Computing and reporting descriptive statistics

Frequency tables for factor variables

Jenine Harris
Brown School

Computing frequencies and frequency distributions

- A **frequency distribution** shows the number of observations in each category for a factor or categorical variable.
    - A frequency distribution is useful to examine how many observations there are for each category of a categorical variable.

Creating a frequency table for transgender status

- The table includes characteristics of transgender participants in the 2014 Behavioral Risk Factor Surveillance Survey.
  - Before getting started, review the 2014 BRFSS codebook found on the CDC BRFSS website.
  - The transgender transition status information is on page 83 of the codebook.
  - The frequency distribution shown in the codebook included:
    - 363 MtF transgender
    - 212 FtM transgender
    - 116 gender nonconforming
    - 150,765 not transgender
    - 1,138 don't know/not sure
    - 1,468 refused
    - 310,602 not asked or missing

Import the data with read.csv()

- Before creating a frequency distribution, first open the BRFSS data file
    - The data file is available in **XPT** or **ASCII** format on the BRFSS website.
    - The XPT is a file from SAS statistical software.
    - R can open the XPT file type by using the **haven** package that is part of the tidyverse.
- A cleaned version of the data set is also available in the book materials as a **csv** (comma separated values) file with the variables needed to recreate the table.
- Use `read.csv()` and the appropriate object naming recommendations to import and name the data frame object.

```
# read the 2014 BRFSS data
brfss.trans.2014 <- read.csv(file = "data/transgender_hc_ch2.csv")
```

Check the data import and summarize with summary()

- Check the Environment tab in the top right pane of R Studio.

- The `brfss.trans.2014` object should be in the Environment tab under the *Data* heading.

  - Next to the object name, it should show 464,664 observations and 9 variables in the data frame.
  - This indicates that the data set contains 464,664 observations, or 464,664 people.
  - This is consistent with the codebook.

- Use summary to learn more about the data before starting to compute other statistics:

```
# examine the data
summary(object = brfss.trans.2014)
```

Summarize the data with summary()

```
##     TRNSGNDR        X_AGEG5YR          X_RACE         X_INCOMG    
##  Min.   :1.00     Min.   : 1.000   Min.   :1.000   Min.   :1.000  
##  1st Qu.:4.00     1st Qu.: 5.000   1st Qu.:1.000   1st Qu.:3.000  
##  Median :4.00     Median : 8.000   Median :1.000   Median :5.000  
##  Mean   :4.06     Mean   : 7.822   Mean   :1.992   Mean   :4.481  
##  3rd Qu.:4.00     3rd Qu.:10.000   3rd Qu.:1.000   3rd Qu.:5.000  
##  Max.   :9.00     Max.   :14.000   Max.   :9.000   Max.   :9.000  
##  NA's   :310602                    NA's   :94                     
##     X_EDUCAG        HLTHPLN1         HADMAM          X_AGE80     
##  Min.   :1.000   Min.   :1.000   Min.   :1.00     Min.   :18.00  
##  1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.00     1st Qu.:44.00  
##  Median :3.000   Median :1.000   Median :1.00     Median :58.00  
##  Mean   :2.966   Mean   :1.108   Mean   :1.22     Mean   :55.49  
##  3rd Qu.:4.000   3rd Qu.:1.000   3rd Qu.:1.00     3rd Qu.:69.00  
##  Max.   :9.000   Max.   :9.000   Max.   :9.00     Max.   :80.00  
##                                  NA's   :208322                  
##     PHYSHLTH   
##  Min.   : 1.0  
##  1st Qu.:20.0  
##  Median :88.0  
##  Mean   :61.2  
##  3rd Qu.:88.0  
##  Max.   :99.0  
##  NA's   :4     
```

Making a basic table of frequencies and percentages

- One way to get a frequency distribution in R is to use `table()`.
- Using `table()` results in a plain table listing each value of a variable and the number of observations that have that value.
- The `table()` function takes the name of the data frame followed by `$` and then the name of the variable for the table (e.g., `data$variable`).
- The data frame was called `brfss.trans.2014` and the variable was called `TRNSGNDR`, so `table(brfss.trans.2014$TRNSGNDR)` will produce a table of frequencies.

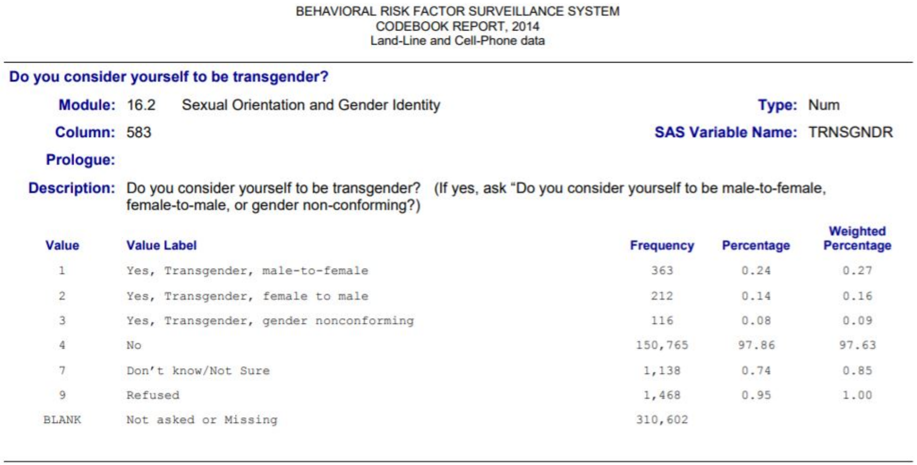
```
table(brfss.trans.2014$TRNSGNDR)
```

```
## 
##      1      2      3      4      7      9 
##    363    212    116 150765   1138   1468
```

Reading the frequency table

```
##
##      1      2      3      4      7      9
##    363    212    116 150765   1138   1468
```

- The output shows a set of numbers with the top row representing the categories and the bottom row giving the number of observations in the category.
- The number of observations in each category is the *frequency*.
- The frequencies in this table match the frequencies in the 2014 BRFSS codebook.
  - For example, the first category of MtF---where MtF stands for Male to Female---shows 363 in the codebook and 363 in the table.



BEHAVIORAL RISK FACTOR SURVEILLANCE SYSTEM
CODEBOOK REPORT, 2014
Land-Line and Cell-Phone data

Do you consider yourself to be transgender?

Module: 16.2 Sexual Orientation and Gender Identity **Type:** Num

Column: 583 **SAS Variable Name:** TRNSGNDR

Prologue:

Description: Do you consider yourself to be transgender? (If yes, ask "Do you consider yourself to be male-to-female, female-to-male, or gender non-conforming?)

Value	Value Label	Frequency	Percentage	Weighted Percentage
1	Yes, Transgender, male-to-female	363	0.24	0.27
2	Yes, Transgender, female to male	212	0.14	0.16
3	Yes, Transgender, gender nonconforming	116	0.08	0.09
4	No	150,765	97.86	97.63
7	Don't know/Not Sure	1,138	0.74	0.85
9	Refused	1,468	0.95	1.00
BLANK	Not asked or Missing	310,602		

Formatting a frequency table

- While the numbers are correct, this table is poorly formatted and there is no way to know what any of the numbers mean.
- A table should include several features to make the contents of the table clear, including:
    - A main title indicating what is in the table with:
        - the overall sample size
        - key pieces of information that describe the data such as the year of data collection and the data source
    - Clear column and row labels with:
        - logical row and column names
        - a clear indication of what the data are such as means or frequencies
        - row and column sample sizes when they are different from overall sample size

Data management to improve table formatting

- One thing to do might be to add **labels** to the transgender variable so that it is clear which categories the frequencies represent.

- Labels are the words that describe each category of a categorical or factor type variable.

- To ensure the data type is correct for the transgender variable (`TRNSGNDR`), examine the data type with the `class()` function.

```r
# check data type for transgender variable
class(x = brfss.trans.2014$TRNSGNDR)
```

```
## [1] "integer"
```

Change data type with mutate()

- The class of `TRNSGNDR` was *integer*.
- The variable has categories and so should be the factor data type in R.
- Change the data type of `TRNSGNDR` to a factor using the **tidyverse** package with the `mutate()` function and `as.factor()`.
- Give the data a new name of `brfss.2014.cleaned` to keep the original data and the cleaned data in separate objects.

```
# open tidyverse for data management
library(package = "tidyverse")

# change variable from numeric to factor
brfss.2014.cleaned <- brfss.trans.2014 %>%
  mutate(TRNSGNDR = as.factor(TRNSGNDR))

# check data type again
class(x = brfss.2014.cleaned$TRNSGNDR)
```

```
## [1] "factor"
```

Add category labels with mutate() and recode_factor()

- The `recode_factor()` function works to recode AND can also change the variable to a factor.

- To save a line of code use `recode_factor()` with `mutate()` instead of using `as.factor()` and then `recode_factor()`.

- `TRNSGNDR` category names in the codebook:
    - 1 = Male to female
    - 2 = Female to male
    - 3 = Gender non-conforming
    - 4 = Not transgender
    - 7 = Not sure
    - 9 = Refused
    - NA

Add category labels and change to factor

- Start with the previous code and change `mutate()` to use `recode_factor()` with the category labels.

- `recode_factor()` requires the original value of the variable on the left side of the `=` and that the original values are enclosed in backticks because the values are considered **names** rather than numbers.

- In R, names are labels given to a category or a variable or another object.

- Names that begin with a number are enclosed in backticks (or quote marks) in order to be recognized by R as a name and not a number.

```
# cleaning the TRNSGNDR variable
brfss.2014.cleaned <- brfss.trans.2014 %>%
  mutate(TRNSGNDR = recode_factor(.x = TRNSGNDR,
                                  `1` = 'Male to female',
                                  `2` = 'Female to male',
                                  `3` = 'Gender non-conforming',
                                  `4` = 'Not transgender',
                                  `7` = 'Not sure',
                                  `9` = 'Refused'))
```

Remembering the quirks of R & finding help

- R (and most coding languages) have lots of little rules to remember, like putting numbers in backticks for `recode_factor()`.

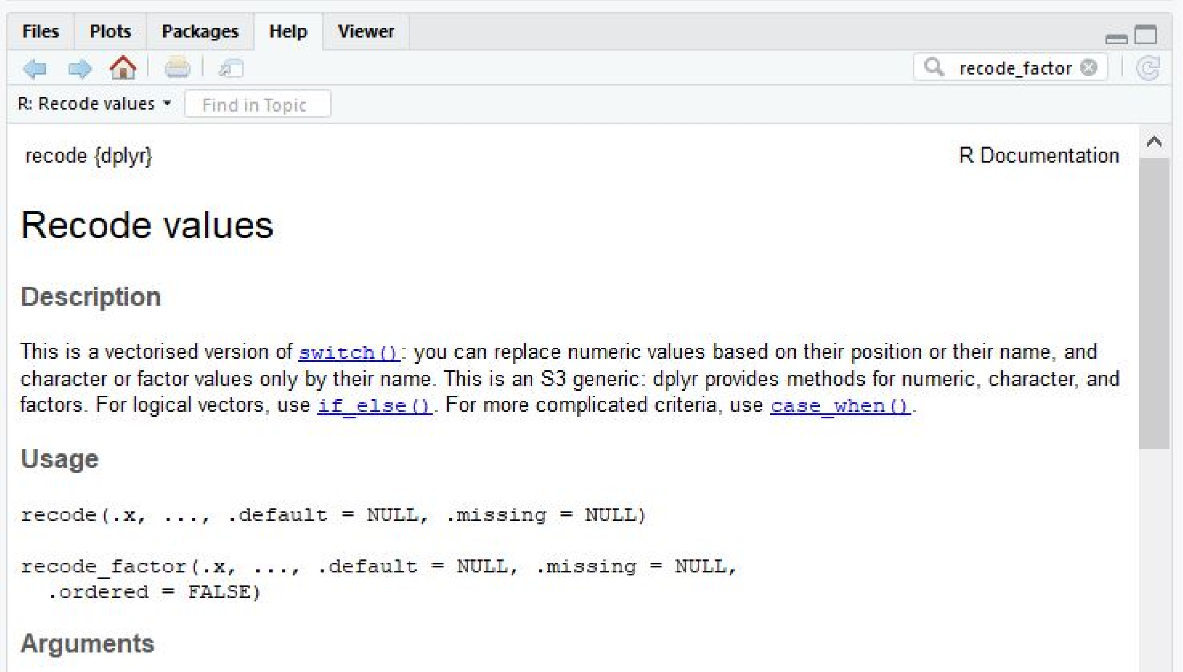
- Some rules you will remember, especially if you use a function a lot, otherwise using the help documentation and Google is a common and daily occurence for most people writing code.

- The help documentation is easy to use directly from R Studio and can be found under one of the tabs in the bottom right pane.

    - Try typing "recode_factor" in the search box at the top of the help pane and the documentation for `recode_factor()` appears in the pane.
    - Another way to get help documentation is to type a single "?" and then the name of the function into the Console (`?recode_factor`) and press Return or Enter, and the help page will appear under the Help tab the lower right pane.
    - Finally, putting the cursor in the function and pressing F1 will also display the help documentation for that function

Using the R help documentation

- One feature of the help documentation is that the package for a function shows at the top left of the help documentation.
- The documentation lists *{dplyr}* in the top left for `recode_factor()`, so it is in the **dplyr** package.



recode {dplyr}   R Documentation

Recode values

Description

This is a vectorised version of `switch()`: you can replace numeric values based on their position or their name, and character or factor values only by their name. This is an S3 generic: dplyr provides methods for numeric, character, and factors. For logical vectors, use `if_else()`. For more complicated criteria, use `case_when()`.

Usage

```
recode(.x, ..., .default = NULL, .missing = NULL)

recode_factor(.x, ..., .default = NULL, .missing = NULL,
  .ordered = FALSE)
```

Arguments

Checking the table after recoding

- Try re-running the `table()` code from above by:
  - Opening the History tab
  - Double clicking on the `table(brfss.2014.cleaned$TRNSGNDR)` code to send it to the console, and
  - Pressing enter or return

```r
# table of transgender status frequencies
table(brfss.2014.cleaned$TRNSGNDR)
```

```
## 
##        Male to female        Female to male Gender non-conforming 
##                   363                   212                   116 
##       Not transgender              Not sure               Refused 
##                150765                  1138                  1468
```

Tricky factor facts

- Each category of a factor is called a level in R and that these levels can also have labels, which is what was added with `recode_factor()` for `TRNSGNDR`.

- The `levels()` function can be used to know the categories for a factor variable.

- One tricky part is that R will treat each unique value of a factor as a different level.

  - So for a vector saved as a factor that is `height <- c("short", "tall", "Short", "tll")`, R would consider it to be have 4 levels: short, tall, Short, and tll even though it seems clear that the first and third observations should be part of the same category.

  - R is case sensitive and will not catch spelling errors.

  - So in order to get a factor with 2 levels, it would need to look like `height <- c("short", "tall", "short", "tall")`.

- Use `levels()` to see if the levels need to be cleaned up.