

Wrangling categorical data in R

Amelia McNamara¹ and Nicholas J Horton²

¹Statistical and Data Sciences Program, Smith College

²Mathematics and Statistics Department, Amherst College

ABSTRACT

Data wrangling is a critical foundation of data science. Wrangling of categorical data is an important component of the analysis cycle. Aspects of these operations can sometimes be tricky, particularly for complex transformations that arise in real-world settings. This paper discusses aspects of categorical variable transformations in R. We consider several motivating examples, suggest defensive coding strategies, and outline principles for data wrangling to help ensure data quality and sound analysis.

Keywords:

INTRODUCTION

Wrangling skills provide an intellectual and practical foundation for data science. Because of the complexity of some transformations, careless data derivation operations can lead to errors or inconsistencies in analysis. The wrangling of categorical data presents particular challenges and is highly relevant because so many variables are categorical (e.g., gender, income bracket, U.S. state). Data ingested into R from spreadsheets can lead to additional problems, many of which are related to categorical data because of the different storage methods possible in both R and the spreadsheets themselves.

In this paper, we consider a number of common idioms related to categorical data 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 with two or more levels).

To ground our work, we will compare and contrast how categorical data are treated in **base R** versus the so-called tidyverse (Wickham, 2014). Tools from the tidyverse, discussed in another paper in this special issue (see <https://github.com/dssc/collection/tidyflow>), are designed to make analysis purer, more predictable, and pipeable. They help facilitate a reproducible workflow where a new version of the data could be supplied in the code with updated results produced (Broman, 2015). Wrangling of categorical data can make this task even more complex (e.g., if a new level of a categorical variable is added in an updated dataset or inadvertently introduced by a careless error in a spreadsheet to be ingested into R).

THE IMPORTANCE OF TOOLING

It's important that statistical and data science tools foster good practice and provide a robust environment for data wrangling and data management. In this section we will describe how factors are used in base R and enumerate problems that arise with their use.

FACTORS IN R

Consider a variable describing gender that includes categories `male`, `female` and `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 uses hashed versions of all character strings, so the storage issue is no longer a consideration (Peng, 2015). For historical reasons, many functions store variables by default as factors.

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 <- ds %>%
  mutate(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

```

44 Instead of creating a new variable with a numeric version of the value of the factor variable `x1f` the
 45 variable is created with a factor number (i.e., 10 is mapped to 1, 20 is mapped to 2, and 40 is mapped to 3).
 46 This result is unexpected because `base::as.numeric()` is intended to recover numeric information
 47 by coercing a character variable. Compare the following:

```

as.numeric(c("hello"))

## [1] NA

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

## [1] 1

```

48 The unfortunate behavior of factors in base **R** has led to an online movement against the default
 49 behavior of many data import functions to take any variable composed as strings and automatically convert
 50 the variable to a factor. The tidyverse is part of this movement, with functions from the **readr** package de-
 51 faulting to leaving strings as-is. (Others have chosen to add `options(stringAsFactors=FALSE)`
 52 into their startup commands.)

53 Although the storage issues have been solved, and there are problems with defaulting strings to factors,
 54 factors are still necessary for some data analytic tasks. The most salient case is in modeling. When
 55 you pass a factor variable into `lm()` or `glm()`, **R** automatically creates indicator (or more pejoratively
 56 ‘dummy’) variables for each of the levels and picks one as a reference group. This behavior is lost if the
 57 variable is stored as a character vector. Factor variables also allow for the possibility of ordering between
 58 classes. Text strings `low`, `medium`, `high` would not preserve the ordering inherent in the groups.
 59 Again, this can be important for modeling when doing ordinal logistic regression and multinomial logistic
 60 regression.

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

64 To motivate this process, we will consider data from the General Social Survey (Smith et al., 2015).
 65 The General Social Survey is a product of the National Data Program for the Social Sciences, and the
 66 survey has been conducted since 1972 by NORC at the University of Chicago. It contains data on many
 67 factors of social life, and is widely used by social scientists. (In this paper we consider data from 2014.)

68 There are some import issues inherent to the data which are not particular to categorical data, so that
 69 processing is processing in Appendix A.. We’ll work with the data that has cleaned variable names.

```

library(dplyr)

```

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

## Observations: 2,540
## Variables: 17
## $ Gss.year.for.this.respondent..... <dbl> 2014, 2014...
## $ Respondent.id.number               <dbl> 1, 2, 3, 4...
## $ LaborStatus                         <fctr> Working f...
## $ Rs.occupational.prestige.score...1970. <dbl> 0, 0, 0, 0...
## $ Marital.status                     <fctr> Divorced,...
## $ Number.of.children                 <dbl> 0, 0, 1, 2...
## $ Age                                <fctr> 53.000000...
## $ Highest.year.of.school.completed   <dbl> 16, 16, 13...
## $ Respondents.sex                    <fctr> Male, Fem...
## $ Race.of.respondent                 <fctr> White, Wh...
## $ Rs.family.income.when.16.yrs.old   <fctr> Below ave...
## $ Total.family.income                 <fctr> $25000 or...
## $ Respondents.income                 <fctr> $25000 or...
## $ Total.family.income.1              <fctr> Not appli...
## $ PolParty                           <fctr> Not str r...
## $ Opinion.of.family.income           <fctr> Above ave...
## $ Sexual.orientation                 <fctr> Heterosex...
```

70 The rest of this paper is organized around case studies related to particular tasks:

71 1. Changing the labels of factor levels,

72 2. Reordering factor levels,

73 3. Combining several levels into one (both string-like labels and numeric, probably go together), and

74 4. Making derived factor variables.

75 CHANGING THE LABELS OF FACTOR LEVELS

76 For this example, we will be considering the labor status variable. It has 0 factor levels. Most of the labels
77 are spelled out fully, but a few are strangely formatted. We want to change this.

78 This is a specific case of the more general problem of changing the text of one (or more) of the factor
79 labels, so it appears more nicely formatted in a **ggplot2** plot, for example.

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

84 Compact but fragile (base R)

```
levels(GSS$Labor.force.status)

## NULL

summary(GSS$Labor.force.status)

## Length Class Mode
##      0   NULL  NULL
```

```
levels(GSS$Labor.force.status) <- c(levels(GSS$Labor.force.status)[1:5],
```

```

        "Temporarily not working",
        "Unemployed, laid off",
        "Working full time",
        "Working part time")

## Error in levels(GSS$Labor.force.status) <- c(levels(GSS$Labor.force.status)[1:5], : attempt
to set an attribute on NULL

summary(GSS$Labor.force.status)

## Length Class Mode
##      0  NULL  NULL

```

85 This method is less than ideal, because it depends on the data coming in with the factor levels ordered
86 in a particular way. By default, R orders factor levels alphabetically. So, “Keeping house” is first not
87 because it is the most common response, but simply because ‘k’ comes first in the alphabet. If the data
88 gets changed outside of R, for example so that responses currently label “Working full time” get labeled
89 “Full time work”, the code will silently fail with invalid results.

90 The workflow will also fail if additional factor levels are added after the fact. In our experience, both
91 with students and scientific collaborators, spreadsheet data can be easily changed in these ways (Leek,
92 2016).

93 Robust but verbose (base R)

94 Another (more robust method) to recode this variable in **base R** is to use subsetting to overwrite particular
95 values in the data.

```

summary(GSS$Political.party.affiliation)

## Length Class Mode
##      0  NULL  NULL

GSS$NewParty <- as.character(GSS$Political.party.affiliation)

## Error in '$<-.data.frame'('*tmp*', "NewParty", value = character(0)): replacement has
0 rows, data has 2540

GSS$NewParty[GSS$Political.party.affiliation=="Ind,near dem"] <- "Independent, near democrat"

## Error in '$<-.data.frame'('*tmp*', "NewParty", value = character(0)): replacement has
0 rows, data has 2540

GSS$NewParty[GSS$Political.party.affiliation == "Ind,near rep"] <- "Independent, near republican"

## Error in '$<-.data.frame'('*tmp*', "NewParty", value = character(0)): replacement has
0 rows, data has 2540

GSS$NewParty[GSS$Political.party.affiliation == "Not str democrat"] <- "Not strong democrat"

## Error in '$<-.data.frame'('*tmp*', "NewParty", value = character(0)): replacement has
0 rows, data has 2540

GSS$NewParty <- factor(GSS$NewParty)

## Error in '$<-.data.frame'('*tmp*', "NewParty", value = structure(integer(0), .Label =
character(0), class = "factor")): replacement has 0 rows, data has 2540

summary(GSS$NewParty)

## Length Class Mode
##      0  NULL  NULL

```

96 This approach is much more robust, because if the labels or ordering of levels changes before this
97 code is run it will not overwrite labels on the incorrect data. However, this approach has a number of
98 limitations in addition to being tedious and error prone. It is possible to miss cases, and misspelling and
99 cut-and-paste errors can mean that pieces of the code do not actually do anything.

Direct and robust (dplyr)

The `recode()` function in the **dplyr** package is a vector function, which combines the robustness of the second base R method while also reducing the verbosity. It still suffers from the problem of misspelling and cut-and-paste errors, because it will not throw errors if you try to recode a level that does not exist.

```
GSS <- GSS %>%
  mutate(dplyrParty =
    recode(Political.party.affiliation, `Not str republican` = "Not a strong republican",
          `Ind,near dem` = "Independent, near democrat",
          `Ind,near rep` = "Independent, near republican",
          `Not str democrat` = "Not a strong democrat"))

## Error in eval(expr, envir, enclos): object 'Political.party.affiliation' not found

summary(GSS$dplyrParty)

## Length Class Mode
##      0  NULL  NULL
```

In the above example, notice the trailing space after “Not str republican” and how the original factor level persists after the recode.

Aside – Editing whitespace out of levels

Whitespace can be dealt with when data is read, or later using string manipulations. This can be done using the `trimws()` function in **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. So, users must have a way to impose order on their factor variables.

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"
```

115 This is both verbose and depends on the number and order of the levels staying the same. If another
 116 factor level is added to the dataset, the above code will throw an error because the number of levels differs.
 117 This example illustrates why it is sometimes dangerous to replace an old version of a data frame with a
 118 new version.

119 Even worse, if the code gets run more than once, the order will be broken. Particularly when working
 120 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"
```

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

122 But it gets worse! It is tempting for new analysts to write code such as the following, which completely
 123 ruins the data set.

```
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 average", "Don't know", "No answer")
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
```

124 Robust method

125 A new addition to the tidyverse is the package **forcats**, a package for categorical data (and, the name is an
 126 anagram of the word factors!). **forcats** includes a `fct_relevel()` function that does exactly what we
 127 want. It allows us to specify the order of our factor levels (either completely or partially) and is robust to
 128 re-running code in an interactive session.

```
# devtools::install_github("hadley/forcats")
library(forcats)
summary(GSS$Opinion.of.family.income)

## 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

GSS <- GSS %>%
  mutate(Opinion.of.family.income =
    fct_relevel(Opinion.of.family.income, "Far above average", "Above average", "Average",
  summary(GSS$Opinion.of.family.income)

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

129 Notice that the levels we did not mention just end up at the back end of the ordering. Running the
 130 code again does not break things.

```
GSS <- GSS %>%
  mutate(Opinion.of.family.income =
    fct_relevel(Opinion.of.family.income, "Far above average", "Above average", "Average", "Below average", "Far below average", "Don't know", "No answer", "NA's"))
summary(GSS$Opinion.of.family.income)
```

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

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$Labor.force.status) <- c("Not employed", "No answer",
  "Other", "Not employed",
  "Not employed", "Not employed",
  "Not employed", "Employed", "Employed")

## Error in levels(GSS$Labor.force.status) <- c("Not employed", "No answer", : attempt
to set an attribute on NULL

summary(GSS$Labor.force.status)
```

| | | |
|-----------|-------|------|
| ## Length | Class | Mode |
| ## 0 | NULL | NULL |

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

Again, `recode()` does what we want.

```
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

Combining numeric-type levels is a problem that often arises 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. However, it may be more 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.

152 Fragile method (base R)

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

```
GSS$BaseAge <- GSS$Age.of.respondent
levels(GSS$BaseAge)

## NULL

levels(GSS$BaseAge) <- c(levels(GSS$BaseAge)[1:71], "89", "No answer")

## Error in levels(GSS$BaseAge) <- c(levels(GSS$BaseAge)[1:71], "89", "No answer"): attempt
to set an attribute on NULL
```

155 When we look at the levels, we can see that the first 71 levels correspond to the ages 18-88, and are
156 in the order we would expect, so we are leaving those as-is. Then we are overwriting the data where
157 BaseAge == "89 or older" with simply 89. Once that is done, we can convert the factor to a
158 character and then to a numeric.

```
GSS$BaseAge <- as.numeric(as.character(GSS$BaseAge))

## Error in `$<-.data.frame`(`*tmp*`, "BaseAge", value = numeric(0)): replacement has 0
rows, data has 2540

summary(GSS$BaseAge)

## Length Class Mode
##      0  NULL  NULL
```

159 We’re avoiding the pitfall from the introduction here by not just using `as.numeric()` on the factor
160 variables (this would convert 18 to 1, 19 to 2, etc.). And of course, we’re cheating a little bit here– if we
161 were going to use this as a numeric variable in an analysis, we wouldn’t necessarily want to turn all the
162 “89 or older” cases into the number “89”. But, we’re just on our way to a two-category factor, so those
163 cases would have gone to the “65 and up” category one way or the other.

164 Now, we can write some conditional logic

```
splitf <- function(x){
  return(ifelse(x<65, "18-64", "65 and up"))
}
summary(GSS$BaseAge)

## Length Class Mode
##      0  NULL  NULL

GSS$BaseAge <- sapply(GSS$BaseAge, splitf) # XX NH could this be combined into a single line?

## Error in `$<-.data.frame`(`*tmp*`, "BaseAge", value = list()): replacement has 0 rows,
data has 2540

GSS$BaseAge <- factor(GSS$BaseAge)

## Error in `$<-.data.frame`(`*tmp*`, "BaseAge", value = structure(integer(0), .Label =
character(0), class = "factor")): replacement has 0 rows, data has 2540

summary(GSS$BaseAge)

## Length Class Mode
##      0  NULL  NULL
```

165 Robust method

166 The **dplyr** method follows similar logic. However, instead of explicitly overwriting 89 or older
167 with the number 89, we use the **tidyr** `extract_numeric()` function to remove the numbers from
168 the factor labels. This works for the labels that already look numeric, like "18.000000" as well as for
169 89 or older. Then, we can include the conditional logic for splitting the variable within a mutate
170 command.


```

library(tidyr)
GSS <- GSS %>%
  mutate(dplyrAge = extract_numeric(Age.of.respondent)) %>%
  mutate(dplyrAge = if_else(dplyrAge < 65, "18-65", "65 and up"),
         dplyrAge = factor(dplyrAge))

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

summary(GSS$dplyrAge)

## Length Class Mode
##      0  NULL  NULL

```

CREATING DERIVED CATEGORICAL VARIABLE

Challenges often arise when data scientists need to create derived categorical variables. As an example, consider an indicator of moderate drinking status. 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) |

These guidelines can be used 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, with all other non-missing values coded as `highrisk`.

```

library(dplyr)
library(mosaic)
library(readr)

```

We make one value missing as a pedagogical tool to check for misbehavior of missing values.

```

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

```

Fragile method (base R)

```

# create empty repository for new variable
drinkstat <- character(length(HELPsmall$i1))
# create abstinent group
drinkstat[HELPsmall$i1==0] = "abstinent"
# create moderate group
drinkstat[(HELPsmall$i1>0 & HELPsmall$i1<=1 &
  HELPsmall$i2<=3 & HELPsmall$sex=="female") |
  (HELPsmall$i1>0 & HELPsmall$i1<=2 &
  HELPsmall$i2<=4 & HELPsmall$sex=="male")] = "moderate"
# create highrisk group
drinkstat[((HELPsmall$i1>1 | HELPsmall$i2>3) & HELPsmall$sex=="female") |
  ((HELPsmall$i1>2 | HELPsmall$i2>4) & HELPsmall$sex=="male")] = "highrisk"
# do we need to account for missing values?
is.na(drinkstat) <- is.na(HELPsmall$i1) | is.na(HELPsmall$i2) |
  is.na(HELPsmall$sex)
tally(~ drinkstat)

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

```

184 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...

HELPsmall <- with(HELPsmall, # this won't work unless HELPsmall is made accessible to mutate() # 1
  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
##          69          373          28

```

185 DEFENSIVE CODING

186 It is always good practice to write conditional testing statements into code using factors. As an example,
 187 we can assert that there are three levels for the drinking status variable in the HELP dataset.

```

library(assertthat)
levels(as.factor(drinkstat))

## [1] "abstinent" "highrisk" "moderate"

assert_that(length(levels(as.factor(drinkstat)))==3)

## [1] TRUE

```

188 Similar code could be used to check other conditions to verify that recoding has been done successfully.
 189 Here is some code that doesn't work:

```

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

```

190 ACKNOWLEDGEMENTS

191 QUERIES FOR REVIEWERS

- 192 1. Is it useful to demonstrate two ways to do each thing (as long as one isn't totally stupid)
- 193 2. Do we clarify why a given task is hard?
- 194 3. Do we clarify why a given approach is error-prone?
- 195 4. Should we focus more on Missing values
- 196 5. Add appendices or online resources for other examples?

197 APPENDIX A: LOADING THE DATA

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

201 Note to reviewers: we will be moving the data to a public datasource

```

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

## Observations: 2,538
## Variables: 17
## $ YEAR      <int> 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014, 2014,...
## $ ID_       <int> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16...
## $ WRKSTAT   <int> 1, 1, 4, 2, 5, 1, 9, 1, 8, 1, 7, 8, 5, 1, 6, 2, 2, 1,...
## $ PRESTIGE  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ MARITAL   <int> 3, 1, 3, 1, 1, 1, 1, 1, 5, 1, 1, 5, 3, 1, 5, 1, 3, 5,...
## $ CHILDS    <int> 0, 0, 1, 2, 3, 1, 2, 2, 4, 3, 2, 0, 5, 2, 0, 3, 3, 0,...
## $ AGE       <int> 53, 26, 59, 56, 74, 56, 63, 34, 37, 30, 43, 56, 69, 4...
## $ EDUC      <int> 16, 16, 13, 16, 17, 17, 12, 17, 10, 15, 5, 11, 8, 11,...
## $ SEX       <int> 1, 2, 1, 2, 2, 2, 1, 1, 2, 2, 2, 1, 1, 2, 2, 1, 2, 1,...
## $ RACE      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 3, 1,...
## $ INCOME16  <int> 2, 3, 2, 2, 4, 4, 2, 3, 3, 1, 1, 2, 2, 2, 2, 3, 2, 3,...
## $ INCOME    <int> 12, 12, 12, 12, 13, 12, 13, 12, 10, 12, 9, 9, 10, 11,...
## $ RINCOME   <int> 12, 12, 0, 9, 0, 12, 13, 12, 0, 12, 0, 0, 0, 11, 12, ...
## $ INCOME72  <int> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ PARTYID   <int> 5, 5, 6, 5, 3, 6, 6, 8, 3, 3, 3, 3, 3, 1, 3, 6, 1, 3,...
## $ FINRELA   <int> 4, 4, 2, 4, 3, 4, 9, 3, 2, 3, 8, 5, 1, 1, 3, 3, 2, 3,...
## $ SEXORNT   <int> 3, 3, 3, 3, 3, 9, 0, 0, 3, 3, 3, 3, 3, 0, 3, 3, 0, 0,...

```

202 Obviously, the result is less than ideal. All of the factor variables are encoded as integers, but their
 203 level labels have been lost. We have to look at a codebook to determine if SEX == 1 indicates male
 204 or female. We would rather preserve the integrated level labels. In order to do this, our best option is to
 205 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)
glimpse(GSS)

## Observations: 2,540
## Variables: 17
## $ Gss.year.for.this.respondent..... <dbl> 2014, 2014...
## $ Respondent.id.number               <dbl> 1, 2, 3, 4...
## $ Labor.force.status                  <chr> "Working f...
## $ Rs.occupational.prestige.score...1970. <dbl> 0, 0, 0, 0...
## $ Marital.status                     <chr> "Divorced"...
## $ Number.of.children                 <dbl> 0, 0, 1, 2...
## $ Age.of.respondent                  <chr> "53.000000...
## $ Highest.year.of.school.completed   <dbl> 16, 16, 13...
## $ Respondents.sex                    <chr> "Male", "F...
## $ Race.of.respondent                 <chr> "White", "...
## $ Rs.family.income.when.16.yrs.old   <chr> "Below ave...
## $ Total.family.income                 <chr> "$25000 or...
## $ Respondents.income                 <chr> "$25000 or...
## $ Total.family.income.1              <chr> "Not appli...
## $ Political.party.affiliation         <chr> "Not str r...
## $ Opinion.of.family.income           <chr> "Above ave...
## $ Sexual.orientation                  <chr> "Heterosex...

```

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

208 One problem is that some of the variable names are full of periods, so are hard to use. But, we can
 209 rename them.

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

```

library(dplyr)
GSS <- GSS %>%
  rename(Year = Gss.year.for.this.respondent.....,
         Occupational.prestige.score.1970 = Rs.occupational.prestige.score...1970.)
write_csv(GSS, path="../data/GSScleaned.csv")

```

212 APPENDIX B: CLOSING EXERCISE

213 We have included the following as a possible closing exercise.

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

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

##          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
```

216 The following codings of substance use involvement were used in the study.

| | value | description |
|-----|-------|-------------|
| | 0 | None |
| | 1 | Alcohol |
| | 2 | Cocaine |
| 217 | 3 | Heroin |
| | 4 | Barbituates |
| | 5 | Benzos |
| | 6 | Marijuana |
| | 7 | Methadone |
| | 8 | Opiates |

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

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

225 SOLUTION:

```
HELPhbase <- with(HELPhbase,
  mutate(HELPhbase,
    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=HELPhbase)
```

```
##
##   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
## alcohol-cocaine    1  2  3  5  6 11 13 15 28 29 51 57 84 99
## 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
```

```
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
```

226 REFERENCES

- 227 Broman, K. (2015). Initial steps toward reproducible research. <http://kbroman.org/steps2rr/>.
228 Leek, J. (2016). How to share data with a statistician. [https://github.com/jtleek/](https://github.com/jtleek/datasharing)
229 [datasharing](https://github.com/jtleek/datasharing).
230 Peng, R. D. (2015). stringsAsFactors: An unauthorized biography. [http://simplystatistics.](http://simplystatistics.org/2015/07/24/stringsasfactors-an-unauthorized-biography/)
231 [org/2015/07/24/stringsasfactors-an-unauthorized-biography/](http://simplystatistics.org/2015/07/24/stringsasfactors-an-unauthorized-biography/).
232 Smith, T. W., Mardsen, P., Hout, M., and Kim, J. (2015). General social surveys, 1972-2014 [machine-
233 readable data file].
234 Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, 59(10).