). Initial steps toward reproducible research. http://kbroman.org/steps2rr/.

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

Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, 59(10).