

Wrangling categorical data in R

Amelia McNamara¹ and Nicholas J Horton²

¹Statistical and Data Sciences Program, Smith College

²Department of Mathematics and Statistics, Amherst College

ABSTRACT

Data wrangling is a critical foundation of data science, and wrangling of categorical data is an important component of this process. However, categorical data can introduce unique issues in data wrangling, particularly in real-world settings with collaborators and periodically-updated dynamic data. This paper discusses common problems arising from categorical variable transformations in R, demonstrate the use of factors, and suggest approaches to address data wrangling challenges. For each problem, we present at least two strategies for management, one in base R and the other from the ‘tidyverse.’ We consider several motivating examples, suggest defensive coding strategies, and outline principles for data wrangling to help ensure data quality and sound analysis.

Keywords: statistical computing; data derivation; data science; data management

INTRODUCTION

Wrangling skills provide an intellectual and practical foundation for data science. Careless data derivation operations can lead to errors or inconsistencies in analysis (Hermans and Murphy-Hill, 2015; FitzJohn et al., 2014). The wrangling of categorical data presents particular challenges and is highly relevant because many variables are categorical (e.g., gender, income bracket, U.S. state) but coded with numerical values. It is easy to break the relationship between category numbers and category labels without realizing it, thus losing the information encoded in a variable. If data sources change upstream (for example, if a domain expert is providing spreadsheet data at regular intervals), code that worked on the initial data may not generate an error message, but could silently produce incorrect results.

Statistical and data science tools need to foster good practice and provide a robust environment for data wrangling and data management. This paper focuses on how R deals with categorical data, and showcases best practices for categorical data manipulation in R to produce reproducible workflows. We consider a number of common idioms related to categorical data that arise frequently in data cleaning and preparation, propose some guidelines for defensive coding, and discuss settings where analysts often get tripped up when working with categorical data.

For example, data ingested into R from spreadsheets can lead to problems with categorical data because of the different storage methods possible in both R and the spreadsheets themselves (Wilson et al., 2016). The examples below will help flag when these issues arise or avoid them altogether.

To ground our work, we will compare and contrast how categorical data are treated in **base R** versus the tidyverse (Wickham, 2014, 2016b). Tools from the tidyverse, discussed in another paper in this special issue (see <https://github.com/dsscollection/tidyflow>), are designed to make analysis purer, more predictable, and pipeable. Key components of the tidyverse that we will address in this paper include **ggplot2**, **dplyr**, **tidyr**, **forcats**, and **readr**. This suite of packages help facilitate a reproducible workflow where a new version of the data could be supplied in the code with updated results produced (Broman, 2015). While R code written in **base** can also have this quality, a common tendency is to use row or column numbers in code, which makes the result less reproducible. 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).

Our goal is to make the case that it is better to work with categorical data using tidyverse packages than with **base R**. Tidyverse code is more human readable, which can help reduce errors from the start, and the functions we highlight have been designed to make it harder to accidentally remove relationships implicit in categorical data. Because these issues are even more salient for new users, we recommend that

47 instructors should teach tidyverse approaches from the start.

48 CATEGORICAL DATA IN R: FACTORS AND STRINGS

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

52 In early versions of R, storing categorical data as a factor variable was considerably more efficient
53 than storing the same data as strings, because factor variables only store the factor labels once (Peng,
54 2015; Lumley, 2015). However, R uses a global string pool, so each unique string is only stored once, so
55 the storage is now less of an issue (Peng, 2015). For historical (or possibly anachronistic) reasons, many
56 functions store variables by default as factors.

57 While factors are important when including categorical variables in regression models, they can be
58 tricky to deal with, since many operations applied to them return different values than when applied to
59 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
```

60 Instead of creating a new variable with a numeric version of the value of the factor variable `x1f`, the
61 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).
62 This result is unexpected because `base::as.numeric()` is intended to recover numeric information
63 by coercing a character variable. Compare the following:

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

## [1] NA

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

## [1] 1
```

64 The unfortunate behavior of factors in R has led to an online movement against the default behavior
65 of many data import functions to make factors out of any variable composed as strings (Peng, 2015).
66 The tidyverse is part of this movement, with functions from the **readr** package defaulting to leaving
67 strings as-is. (Others have chosen to add `options(stringAsFactors=FALSE)` into their startup
68 commands.)

69 Although the storage issues have been solved, and there are problems with defaulting strings to factors,
70 factors are still necessary for some data analytic tasks. The most salient case is in modeling. When
71 you pass a factor variable into `lm()` or `glm()`, R automatically creates indicator (or more colloquially
72 ‘dummy’) variables for each of the levels and picks one as a reference group.

73 For simple cases, this behavior can also be achieved with a character vector. However, to choose which
74 level to use as a reference level or to order classes, factors must be used. For example, if a factor encodes
75 income levels as `low`, `medium`, `high`, it might make sense to use the lowest income level (`low`) as
76 the reference class so that all the other coefficients can be interpreted in comparison to it. However, R
77 would use `high` as the reference by default because ‘h’ comes before ‘l’ in the alphabet.

78 While ordering is particularly important when doing ordinal logistic regression and multinomial
79 logistic regression, the use of alphabetic ordering by default means even simple linear regression can be
80 affected.

While factors are important, 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 are lost. They present a steep learning curve for new users. In this paper, we will suggest best practices for working with factor data.

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

There are some import issues inherent to the data which are not particular to categorical data (see Appendix A for details). We'll work with the data with slightly cleaned up variable names.

```
library(dplyr)
GSS <- read.csv("../data/GSSoriginal.csv")
glimpse(GSS)

## Observations: 2,540
## Variables: 17
## $ Year <dbl> 2014, 2014, 2014, 2014, 2014,...
## $ Respondent.id.number <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10...
## $ Labor.force.status <fctr> Working fulltime, Working fu...
## $ Occupational.prestige.score.1970 <dbl> 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,...
## $ Marital.status <fctr> Divorced, Married, Divorced,...
## $ Number.of.children <dbl> 0, 0, 1, 2, 3, 1, 2, 2, 4, 3,...
## $ Age.of.respondent <fctr> 53.000000, 26.000000, 59.000...
## $ Highest.year.of.school.completed <dbl> 16, 16, 13, 16, 17, 17, 12, 1...
## $ Respondents.sex <fctr> Male, Female, Male, Female, ...
## $ Race.of.respondent <fctr> White, White, White, White, ...
## $ Rs.family.income.when.16.yrs.old <fctr> Below average, Average, Belo...
## $ Total.family.income <fctr> $25000 or more, $25000 or mo...
## $ Respondents.income <fctr> $25000 or more, $25000 or mo...
## $ Total.family.income.1 <fctr> Not applicable, Not applicab...
## $ Political.party.affiliation <fctr> Not str republican, Not str ...
## $ Opinion.of.family.income <fctr> Above average, Above average...
## $ Sexual.orientation <fctr> Heterosexual or straight, He...
```

The remainder of this paper is organized around case studies (examples) to carry out four specific and useful tasks:

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), and
4. Making derived factor variables.

Each case study begins with a problem, and presents several solutions. Typically, we contrast a method that uses the functionality of **base R** functions with an approach from the tidyverse along with some annotations of the code as needed. We will argue that while both approaches can solve the problem, the tidyverse solution tends to be simpler, easier to learn, and more robust.

CHANGING THE LABELS OF FACTOR LEVELS

In our first example, we will be considering the labor status variable. It is a categorical variable with 9 levels. Most of the labels are spelled out fully, but a few are strangely formatted. We want to change this.

This is a specific case of the more general problem of changing the text of factor labels, so they appear more nicely formatted in a 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 more human readable method, while also supporting reproducibility.

Compact but fragile (base R)

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

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

summary(GSS$Labor.force.status)

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

summary(GSS$Labor.force.status)

##      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
```

110 This method is less than ideal, because it depends on the data coming in with the factor levels ordered
 111 in a particular way. We call this a *fragile* process since future datasets may cause a workflow to break (a
 112 related concept in computer science is *software brittleness*). Why is this fragile? By default, R orders
 113 factor levels alphabetically. So, “Keeping house” is first not because it is the most common response,
 114 but simply because ‘k’ comes first in the alphabet. If the data gets changed outside of R, for example so
 115 responses currently labeled “Working full time” get labeled “Full time work”, the code will not generate an
 116 error message, but will mislabel all the data such that the `Labor.force.status` variable is essentially
 117 meaningless. (Another possible issue arises with strings that include non-ASCII characters, where the
 118 default of order levels may vary from locale to locale.)

119 The workflow will also fail if additional factor levels are added after the fact. In our experience, both
 120 with students and scientific collaborators, spreadsheet data can be easily changed in these ways. Others
 121 have noted this concern (Leek, 2016).

122 On a similar note, the following code silently makes a missing value.

```
factor("a", levels="c")

## [1] <NA>
## Levels: c
```

123 Robust but verbose (base R)

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

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

```
##           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

GSS$Labor.force.status <- as.character(GSS$Labor.force.status)
GSS$Labor.force.status[GSS$Labor.force.status=="Working full time"] <-
  "Full time"
GSS$Labor.force.status[GSS$Labor.force.status=="Working part time"] <-
  "Part time"
GSS$Labor.force.status[GSS$Labor.force.status=="Unemployed, laid off"] <-
  "Unemployed"
GSS$Labor.force.status <- factor(GSS$Labor.force.status)
summary(GSS$NewParty)

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

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

130 Direct and robust (dplyr)

131 The `recode()` function in the **dplyr** package is a vectorized function, which combines the robustness
 132 of the second base R approach while also reducing the verbosity. It still suffers from the problem of
 133 misspelling and cut-and-paste errors, because it will not generate an error message if you try to recode a
 134 non-existent level.

```
GSS <- GSS %>%
  mutate(Labor.force.status =
    recode(Labor.force.status,
      `Temporarily not working` = "Not working",
      `Keeping house` = "Homemaker",
      `School` = "In school",
      `Part time` = "Working part time",
      `Full time` = "Working full time"))
  summary(GSS$Labor.force.status)

##           Working full time           Homemaker           No answer
##           1230                263                2
##           Other                Working part time           Retired
##           76                273                460
##           In school Temporarily not working           Unemployed
##           90                40                104
##           NA's
##           2
```

135 In the above example, notice the trailing space in ``Temporarily not working`` in the
 136 `recode()` call. Because of this typo (the original factor level is actually ``Temporarily not working'``),
 137 the original factor level persists after the `recode`.

138 Aside – Editing whitespace out of levels

139 Whitespace can be dealt with when data is read, or later using string manipulations. This can be done
 140 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)

One common way to make this sort of change is to pass an argument to `levels` within the `factor()` function. However, this is fragile with respect to spelling issues and trailing whitespace.

```

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

##      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

test <- factor(famIncome, levels = c("Far above average", "Above average", "Average ",
                                     "Below Average", "Far below average", "Don't know",
                                     "No answer"))

summary(famIncome)

##      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

```

Note that many of the category totals come through appropriately, but several totals get set to 0 ('Average' because of the trailing whitespace and 'Below Average' because of the mistaken capitalization). These errors can be exceedingly frustrating to troubleshoot.

An approach that looks similar upon inspection but actually performs quite differently is to overwrite the `levels()` of the factor outside the `factor()` command. It is tempting for new analysts to write code such as the following, which completely breaks the association between rows and factor labels the data set.

```

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

##      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(famIncome) <- c("Far above average", "Above average", "Average", "Below Average",
                       "Far below average", "Don't know", "No answer")
summary(famIncome)

## 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

```

An approach that will not suffer from spelling mistakes is to use numeric indexing to reorder the levels.

```
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"
```

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

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

164 The more times the code is run, the more mixed up the labels and observations get.

165 Robust method

166 Because of the fragility and potential for frustration and mistakes associated with reordering levels in
 167 base R, we recommend the use of a tidyverse package. The package **forcats** (where the name is an
 168 anagram of the word factors!) (Wickham, 2016a). **forcats** is included in the tidyverse. It includes a
 169 `fct_relevel()` function that does exactly what we want. It allows us to specify the order of our
 170 factor levels (either completely or partially) and is robust to 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",
      "Below average",
      "Far below 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
```

171 Notice the levels we did not mention end up at the back end of the ordering. Running the code again
 172 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"))
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
```

173 COMBINING SEVERAL LEVELS INTO ONE

174 Combining discrete levels

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

178 *Fragile method (base R)*

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

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

## Not employed No answer Other Employed NA's
##          2129          263          2          144          2
```

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

182 **Robust method**
 183 The `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"
```

184 Combining numeric-type levels

185 Combining numeric-type levels is a frequently-occurring problem even when `stringsAsFactors = FALSE`.
 186 Often variables like age or income are right-censored, so there is a final category that lumps the remainder
 187 of people into one group. This means the data is necessarily at least a character string if not a factor.
 188 However, it may be more natural to work with numeric expressions when recoding this data.

189 In this data, age is provided as an integer for respondents 18-88, but also includes the possible answer
 190 “89 or older” as well as a possible “No answer” and NA values.

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

194 Fragile method (base R)

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

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

## [1] "18.000000" "19.000000" "20.000000" "21.000000" "22.000000"
## [6] "23.000000" "24.000000" "25.000000" "26.000000" "27.000000"
## [11] "28.000000" "29.000000" "30.000000" "31.000000" "32.000000"
## [16] "33.000000" "34.000000" "35.000000" "36.000000" "37.000000"
## [21] "38.000000" "39.000000" "40.000000" "41.000000" "42.000000"
## [26] "43.000000" "44.000000" "45.000000" "46.000000" "47.000000"
## [31] "48.000000" "49.000000" "50.000000" "51.000000" "52.000000"
## [36] "53.000000" "54.000000" "55.000000" "56.000000" "57.000000"
## [41] "58.000000" "59.000000" "60.000000" "61.000000" "62.000000"
## [46] "63.000000" "64.000000" "65.000000" "66.000000" "67.000000"
## [51] "68.000000" "69.000000" "70.000000" "71.000000" "72.000000"
## [56] "73.000000" "74.000000" "75.000000" "76.000000" "77.000000"
## [61] "78.000000" "79.000000" "80.000000" "81.000000" "82.000000"
## [66] "83.000000" "84.000000" "85.000000" "86.000000" "87.000000"
## [71] "88.000000" "89 or older" "No answer"

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

197 When we look at the levels, we can see the first 71 levels correspond to the ages 18-88, and are
 198 in the order we would expect, so we are leaving those as-is. Then we are overwriting the data where
 199 `BaseAge == "89 or older"` with simply 89. Finally, we can convert the factor to a character
 200 vector and then to a numeric one.

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

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

201 We’re avoiding the pitfall from the introduction here by not simply using `as.numeric()` on the
 202 factor variables (this would convert 18 to 1, 19 to 2, etc.). And of course, we’re cheating a little bit here—

203 if we were going to use this as a numeric variable in an analysis, we wouldn't necessarily want to turn all
 204 the "89 or older" cases into the number "89". But, we're on our way to a two level factor, so those cases
 205 would have gone to the "65 and up" category one way or the other.
 206 Now, we can write some conditional logic

```
summary(GSS$BaseAge)

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

GSS$BaseAge <- ifelse(GSS$BaseAge > 65, "18-64", "65 and up")
GSS$BaseAge <- factor(GSS$BaseAge)
summary(GSS$BaseAge)

##      18-64 65 and up      NA's
##      478      2051      11
```

207 **Robust method**

208 The **dplyr** method follows similar logic. However, instead of explicitly overwriting 89 or older
 209 with the number 89, we use the **readr** `parse_number()` function to remove the numbers from the
 210 factor labels. This works for the labels that already look numeric, like "18.000000" as well as for
 211 "89 or older". Then, we can include the conditional logic for splitting the variable within a `mutate`
 212 command.

```
library(readr)
GSS <- GSS %>%
  mutate(dplyrAge = parse_number(Age.of.respondent)) %>%
  mutate(dplyrAge = if_else(dplyrAge < 65, "18-65", "65 and up"),
         dplyrAge = factor(dplyrAge))
summary(GSS$dplyrAge)

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

213 Note that you need to be very sure that the strings with a number have a relevant number. You could
 214 accidentally add a number that is not meaningful if numbers appear in unanticipated ways.

215 **CREATING DERIVED CATEGORICAL VARIABLES**

216 Challenges often arise when data scientists need to create derived categorical variables. As an exam-
 217 ple, consider an indicator of moderate drinking status. The National Institutes of Alcohol Abuse and
 218 Alcoholism have published guidelines for moderate drinking National Institute of Alcohol Abuse and
 219 Alcoholism (2016). These guidelines state that women (or men aged 65 or older) should drink no more
 220 than one drink per day on average and no more than three drinks on any single day or at a sitting. Men
 221 under age 65 should drink no more than two drinks per day on average and no more than four drinks
 222 on any single day. The `HELPMISS` dataset from the **mosaicData** package includes baseline data from a
 223 randomized clinical trial (Health Evaluation and Linkage to Primary Care) Samet et al. (2003). These
 224 subjects were recruited from a detoxification center, hence those that reported alcohol as their primary
 225 substance of abuse have extremely high rates of drinking.

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)

227 These guidelines can be used to create a new variable called `abstinent` for those reporting no
 228 drinking based on the value of their `i1` variable and `moderate` for those that do not exceed the NIAAA
 229 guidelines, with all other non-missing values coded as `highrisk`.

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

230 Because missing values can become especially problematic in more complex derivations, we will
231 make one value missing so we can ensure our data wrangling accounts for the missing value.

```
data(HELPmiss)
HELPsmall <- HELPmiss %>%
  mutate(i1 = ifelse(id==1, NA, i1)) %>% # make one value missing
  select(sex, i1, i2, age)
head(HELPsmall, 2)

##      sex i1 i2 age
## 1 male NA 26 37
## 2 male 56 62 37
```

232 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"
# account for missing values
is.na(drinkstat) <- is.na(HELPsmall$i1) | is.na(HELPsmall$i2) |
  is.na(HELPsmall$sex)
drinkstat <- factor(drinkstat)
table(drinkstat, useNA = "always")

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

233 While this approach works, it is hard to follow and to check or debug. The logical conditions are all
234 correctly coded, but require many repetitions of `HELPsmall$variable`, and the missing value was
235 not handled by default (without the `is.na()` call, the missing value would default to be "highrisk"
236 because of their extreme value for `i2`).

237 Robust method (dplyr)

```
HELPsmall <- with(HELPsmall, # this won't work with current dplyr
  # unless HELPsmall is made accessible to mutate() through with()
  # Hadley was aware of this issue with case_when(), though the issue is closed at
  # https://github.com/hadley/dplyr/issues/1996
  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",
      is.na(i1) ~ "missing", # this can't be NA
      TRUE ~ "highrisk"
    ))
  tally(~ drink_stat, exclude=NULL, data = HELPsmall)

## drink_stat
## abstinent highrisk missing moderate <NA>
##          69       372        1       28        0
```

238 In the robust tidyverse method, the same logic is used, but the conditions are clearer and more
239 comprehensible. Instead of one complex Boolean condition for `moderate`, three separate lines can be
240 used to match the different options. While the end result is the same, this code is more human readable
241 and it is harder to miss possible edge cases.

242 DEFENSIVE CODING

243 It is always good practice to code in a defensive manner. Investing a little time up front can help avoid
244 painful errors later. For the setting we are considering, defensive coding might involve adding conditional
245 testing statements into code creating or modifying factors. These testing statements can help ensure the
246 data have not changed from one session to another, or as the result of changes to the raw data.

247 As an example, we might want to check there are exactly three levels for the drinking status variable
248 in the HELP dataset. If there were fewer or more than three levels, something would have gone wrong
249 with our code. We can use the **assertthat** package to help with this.

```
library(assertthat)
levels(drinkstat)

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

assert_that(length(levels(drinkstat))==3)

## [1] TRUE
```

250 We also might want to ensure the factor labels are exactly what we were expecting. Perhaps we want
251 to make sure our Race variable has been collapsed into two categories, with particular levels. We can use
252 `expect_equivalent()` and `expect_equal()` from the **testthat** package to make this check.

```
library(testthat)
str(levels(GSS$Race.of.respondent))

## chr [1:2] "Nonwhite" "White"

str(c("White", "Nonwhite"))

## chr [1:2] "White" "Nonwhite"

str(sort(c("White", "Nonwhite")))

## chr [1:2] "Nonwhite" "White"

#expect_equivalent(levels(GSS$Race.of.respondent),
# c("White", "Nonwhite")) # This doesn't work, but we wish it did.
expect_equivalent(levels(GSS$Race.of.respondent), c("Nonwhite", "White"))
expect_equal(levels(GSS$Race.of.respondent), c("Nonwhite", "White"))
expect_equivalent(levels(GSS$Race.of.respondent), sort(c("Nonwhite", "White")))
```

253 CONCLUSION

254 Categorical variables arise commonly in most datasets. Aspects of data wrangling involving categorical
255 variables can be problematic and error-prone. In this paper we have outlined some example case studies
256 where analytic tasks can be simplified and made more robust through use of new tools available in the
257 tidyverse. We believe further work is needed to continue to make it easier to undertake analyses requiring
258 data wrangling (particularly with respect to categorical data). New tools and an increased emphasis on
259 defensive coding may help improve the quality of data science moving forward.

260 ACKNOWLEDGEMENTS

261 Thanks to Hadley Wickham, Mine Cetinkaya-Rundel, Johanna Hardin, Zev Ross, and Colin Rundel for
262 helpful comments and suggestions on an earlier draft.

263 APPENDIX A: LOADING THE DATA

264 Since this is a reproducible special issue, we want to make sure our data ingestion process is as
265 reproducible as possible. We are using the General Social Survey (GSS) data, which includes many years
266 of data (1972-2014) and many possible variables (150-800 variables, depending on the year) (Smith et al.,
267 2015). However, the GSS data has some idiosyncrasies. So, we are attempting good-enough practices for
268 data ingest (Wilson et al., 2016).

269 The major issue related to reproducibility is the fact that the dataset is not available through an API.
 270 For SPSS and Stata users, yearly data are available for direct download on the website. For more format
 271 possibilities, users must go through an online wizard to select variables and years for the data they wish
 272 to download (NORC at the University of Chicago, 2016). For this paper, we selected a subset of the
 273 demographic variables and the year 2014. The possible output options from the wizard are Excel (either
 274 data and metadata or metadata only), SPSS, SAS, Stata, DDI, or R script. We selected both the Excel and
 275 R formats to look at the differences.

276 The R format provided by the GSS is actually a Stata file and custom R script using the **foreign**
 277 package to do the translation for you. Here is the result of that process.

```
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, 1, 2, 1, ...
## $ RACE      <int> 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 3, 1, 1, 1, 1, 1, 3, 1, ...
## $ INCOM16   <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, ...
```

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

```
library(readxl)
GSS <- read_excel("../data/GSS.xls")
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               <chr> "Not appli...
## $ Political party affiliation       <chr> "Not str r...
## $ Opinion of family income         <chr> "Above ave...
## $ Sexual orientation                <chr> "Heterosex..."
```

282 This is a little better. Now we have preserved the character strings. But, the data is not yet usable in
 283 an analysis. One problem is some of the variable names include spaces, so they are hard to use. Also, one
 284 variable name is repeated, perhaps because of an error in the data wizard. To fix these issues, we need to
 285 rename the variables so all variables have unique names without spaces.

```
names(GSS) <- make.names(names(GSS), unique=TRUE)
names(GSS)

## [1] "Gss.year.for.this.respondent....."
## [2] "Respondent.id.number"
## [3] "Labor.force.status"
## [4] "Rs.occupational.prestige.score...1970."
## [5] "Marital.status"
## [6] "Number.of.children"
## [7] "Age.of.respondent"
## [8] "Highest.year.of.school.completed"
## [9] "Respondents.sex"
## [10] "Race.of.respondent"
## [11] "Rs.family.income.when.16.yrs.old"
## [12] "Total.family.income"
## [13] "Respondents.income"
## [14] "Total.family.income.1"
## [15] "Political.party.affiliation"
## [16] "Opinion.of.family.income"
## [17] "Sexual.orientation"
```

286 These names are an improvement, but now some are full of periods. We'd like to rename the most
 287 extreme cases to make the names more human readable. As with all the tasks in this paper, there is
 288 a fragile way to do this in **base R**, but we'll use the more robust `rename()` function from the **dplyr**
 289 package. `rename()`

```
library(dplyr)
GSS <- GSS %>%
  rename(Year = Gss.year.for.this.respondent.....,
         Occupational.prestige.score.1970 = Rs.occupational.prestige.score...1970.)
names(GSS)

## [1] "Year" "Respondent.id.number"
## [3] "Labor.force.status" "Occupational.prestige.score.1970"
## [5] "Marital.status" "Number.of.children"
## [7] "Age.of.respondent" "Highest.year.of.school.completed"
## [9] "Respondents.sex" "Race.of.respondent"
## [11] "Rs.family.income.when.16.yrs.old" "Total.family.income"
## [13] "Respondents.income" "Total.family.income.1"
## [15] "Political.party.affiliation" "Opinion.of.family.income"
## [17] "Sexual.orientation"
```

290 With the data loaded and the names adjusted, we can write the data to a new file for use in the body of
 291 the paper.

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

292 A version of this file is used as our motivating example.

293 APPENDIX B: CLOSING EXERCISE

294 We have included the following as a possible supplementary exercise.

295 Subjects in the HELP study were also categorized into categories of primary and secondary drug and
 296 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
```

297 The following coding of substance use involvement was used in the study.

	value	description
	0	None
	1	Alcohol
	2	Cocaine
	3	Heroin
298	4	Barbituates
	5	Benzos
	6	Marijuana
	7	Methadone
	8	Opiates

299 Create a new variable called `primsub` combining 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.

304 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?

305 SOLUTION:

```
HELPhbase <- HELPhbase %>%
  mutate(
    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)

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

tally(~ second, data=HELPhbase)

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

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

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

```
merged <- merged %>%
  mutate(
    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 57  0  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)

## title == "alcohol-none"
## TRUE FALSE <NA>
##    99   370     1

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

## title == "alcohol-other"
## TRUE FALSE <NA>
##     5   464     1

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

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

REFERENCES

- 307
308 Broman, K. (2015). Initial steps toward reproducible research. <http://kbroman.org/steps2rr/>.
309 FitzJohn, R., Pennell, M., Zanne, A., and Cornwell, W. (2014). Reproducible research is still a challenge.
310 Technical report, rOpenSci.
311 Hermans, F. and Murphy-Hill, E. (2015). Enron's spreadsheets and related emails: A dataset and analysis.
312 In *ICSE*.
313 Leek, J. (2016). How to share data with a statistician. [https://github.com/jtleek/](https://github.com/jtleek/datasharing)
314 datasharing.
315 Lumley, T. (2015). stringsAsFactors = `jsigh`. [http://notstatschat.tumblr.com/post/](http://notstatschat.tumblr.com/post/124987394001/stringsasfactors-sigh)
316 124987394001/stringsasfactors-sigh.
317 National Institute of Alcohol Abuse and Alcoholism (2016). Retinking drinking: What's 'low-risk' drink-
318 ing? [http://rethinkingdrinking.niaaa.nih.gov/How-much-is-too-much/](http://rethinkingdrinking.niaaa.nih.gov/How-much-is-too-much/Is-your-drinking-pattern-risky/Whats-Low-Risk-Drinking.aspx)
319 Is-your-drinking-pattern-risky/Whats-Low-Risk-Drinking.aspx.

320 NORC at the University of Chicago (2016). GSS data explorer. <https://gssdataexplorer.norc.org/>.

321

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

323

324 Samet, J. H., Larson, M. J., Horton, N. J., Doyle, K., Winter, M., and Saitz, R. (2003). Linking alcohol and drug dependent adults to primary medical care: A randomized controlled trial of a multidisciplinary health intervention in a detoxification unit. *Addiction*, 98(4):509–516.

325

326 Smith, T. W., Mardsen, P., Hout, M., and Kim, J. (2015). General social surveys, 1972-2014 [machine-readable data file].

327

328

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

330 Wickham, H. (2016a). *forcats: Tools for Working with Categorical Variables (Factors)*.

331 Wickham, H. (2016b). The tidy tools manifesto. <https://cran.r-project.org/web/packages/tidyverse/vignettes/manifesto.html>.

332

333 Wilson, G., Bryan, J., Cranston, K., Kitzes, J., Nederbragt, L., and Teal, T. K. (2016). Good enough practices for scientific computing. <https://swcarpentry.github.io/good-enough-practices-in-scientific-computing/>.

334

335