Preparing data for analysis and visualization in R

Identifying and treating missing values

Jenine Harris
Brown School

Identifying and treating missing values

- In addition to making sure the variables used are an appropriate type, it is also important to make sure that missing values were treated appropriately by R.
- In R, missing values are recorded as `NA` which stands for *no answer*.
- Researchers code missing values in many different ways when collecting and storing data. Some of the more common ways are:
    - blank
    - 777, -777, 888, -888, 999, -999 or something similar
    - a single period
    - -1
    - NULL
- Other responses like "Don't know" or "Inapplicable" may sometimes be treated as missing or treated as response categories depending on what is most appropriate given the characteristics of the data and the analysis goals.

Recoding missing values to NA

- In the summary of the GSS data in the codebook (page 304) the grass variable has five possible values: DK (don't know), IAP (inapplicable), LEGAL, NOT LEGAL, and `NA` (no answer).
- The DK, IAP, and `NA` could all be considered missing values.
- However, R only treats `NA` as missing.
- Before conducting any analyses, the DK and IAP values can be converted to `NA` to be treated as missing in any analyses.
- Note that `NA` is a **reserved word** in R.
- In order to use `NA`, both letters must be uppercase (`Na` or `na` are not permitted), and there can be no quotation marks (R will treat `"NA"` as a character, rather than a true missing value).

Using tidyverse to recode missing values to NA

- One way to recode is to use the **tidyverse** package.
- If the **tidyverse** is installed, open it with:

```
# open tidyverse package
library(package = "tidyverse")
```

- If the **tidyverse** package is not installed before using `library()` to open it, the library function will show an error.

Review the data first

```
# Import the data if needed
# and summarize
gss.2016 <-
  data.table::fread(input = "data/legal_weed_age_GSS2016_ch1.csv")
summary(object = gss.2016)
```

```
##     grass                age
##  Length:2867        Length:2867
##  Class :character   Class :character
##  Mode  :character   Mode  :character
```

Fix the data type before recoding

- The tidyverse uses the pipe feature, `%>%`, for data management and other tasks.
- The `%>%` works to send or *pipe* information through a function or set of functions.
- The `mutate()` function takes the name of the variable to recode and then information on how to recode it.
- In this case, pipe the `gss.2016` data set into a `mutate()` function that can be used to fix the data type.

```
gss.2016.cleaned <- gss.2016 %>%
  mutate(grass = as.factor(grass))
```

```
# check the summary, there should be
summary(object = gss.2016.cleaned)
```

```
##         grass          age
##  DK        : 110   Length:2867
##  IAP       : 911   Class :character
##  LEGAL     :1126   Mode  :character
##  NOT LEGAL: 717
##  NA's      :   3
```

Use mutate() to recode missing values to NA

- Use the `mutate()` function to recode `'DK'`

```
gss.2016.cleaned <- gss.2016 %>%
  mutate(grass = as.factor(grass)) %>%
  mutate(grass = na_if(x = grass, y = 'DK'))
```

- First is the `gss.2016.cleaned <-` which indicates that whatever happens on the right hand side of the `<-` will be assigned to the `gss.2016.cleaned` object name.
- The first thing after the arrow is `gss.2016 %>%` which indicates that the `gss.2016` data is being piped into whatever comes on the next line, in this case, it is being piped into the `mutate()` function
- The `mutate()` function on the next line uses the `na_if()` function to make the `grass` variable equal to `NA` if the `grass` variable is currently coded as "DK".

Check the recoding

```
# check the summary, there should be
# 110 + 3 in the NA category
summary(object = gss.2016.cleaned)
```

```
##         grass           age
##  DK        :   0   Length:2867
##  IAP       : 911   Class :character
##  LEGAL     :1126   Mode  :character
##  NOT LEGAL: 717
##  NA's      : 113
```

Replace DK and IAP with NA

- Using the pipe, add the IAP recoding to the code to replace IAP with `NA`:

```
gss.2016.cleaned <- gss.2016 %>%
  mutate(grass = as.factor(grass)) %>%
  mutate(grass = na_if(x = grass, y = 'DK')) %>%
  mutate(grass = na_if(x = grass, y = 'IAP'))
```

```
# check the summary, there should now be
# 110 + 911 + 3 in the NA category
summary(object = gss.2016.cleaned)
```

```
##        grass          age
##  DK        :   0   Length:2867
##  IAP       :   0   Class :character
##  LEGAL     :1126   Mode  :character
##  NOT LEGAL: 717
##  NA's      :1024
```

Remove unused categories with droplevels()

- The summary information is accurate, with zero observations coded as `DK` or `IAP`.
- However, the `DK` and `IAP` category labels are still listed even though there are no observations with these coded values.
- To remove unused categories, use the `droplevels()` function:

```
gss.2016.cleaned <- gss.2016 %>%
  mutate(grass = as.factor(grass)) %>%
  mutate(grass = na_if(x = grass, y = 'DK')) %>%
  mutate(grass = na_if(x = grass, y = 'IAP')) %>%
  mutate(grass = droplevels(x = grass))
```

```
# check the summary
summary(object = gss.2016.cleaned)
```

```
##         grass          age
##  LEGAL    :1126   Length:2867
##  NOT LEGAL: 717   Class :character
##  NA's     :1024   Mode  :character
```

Recoding continuous to categorical

- In addition to recoding missing values, `mutate()` is useful to create categories from numeric variables.
- The age variable currently holds the age in years rather than age categories.
- Recode to measure age in four categories:
  - 18-29
  - 30-59
  - 60-74
  - 75+

Using the cut() function to recode numeric to factor

- The tidyverse can be used to complete the recoding of "89 OR OLDER" to 89 and the changing of data types with `as.numeric()` and `as.factor()` added to the `mutate()` functions.
- The `cut()` function can be used to divide a continuous variable into categories by cutting it into pieces and adding a label to each piece.
    - To create cut takes a variable like `age` as the first argument, so it would look like `cut(age,...`.
    - The second thing to add after the variable name is a vector made up of the **breaks**.
    - Breaks specify the lower and upper limit of each category of values.
    - The first entry is the lowest value of the first category, the second entry is the upper value of the first category, the third entry is the upper value of the second category, and so on.
    - The function would look like `cut(age, breaks = c(-Inf, 29, 59, 74, Inf),`.
- The final thing to add is a vector made up of the **labels** for the categories, with each label inside quote marks, like this: `labels = c('< 30', '30 - 59', '60 - 74', '75+' )`.

Code for using the cut() function

```
# use tidyverse to change data types and recode
gss.2016.cleaned <- gss.2016 %>%
  mutate(age = recode(age, '89 OR OLDER' = '89')) %>%
  mutate(age = as.numeric(x = age)) %>%
  mutate(grass = as.factor(x = grass)) %>%
  mutate(grass = na_if(x = grass, y = 'DK')) %>%
  mutate(grass = na_if(x = grass, y = 'IAP')) %>%
  mutate(grass = droplevels(x = grass)) %>%
  mutate(age.cat = cut(x = age,
                       breaks = c(-Inf, 29, 59, 74, Inf),
                       labels = c('< 30', '30 - 59',
                                  '60 - 74', '75+' )))
```

```
summary(object = gss.2016.cleaned)
```

```
##        grass           age           age.cat
##  LEGAL    :1126   Min.   :18.00   < 30   : 481
##  NOT LEGAL: 717   1st Qu.:34.00   30 - 59:1517
##  NA's     :1024   Median :49.00   60 - 74: 598
##                   Mean   :49.16   75+    : 261
##                   3rd Qu.:62.00   NA's   :  10
##                   Max.   :89.00
##                   NA's   :10
```

Full code for recoding

```
#########################################################
# Project: First week of biostats
# Purpose: Clean GSS 2016 data
# Author: Jenine
# Edit date: July 27, 2020
# Data: GSS 2016 subset of age and marijuana use variables
#########################################################

# bring in GSS 2016 data from the web and examine it
library(package = "data.table")
gss.2016 <- fread(file = "data/legal_weed_age_GSS2016_ch1.csv")
# use tidyverse to clean the data
library(package = "tidyverse")
gss.2016.cleaned <- gss.2016 %>%
  mutate(age = recode(age, '89 OR OLDER' = '89')) %>%
  mutate(age = as.numeric(x = age)) %>%
  mutate(grass = as.factor(x = grass)) %>%
  mutate(grass = na_if(x = grass, y = 'DK')) %>%
  mutate(grass = na_if(x = grass, y = 'IAP')) %>%
  mutate(grass = droplevels(x = grass)) %>%
  mutate(age.cat = cut(x = age, breaks = c(-Inf, 29, 59, 74, Inf),
                       labels = c('< 30', '30 - 59',
                                  '60 - 74', '75+' )))
# check the summary
summary(object = gss.2016.cleaned)
```

Check your understanding

Describe what `mutate()`, `na_if()`, and `%>%` did in the final code.

Answer

- `mutate()` changes a variable in some way. In this case, mutate was used to recode age, make age numeric, make grass a factor, recode DK and IAP to be NA, drop unused levels, and recode age to a new categorical age variable.
- `na_if` changes specified values to be missing values, which are represented by `NA` in R
- `%>%` is a pipe that sends whatever is on the left through the function on the right