2

COMPUTING AND REPORTING DESCRIPTIVE STATISTICS

The R-Team and the Troubling Transgender Health Care Problem

Leslie was excited for the second meeting of the R-Team. She met Kiara and Nancy at a local café for lunch.

Nancy said, "Today, I thought we would discuss descriptive statistics." Consistent with the name, she explained, descriptive statistics are statistics that describe whatever is being studied, like people or organizations. Descriptive statistics are important since they provide information about the characteristics of the observations in a data set and provide context for interpretation of the results from complex statistical tests. Descriptive statistics may also be used to help choose the correct statistical test to use to answer your questions.

"Sounds good," Leslie said. Then she shared the rest of her fries with the team. After getting used to the R environment and preparing data for analysis with the pot policy problem, Leslie had her own idea to propose. "So I was listening to a podcast about the issue of transgender health and how transgender patients are treated in a system that has traditionally focused on two sexes, and I wondered if we might tackle that with R?"

"Totally," said Kiara.

"Right after I finish my sandwich," Nancy said.

Kiara chuckled and said, "After that, let's develop a list of R achievements to improve Leslie's R skills while we work together on understanding transgender health care."

2.1 Achievements to unlock

- Achievement 1: Understanding variable types and data types
- Achievement 2: Choosing and conducting descriptive analyses for categorical (factor) variables
- Achievement 3: Choosing and conducting descriptive analyses for continuous (numeric) variables
- Achievement 4: Developing clear tables for reporting descriptive statistics

2.2 The transgender health care problem

Before coming to meet with Nancy and Kiara, Leslie had done some reading on the problem, including a recent paper on mammography that demonstrated some of the challenges for transgender people in the current system. Over lunch, she summarized what she'd found.

First, Leslie wanted to clarify the definition of transgender to make sure they were all on the same page. Leslie told them she thought the term *sex* referred to biology and the term *gender* referred to identity. Nancy agreed with the definition of sex, but had heard broader definitions of gender that included biological, psychosocial, and cultural factors along with identity (Mayer et al., 2008). She looked up the World Health Organization's definition for gender:

> Gender refers to the socially constructed characteristics of women and men—such as norms, roles and relationships of and between groups of women and men. It varies from society to society and can be changed. While most people are born either male or female, they are taught appropriate norms and behaviours—including how they should interact with others of the same or opposite sex within households, communities and work places. (World Health Organization [WHO], 2019)

Note: In shaded sections throughout this text, the rows starting "##" show the output that will appear after running the R code just above it.

Leslie was glad they looked this up because it was much clearer than her initial description. She went on to explain that *transgender* people are people whose *biological sex* is not consistent with their *gender* (Mayer et al., 2008). She quoted an article from 2008 that she thought had a definition that might be helpful: "Transgender is an inclusive term to describe people who have gender identities, expressions, or behaviors not traditionally associated with their birth sex" (Mayer et al., 2008, p. 990).

The paper Leslie read gave a few examples of transgender, including a person who is biologically female but identifies as a man, a person who is biologically male but identifies as a woman, or someone who feels like both genders or neither gender (Mayer et al., 2008). Nancy looked at the Centers for Disease Control and Prevention (CDC) website, which explained that gender identity is the gender a person identifies as, and gender expression is the gender a person outwardly shows (CDC, n.d.). When describing individuals, typically the person's chosen identity is included. So, a person who is biologically female but identifies as a man would be described as a "transgender man." Kiara said that was consistent with what she'd experienced when she learned that her second cousin was transgender. For a while, it had been difficult to remember to refer to her cousin, who had previously been *she*, as *he*. Leslie and Nancy nodded. Sensing they were ready to proceed, Leslie gave some background on health and health care for transgender people.

Transgender people face serious physical and mental health disparities (Safer et al., 2016), often related to discrimination (Bradford, Reisner, Honnold, & Xavier, 2013). Among the most prevalent and concerning disparities are extremely high rates of HIV and other sexually transmitted infections, poor mental health, and high rates of substance use and abuse. For example, transgender women have a 19.1% prevalence of HIV worldwide (Baral et al., 2013), while reproductive-aged adults have an HIV prevalence of 0.4%. A survey of transgender adults in the United States found that 44.1% have clinical depression, 33.2% have anxiety, and 27.5% have somatization symptoms (i.e., physical symptoms stemming from depression, anxiety, or another psychiatric condition; Lombardi, 2010). Attempted suicide rates range from 32% to 50% of transgender people worldwide, with higher rates among younger age groups (Virupaksha, Muralidhar, & Ramakrishna, 2016).

In 2001, an Institute of Medicine report found that 35.5% of transgender adults are current smokers; this is more than twice the percentage of cisgender (i.e., people whose gender identity matches their biological sex) adults (14.9% smoking rate) (Institute of Medicine, 2001). Transgender adults also have higher rates of alcohol and substance abuse (Benotsch et al., 2013; Santos et al., 2014) and of overweight and obesity (Boehmer, Bowen, & Bauer, 2007). Exacerbating these health disparities are limitations on access to health care (Meyer, Brown, Herman, Reisner, & Bockting, 2017) and limitations of the health care system to effectively address the unique health needs of transgender people (Daniel & Butkus, 2015; Reisner et al., 2016).

Given the high smoking rate, substance and alcohol use, and the prevalence of overweight and obesity, it is important that transgender people are adequately screened for cancer and other health conditions. Leslie found conflicting evidence about screening for transgender people. A 2015 study that examined medical charts from 2012–2013 found that transgender adults had 47% lower odds of breast cancer screening than cisgender adults (Bazzi, Whorms, King, & Potter, 2015). However, a different study published in 2017 and based on 2014 survey data reported that transgender and nontransgender adults have comparable rates of mammography screening (Narayan, Lebron-Zapata, & Morris, 2017). Nancy suggested maybe the rate is increasing over time, but Kiara thought the data were not collected far enough apart for that to be the case. She thought the survey data, which were a representative sample of U.S. residents from the Behavioral Risk Factor Surveillance System (BRFSS) survey, in the 2017 study likely measured people with different characteristics from the people in the 2015 study, which reviewed charts from people seeking care at an urban clinic.

Leslie looked up the BRFSS (https://www.cdc.gov/brfss/index.html) and found that it is a large national survey conducted by the CDC in the United States by phone every year. The 2017 mammogram paper was based on data from participants who responded to the 2014 BRFSS survey. The 2014 BRFSS was

administered by states and had 464,664 participants. The BRFSS sampling process was designed so that the sample would represent the U.S. population as much as possible. The survey included questions about characteristics like age, sex, and income, along with questions about health behaviors and outcomes.

In 2014, 19 states and Guam included a question on the BRFSS about transgender transition status, with choices of male to female (MtF), female to male (FtM), or non-conforming. The entire sample from the 19 states and Guam contained 151,456 people. Of the 151,456 participants, the study authors identified 222 survey participants who reported they were transgender, were 40–74 years old (the age range when mammograms are recommended), and were asked about having a mammogram (Narayan, Lebron-Zapata, & Morris, 2017). The mammogram question was only asked of those who described themselves as being female sex.

Leslie shared the first table of statistics, published as Table 1 in Narayan et al. (2017), with Kiara and Nancy.

Kiara explained that, prior to examining statistical relationships among variables, published studies in academic journals usually present descriptive statistics like the ones in Table 2.1. She added that descriptive statistics are often displayed in the first table in a published article or report and are therefore sometimes referred to as the *Table 1* statistics or, even more commonly, as the *descriptives*. She also pointed out that, in this case, all the descriptives in the transgender health paper were percentages

TABLE 2.1 Reproduced Table 1, "Transgender Survey Participant Demographics," from Narayan et al. (2017)

Survey participant demographics (*n* = 220)	Percent
Transition status	
Male to female	35.0
Female to male	50.9
Gender non-conforming	14.1
Age category	
40–44	12.2
45–49	12.2
50–54	14.4
55–59	19.8
60–64	19.8
65–69	10.8
70–74	10.8
Race/ethnicity	
White	68.5
Black	14.0
Native American	1.8
Asian/Pacific Islander	2.7
Other	13.1

(Continued)

TABLE 2.1 (Continued)

Survey participant demographics (n = 220)	Percent
Income category	
Less than $15,000	20.7
$15,000 to less than $25,000	19.8
$25,000 to less than $35,000	8.6
$35,000 to less than $50,000	11.7
$50,000 or more	29.3
Don't know/not sure/missing	9.9
Education category	
Did not graduate high school	10.8
Graduated high school	38.7
Attended college/technical school	30.6
Graduated from college/technical school	19.8
Health insurance?	
Yes	89.2

showing the percent of people in each category. She explained that percentages were computed for this table because of the types of variables the researchers examined. Now that they had some background and an idea of the topic and data they were working with, Nancy and Kiara were ready to get started.

2.3 Data, codebook, and R packages for learning about descriptive statistics

Before they examined the data, Kiara made a list of all the data, documentation, and R packages needed for learning descriptive statistics.

- Two options for accessing the data*
  - Download the clean data file **transgender_hc_ch2.csv** from **edge.sagepub.com/harris1e**
  - Follow the instructions in Box 2.1 to download and clean the data directly from the Internet
- Two options for accessing the codebook
  - Download **brfss_2014_codebook.pdf** from **edge.sagepub.com/harris1e**
  - Download the codebook from the Behavioral Risk Factor Surveillance Survey website (https://www.cdc.gov/brfss/index.html)

*While many internet data sources are stable, some will be periodically updated or corrected. Use of updated data sources may result in slightly different results when following along with the examples in the text.

- Install R packages if not already installed
  - `tidyverse`, by Hadley Wickham (https://www.rdocumentation.org/packages/tidyverse/)
  - `haven`, by Hadley Wickham (https://www.rdocumentation.org/packages/haven/)
  - `Hmisc`, by Frank E. Harrell Jr. (https://www.rdocumentation.org/packages/Hmisc/)
  - `descr`, by Jakson Alves de Aquino (https://www.rdocumentation.org/packages/descr/)
  - `tableone` (Yoshida, n.d.)
  - `kableExtra`, by Hao Zhu (https://www.rdocumentation.org/packages/kableExtra/)
  - `semTools`, by Terry Jorgensen (https://www.rdocumentation.org/packages/semTools/)
  - `qualvar`, by Joel Gombin (https://cran.r-project.org/web/packages/qualvar/index.html)
  - `labelled`, by Joseph Larmarange (https://www.rdocumentation.org/packages/labelled/)
  - `knitr`, by Yihui Xie (https://www.rdocumentation.org/packages/knitr/)

2.4 Achievement 1: Understanding variable types and data types

2.4.1 DATA TYPES FOR CATEGORICAL VARIABLES

In Section 1.7, Kiara introduced Leslie to several data types, including factor, numeric, integer, and character. Arguably, the two data types most commonly used in social science research are factor and numeric. Kiara reminded Leslie that the factor data type is used for information that is measured or coded in categories, or categorical variables. Leslie remembered from the previous meeting that there were two types of categorical variables, nominal and ordinal.

Nominal categorical variables are variables measured in categories that do not have any logical order, like marital status, religion, sex, and race. Ordinal categorical variables have categories with a logical order. One way Leslie liked to remember this was to notice that ordinal and order both start with *o-r-d*, so ordinal variables have an order. For example, income can be measured in categories such as <$10,000, $10,000–$24,999, $25,000–$99,999, and $100,000+. These categories have a logical order from the lowest income group to the highest income group.

One common ordinal way of measuring things is the use of a ***Likert scale***. Likert scales have categories that go in a logical order from *least to most* or *most to least*. For example, Leslie suggested that measuring agreement with the statement "R is awesome" could use a Likert scale with the following options: *strongly agree, somewhat agree, neutral, somewhat disagree, strongly disagree*. Leslie explained that often people refer to the number of categories as the *points* in a Likert scale. The agreement scale from *strongly agree* to *strongly disagree* is a 5-point Likert scale because there are five options along the scale.

2.4.2 DATA TYPES FOR CONTINUOUS VARIABLES

On the other hand, explained Kiara, the numeric data type in R is used for continuous variables. Leslie confirmed that continuous variables are the variables that can take *any* value along some continuum, hence continuous. Just like *o-r-d* is in order and ordinal, *c-o-n* is in continuum and continuous, which can be a good way to remember this variable type. Examples of continuous variables include age, height, weight, distance, blood pressure, temperature, and so on.

Kiara explained that the numeric data type is also often used for variables that do not technically qualify as continuous but are measured along some sort of a continuum. Age in years would be one example since it falls along a continuum from 0 years old to over 100. However, by specifying age *in years,* the

values for this variable cannot be any value between 0 and over 100, but instead are limited to the whole numbers in this range. Likewise, the number of cars in a parking lot, or the number of coins in a piggy bank, would be numeric but not truly continuous. If the variable is a whole number, the integer data type could also be used, but it has more limitations for analysis than does the numeric data type in R.

Leslie knew that each different type of variable required a different approach for analysis. Before getting into any complex statistical modeling, she remembered that descriptive statistics are key for understanding who or what is in the data set and often help in choosing an appropriate statistical test.

2.4.3 ACHIEVEMENT 1: CHECK YOUR UNDERSTANDING

Identify the most appropriate data type for the following variables:

- Number of healthy days per month
- Marital status
- Religious affiliation
- Smoking status
- Number of alcoholic beverages per week

2.5 Achievement 2: Choosing and conducting descriptive analyses for categorical (factor) variables

Since the table from the paper (Table 2.1) shows all categorical variables, the R-Team started there. Leslie knew that the two most commonly used and reported descriptive statistics for categorical (or factor) data types are frequencies and percentages. For a categorical variable, frequencies are the number of observations—often people or organizations—that fall into a given category. Percentages are the proportion of all observations that fall into a given category.

2.5.1 COMPUTING FREQUENCIES AND FREQUENCY DISTRIBUTIONS

Leslie also remembered that a ***frequency distribution*** shows the number of observations in each category for a factor or categorical variable. She suggested that they could use a frequency distribution to examine how many observations there were for each transgender transition category in the BRFSS data set from the mammogram paper.

Before getting started, the team looked through the 2014 BRFSS codebook together, which they found on the CDC BRFSS website (CDC, 2015). They found the transgender transition status information on page 83 of the codebook. The frequency distribution shown in the codebook indicated 363 MtF transgender, 212 FtM transgender, 116 gender non-conforming, 150,765 not transgender, 1,138 don't know/not sure, 1,468 refused, and 310,602 not asked or missing (Figure 2.1).

Kiara suggested that Leslie re-create this frequency distribution in R by first opening the BRFSS data file and then using an R function for frequencies. On the BRFSS website (https://www.cdc.gov/brfss/index.html), the data file is available in ***xpt*** or ***ASCII*** format. The xpt is a file format used by SAS statistical software; Kiara explained that one of the great features of R is that it can open most data sets from most software packages. This is one of the features of R that makes it easier to do reproducible work. She said R can open the xpt file type by using the `haven` package that is part of the `tidyverse` in R. For now, Kiara said she had already downloaded, opened, and cleaned the data, so they did not need to do all of

FIGURE 2.1 2014 Behavioral Risk Factor Surveillance Survey (BRFSS) codebook page 83

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		

2.1 Kiara's reproducibility resource: Bringing data in directly from the Internet

The more that can be done in R with documented and organized code, the more reproducible analyses will be. If the entire process of importing, managing, analyzing, and reporting results can all be together in a single document, the opportunity for errors is greatly decreased.

The BRFSS data set can be found on the CDC website in a zipped SAS transport file with the file extension of xpt. It is possible to bring data directly into R from websites. One of the ways to do this is with the **haven** package. The process is not straightforward like the `read.csv()` function or other importing functions are. The **haven** package cannot unzip and import an xpt file directly from the Internet, so the file has to be imported into a temporary location on a local computer and then read using `read_xpt()` from **haven**. Kiara wrote out the steps for downloading, importing, and exporting the data from a zipped xpt file online. Note that the `download.file()` function will take several minutes to run on most personal computers.

```
# open the haven package to read an xpt
library(package = "haven")

# create a temporary file to store the zipped file
# before you open it
temp <- tempfile(fileext = ".zip")
```

(Continued)

(Continued)

```
# use download.file to put the zipped file in the temp file
# this will take a couple of minutes depending on computer speed
download.file(url = "http://www.cdc.gov/brfss/annual_data/2014/files/
LLCP2014XPT.zip", destfile = temp)

# unzip it and read it
brfss.2014 <- read_xpt(file = temp)

# open tidyverse to select variables
library(package = "tidyverse")

# select variables for analysis
# use ` around variable names starting with underscores
transgender_hc_ch2 <- brfss.2014 %>%
  select(TRNSGNDR, `_AGEG5YR`, `_RACE`, `_INCOMG`, `_EDUCAG`,
      HLTHPLN1, HADMAM, `_AGE80`, PHYSHLTH)

# export the data set to a csv file in a local folder called data
write.csv(x = transgender_hc_ch2,
          file = "[data folder location]/data/transgender_hc_ch2.csv",
          row.names = FALSE)
```

that. Leslie was interested in understanding how she did this, so Kiara shared with Leslie the code she used to download the data so that she could review it later (Box 2.1).

After importing the data, Kiara exported a ***csv*** (comma separated values) file with the variables needed to re-create the table. Nancy reminded Leslie of the `read.csv()` function to import the data from their previous meeting. Leslie wrote a line of code, being sure to use the object naming recommendations she'd learned during the data preparation day. She entered the location where the **transgender_hc_ch2.csv** data file was saved between the quote marks after `file =` in the `read.csv()` function.

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

Leslie highlighted the code and used the Run button to import the data. The data appeared in the Environment tab in the top right pane of RStudio. The entry in the Environment tab was under the Data heading and was called `brfss.trans.2014` (the name given to the object). Next to the data frame name, Leslie saw that there were 464,664 observations and nine variables in the data frame.

This indicates that the data set contains 464,664 observations, or 464,664 people in this case. This was consistent with the codebook, so Leslie was assured that these were the data she was looking for.

Leslie used `summary()` to learn more about the data before starting to compute other statistics.

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

Leslie thought it was good to see the variable names and some information about each variable before she started on the frequency tables. She slid her laptop over to Nancy, who looked ready to code.

2.5.2 MAKING A BASIC TABLE OF FREQUENCIES AND PERCENTAGES

Nancy started by showing Leslie an easy way to get a frequency distribution in R, which 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 Nancy wrote `table(brfss.trans.2014$TRNSGNDR)` and used a keyboard shortcut to run the line of code.

```
# frequency distribution for transgender
# participants in the 2014 BRFSS
table(brfss.trans.2014$TRNSGNDR)
##
##      1      2      3      4      7      9
##    363    212    116 150765   1138   1468
```

The output showed a set of numbers with the top row representing the categories and the bottom row giving the number of observations in each category. The number of observations in each category is the *frequency*. The frequencies in this simple table matched 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 from Nancy's code.

While the numbers were correct, Nancy suggested that this table was poorly formatted, and there was no way to know what any of the numbers meant without more information. She explained that a table should include several features to make the contents of the table clear, such as the following:

- A main title indicating what is in the table, including
  - 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 that have
  - 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 sizes

2.5.3 DATA MANAGEMENT

In order to make a clear frequency table, Nancy advised Leslie to use some of the tools from their first meeting along with some new tools. The first thing to do might be to add labels to the `TRNSGNDR` variable so that it is clear which categories the frequencies represent. Leslie reminded herself that labels are the words that describe each category of a categorical- or factor-type variable. Before she could add labels, Leslie needed to know the data type of the `TRNSGNDR` variable. She started by examining the data type of the variable with the `class()` function. To get the class of `TRNSGNDR`, Leslie added `brfss.trans.2014$TRNSGNDR` to the `class()` function.

```
# check data type for TRNSGNDR variable
class(x = brfss.trans.2014$TRNSGNDR)
## [1] "integer"
```

The class was *integer*, which was not what Leslie was expecting. Leslie knew that this variable had categories and so should be the factor data type in R (Section 1.7). She started her data management by changing the data type of `TRNSGNDR` to a factor using the **`tidyverse`** package with the `mutate()` function and `as.factor()`. She gave the data a new name of `brfss.2014.cleaned` so that she had the original data and the cleaned data in separate objects.

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

# change variable from integer 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"
```

Now Leslie wanted to add the category labels to the variable. She decided to add this onto the data management code she had already started. Nancy said this sounded great, and that the `recode_factor()` they used in the data preparation activities could work to recode *and* to change the variable to a factor. She suggested Leslie could save a line of code and just use `recode_factor()` with `mutate()` instead of using `as.factor()` and then `recode_factor()`. Leslie looked skeptical but went ahead and found the category names in the codebook (https://www.cdc.gov/brfss/annual_data/2014/pdf/CODEBOOK14_LLCP.pdf).

- 1 = Male to female
- 2 = Female to male
- 3 = Gender non-conforming
- 4 = Not transgender
- 7 = Not sure
- 9 = Refused
- NA

While Leslie was distracted with the codebook, Nancy seized her chance to do some of the coding. She slid the laptop over and changed the `mutate()` code to use `recode_factor()` with the category labels. When she was done, she shared her work with Leslie. Nancy showed Leslie that `recode_factor()` requires the original value of the variable on the left side of the = and that the original values are enclosed in backticks (`` `value` ``) 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'))
```

Leslie wondered aloud how she would ever be able to remember the rule about putting numbers in backticks for `recode_factor()`. Nancy said that some code is easy to remember for her, especially if she uses it a lot, but otherwise she uses the help documentation and the Internet almost every day she is coding. She reminded Leslie that the help documentation is easy to use directly from RStudio and can be found under one of the tabs in the bottom right pane. She showed Leslie how to type "recode_factor" in the search box at the top of the help pane, and the documentation for `recode_factor()` appears in the pane. She also shared that Leslie can 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 in the lower right pane.

While she had the help documentation open, Nancy showed Leslie one feature of the help documentation, which is that the package for a function shows at the top left of the help documentation (Figure 2.2). In this case, the documentation lists *{dplyr}* in the top left, so `recode_factor()` is in the **`dplyr`** package. Occasionally, Kiara said, she remembers some function to use but not the package it comes from, so the help documentation can be useful for that as well as for how to use a function.

Once they were done looking at the help pane, Nancy suggested Leslie rerun `table()` from above. Leslie used the trick of opening the History tab and double-clicking on the `table(brfss.2014.cleaned$TRNSGNDR)` code to send it to the Console. She pressed Enter to run the code from there.

```
# 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
```

Nancy wanted to give a quick caution about factor variable recoding. Since factors are particularly useful for social scientists, it is worthwhile to understand a bit more about how R treats factors because they can

FIGURE 2.2 Screenshot of recode help documentation

sometimes be tricky. She explained that each category of a factor is called a level in R and that these levels can be plain numbers or have labels, which is what they had added with `recode_factor()` for `TRNSGNDR`. She explained that the `levels()` function can be used to know what the categories are for a factor variable.

The 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 looks like `height <- c("short", "tall", "Short", "tll")`, R would consider it to have four levels: short, tall, Short, and tll. Leslie looked confused because the first and third observations should be part of the same category. Nancy said that short and Short are different categories because the first letter is capitalized in one and not the other. The same thing is true with tall and tll—spelling matters! So in order to get a factor with two levels, it would need to look like `height <- c("short", "tall", "short", "tall")`. Nancy suggested that Leslie could check the levels of a factor using `levels()` to see if the levels need to be cleaned up.

For now, the table looked better, but it still needed some work on the formatting before it was ready to share with others. Leslie started thinking about the `summary()` function from their last session and wondered if that might be a good way to get frequencies for more than one variable at a time. Nancy thought this was an interesting idea and they tried using `summary()`.

```
# use summary for frequencies
summary(object = brfss.2014.cleaned)
##                    TRNSGNDR        X_AGEG5YR        X_RACE
##  Male to female       :   363   Min.   : 1.000  Min.   : 1.000
##  Female to male       :   212   1st Qu.: 5.000  1st Qu.: 1.000
##  Gender non-conforming:   116   Median : 8.000  Median : 1.000
##  Not transgender      :150765   Mean   : 7.822  Mean   : 1.992
##  Not sure             :  1138   3rd Qu.:10.000  3rd Qu.: 1.000
##  Refused              :  1468   Max.   :14.000  Max.   : 9.000
##  NA's                 :310602                   NA's   :94
##     X_INCOMG         X_EDUCAG         HLTHPLN1          HADMAM
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.00
##  1st Qu.:3.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.00
##  Median :5.000   Median :3.000   Median :1.000   Median :1.00
##  Mean   :4.481   Mean   :2.966   Mean   :1.108   Mean   :1.22
##  3rd Qu.:5.000   3rd Qu.:4.000   3rd Qu.:1.000   3rd Qu.:1.00
##  Max.   :9.000   Max.   :9.000   Max.   :9.000   Max.   :9.00
##                                                  NA's   :208322
##     X_AGE80         PHYSHLTH
##  Min.   :18.00   Min.   : 1.0
##  1st Qu.:44.00   1st Qu.:20.0
##  Median :58.00   Median :88.0
##  Mean   :55.49   Mean   :61.2
##  3rd Qu.:69.00   3rd Qu.:88.0
##  Max.   :80.00   Max.   :99.0
##                  NA's   :4
```

They reviewed the results and decided that `summary()` was really similar to `table()`, and the format does not look even close to what they want for the formatted table. Leslie wondered if there was any other code like `summary()` that summarized all the variables at once. Nancy said she has used `describe()` from the **`Hmisc`** package. Leslie recalled that a package must be installed once, but after it's installed there are two ways to access the functions that come from the package. The first way is to load the entire package using `library(package = "Hmisc")` or using the `::` structure. Since they just wanted to test out `describe()`, Leslie tried using `::`.

```
# trying describe for descriptive statistics
Hmisc::describe(x = brfss.2014.cleaned)
## brfss.2014.cleaned
##
##  9  Variables      464664  Observations
## ---------------------------------------------------------------------------
## TRNSGNDR
##        n  missing distinct
##   154062   310602        6
##
## Male to female (363, 0.002), Female to male (212, 0.001), Gender
## non-conforming (116, 0.001), Not transgender (150765, 0.979), Not sure
## (1138, 0.007), Refused (1468, 0.010)
## ---------------------------------------------------------------------------
## X_AGEG5YR
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   464664        0       14    0.993    7.822    3.952        1        3
##      .25      .50      .75      .90      .95
##        5        8       10       12       13
##
## Value          1     2     3     4     5     6     7     8     9    10
## Frequency  24198 19891 23662 25444 28597 32686 43366 49432 52620 50680
## Proportion 0.052 0.043 0.051 0.055 0.062 0.070 0.093 0.106 0.113 0.109
##
## Value         11    12    13    14
## Frequency  40160 29310 38840  5778
## Proportion 0.086 0.063 0.084 0.012
## ---------------------------------------------------------------------------
## X_RACE
##        n  missing distinct     Info     Mean      Gmd
##   464570       94        9    0.546    1.992    1.687
##
## Value          1      2      3      4      5      6      7      8      9
```

```
## Frequency  356808  35062   7076   9314   1711   2109   8824  35838  7828
## Proportion  0.768  0.075 0.015a  0.020  0.004  0.005 0.019   0.077 0.017
## ------------------------------------------------------------------------
## X_INCOMG
##        n  missing distinct     Info     Mean      Gmd
##   464664        0        6    0.932    4.481    2.522
##
## Value           1      2      3      4      5      9
## Frequency   44142  68042  44315  57418 179351  71396
## Proportion  0.095  0.146  0.095  0.124  0.386  0.154
## ------------------------------------------------------------------------
## X_EDUCAG
##        n  missing distinct     Info     Mean      Gmd
##   464664        0        5    0.911    2.966    1.153
##
## Value           1      2      3      4      9
## Frequency   37003 131325 125635 166972   3729
## Proportion  0.080  0.283  0.270  0.359  0.008
## ------------------------------------------------------------------------
## HLTHPLN1
##        n  missing distinct     Info     Mean      Gmd
##   464664        0        4    0.233    1.108   0.2022
##
## Value           1      2      7      9
## Frequency  425198  37642    934    890
## Proportion  0.915  0.081  0.002  0.002
## ------------------------------------------------------------------------
## HADMAM
##        n  missing distinct     Info     Mean      Gmd
##   256342   208322        4    0.483    1.215   0.3488
##
## Value           1      2      7      9
## Frequency  204705  51067    253    317
## Proportion 0.799   0.199  0.001  0.001
## ------------------------------------------------------------------------
## X_AGE80
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   464664        0       63    0.999    55.49    19.22    24 30
##      .25      .50      .75      .90      .95
```

```
##       44       58       69       78       80
##
## lowest : 18 19 20 21 22, highest: 76 77 78 79 80
## ------------------------------------------------------------------------
## PHYSHLTH
##        n  missing distinct     Info     Mean      Gmd      .05      .10
##   464660        4       33    0.752     61.2    36.37        2        3
##      .25      .50      .75      .90      .95
##       20       88       88       88       88
##
## lowest :  1  2  3  4  5, highest: 29 30 77 88 99
## ------------------------------------------------------------------------
```

They looked through the output from `describe()` and saw that proportions were included, which is something that was not in the `summary()` output. However, there was a lot of output to read through to find specific information. Kiara mentioned that she prefers to use functions with output that is more targeted to what she is interested in, so she just uses `summary()` and `describe()` to get a sense of the data or to confirm that her data management tasks have worked, but uses other options once she is ready to create tables or other displays of data. Leslie thought this was a good strategy, although she was curious about alternatives from other packages.

Nancy said that the **`descr`** (short for descriptives) package has a good option for a basic table of frequencies and percentages with the `freq()` function. Nancy also knew that a graph was automatically printed with the `freq()` output and the graph is not useful at this point, so she showed Leslie how she uses the `plot = FALSE` option with `freq()` to stop the graph from printing with the output. Since she just wanted the one function from **`descr`**, Nancy used the `::` format to access the `freq()` function.

```
# use freq from the descr package to make a table of frequencies and percents
# suppress the bar plot that automatically prints
descr::freq(x = brfss.2014.cleaned$TRNSGNDR, plot = FALSE)
## brfss.2014.cleaned$TRNSGNDR
##                          Frequency   Percent Valid Percent
## Male to female                 363   0.07812       0.23562
## Female to male                 212   0.04562       0.13761
## Gender non-conforming          116   0.02496       0.07529
## Not transgender             150765  32.44603      97.85995
## Not sure                      1138   0.24491       0.73866
## Refused                       1468   0.31593       0.95286
## NA's                        310602  66.84443
## Total                       464664 100.00000     100.00000
```

Leslie noticed that there were two columns in the output that show percentages. She reviewed the columns and found that the Percent column includes the missing data (`NA`) in the calculation of the

percentage of observations in each category. The Valid Percent column removes the NA values and calculates the percentage of observations that falls into each category *excluding the observations missing values on this variable*.

Since they were already using the **tidyverse** package, Nancy suggested they try using it for this instead of opening a new package. She wrote some alternate code to show Leslie how she might create this table.

```
# use tidyverse to make table of frequency and percent
brfss.2014.cleaned %>%
  group_by(TRNSGNDR) %>%
  summarize(freq.trans = n()) %>%
  mutate(perc.trans = 100 * (freq.trans / sum(freq.trans)))
## # A tibble: 7 x 3
##   TRNSGNDR              freq.trans perc.trans
##   <fct>                      <int>      <dbl>
## 1 Male to female               363     0.0781
## 2 Female to male               212     0.0456
## 3 Gender non-conforming        116     0.0250
## 4 Not transgender           150765    32.4
## 5 Not sure                    1138     0.245
## 6 Refused                     1468     0.316
## 7 <NA>                      310602    66.8
```

Leslie was not impressed. She thought this looked like a lot more code to get the same information they got from `freq()`. Nancy agreed that the code was more complicated to get the same table. She explained that sometimes it was worth the time to use the more complicated code in order to keep a consistent style or to be able to use the code again later. Leslie nodded but remained skeptical. Kiara pointed out that this is an example of how you can do the same thing in a number of different ways in R. Often, there is no single correct way.

Nancy noted that both the `freq()` and **tidyverse** tables included the information needed for reporting, but they were still not formatted in a way that could be directly incorporated into a report or other document. Leslie suggested that they copy these results into a word-processing program and format the table there.

Kiara cringed at this suggestion. Copying from statistical output into another document is where many errors occur in scientific reporting (Harris et al., 2019) and is a major reason why some research results are not reproducible. It is more efficient and accurate to create a formatted table in R and use it without transcribing or changing the contents. Nancy suggested there were two great options for creating reproducible tables directly in R using the **tableone** and **kableExtra** packages. Kiara promised to explain how to make pretty tables later, but for now the R-Team agreed to use the **tidyverse** to make their table.

Before they moved on to numeric variables, Leslie wanted to make sure she understood how to report and interpret the descriptive statistics for factor variables. She wrote a few sentences to share with Nancy and Kiara.

> The 2014 BRFSS had a total of 464,664 participants. Of these, 310,602 (66.8%) were not asked or were otherwise missing a response to the transgender status question. A few participants refused to answer ($n = 1{,}468$; 0.32%), and a small number were unsure of their status ($n = 1{,}138$; 0.24%). Most reported being not transgender ($n = 150{,}765$; 32.4%), 116 were gender non-conforming (0.03%), 212 were female to male (0.05%), and 363 were male to female (0.08%).

Nancy and Kiara agreed that this was good for reporting frequencies and percentages. Leslie thought it might be more useful to just include the *valid* percentages for people who responded to the question. Without a word, Nancy used `[]` to omit the `NA` from `TRNSGNDR` to calculate the valid percentages.

```
# use tidyverse to make table of frequency and percent
brfss.2014.cleaned %>%
  group_by(TRNSGNDR) %>%
  summarize(freq.trans = n()) %>%
  mutate(perc.trans = 100 * (freq.trans / sum(freq.trans))) %>%
  mutate(valid.perc = 100 * (freq.trans / (sum(freq.trans[na.omit(object =
TRNSGNDR)]))))
## # A tibble: 7 x 4
##   TRNSGNDR              freq.trans perc.trans valid.perc
##   <fct>                      <int>      <dbl>      <dbl>
## 1 Male to female               363     0.0781     0.236
## 2 Female to male               212     0.0456     0.138
## 3 Gender non-conforming        116     0.0250     0.0753
## 4 Not transgender           150765    32.4       97.9
## 5 Not sure                    1138     0.245      0.739
## 6 Refused                     1468     0.316      0.953
## 7 <NA>                      310602    66.8      202
```

Nancy said to just ignore the <NA> `valid.perc` for now; it is a tricky data management problem to delete this value to have a perfect table. Leslie understood and updated her sentences to share with Nancy and Kiara.

> The 2014 BRFSS had a total of 464,664 participants. Of these, 310,602 (66.8%) were not asked or were otherwise missing a response to the transgender status question. Of the 33.2% who responded, some refused to answer ($n = 1{,}468$; 0.95%), and a small number were unsure of their status ($n = 1{,}138$, 0.74%). Most reported being not transgender ($n = 150{,}765$; 97.9%), 116 were gender non-conforming (0.08%), 212 were female to male (0.14%), and 363 were male to female (0.24%).

Leslie felt much better about this interpretation. Her R-Team friends agreed.

2.5.4 ACHIEVEMENT 2: CHECK YOUR UNDERSTANDING

Use one of the methods shown to create a table of the frequencies for the `HADMAM` variable, which indicates whether or not each survey participant had a mammogram. Review the question and response options in the codebook and recode to ensure that the correct category labels show up in the table before you begin.

2.6 Achievement 3: Choosing and conducting descriptive analyses for continuous (numeric) variables

2.6.1 WHY FREQUENCY DISTRIBUTIONS DO NOT WORK FOR NUMERIC VARIABLES

Now that they had a good table of descriptive statistics for a factor variable, Nancy and Leslie switched to numeric variables. Numeric variables are measured on a continuum and can be truly continuous or just close to continuous. Leslie knew these types of variables were not well described using frequency distributions. For example, a frequency table of the age variable (`X_AGE80`) looked like this:

```
# table with frequencies from the age variable
table(brfss.2014.cleaned$X_AGE80)
##
##    18    19    20    21    22    23    24    25    26    27    28    29
##  3447  3209  3147  3470  3470  3632  3825  3982  3723  3943  4191  4054
##    30    31    32    33    34    35    36    37    38    39    40    41
##  4719  4169  4988  4888  4925  5373  5033  5109  5152  4891  5897  4672
##    42    43    44    45    46    47    48    49    50    51    52    53
##  6029  6215  6091  6463  6252  6963  6994  7019  8925  7571  9060  9015
##    54    55    56    57    58    59    60    61    62    63    64    65
##  9268  9876  9546 10346 10052 10293 11651 10310 11842 10955 10683 11513
##    66    67    68    69    70    71    72    73    74    75    76    77
## 10704 11583  9129  8092  9305  8388  8239  7381  6850  6844  6048  5845
##    78    79    80
##  5552  5021 38842
```

Nancy agreed that this display of age data provides very little useful information and pointed out that there are descriptive statistics to examine numeric variables directly. Leslie knew all about these statistics, which include measures of the central tendency and ***spread*** of the values for a numeric variable.

2.6.2 DEFINING AND CALCULATING CENTRAL TENDENCY

Leslie remembered from her statistics courses that central tendency was a measure of the center, or typical value, of a variable and that there were three measures of central tendency: mean, median, and mode.

- The *mean* is the sum of the values divided by the number of values.
- The *median* is the middle value (or the mean of the two middle values if there is an even number of observations).
- The *mode* is the most common value or values.

2.6.2.1 USING THE MEAN

The most well-understood and widely used measure of central tendency is the mean. Leslie knew that the mean is calculated by adding up all the values of a variable and dividing by the number of values. She wrote out Equation (2.1).

$$m_x = \frac{\sum_{i=1}^{n} x_i}{n} \tag{2.1}$$

Nancy was not as interested in the statistical theory, and Leslie saw her eyes glaze over as soon as she saw Equation (2.1), but Leslie explained anyway. She said that the ***numerator*** (top of the fraction) is the sum (Σ) of all the values of x from the first value ($i = 1$) to the last value (n) divided by the number of values (n). Leslie remembered that, to use the mean, a variable should be ***normally distributed***, or have values that resemble a bell curve when graphed. Nancy saw an opportunity and quickly coded a graph of a normal distribution to show Leslie (Figure 2.3). Leslie was impressed and confirmed that this is how a variable should be distributed in order to use the mean.

Nancy decided to show off a little and added the bell-shaped normal curve to the normal distribution graph (Figure 2.4).

Leslie recognized that the underlying graph in Figures 2.3 and 2.4 is a ***histogram***, which shows the values of a variable along the horizontal **x-*axis*** and some kind of probability (or sometimes the frequency) of each value of the variable along the vertical **y-*axis***. That was all Leslie could remember about this graph, and Kiara assured her that they would talk more about histograms in detail at their next meeting. For now, they would just remember that ***normality*** is signified by the shape of the graph.

Leslie remembered that, for a variable with a distribution that is normal, or near normal, the mean value would be a good representation of the middle of the data. But if the data look ***skewed***, with some very large values on the right (**right** or ***positively skewed***) or left (**left** or ***negatively skewed***), the median would be more appropriate. Nancy got her coding fingers ready and made a couple of skewed histograms as examples (Figure 2.5).

FIGURE 2.3 The shape of the distribution for a normally distributed variable

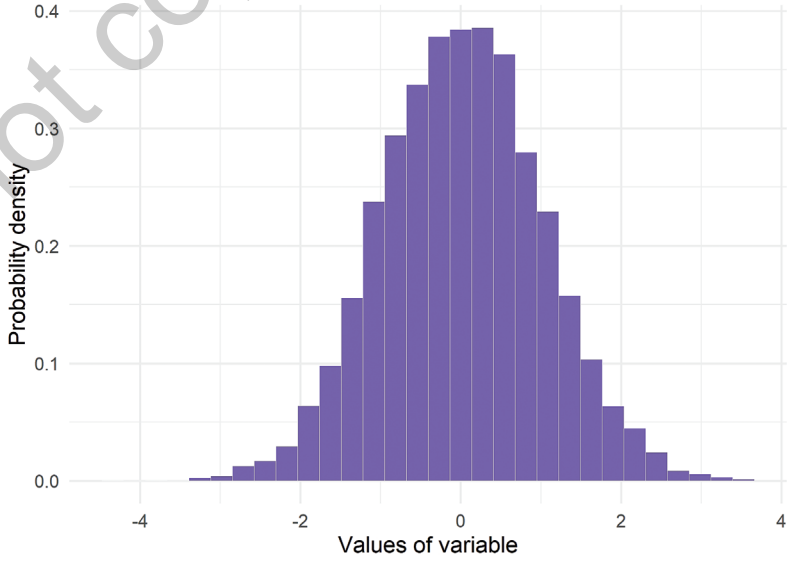


FIGURE 2.4 Normally distributed variable with normal curve

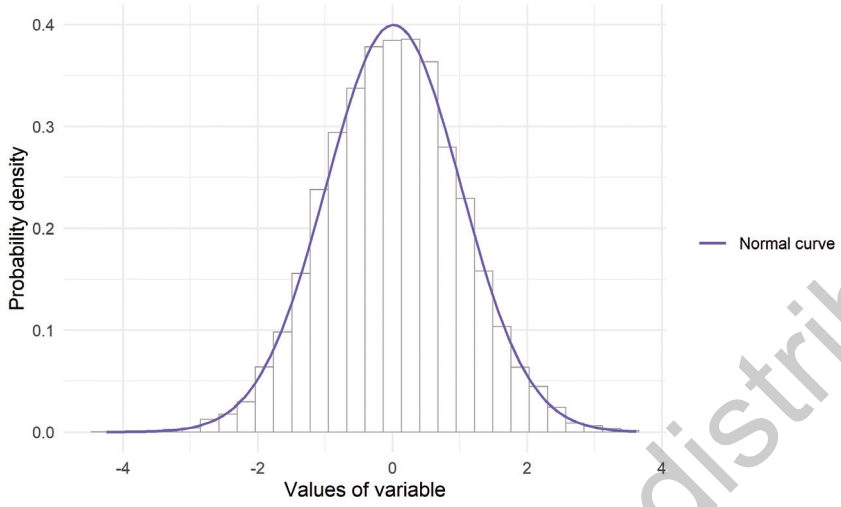
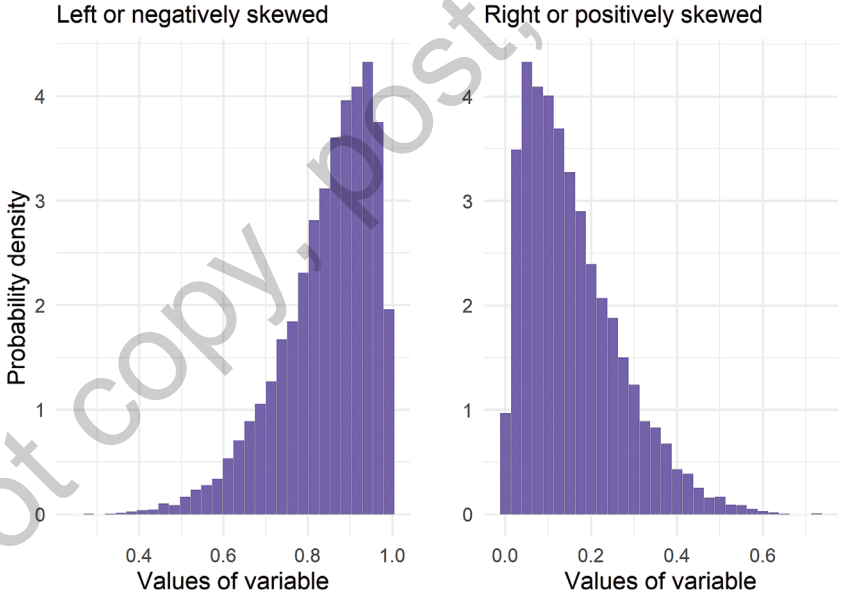


FIGURE 2.5 Examples of skewed distributions



Leslie explained that the reason the mean is not a good representation of skewed data is that adding together a set of values that includes a few very large or very small values like those on the far left of a ***left-skewed*** distribution or the far right of a ***right-skewed*** distribution will result in a large or small total value in the numerator of Equation (2.1), and therefore the mean will be a large or small value relative to the actual middle of the data.

That is, she said, the mean is influenced by the very large or very small values and is therefore not representative of a typical value or the middle of the data; instead, the mean of a skewed variable is larger

or smaller than the actual middle values in the data. Nancy remembered one example of this that is often used is income. Say you collected the incomes of five of your friends and billionaire Bill Gates. Your friends made the following annual salaries in U.S. dollars: $25,000, $62,000, $41,000, $96,000, and $41,000. Adding these five values together and dividing by five would result in a mean salary among your friends of $53,000. Given the salaries listed, this seems like a reasonable value that accurately represents a typical salary of your friends.

While she explained this to Leslie, Nancy introduced the `mean()` function.

```
# create salaries vector and find its mean
salaries <- c(25000, 62000, 41000, 96000, 41000)
mean(x = salaries)
## [1] 53000
```

However, Nancy explained, if Bill Gates just happened to be a friend of yours, you would include his estimated $11.5 billion annual earnings by adding it to the existing `salaries` vector. Adding a value to a vector can be done by concatenating the existing values with the new values in a new vector using `c()`, like this:

```
# add Bill Gates
salaries.gates <- c(salaries, 11500000000)

# find the mean of the vector with gates
mean(x = salaries.gates)
## [1] 1916710833
```

Now it appears that your friends make a mean salary of $1,916,710,833. While technically this may be true if Bill Gates is among your friends, it is not a good representation of a typical salary for one of the friends. In the case of skewed data, the median is a better choice.

Leslie looked away for a minute, and when she looked back, the code magician (Nancy) had coded a normal distribution with the mean represented by a line in the middle of the distribution (Figure 2.6). Visually, the mean for a normally distributed variable would be in the middle like the vertical line in Nancy's fancy distribution.

Nancy also added means to the skewed-variables graphic (Figure 2.7). This demonstrated that the mean was not in the middle, but instead was toward the longer *tail* of data. The tails of a distribution are the values to the far right and far left in the distribution. Often, there are fewer values in these parts of the distribution, so they taper off and look like a tail.

2.6.2.2 USING THE MEDIAN

Leslie reminded everyone that the median is the middle number when numbers are in order from smallest to largest. When there are an even number of numbers, the median is the mean of the middle two numbers. Nancy typed some quick code to show that, in the examples above, the median for the friend groups with and without Bill Gates would be as follows:

FIGURE 2.6 Normally distributed variable with mean

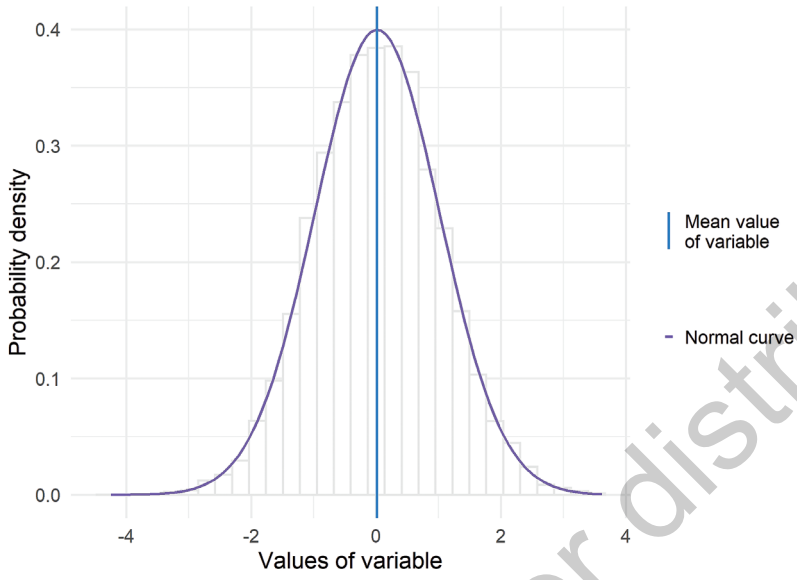
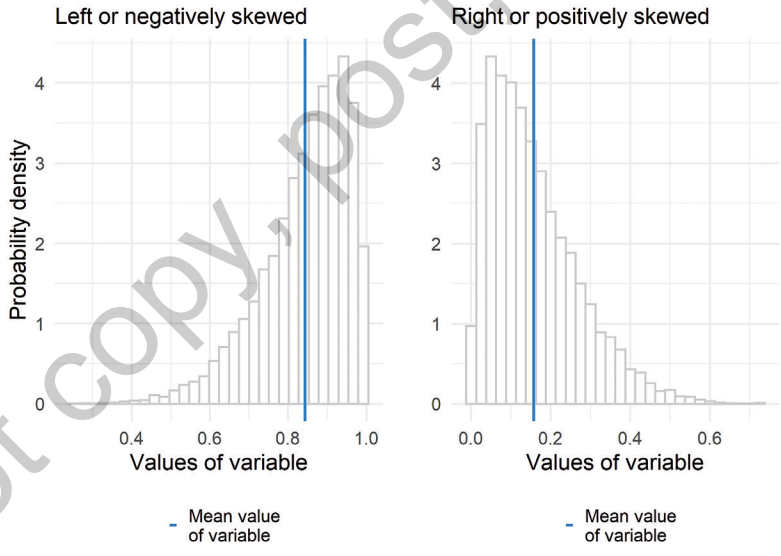


FIGURE 2.7 Examples of skewed distributions showing mean values




```
# median salary without Bill Gates
median(x = salaries)
## [1] 41000
# median salary with Bill Gates
median(x = salaries.gates)
## [1] 51500
```

While Leslie noticed that the median increased with the addition of Bill Gates to her friend group, the result was still a reasonable representation of the middle of the salaries of this group of friends, unlike the mean of $1,916,710,833.

Leslie went on to explain that in a perfect normal distribution, the median would be exactly the same as the mean. In skewed distributions, the median will not be the same as the mean. For a variable that is left skewed, the median will usually be to the right of the mean (von Hippel, 2005). For a variable that is right skewed, the median will usually be to the left of the mean. Nancy was not paying close attention but realized she could write some code to show the mean and the median on the skewed distribution to demonstrate Leslie's point (Figure 2.8).

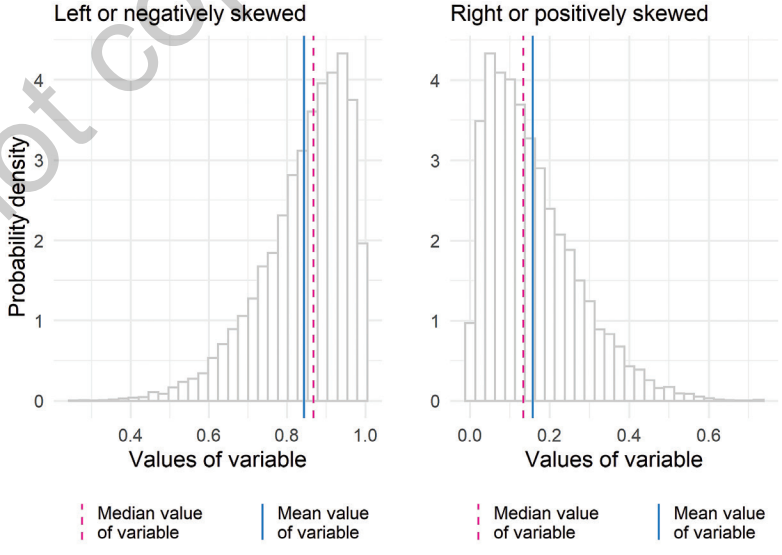
In this case, the median is not too far from the mean. Leslie was curious about how different from normal a distribution has to be before people use the median instead of the mean. From the examples so far, she thought that median salary was a more accurate representation of the middle of the `gates.salaries` vector than the mean of `gates.salaries`. However, for the two skewed distribution histogram examples in Figure 2.8, the median and mean are very close to each other, and since the mean seems like a more widely used and understood measure, might it be a better choice?

She looked through her statistics materials and found that one way some people make this decision is to calculate a measure of skewness. ***Skewness*** is a measure of the extent to which a distribution is skewed. There are a few variations on the formula, but she found one that is commonly used (von Hippel, 2005). She showed Nancy and Kiara Equation (2.2).

$$skewness_x = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - m_x}{s_x}\right)^3 \tag{2.2}$$

In Equation (2.2), n is the sample size, x_i is each observation of x, m_x is the sample mean of x, and s_x is the sample standard deviation of x. Leslie reviewed the equation slowly. The first thing she found to note is that the mean of x, m_x, is subtracted from each observation x_i. These differences between the mean and

FIGURE 2.8 Examples of skewed distributions with means and medians



each observed value are called ***deviation scores***. Each deviation score is divided by the standard deviation, *s*, and the result is cubed. Finally, sum the cubed values and divide by the number of cubed values (which is taking the mean) to get the skewness of *x*. The resulting value is usually *negative* when the skew is to the left and usually *positive* when the skew is toward the right. Because skewness is strongly impacted by the sample size with $\frac{1}{n}$ in Equation (2.2), the value of skew that is considered *too skewed* differs depending on sample size.

Leslie found this helpful and asked Nancy if there was a way to compute skewness in R. Nancy said the **semTools** package has `skew()` to compute this value and a few related values. She typed in the code and tried `skew()` for the salary vector with Bill Gates and for the two variables used to create the skewed histograms in Figure 2.8, `left.skewed` for the left-skewed histogram and `right.skewed` for the right-skewed histogram. Leslie did not have the data Nancy used for these variables, so she could only follow along with the `skew()` calculation for the `salaries.gates` object.

```
# skewness of salaries variable
semTools::skew(object = salaries.gates)
##  skew (g1)         se          z          p
## 2.44948974 1.00000000 2.44948974 0.01430588

# skewness of Figure 2.8 left-skewed distribution variable
semTools::skew(object = left.skewed)
##   skew (g1)          se           z          p
##  -1.0739316   0.0244949 -43.8430756  0.0000000

# skewness of Figure 2.8 right-skewed distribution variable
semTools::skew(object = right.skewed)
##   skew (g1)         se          z          p
##  1.0454195  0.0244949 42.6790709  0.0000000
```

Leslie was expecting a single number, but instead each `skew()` calculation contained four values. She looked in the help documentation and found that the first value is the skewness and the second value is the *se*, or the ***standard error*** of the skew. Leslie remembered that the standard error is often used to quantify how much variation there is in data, but she decided to leave that for later. The third number, *z*, is the value of the skew divided by its *se*. She determined from some of the readings she had in her statistics materials that *z* is the number useful for determining if the level of skew is too much for the variable to be treated as normally distributed (Kim, 2013). If the sample size is small ($n < 50$), *z* values outside the −2 to 2 range are a problem. If the sample size is between 50 and 300, *z* values outside the −3.29 to 3.29 range are a problem. For large samples ($n > 300$), using a visual is recommended over the statistics, but generally *z* values outside the range of −7 to 7 can be considered problematic.

Leslie looked back at the *se*, *z*, and *p* statistics in the output and decided that gaining a deep understanding of these statistics was a little too much for what they were doing in this meeting. She decided to focus on gaining a little more foundation in descriptive statistics and sampling instead. For now, she relied on graphs and the general rules. Since the *z* for the skewness of `salaries.gates` is greater than 2 ($z = 2.45$) and the sample size is small, the data have a skew problem. This makes sense given the huge Bill Gates salary included in these data. The *z* statistics for the data underlying the `left.skewed` and `right.skewed` distributions in Figure 2.8 were much larger in magnitude and far beyond the threshold for problematic, which made sense to Leslie since those distributions were clearly not normal.

2.6.2.3 USING THE MODE

Finally, the mode is the most common value in a data set. Leslie suggested that the mode is not widely used in the research she had read, but it was still good to know that it existed as an option. She did recall that, for numeric variables, the mode can be used along with the mean and median as an additional indicator of a normal distribution. In a perfect normal distribution, the mean, median, and mode are all exactly the same.

Nancy explained that, unfortunately, perhaps because it is rarely used, there is no mode function. Instead, she showed Leslie how to use `sort()` with `table()` to sort a table from highest to lowest (decreasing order) or lowest to highest (increasing order). Nancy walked Leslie through computing the mode in R. Start with a table, `table(salaries)`, which will show the frequencies of each value in the data.

```
# table showing salaries frequencies
table(salaries)
## salaries
## 25000 41000 62000 96000
##     1     2     1     1
```

There is one salary of $25,000, two salaries of $41,000, and so on. With data this simple, it would be easy to stop here and notice that $41,000 is the mode. However, most data sets will be much larger, and it will be more efficient to have a set of R functions to use. Nancy put the table in order from largest frequency to smallest frequency using `sort()` with `decreasing = TRUE`.

```
# table showing salaries frequencies
sort(x = table(salaries), decreasing = TRUE)
## salaries
## 41000 25000 62000 96000
##     2     1     1     1
```

This orders the table starting with the salary with the highest frequency, or the most common salary. This salary is the mode. To just have R print the salary (not its frequency), use `names()`, which will print the salary category names shown in the top row of the table.

```
# table showing salaries frequencies
names(x = sort(x = table(salaries), decreasing = TRUE))
## [1] "41000" "25000" "62000" "96000"
```

Finally, to just print the first category name, use the square brackets, `[ ]`. In R, the square brackets are used to grab a particular location (sometimes called an "index"). For example, the `salaries` vector has 5 observations: 25000, 62000, 41000, 96000, 41000. To get the 4th observation of `salaries`, you can type `salaries[4]`, which is 96000. Here, to get the first category name, use the same bit of code, and add a `[1]` to the end.

```
# no mode function so find the mode using a table
# and sort values in decreasing order
# so the most common value comes first
names(x = sort(x = table(salaries), decreasing = TRUE))[1]
## [1] "41000"
```

All together, the three measures can be computed for the `salaries` vector.

```
# mean, median, and mode of salaries
mean(x = salaries)
## [1] 53000
median(x = salaries)
## [1] 41000
names(x = sort(x = table(salaries), decreasing = TRUE))[1]
## [1] "41000"
```

The mean is $53,000, and the median and mode are both $41,000. So the mean, median, and mode were not exactly the same, and Leslie concluded that the data are not *perfectly* normally distributed.

The mode is sometimes used to identify the most common (or typical) category of a factor variable. For example, in the tables above, the mode of the `TRNSGNDR` variable is "Not transgender," with more than 150,000 observations in that category.

2.6.3 MESSY DATA AND COMPUTING MEASURES OF CENTRAL TENDENCY

Using these functions with variables from a data frame like `brfss.2014.cleaned` is similar. Nancy wanted to show Leslie how to create a histogram and compute the mean, median, and mode for the `PHYSHLTH` variable from the 2014 BRFSS data set. `PHYSHLTH` is the number of physically unhealthy days a survey participant has had in the last 30 days. On page 11 of the BRFSS codebook, the values of `77` and `99` are `Don't know/ Not sure` and `Refused`, so they could be coded as missing before examining the variable. It also looks like `88` is `None` for the number of unhealthy days and should be coded as zero. Leslie copied and pasted her data management code from earlier. Nancy was eager to keep coding and slid the laptop over to type in the new code for changing `77` and `99` to `NA`. She also recoded `88` in `PHYSHLTH` to be `0`, which was a little complicated.

```
# pipe in the original data frame
# recode the TRNSGNDR factor so it's easy to read
# recode 77, 88, 99 on PHYSHLTH
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')) %>%
  mutate(PHYSHLTH = na_if(x = PHYSHLTH, y = 77)) %>%
  mutate(PHYSHLTH = na_if(x = PHYSHLTH, y = 99)) %>%
  mutate(PHYSHLTH = as.numeric(recode(.x = PHYSHLTH, `88` = 0L)))
```

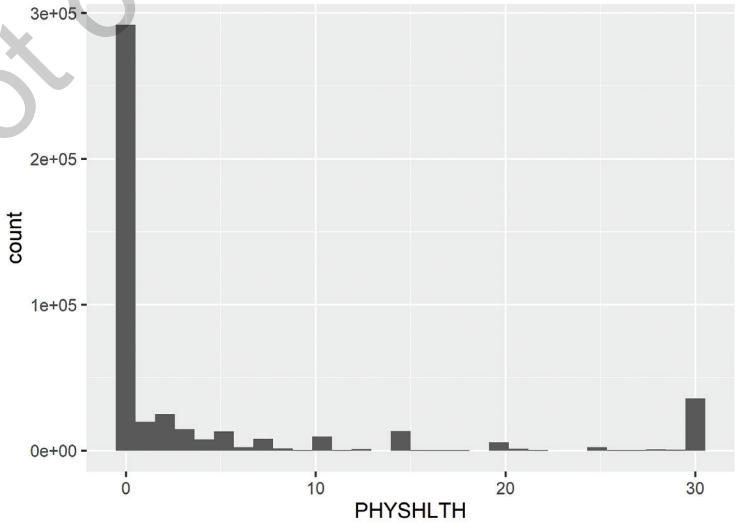
Once Nancy had completed and run the code, she checked the recoding of `PHYSHLTH`.

```
# examine PHYSHLTH to check data management
summary(object = brfss.2014.cleaned$PHYSHLTH)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.    NA's
##   0.000   0.000   0.000   4.224   3.000  30.000   10303
```

The maximum number of unhealthy days shown in the output for `summary()` was 30, so the recoding worked as planned. Nancy showed Leslie how to make a histogram from the cleaned data. The code was similar to what they used at the end of the data preparation meeting, but the `geom_`, or geometry, layer this time is `geom_histogram()` to make a histogram (Figure 2.9).

```
# make a histogram
brfss.2014.cleaned %>%
  ggplot(aes(x = PHYSHLTH)) +
  geom_histogram()
```

FIGURE 2.9 Distribution of the `PHYSHLTH` variable



The histogram showed most people have between 0 and 10 unhealthy days per 30 days. Leslie wanted to take a turn coding and wrote the code for the measures of central tendency for the `PHYSHLTH` variable.

```
# get mean, median, mode
mean(x = brfss.2014.cleaned$PHYSHLTH)
## [1] NA
median(x = brfss.2014.cleaned$PHYSHLTH)
## [1] NA
names(x = sort(x = table(brfss.2014.cleaned$PHYSHLTH), decreasing = TRUE))[1]
## [1] "0"
```

Well, that didn't work very well! The mode looks like 0 unhealthy days. The mean and the median show `NA`. Nancy explained that the `mean()` and `median()` functions require removal of `NA` values. They worked for the `salaries` vector because there were no `NA` values. To fix this, she needed to include a second argument that tells the `mean()` function how to treat missing data. She added the option `na.rm = TRUE`, where `na` stands for `NA` and `rm` stands for "remove."

```
# get mean, median, mode
mean(x = brfss.2014.cleaned$PHYSHLTH, na.rm = TRUE)
## [1] 4.224106
median(x = brfss.2014.cleaned$PHYSHLTH, na.rm = TRUE)
## [1] 0
names(x = sort(table(brfss.2014.cleaned$PHYSHLTH), decreasing = TRUE))[1]
## [1] "0"
```

Leslie admitted at this point that she was concerned that she would never be able to remember all the arguments and rules for the different functions. Kiara reassured her that she does *not* need to memorize them! First, Kiara explained, Leslie can always check the help documentation. The help documentation will list what a function does, the arguments it can take, and what the output of the function means, and there are often examples at the bottom of the help documentation that show how to use the function.

Additionally, Kiara showed Leslie how to use tab completion to make her life easier. Kiara told Leslie to start typing the letters "me" in the Console and then hit her laptop's Tab key. Leslie noticed that a lot of different functions starting with "me" appear, and that `mean()` was first! Since `mean()` was first and highlighted, Kiara told Leslie to press the Enter or Return key on her laptop keyboard. Leslie was relieved that she didn't need to type out the rest of the word "mean" *and* that the parentheses automatically appeared!

Kiara told Leslie to hit the Tab key once again. The first argument `x` appeared, and a brief description of that argument. Here, `x` is the variable that you want to get the mean of. Leslie pressed Enter and the `x =` argument appeared in her code and was ready to be filled in with `brfss.2014.cleaned$PHYSHLTH`. Leslie started typing the letters "brf" and hit Tab. Sure enough, she saw the name of the data frame she needed and highlighted before she pressed Enter and added the `$`. After she added `$`, the same thing happened when she started to type the `PHYSHLTH` variable name. Kiara explained that arguments within a function are separated by commas. She suggested Leslie type a comma after `PHYSHLTH` and press Tab. Leslie saw the `na.rm` argument appear! It was like magic. Leslie finished the line of code using the Tab-complete trick and was relieved that she didn't have to remember or type so much.

Kiara suggested that **tidyverse** code might be clearer to get the central tendency measures since the name of the data frame would not have to be included in all three functions. To pipe the data frame into the central tendency functions, use `summarize()`. Nancy mentioned that some code uses British spellings, but the American spellings usually work, too. For example, in **tidyverse**, `summarize()` is the same as `summarise()`.

```
# get mean, median, mode
brfss.2014.cleaned %>%
  summarize(mean.days = mean(x = PHYSHLTH,
                             na.rm = TRUE),
            med.days = median(x = PHYSHLTH,
                              na.rm = TRUE),
            mode.days = names(x = sort(table(PHYSHLTH),
                                       decreasing = TRUE))[1])
##   mean.days med.days mode.days
## 1  4.224106        0         0
```

So the mean number of unhealthy days per month is 4.22 and the median and mode are 0. This makes sense given the right skew of the histogram. The people with 30 poor health days are making the mean value higher than the median. A few very large (or very small) values relative to the rest of the data can make a big difference for a mean.

Leslie wanted to calculate the skewness to see what it is for a variable that looks this skewed. Based on what she learned above, she expected the *z* for skewness of `PHYSHLTH` to be a positive number and greater than 7. She wrote and ran the code.

```
# skewness for PHYSHLTH
semTools::skew(object = brfss.2014.cleaned$PHYSHLTH)
##    skew (g1)           se            z            p
## 2.209078e+00 3.633918e-03 6.079054e+02 0.000000e+00
```

Leslie noticed that the results were shown using ***scientific notation***. While she was familiar with scientific notation, which is useful for printing large numbers in small spaces, she knew it is not well-understood by most audiences. Kiara said they would see more of it later (Box 3.2), but for now, they should just remember that the 2.209078e+00, 3.633918e–03, 6.079054e+02, and 0.000000e+00 were shorthand ways of writing 2.209078×10^0, 3.633918×10^{-3}, 6.079054×10^2, and 0.000000×10^0. Kiara suggested that they think of the +00, –03, and +02 as the direction and how many places to move the decimal point. So, +00 is move the decimal point 0 places to the right, –03 is move the decimal point 3 places to the left, and +02 is move the decimal point 2 places to the right.

Since 2.209078×10^0 is just 2.209078 after moving the decimal point 0 places (or after recognizing that anything raised to the power of zero is 1, so $10^0 = 1$), `PHYSHLTH` has a skewness of 2.209078. After moving the decimal point 2 places to the right, *z* is 607.9054, which is much higher than 7. The graph showed a clear right skew, so there is plenty of evidence that this variable is not normally distributed.

2.6.4 DEFINING AND CALCULATING SPREAD

In addition to using central tendency to characterize a variable, Leslie remembered that reporting a corresponding measure of how spread out the values are around the central value is also important to understanding numeric variables. Each measure of central tendency has one or more corresponding measures of spread.

- Mean: use ***variance*** or ***standard deviation*** to measure spread
- Median: use ***range*** or ***interquartile range*** (IQR) to measure spread
- Mode: use an ***index of qualitative variation*** to measure spread

2.6.4.1 SPREAD TO REPORT WITH THE MEAN

The variance is the average of the squared differences between each value of a variable and the mean of the variable. Leslie showed Nancy the formula from an old statistics textbook, and Nancy wrote Equation (2.3) so they could review how it worked.

$$s_x^2 = \frac{\sum_{i=1}^{n}(x_i - m_x)^2}{n-1} \tag{2.3}$$

Leslie saw that the s_x^2 is the variance of x, the Σ symbol is sum, the x_i is each individual value of x, the m_x is the mean of x, and n is the sample size. The variance is the sum of the squared differences between each value of x and the mean of x (or the sum of squared deviation scores) divided by the sample size minus 1. "Fun fact," said Nancy, "the n – 1 in the formula is called the Bessel correction."

"I'm not sure how that's fun," Leslie said, smiling, "but OK."

Nancy showed her that, in R, the `var()` function finds the variance.

```
# variance of unhealthy days
var(x = brfss.2014.cleaned$PHYSHLTH, na.rm = TRUE)
## [1] 77.00419
```

After looking it up in her statistics textbook and online, Leslie determined there was no direct interpretation of the variance. It is a general measure of how much variation there is in the values of a variable. She found that a more useful measure of spread is the standard deviation, which is the square root of the variance. She shared Equation (2.4) showing the standard deviation of x, or s_x.

$$s_x = \sqrt{\frac{\sum_{i=1}^{n}(x_i - m_x)^2}{n-1}} \tag{2.4}$$

Nancy continued her coding and added the variance and standard deviation to the **`tidyverse`** descriptive statistics code they had been accumulating.

```
# get mean, median, mode, and spread
brfss.2014.cleaned %>%
```

```
  summarize(mean.days = mean(x = PHYSHLTH, na.rm = TRUE),
            sd.days = sd(x = PHYSHLTH, na.rm = TRUE),
            var.days = var(x = PHYSHLTH, na.rm = TRUE),
            med.days = median(x = PHYSHLTH, na.rm = TRUE),
            mode.days = names(x = sort(x = table(PHYSHLTH),
                                       decreasing = TRUE))[1])
##   mean.days sd.days var.days med.days mode.days
## 1  4.224106 8.775203 77.00419        0         0
```

Leslie remembered that the standard deviation was sometimes interpreted as the average amount an observation differs from the mean. After walking through Equation (2.4) one more time, Leslie concluded that this is conceptually close and a good way to think about it, but not 100% accurate. Regardless, she found that the standard deviation was the best measure of spread to report with means. Nancy thought a visual representation might help and added lines to the normal distribution graph showing how far away one standard deviation is from the mean in the histogram in Figure 2.10.

The R-Team noticed that most of the observations of the variable were between the standard deviation lines. Most observations are within one standard deviation away from the mean. This reminded Kiara of a picture she took in Brussels at an R conference there in 2017 (Figure 2.11).

Leslie had a feeling that there was something else they needed to look at before they were done discussing normal distributions. Kiara suggested that maybe she was remembering ***kurtosis***. Kiara explained that kurtosis measures how many observations are in the tails of a distribution. She said that some distributions look bell-shaped, but have a lot of observations in the tails (platykurtic) or very few observations in the tails (leptokurtic) (Westfall, 2004). Leslie asked Kiara whether she was remembering correctly that ***platykurtic*** distributions are more *flat* (as in a *platy*pus has a flat beak) and ***leptokurtic***

FIGURE 2.10 Normally distributed variable showing mean plus and minus one standard deviation

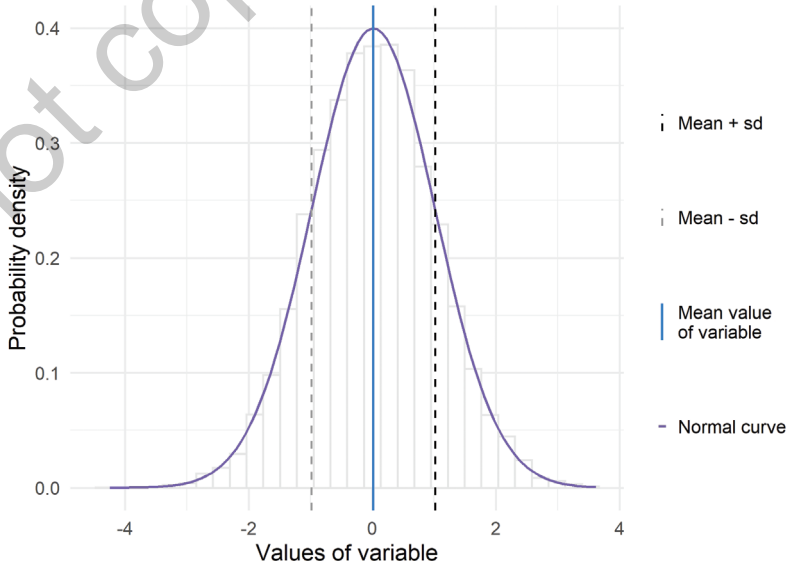


FIGURE 2.11 Street sign from Brussels, Belgium (2017)

Source: Jenine K. Harris.

distributions are more *pointy*. Kiara said this is a common way of describing the shapes of these distributions, but that, technically, kurtosis measures whether there are many or few observations in the tails of the distribution (Westfall, 2004).

Nancy saw an opening for more coding and made Figure 2.12 with histograms that had problems with kurtosis so the team could get an idea of what they are looking for.

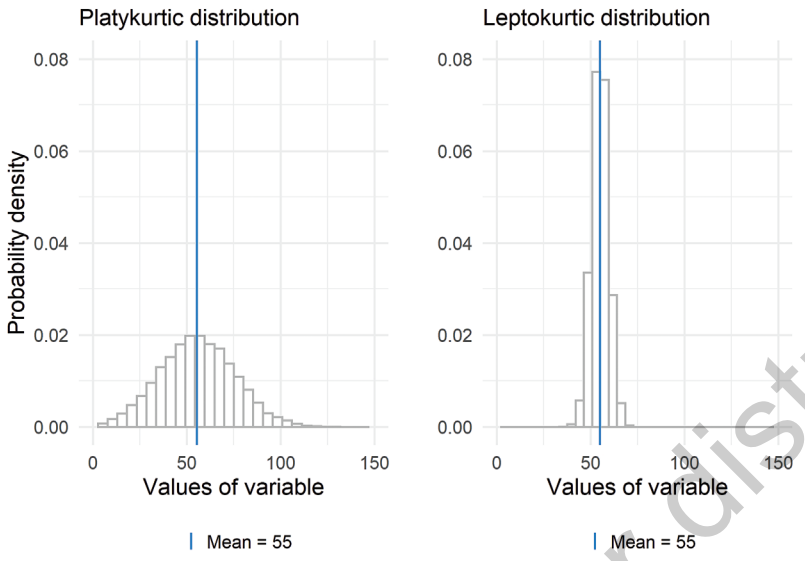
From her data management work, Kiara learned that platykurtic and leptokurtic deviations from normality do not necessarily influence the mean, since it will still be a good representation of the middle of the data *if the distribution is symmetrical and not skewed*. However, platykurtic and leptokurtic distributions will have smaller and larger values of variance and standard deviation, respectively, compared to a normal distribution. The variance and standard deviation are not only used to quantify spread, but also used in many of the common statistical tests.

Leslie found the formula for kurtosis (Westfall, 2004) and showed Equation (2.5) to the team.

$$kurtosis_x = \frac{1}{n}\sum_{i=1}^{n}\left(\frac{x_i - m_x}{s_x}\right)^4 \tag{2.5}$$

Leslie read that, in Equation (2.5), n is the sample size, s_x is the standard deviation of x, x_i is each value of x, and m_x is the mean of x. The $\sum_{i=1}^{n}$ symbol indicates the values from the first value of x ($i = 1$) to

FIGURE 2.12 Examples of different-shaped distributions with means of 55



the last value of x ($i = n$) should be summed. A normal distribution will have a kurtosis value of 3; distributions with kurtosis = 3 are described as ***mesokurtic***. If kurtosis is above or below 3, there is excess kurtosis. Values of kurtosis above 3 indicate the distribution is *leptokurtic*, with fewer observations in the tails than a normal distribution (the fewer observations in the tails often give a distribution a pointy look). Values of kurtosis below 3 indicate the distribution is *platykurtic*, with more observations in the tails than a normal distribution would have given the mean, standard deviation, and sample size.

Leslie looked at the **semTools** package documentation and found `kurtosis()` to compute the kurtosis and a few related values. The `kurtosis()` function subtracts 3 from the kurtosis, so positive values will indicate a leptokurtic distribution and negative will indicate a platykurtic distribution. The same cutoff values from skew also apply for the z for small, medium, and large sample sizes in kurtosis. Leslie tried it for the two variables used in the leptokurtic and platykurtic graph above, saved as `var1` in `lepto.plot` and `platy.plot` respectively.

```
# kurtosis of Figure 2.12 leptokurtic distribution variable
semTools::kurtosis(object = lepto.plot$var1)
## Excess Kur (g2)              se               z               p
##     0.05206405      0.04898979      1.06275295      0.28789400

# kurtosis of Figure 2.12 platykurtic distribution variable
semTools::kurtosis(object = platy.plot$var1)
## Excess Kur (g2)              se               z               p
##    -0.04920369      0.04898979     -1.00436604      0.31520221
```

The values of z for the two variables used in the example graphs are relatively small and so are not problematic regardless of sample size, so using statistics that rely on a normal distribution seems OK. Kiara mentioned that they would learn some additional statistical tests later that are useful for identifying

non-normal distributions. Often, values of kurtosis and skewness will be reported with these statistical tests as additional information describing the nature of the non-normality (DeCarlo, 1997), rather than as the only piece of information on which to base a decision about normality.

2.6.4.2 SPREAD TO REPORT WITH THE MEDIAN

When distributions are not normally distributed, the median is often a better choice than the mean. For medians, however, the variance and standard deviation will not work to report the spread. Just like the very large values influence the mean, they also influence the standard deviation since the mean is part of the standard deviation formula. The R-Team found two options for reporting spread for non-normal variables. First, the range is the span between the largest and smallest values of a variable. Nancy added the ranges to the skewed distributions (Figure 2.13).

The range does not seem too informative on these graphs, Leslie noted, because it marks the very highest and very lowest values for a variable, but there is no indication of how the observations are spread out between these values. In order to get exact values, Nancy demonstrated by computing the range for the `PHYSHLTH` variable.

```
# range of days of physical health
range(brfss.2014.cleaned$PHYSHLTH, na.rm = TRUE)
## [1]  0 30
```

Leslie noticed that they *finally* had values that were easy to interpret: The range of unhealthy days in a month is 0 to 30. For the `PHYSHLTH` variable, the ends of the range are the highest and lowest possible values of the variable. The range does not provide any indication of how the data are distributed across the possible values. For example, maybe there is one person who has 30 days of poor physical health and everyone else is between 0 and 10. Or, maybe half the people have 0 and half the people have 30 and no people have anything in between. Leslie thought another option would be better for understanding spread.

FIGURE 2.13 Examples of skewed distributions with medians and range boundaries

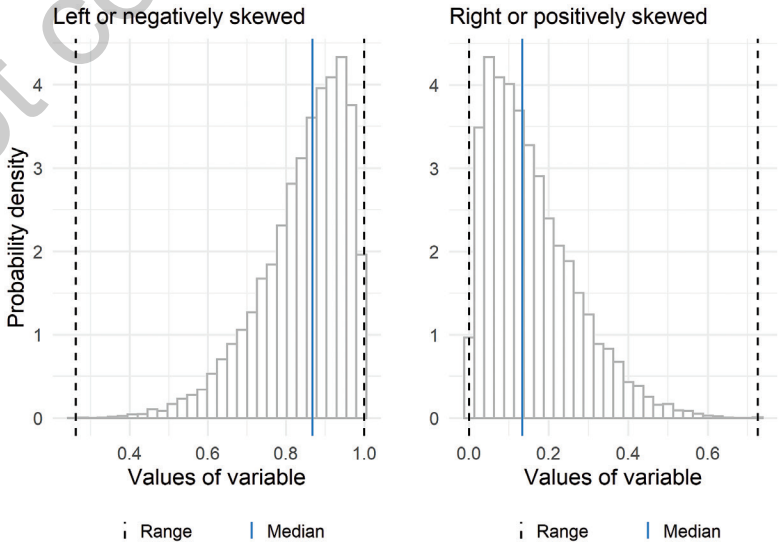
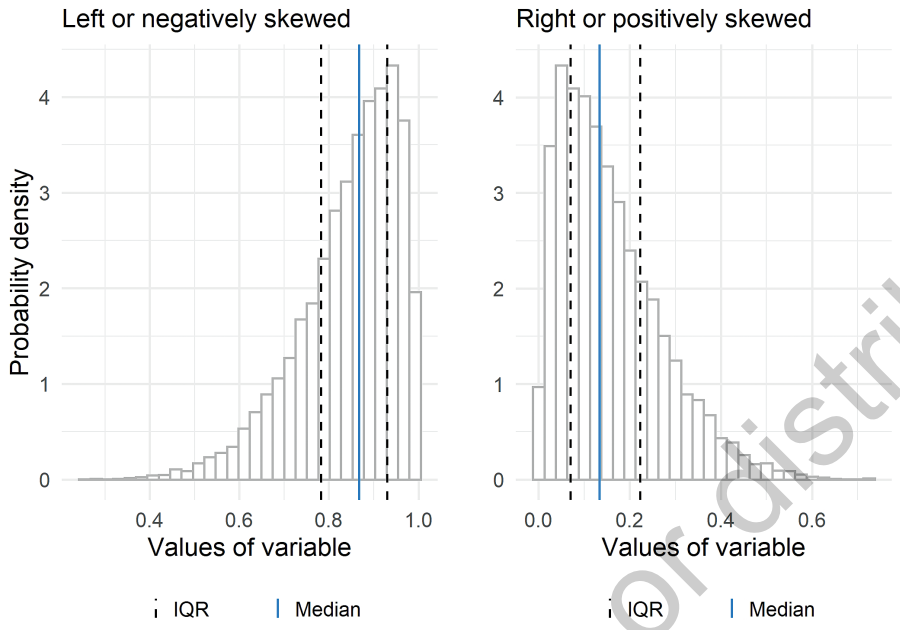


FIGURE 2.14 Examples of skewed distributions with medians and IQR boundaries



Leslie looked up the interquartile range, or IQR, and found that it might be more appropriate for these data, or for data that are highly skewed. The IQR is the difference between the first and third quartiles. A quartile is one-quarter of the data, so the difference between the first and third quartiles would be the boundaries around the middle 50% of the data. Nancy plotted the IQR on the skewed distributions (Figure 2.14).

This was more useful to Leslie. To find the IQR of `PHYSHLTH`, Nancy used `IQR()` and added it to the **`tidyverse`** descriptive statistics list.

```
# get descriptive statistics for PHYSHLTH
brfss.2014.cleaned %>%
  summarize(mean.days = mean(x = PHYSHLTH, na.rm = TRUE),
            sd.days = sd(x = PHYSHLTH, na.rm = TRUE),
            var.days = var(x = PHYSHLTH, na.rm = TRUE),
            med.days = median(x = PHYSHLTH, na.rm = TRUE),
            iqr.days = IQR(x = PHYSHLTH, na.rm = TRUE),
            mode.days = names(x = sort(x = table(PHYSHLTH),
                                       decreasing = TRUE))[1])
##   mean.days  sd.days var.days med.days iqr.days mode.days
## 1  4.224106 8.775203 77.00419        0        3         0
```

Leslie noticed that they had typed `na.rm = TRUE` for all but one of the statistics in the `summarize()` code. She asked Nancy if there was some way to remove `NA` to make the code less repetitive. Nancy was delighted that Leslie was starting to think about the format of her code. She reminded Leslie that there was a way to omit `NA` values using `drop_na()` with the name of the variable of interest, like this:

```
# get descriptive statistics for PHYSHLTH
brfss.2014.cleaned %>%
  drop_na(PHYSHLTH) %>%
  summarize(mean.days = mean(x = PHYSHLTH),
            sd.days = sd(x = PHYSHLTH),
            var.days = var(x = PHYSHLTH),
            med.days = median(x = PHYSHLTH),
            iqr.days = IQR(x = PHYSHLTH),
            mode.days = names(x = sort(x = table(PHYSHLTH),
                                       decreasing = TRUE))[1])
##   mean.days  sd.days var.days med.days iqr.days mode.days
## 1  4.224106 8.775203 77.00419        0        3         0
```

Leslie examined the statistics so far. She noted that the IQR value makes sense; there is a 3-day difference between the first and third quartiles for unhealthy days. She remembered she had usually seen the IQR reported without much interpretation, like this: IQR = 3.

Leslie would rather see the upper and lower bounds of the IQR. Nancy said there was no way to do this in the `IQR()` function, but the `quantile()` function could be used to find the bounds around the middle 50%, which are the IQR boundaries.

```
# interquartile range of unhealthy days
quantile(x = brfss.2014.cleaned$PHYSHLTH, na.rm = TRUE)
##   0%  25%  50%  75% 100%
##    0    0    0    3   30
```

The middle 50% of the data is between the 25% and 75% quantiles. Leslie could now report the bounds around the middle 50% of unhealthy days, 0 to 3. Fifty percent of observations (people) in this data set have between 0 and 3 physically unhealthy days per month. Nancy added the median and IQR boundaries to the `PHYSHLTH` plot with new layers and did a little formatting so they are easier to see (Figure 2.15).

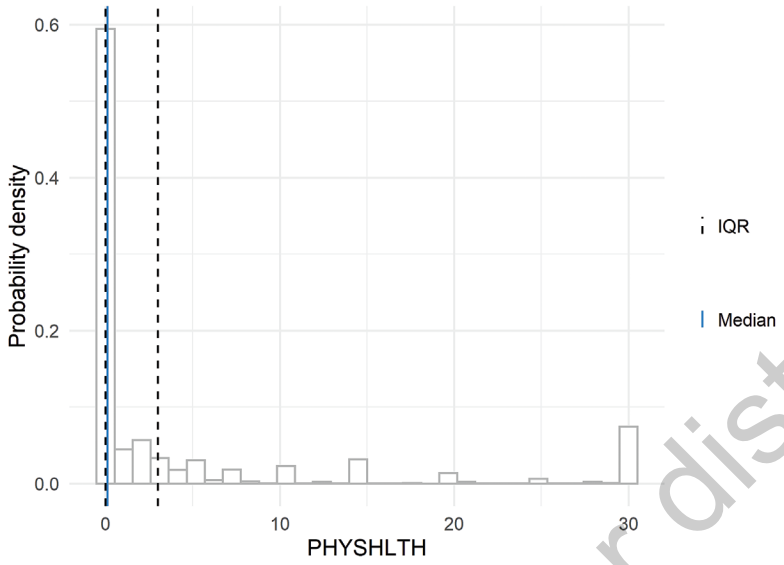
Leslie noticed that the lower boundary of the IQR is the same as the median in the graph. Kiara said this was due to so many values being zero and this being the lowest possible value of the variable. Leslie was curious about the graph code, but Nancy said it had some complex features and suggested they save it for next time when they are working on graphs.

2.6.4.3 SPREAD TO REPORT WITH THE MODE

Leslie remembered that spread is not often reported for categorical variables. Leslie and Kiara discussed what the idea of spread or variability even means for a categorical variable. They reviewed some options in textbooks that describe how observations are spread across categories, sometimes described as *diversity*.

Kiara and Leslie found several options for the *index of qualitative variation*, which quantified how much the observations are spread across categories of a categorical variable. While these indexes are computed in different ways, they all have a range from 0 to 1. The resulting values are *high* when observations are

FIGURE 2.15 Distribution of days of poor physical health showing median and IQR boundaries



spread out among categories and *low* when they are not. For example, if a data set had a marital status variable and there were 3 people in each marital status category, the data would be considered perfectly spread across groups and the index value would be 1. Likewise, if everyone in the data set was in one category (e.g., unmarried), the index value would be 0 for no spread at all.

After reading through the R help documentation for the **qualvar** package, which includes several options, Kiara and Leslie found the ***B index*** to be their favorite.

They shared their finding with Nancy, who was playing on her phone because she had run out of things to code for the moment. Nancy had not used the B index, but she found `B()` in the **qualvar** package. She looked up the help documentation for the `B()` function and saw that the first argument is `x`, which is described as "a vector of frequencies." The `PHYSHLTH` variable is numeric, so it would not use the B index. Instead, Leslie went back to the `TRNSGNDR` variable. To get a vector of frequencies for `TRNSGNDR`, Nancy thought the `table()` function might work. Instead of making a table as a new object, Nancy added the `table()` code directly into the `B()` function. It worked perfectly.

```
# B index of TRNSGNDR variable
qualvar::B(x = table(brfss.2014.cleaned$TRNSGNDR))
## [1] 0.0009940017
```

The resulting value of 0.00099 is close to zero and therefore indicates that observations in this data set are not well spread out among the six categories of `TRNSGNDR`. While it is true that there are people in all categories, the "Not transgender" category contains a much larger number of observations than any of the other categories, so the small value of B reflects this lack of even spread of observations across categories of `TRNSGNDR`.

Leslie wanted to make sure she had the chance to practice reporting and interpreting the descriptive statistics for numeric variables, so she wrote a few sentences to make sure she understood.

> The mean number of days of poor health per month for participants in the 2014 BRFSS was 4.22 ($s = 8.78$).

Leslie recognized that she would not report the mean and the median for the variable; instead, she would choose the most appropriate one. However, for the sake of practicing, she wrote an interpretation of the median.

> The median number of days of poor health per month for participants in the 2014 BRFSS was 0 (IQR = 3).

The mode would not commonly be reported for the `PHYSHLTH` variable, so she reported the mode and B index for `TRNSGNDR` instead.

> The most common response (mode) to the transgender question was "Not transgender." The responses were not spread out very evenly among the categories, with over 150,000 in the "Not transgender" category and just 116 in the "Gender non-conforming" category (B = .00099).

Kiara and Nancy approved of these interpretations and told Leslie her writing was similar to interpretations in published manuscripts.

2.6.5 ACHIEVEMENT 3: CHECK YOUR UNDERSTANDING

Find central tendency and spread for the age variable (**`X_AGE80`**). Examine the variable and the codebook first to see if it needs to be cleaned.

2.7 Achievement 4: Developing clear tables for reporting descriptive statistics

Although Leslie felt like she now had a great grasp of descriptive statistics, Kiara explained that they were only halfway done! After all, Kiara stated, statistics are only useful when people can understand them. In the case of descriptive statistics, she said, making a clear table is the key to good communication. Clear tables tend to have the following features:

- A title that explains what is in the table
  - The number of observations if possible
  - Key pieces of information that describe the sample such as the year of data collection and the data source
  - The units of measurement (people, organizations, etc.)
- Consistent use of the same number of decimal places throughout the table
- Numbers aligned to the right so that the decimal points line up
- Words aligned to the left
- Indentation and shading to differentiate rows or sections
- Limited internal lines
- Clearly labeled rows and columns

If the table contains only factor-type variables, Kiara explained, it may look like Table 2.1 including only percentages. They decided that reproducing that table would be good practice for making publication-quality tables. To reproduce Table 2.1, Kiara had Leslie note the variables included in the table and find them in the BRFSS codebook (https://www.cdc.gov/brfss/annual_data/2014/pdf/CODEBOOK14_LLCP.pdf). After she reviewed the variables in the codebook, the team got to work to reproduce Table 2.1. Kiara recommended they refer to this table as Table 2.1 as they work, with the understanding that it was a reproduction of the Table 1 printed in the Narayan et al. (2017) paper.

2.7.1 DATA CLEANING BEFORE ANALYSIS

2.7.1.1 CREATING A SMALLER DATA FRAME TO WORK WITH

Looking back at Table 2.1 and reviewing the Narayan et al. paper (2017), Leslie and Kiara determined that it contained only those who answered the transgender status question, were in the 40- to 74-year-old age groups, and were asked the mammogram question. They found these three variables in the codebook:

- `TRNSGNDR`: codebook page 83
- `_AGEG5YR`: codebook page 108
- `HADMAM`: codebook page 37

Kiara shared one of her favorite data management tips with Leslie, which was to create a smaller data set by limiting the data to the observations that will be analyzed. There are several ways to create a subset that contains only particular observations. One method is to simply grab the observations wanted using the square brackets `[ ]`. Kiara reminded Leslie that she had already used `[ ]` to get the 4th element of the `salaries` vector using `salaries[4]`. Vectors are one-dimensional, and therefore Leslie would just put one number inside the brackets. Data frames are two-dimensional, Kiara explained, since there are both rows and columns. So to index a data frame, the square brackets require two numbers, separated by a comma, and the order will always be `[rows, columns]`. Kiara wrote a few examples.

- Get the value of the 3rd row and 2nd column: `data.frame[3, 2]`.
- Get all of the values in the 5th column: `data.frame[ ,5]`. A blank in the rows part means "all," but it still needs to be separated by a comma.
- Get the values for rows 2 through 6 and columns 4 through 10: `data.frame[2:6, 4:10]`. The colon `:` can be read as "through."
- Get the values for rows 1, 7, 18 and columns 4 through 10: `data.frame[c(1, 7, 18), 4:10]`.

Since the R-Team has been using **`tidyverse`**, Nancy thought they should create a subset with the `filter()` function instead of the square brackets. She suggested that Leslie look up the help documentation for the `filter()` function on her own. Leslie found that `filter()` "choose[s] rows/cases where conditions are true." Nancy explained that this meant she should write a statement that is either `TRUE` or `FALSE` and R will use this to keep the observations where the statement is true and filter out the observations where the statement is false. Nancy knew the importance of understanding logical statements, so she gave Leslie a few examples to try before she wrote `filter()` code.

- Is the object on the left equal to the object on the right? Use ==.
  - `3 == 9` is `FALSE` because 3 does not equal 9.
  - If `a = 10` and `b = 10`, then `a == b` is `TRUE`.

- Is the object on the left *not* equal to the object on the right? Use `!=`.
  - `3 != 9` is `TRUE` because 3 does not equal 9.
  - If `a = 10` and `b = 10`, then `a != b` is `FALSE`.
- Is the object on the left greater than or less than the object on the right? Use > or <.
  - `3 > 9` is `FALSE`.
  - `3 < 9` is `TRUE`.
  - `3 < 3` is `FALSE`.
  - `3 <= 3` is `TRUE`.

Nancy then told Leslie that she could combine these logical statements. To require two conditions to both be true, use the `&`, so `3 < 9 & 4 == 4` would be `TRUE`. This is because *both* statements are true. However, `3 > 9 & 4 == 4` is `FALSE`. This is because the first statement `3 > 9` is `FALSE`, therefore it doesn't matter that the second part of the statement (`4 == 4`) is OK—the whole combined statement is `FALSE`.

Nancy explained that if you want to know if one *or* the other is true, use the `|` symbol. `3 > 9 | 4 == 4` is `TRUE` because `4 == 4` is `TRUE`.

Leslie had a better grasp of logical conditions after that and was ready to use `filter()`. When she reviewed the data frame before coding, Leslie noticed that the `_AGEG5YR` variable was imported with an `X` at the beginning of the variable name, so she incorporated this into the code.

```
# create a subset of the data set to keep
# transgender status of MtF OR FtM OR Gender non-conforming
# age group higher than group 4 and lower than group 12
# was asked mammogram question
brfss.2014.small <- brfss.2014.cleaned %>%
  filter(TRNSGNDR == 'Male to female'|
           TRNSGNDR == 'Female to male'|
           TRNSGNDR == 'Gender non-conforming') %>%
  filter(X_AGEG5YR > 4 & X_AGEG5YR < 12) %>%
  filter(!is.na(HADMAM))

# check the new data frame
summary(object = brfss.2014.small)
##                TRNSGNDR     X_AGEG5YR          X_RACE
##  Male to female       : 77   Min.   : 5.000   Min.   :1.000
##  Female to male       :113   1st Qu.: 7.000   1st Qu.:1.000
##  Gender non-conforming: 32   Median : 8.000   Median :1.000
##  Not transgender      :  0   Mean   : 7.986   Mean   :2.054
##  Not sure             :  0   3rd Qu.: 9.000   3rd Qu.:2.000
##  Refused              :  0   Max.   :11.000   Max.   :9.000
##
##     X_INCOMG        X_EDUCAG        HLTHPLN1         HADMAM
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
```

```
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000
##  Median :4.000   Median :3.000   Median :1.000   Median :1.000
##  Mean   :3.685   Mean   :2.595   Mean   :1.108   Mean   :1.171
##  3rd Qu.:5.000   3rd Qu.:3.000   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :9.000   Max.   :4.000   Max.   :2.000   Max.   :9.000
##
##     X_AGE80         PHYSHLTH
##  Min.   :40.00   Min.   : 0.000
##  1st Qu.:50.00   1st Qu.: 0.000
##  Median :57.00   Median : 1.000
##  Mean   :56.83   Mean   : 7.528
##  3rd Qu.:63.75   3rd Qu.:11.000
##  Max.   :74.00   Max.   :30.000
##                  NA's   :10
```

Leslie summarized what she wrote. The first `filter()` chose observations that were any one of the three categories of transgender included in the data. She used the `|` "or" operator for this `filter()`. The second `filter()` chose people in an age category above category 4 but below category 12, in the age categories 5 through 11. Leslie asked why they did not use the `X_AGE80` to choose people 40 to 74. Nancy replied that she had tried that first while Leslie was reading the codebook, but she found a few observations in `X_AGE80` to be coded strangely, and since the age categories variable `X_AGEG5YR` was the one used in the table they were trying to reproduce, she thought that was a better idea.

The last `filter()` used the `!is.na` to choose observations where the `HADMAM` variable was not `NA`. Applying all of these filters resulted in a smaller data frame with 222 observations. Leslie noticed that the breast cancer screening article included 220 people who fit the criteria and wondered why she and Nancy had 2 additional people in their data frame. She decided she would review the percentages closely in creating the table to see where these 2 people fit in and to determine if they should be excluded.

Kiara told Leslie that small mistakes seem to be common in published research, often due to errors in transferring numbers from statistical software into a word-processing program. She suggested that creating fully formatted tables directly in R or another software program could reduce errors and increase the reproducibility of published research.

Now that the data set contained the observations used to create the table, Kiara suggested that it may be useful to reduce the data set to contain only the variables used to create the table. In addition to transgender status, age categories, and mammogram information, the table contained percentages for race/ethnicity, income category, education category, and health insurance status. The complete list of variables for the table is as follows:

- `TRNSGNDR`
- `X_AGEG5YR`
- `X_RACE`
- `X_INCOMG`
- `X_EDUCAG`

- HLTHPLN1
- HADMAM

Nancy helped Leslie write some variable selection code to add to the filtering code they just wrote.

```
# create a subset of observations and variables
brfss.2014.small <- brfss.2014.cleaned %>%
  filter(TRNSGNDR == 'Male to female'|
           TRNSGNDR == 'Female to male'|
           TRNSGNDR == 'Gender non-conforming') %>%
  filter(X_AGEG5YR > 4 & X_AGEG5YR < 12) %>%
  filter(!is.na(HADMAM)) %>%
  select(TRNSGNDR, X_AGEG5YR, X_RACE, X_INCOMG, X_EDUCAG, HLTHPLN1, HADMAM)
```

Leslie used the `summary()` function to examine the new data set to see what it contained.

```
# summary statistics for the new data frame
summary(object = brfss.2014.small)
##                   TRNSGNDR     X_AGEG5YR          X_RACE
##  Male to female       : 77   Min.   : 5.000   Min.   :1.000
##  Female to male       :113   1st Qu.: 7.000   1st Qu.:1.000
##  Gender non-conforming: 32   Median : 8.000   Median :1.000
##  Not transgender      :  0   Mean   : 7.986   Mean   :2.054
##  Not sure             :  0   3rd Qu.: 9.000   3rd Qu.:2.000
##  Refused              :  0   Max.   :11.000   Max.   :9.000
##      X_INCOMG       X_EDUCAG        HLTHPLN1         HADMAM
##  Min.   :1.000   Min.   :1.000   Min.   :1.000   Min.   :1.000
##  1st Qu.:2.000   1st Qu.:2.000   1st Qu.:1.000   1st Qu.:1.000
##  Median :4.000   Median :3.000   Median :1.000   Median :1.000
##  Mean   :3.685   Mean   :2.595   Mean   :1.108   Mean   :1.171
##  3rd Qu.:5.000   3rd Qu.:3.000   3rd Qu.:1.000   3rd Qu.:1.000
##  Max.   :9.000   Max.   :4.000   Max.   :2.000   Max.   :9.000
```

Leslie noticed that some of the variables were the wrong data type, since R had computed the mean and median for each one when they were all categorical and should have been the factor data type. Luckily, Nancy knew of a variation on `mutate()` that could be used to change all the variables in this small data set to factor types. The `mutate_all()` function can be used to do something to every variable in a data frame. Nancy added `mutate_all(as.factor)` to the code and they tested to see if it worked.

```
# change variables to factor data types
brfss.2014.small <- brfss.2014.cleaned %>%
    filter(TRNSGNDR == 'Male to female'|
```

```
            TRNSGNDR == 'Female to male'|
            TRNSGNDR == 'Gender non-conforming') %>%
  filter(X_AGEG5YR > 4 & X_AGEG5YR < 12) %>%
  filter(!is.na(HADMAM)) %>%
  select(TRNSGNDR, X_AGEG5YR, X_RACE, X_INCOMG,
         X_EDUCAG, HLTHPLN1, HADMAM) %>%
  mutate_all(as.factor)

# summary statistics for the new data frame
summary(object = brfss.2014.small)
##                   TRNSGNDR    X_AGEG5YR     X_RACE     X_INCOMG X_EDUCAG
##  Male to female       : 77   5 :27     1       :152     1:46     1:24
##  Female to male       :113   6 :27     2       : 31     2:44     2:86
##  Gender non-conforming: 32   7 :32     8       : 11     3:19     3:68
##  Not transgender      :  0   8 :44     7       :  8     4:26     4:44
##  Not sure             :  0   9 :44     5       :  7     5:65
##  Refused              :  0   10:24     4       :  6     9:22
##                              11:24     (Other): 7
## HLTHPLN1 HADMAM
##  1:198    1:198
##  2: 24    2: 22
##           9:  2
##
##
##
##
```

2.7.1.2 ADDING LABELS TO VARIABLES

Leslie noticed that, while the variables were all factors, many did not have labels for each category. Nancy reminded Leslie that they could use `mutate()` and `recode_factor()` to add the category labels like they did for the `TRNSGNDR` variable. Leslie and Nancy reviewed the codebook and worked together to write the code to add the labels. Leslie was having some trouble with the apostrophe in "Don't know," and Nancy told her that adding the \ character before punctuation allows R to read the punctuation correctly. Leslie added the \ and it appeared to work.

```
# add labels to factor variables
brfss.2014.small <- brfss.2014.cleaned %>%
  filter(TRNSGNDR == 'Male to female'|
           TRNSGNDR == 'Female to male'|
           TRNSGNDR == 'Gender non-conforming') %>%
  filter(X_AGEG5YR > 4 & X_AGEG5YR < 12) %>%
```

```
  filter(!is.na(HADMAM)) %>%
  select (TRNSGNDR, X_AGEG5YR, X_RACE, X_INCOMG,
          X_EDUCAG, HLTHPLN1, HADMAM) %>%
  mutate_all(as.factor) %>%
  mutate(X_AGEG5YR = recode_factor(.x = X_AGEG5YR,
                                   `5` = '40-44',
                                   `6` = '45-49',
                                   `7` = '50-54',
                                   `8` = '55-59',
                                   `9` = '60-64',
                                   `10` = '65-69',
                                   `11` = '70-74')) %>%
  mutate(X_INCOMG = recode_factor(.x = X_INCOMG,
                                  `1` = 'Less than $15,000',
                                  `2` = '$15,000 to less than $25,000',
                                  `3` = '$25,000 to less than $35,000',
                                  `4` = '$35,000 to less than $50,000',
                                  `5` = '$50,000 or more',
                                  `9` = 'Don\'t know/not sure/missing')) %>%
  mutate(X_EDUCAG = recode_factor(.x = X_EDUCAG,
                                  `1` = 'Did not graduate high school',
                                  `2` = 'Graduated high school',
                                  `3` = 'Attended college/technical school',
                                  `4` = 'Graduated from college/technical school',
                                  `9` = NA_character_)) %>%
  mutate(HLTHPLN1 = recode_factor(.x = HLTHPLN1,
                                  `1` = 'Yes',
                                  `2` = 'No',
                                  `7` = 'Don\'t know/not sure/missing',
                                  `9` = 'Refused'))

# check the work so far
summary(object = brfss.2014.small)
##                   TRNSGNDR   X_AGEG5YR      X_RACE
##  Male to female       : 77   40-44:27   1      :152
##  Female to male       :113   45-49:27   2      : 31
##  Gender non-conforming: 32   50-54:32   8      : 11
##  Not transgender      :  0   55-59:44   7      :  8
##  Not sure             :  0   60-64:44   5      :  7
##  Refused              :  0   65-69:24   4      :  6
##                              70-74:24   (Other):  7
```

```
##                          X_INCOMG
##  Less than $15,000           :46
##  $15,000 to less than $25,000:44
##  $25,000 to less than $35,000:19
##  $35,000 to less than $50,000:26
##  $50,000 or more             :65
##  Don't know/not sure/missing :22
##
##                                    X_EDUCAG  HLTHPLN1  HADMAM
##  Did not graduate high school          :24   Yes:198   1:198
##  Graduated high school                 :86   No : 24   2: 22
##  Attended college/technical school     :68             9:  2
##  Graduated from college/technical school:44
##
##
##
```

Everything looked good so far, but `X_RACE` and `HADMAM` still needed to be recoded. Leslie noticed that, based on the percentages reported in Table 2.1, Pacific Islanders were included in the "Other race" category and not in the "Asian/Pacific Islander" category. To reproduce the exact percentages in the table, she needed to include Pacific Islanders as "Other." Reviewing the BRFSS codebook, page 106, she found the following:

1) White only, non-Hispanic
2) Black only, non-Hispanic
3) American Indian or Alaskan Native only, Non-Hispanic
4) Asian only, non-Hispanic
5) Native Hawaiian or other Pacific Islander only, Non-Hispanic
6) Other race only, non-Hispanic
7) Multiracial, non-Hispanic
8) Hispanic
9) Don't know/Not sure/Refused

Table 2.1 used the following categories:

1) White
2) Black
3) Native American
4) Asian/Pacific Islander
5) Other

Leslie mapped the categories in the codebook into the categories in the table.

- Category 1 (White only, non-Hispanic) from the BRFSS data was labeled as White in the table
- Category 2 (Black only, non-Hispanic) from the BRFSS data was labeled as Black in the table
- Category 3 (American Indian or Alaskan Native only, Non-Hispanic) from BRFSS was Native American in the table
- Category 4 (Asian only, non-Hispanic) from BRFSS was Asian/Pacific Islander in the table
- Due to the mistake in labeling in the paper, categories 5, 6, 7, and 8 from BRFSS were Other in the table

Nancy watched closely while Leslie wrote the recoding for the levels of `X_RACE` exactly as it was written in the 2017 paper for the purposes of reproducing the table.

```
# add labels to factor variables
brfss.2014.small <- brfss.2014.cleaned %>%
  filter(TRNSGNDR == 'Male to female'|
           TRNSGNDR == 'Female to male'|
           TRNSGNDR == 'Gender non-conforming') %>%
  filter(X_AGEG5YR > 4 & X_AGEG5YR < 12) %>%
  filter(!is.na(HADMAM)) %>%
  select(TRNSGNDR, X_AGEG5YR, X_RACE, X_INCOMG,
         X_EDUCAG, HLTHPLN1, HADMAM) %>%
  mutate_all(as.factor) %>%
  mutate(X_AGEG5YR = recode_factor(.x = X_AGEG5YR,
                                   `5` = '40-44',
                                   `6` = '45-49',
                                   `7` = '50-54',
                                   `8` = '55-59',
                                   `9` = '60-64',
                                   `10` = '65-69',
                                   `11` = '70-74')) %>%
  mutate(X_INCOMG = recode_factor(.x = X_INCOMG,
                                  `1` = 'Less than $15,000',
                                  `2` = '$15,000 to less than $25,000',
                                  `3` = '$25,000 to less than $35,000',
                                  `4` = '$35,000 to less than $50,000',
                                  `5` = '$50,000 or more',
                                  `9` = 'Don\'t know/not sure/missing')) %>%
  mutate(X_EDUCAG = recode_factor(.x = X_EDUCAG,
                                  `1` = 'Did not graduate high school',
                                  `2` = 'Graduated high school',
```

```
                                  `3` = 'Attended college/technical school',
                                  `4` = 'Graduated from college/technical school',
                                  `9` = NA_character_)) %>%
  mutate(HLTHPLN1 = recode_factor(.x = HLTHPLN1,
                                  `1` = 'Yes',
                                  `2` = 'No',
                                  `7` = 'Don\'t know/not sure/missing',
                                  `9` = 'Refused')) %>%
  mutate(X_RACE = recode_factor(.x = X_RACE,
                                `1` = 'White',
                                `2` = 'Black',
                                `3` = 'Native American',
                                `4` = 'Asian/Pacific Islander',
                                `5` = 'Other',
                                `6` = 'Other',
                                `7` = 'Other',
                                `8` = 'Other',
                                `9` = 'Other')) %>%
  mutate(HADMAM = recode_factor(.x = HADMAM,
                                `1` = 'Yes',
                                `2` = 'No',
                                `7` = 'Don\'t know/not sure/missing',
                                `9` = 'Refused'))

#check the work so far
summary(object = brfss.2014.small)
##                    TRNSGNDR   X_AGEG5YR                      X_RACE
##  Male to female        : 77   40-44:27   White                  :152
##  Female to male        :113   45-49:27   Black                  : 31
##  Gender non-conforming: 32   50-54:32   Native American        :  4
##  Not transgender       :  0   55-59:44   Asian/Pacific Islander:  6
##  Not sure              :  0   60-64:44   Other                  : 29
##  Refused               :  0   65-69:24
##                               70-74:24
##                          X_INCOMG
##  Less than $15,000           :46
##  $15,000 to less than $25,000:44
##  $25,000 to less than $35,000:19
##  $35,000 to less than $50,000:26
##  $50,000 or more             :65
##  Don't know/not sure/missing :22
```

```
##
##                                    X_EDUCAG  HLTHPLN1      HADMAM
##  Did not graduate high school           :24   Yes:198   Yes    :198
##  Graduated high school                  :86   No : 24   No     : 22
##  Attended college/technical school      :68             Refused:  2
##  Graduated from college/technical school:44
##
##
##
```

2.7.1.3 CHECKING WORK AND RECODING PROBLEMATIC VALUES

Leslie wanted to figure out why there were 222 observations in her data frame and 220 in Table 2.1, which was showing the exact numbers from the Narayan et al. paper. Since the table only contained percentages, she thought she would review the percentages to find where the problem was. She remembered that percentages were produced with the `prop.table()` function, which needs a `table()` as input. To get a table of transgender status percentages, she used the following:

```
# get percents for TRNSGNDR
prop.table(x = table(brfss.2014.small$TRNSGNDR))
##
##        Male to female        Female to male Gender non-conforming
##             0.3468468             0.5090090             0.1441441
##       Not transgender              Not sure               Refused
##             0.0000000             0.0000000             0.0000000
```

These values were slightly different from those in the original table. Kiara thought this was likely due to the two-observation difference. Using her well-developed data-sleuthing skills, Kiara found that the difference was because the two observations where the `HADMAM` variable was coded as 9, or "Refused," were dropped before computing percentages of the `TRNSGNDR` variable but were kept for computing the percentages of all the other variables.

This was a really tricky data management problem! Leslie asked if they could change `TRNSGNDR` to `NA` when the `HADMAM` variable was category 9, which is the code for "Refused." Kiara said this would work, but it required learning a new function, `if_else()`.

Kiara explained that the `if_else()` function takes three arguments. The first argument is a logical statement (or condition) that must be either `TRUE` or `FALSE`. The second argument is `true =`. This is where you tell R what to do if the statement from the first argument is `TRUE`. The third argument, `false =`, is what you want to happen if the statement from the first argument is `FALSE`. The second and third arguments have to be the same data type. Before they wrote the code, Kiara first had Leslie say out loud what she wanted the `if_else()` function to do. She responded with the following:

> "For each person in the data set, if that person's value in `HADMAM` was *not* equal to 9, then leave their `TRNSGNDR` value as it is (do nothing). For everyone else that *does* have a value of 9 in `HADMAM`, change their `TRNSGNDR` value to be `NA`."

Kiara and Nancy looked at each other and were quite impressed with Leslie! Nancy added a line to the data management before `select()` and tested it.

```
# complete data management code
brfss.2014.small <- brfss.2014.cleaned %>%
  filter(TRNSGNDR == 'Male to female'|
           TRNSGNDR == 'Female to male'|
           TRNSGNDR == 'Gender non-conforming') %>%
  filter(X_AGEG5YR > 4 & X_AGEG5YR < 12) %>%
  filter(!is.na(HADMAM)) %>%
  mutate(TRNSGNDR = if_else(condition = HADMAM != 9,
                            true = TRNSGNDR,
                            false = factor(NA))) %>%
  select(TRNSGNDR, X_AGEG5YR, X_RACE, X_INCOMG, X_EDUCAG, HLTHPLN1) %>%
  mutate_all(as.factor) %>%
  mutate(X_AGEG5YR = recode_factor(.x = X_AGEG5YR,
                                   `5` = '40-44',
                                   `6` = '45-49',
                                   `7` = '50-54',
                                   `8` = '55-59',
                                   `9` = '60-64',
                                   `10` = '65-69',
                                   `11` = '70-74')) %>%
  mutate(X_INCOMG = recode_factor(.x = X_INCOMG,
                                  `1` = 'Less than $15,000',
                                  `2` = '$15,000 to less than $25,000',
                                  `3` = '$25,000 to less than $35,000',
                                  `4` = '$35,000 to less than $50,000',
                                  `5` = '$50,000 or more',
                                  `9` = 'Don\'t know/not sure/missing')) %>%
  mutate(X_EDUCAG = recode_factor(.x = X_EDUCAG,
                                  `1` = 'Did not graduate high school',
                                  `2` = 'Graduated high school',
                                  `3` = 'Attended college/technical school',
                                  `4` = 'Graduated from college/technical school',
                                  `9` = NA_character_)) %>%
  mutate(HLTHPLN1 = recode_factor(.x = HLTHPLN1,
                                  `1` = 'Yes',
                                  `2` = 'No',
                                  `7` = 'Don\'t know/not sure/missing',
                                  `9` = 'Refused')) %>%
```

```
  mutate(X_RACE = recode_factor(.x = X_RACE,
                                `1` = 'White',
                                `2` = 'Black',
                                `3` = 'Native American',
                                `4` = 'Asian/Pacific Islander',
                                `5` = 'Other',
                                `6` = 'Other',
                                `7` = 'Other',
                                `8` = 'Other',
                                `9` = 'Other')) %>%
  droplevels()

#check the work
prop.table(x = table(brfss.2014.small$TRNSGNDR))
##
##        Male to female        Female to male Gender non-conforming
##             0.3500000             0.5090909             0.1409091
```

It worked. The R-Team took a second to admire the results before continuing their conversation. Leslie noticed that there was now a `droplevels()` line at the end of the code. Nancy forgot to say she had snuck that into the code to remove the empty levels after recoding. After all that, Leslie vowed again that she would use good practices for organizing and annotating her code to avoid data management issues. She raised her right hand as if in court and swore to the team that she would annotate and organize her code. Kiara took that moment to explain another useful strategy for ensuring reproducibility, and that was to work with a ***co-pilot*** whenever coding. A co-pilot is another person who can write the code with you or run the code you have written and let you know if it worked.

2.7.2 CREATING A TABLE FROM THE CLEAN DATA

Creating well-formatted tables easily is one of the few things that R *does not do very well*. Nancy thought the **tableone** package (Yoshida, n.d.) was the best place to start, even though the tables it creates are not in the easiest format to use for formal reporting. The **tableone** package can create a table that includes all the variables in a data frame, and it automatically selects descriptive statistics to report based on the variable type. Nancy wrote a little code and showed Leslie how easy it is to create a table using `CreateTableOne()`.

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

# create a basic table
CreateTableOne(data = brfss.2014.small)
##
##                                                Overall
```

```
##   n                                          222
##   TRNSGNDR (%)
##      Male to female                           77 (35.0)
##      Female to male                          112 (50.9)
##      Gender non-conforming                    31 (14.1)
##   X_AGEG5YR (%)
##      40-44                                    27 (12.2)
##      45-49                                    27 (12.2)
##      50-54                                    32 (14.4)
##      55-59                                    44 (19.8)
##      60-64                                    44 (19.8)
##      65-69                                    24 (10.8)
##      70-74                                    24 (10.8)
##   X_RACE (%)
##      White                                   152 (68.5)
##      Black                                    31 (14.0)
##      Native American                           4 ( 1.8)
##      Asian/Pacific Islander                    6 ( 2.7)
##      Other                                    29 (13.1)
##   X_INCOMG (%)
##      Less than $15,000                        46 (20.7)
##      $15,000 to less than $25,000             44 (19.8)
##      $25,000 to less than $35,000             19 ( 8.6)
##      $35,000 to less than $50,000             26 (11.7)
##      $50,000 or more                          65 (29.3)
##      Don't know/not sure/missing              22 ( 9.9)
##   X_EDUCAG (%)
##      Did not graduate high school             24 (10.8)
##      Graduated high school                    86 (38.7)
##      Attended college/technical school        68 (30.6)
##      Graduated from college/technical school  44 (19.8)
##   HLTHPLN1 = No (%)                           24 (10.8)
```

After all the complicated data management, that was surprisingly easy! Leslie compared the table to Table 2.1 and noticed a few things to work on to reproduce it more closely:

- Table 2.1 only has percentages and not frequencies
- The `HLTHPLN1` variable shows the "Yes" group in Table 2.1
- The headings for different sections are not variable names in Table 2.1
- The percent signs can be removed from the section headers

Nancy had used **tableone** only a few times, so she didn't know if these things could be changed. She checked the help documentation by typing `CreateTableOne` into the search box in the help tab. She noticed a `print()` function in the help documentation that had some options for changing how the table prints, including an option to add variable labels, which will address the third bullet. To use this feature, the variables must have labels in the data frame, though.

To check and see if the variables had labels, Nancy used `str(brfss.2014.small)`, which shows the *structure* of the data frame.

```
# check the labels for the data frame
str(object = brfss.2014.small)
## 'data.frame':    222 obs. of  6 variables:
## $ TRNSGNDR : Factor w/ 3 levels "Male to female",..: 1 1 2 3 1 3 2 1 1 2 ...
## $ X_AGEG5YR: Factor w/ 7 levels "40-44","45-49",..: 4 7 2 6 3 4 1 2 4 1 ...
## $ X_RACE   : Factor w/ 5 levels "White","Black",..: 1 1 5 5 1 1 1 1 1 5 ...
## $ X_INCOMG : Factor w/ 6 levels "Less than $15,000",..: 6 5 6 1 3 2 5 1
5 6 ...
## $ X_EDUCAG : Factor w/ 4 levels "Did not graduate high school",..: 2 4 4
1 2 2 4 2 4 1 ...
## $ HLTHPLN1 : Factor w/ 2 levels "Yes","No": 2 1 1 2 2 1 2 1 1 2 ...
```

The output from `str()` did not include anything that looked like labels. Nancy noticed that the **tableone** package uses labels created in the **labelled** package. She read the documentation for the **labelled** package and wrote some code to add labels to the `brfss.2014.small` data frame using `var_label()`, with the labels for the table listed in a vector. To indicate that the `TRNSGNDR` variable was missing two of the observations, Nancy included "($n = 220$)" in the label for the transgender status variable.

```
# add variable labels to print in table
labelled::var_label(x = brfss.2014.small) <- c("Transition status (n = 220)",
                                               "Age category",
                                               "Race/ethnicity",
                                               "Income category",
                                               "Education category",
                                               "Health insurance?")

# check data frame for labels
str(object = brfss.2014.small)
## 'data.frame':    222 obs. of  6 variables:
## $ TRNSGNDR : Factor w/ 3 levels "Male to female",..: 1 1 2 3 1 3 2 1 1 2 ...
## ..- attr(*, "label")= chr "Transition status (n = 220)"
## $ X_AGEG5YR: Factor w/ 7 levels "40-44","45-49",..: 4 7 2 6 3 4 1 2 4 1 ...
## ..- attr(*, "label")= chr "Age category"
## $ X_RACE   : Factor w/ 5 levels "White","Black",..: 1 1 5 5 1 1 1 1 1 5 ...
```

```
## ..- attr(*, "label")= chr "Race/ethnicity"
## $ X_INCOMG : Factor w/ 6 levels "Less than $15,000",..: 6 5 6 1 3 2 5 1
5 6 ...
## ..- attr(*, "label")= chr "Income category"
## $ X_EDUCAG : Factor w/ 4 levels "Did not graduate high school",..: 2 4 4
1 2 2 4 2 4 1 ...
## ..- attr(*, "label")= chr "Education category"
## $ HLTHPLN1 : Factor w/ 2 levels "Yes","No": 2 1 1 2 2 1 2 1 1 2 ...
## ..- attr(*, "label")= chr "Health insurance?"
```

The data frame now showed labels with each variable. Now that the labels were in place, Nancy showed Leslie the `print()` function used with `CreateTableOne()`. To use `print()`, the table is first saved as an object with a name.

```
# create a basic table as an object
trans.hc.table <- CreateTableOne(data = brfss.2014.small)

# use print to show table with labels
print(x = trans.hc.table, varLabels = TRUE)
##
##                                          Overall
##   n                                      222
##   Transition status (n = 220) (%)
##      Male to female                       77 (35.0)
##      Female to male                      112 (50.9)
##      Gender non-conforming                31 (14.1)
##   Age category (%)
##      40-44                                27 (12.2)
##      45-49                                27 (12.2)
##      50-54                                32 (14.4)
##      55-59                                44 (19.8)
##      60-64                                44 (19.8)
##      65-69                                24 (10.8)
##      70-74                                24 (10.8)
##   Race/ethnicity (%)
##      White                               152 (68.5)
##      Black                                31 (14.0)
##      Native American                       4 ( 1.8)
##      Asian/Pacific Islander                6 ( 2.7)
##      Other                                29 (13.1)
##   Income category (%)
```

```
##      Less than $15,000                          46 (20.7)
##      $15,000 to less than $25,000               44 (19.8)
##      $25,000 to less than $35,000               19 ( 8.6)
##      $35,000 to less than $50,000               26 (11.7)
##      $50,000 or more                            65 (29.3)
##      Don't know/not sure/missing                22 ( 9.9)
##   Education category (%)
##      Did not graduate high school               24 (10.8)
##      Graduated high school                      86 (38.7)
##      Attended college/technical school          68 (30.6)
##      Graduated from college/technical school    44 (19.8)
##   Health insurance? = No (%)                    24 (10.8)
```

Now that the labels were working, the R-Team wanted to limit the numbers reported to just percentages. Nancy and Leslie read through all the help documentation under the help tab, and Nancy noticed a way to just print the percentages. She added the `format = "p"` argument to the `print()` function. She then saw that the percent (%) symbols could be removed with `explain = FALSE`, so she added this as well.

```
# use print to show table with labels and percent
print(x = trans.hc.table,
      varLabels = TRUE,
      format = "p",
      explain = FALSE)
##
##                                            Overall
##   n                                         222
##   Transition status (n = 220)
##      Male to female                         35.0
##      Female to male                         50.9
##      Gender non-conforming                  14.1
##   Age category
##      40-44                                  12.2
##      45-49                                  12.2
##      50-54                                  14.4
##      55-59                                  19.8
##      60-64                                  19.8
##      65-69                                  10.8
##      70-74                                  10.8
##   Race/ethnicity
```

```
##      White                                      68.5
##      Black                                      14.0
##      Native American                             1.8
##      Asian/Pacific Islander                      2.7
##      Other                                      13.1
##   Income category
##      Less than $15,000                          20.7
##      $15,000 to less than $25,000               19.8
##      $25,000 to less than $35,000                8.6
##      $35,000 to less than $50,000               11.7
##      $50,000 or more                            29.3
##      Don't know/not sure/missing                 9.9
##   Education category
##      Did not graduate high school               10.8
##      Graduated high school                      38.7
##     Attended college/technical school           30.6
##     Graduated from college/technical school     19.8
##   Health insurance? = No                        10.8
```

The last thing they wanted to fix was to show the "Yes" category for the health insurance variable; however, they did not see an option to do this in the help documentation. They decided to leave this for another day, and Leslie looked forward to figuring it out on her own before the next meeting, unless Nancy figured it out first!

Nancy and Leslie reviewed the table and found that it looked ready to use in a report or manuscript and had almost identical content to Table 2.1. Leslie complained that it felt like a lot more work than just retyping the numbers from R into a word-processing program. Kiara reminded her that this may be why they had to spend so much time to find the two observations and suggested three reasons to spend the time developing tables directly in R or another software program:

1. Transcribing the numbers can (and does) result in errors.
2. Small changes to table contents can be made more easily.
3. Code can be reused, so developing new tables will take less time after the first one is complete.

Leslie begrudgingly agreed and asked Nancy if they could try creating a table that included numeric variables as well. She explained that many of the articles she has read have long tables with both categorical and continuous variables displayed together.

Nancy thought this was a great idea, especially since it would allow her to write some more code. In order to create this table, the data frame needed to include continuous variables. Leslie went back to the data management code they created and added the `PHYSHLTH` variable to the `select()` list. `PHYSHLTH` shows the number of days of poor physical health in the last 30 days, so it is not categorical. Nancy removed the `mutate_all(as.factor) %>%` function since the `recode_factor()` worked to change these variables anyhow. Leslie wondered how long ago Nancy had realized that they did not need the `mutate_all(as.factor) %>%` once they had all the factors recoding with `recode_factor()`.

```
# complete data management code
brfss.2014.small <- brfss.2014.cleaned %>%
  filter(TRNSGNDR == 'Male to female'|
           TRNSGNDR == 'Female to male'|
           TRNSGNDR == 'Gender non-conforming') %>%
  filter(X_AGEG5YR > 4 & X_AGEG5YR < 12) %>%
  filter(!is.na(HADMAM)) %>%
  mutate(TRNSGNDR = if_else(HADMAM != 9, TRNSGNDR, factor(NA))) %>%
  select(TRNSGNDR, X_AGEG5YR, X_RACE, X_INCOMG,
         X_EDUCAG, HLTHPLN1, PHYSHLTH) %>%
  mutate(X_AGEG5YR = recode_factor(.x = X_AGEG5YR,
                                   `5` = '40-44',
                                   `6` = '45-49',
                                   `7` = '50-54',
                                   `8` = '55-59',
                                   `9` = '60-64',
                                   `10` = '65-69',
                                   `11` = '70-74')) %>%
  mutate(X_INCOMG = recode_factor(.x = X_INCOMG,
                                  `1` = 'Less than $15,000',
                                  `2` = '$15,000 to less than $25,000',
                                  `3` = '$25,000 to less than $35,000',
                                  `4` = '$35,000 to less than $50,000',
                                  `5` = '$50,000 or more',
                                  `9` = 'Don\'t know/not sure/missing')) %>%
  mutate(X_EDUCAG = recode_factor(.x = X_EDUCAG,
                                  `1` = 'Did not graduate high school',
                                  `2` = 'Graduated high school',
                                  `3` = 'Attended college/technical school',
                                  `4` = 'Graduated from college/technical school',
                                  `9` = NA_character_)) %>%
  mutate(HLTHPLN1 = recode_factor(.x = HLTHPLN1,
                                  `1` = 'Yes',
                                  `2` = 'No',
                                  `7` = 'Don\'t know/not sure/missing',
                                  `9` = 'Refused')) %>%
  mutate(X_RACE = recode_factor(.x = X_RACE,
                                `1` = 'White',
                                `2` = 'Black',
                                `3` = 'Native American',
```

```
                                `4` = 'Asian/Pacific Islande',
                                `5` = 'Other',
                                `6` = 'Other',
                                `7` = 'Other',
                                `8` = 'Other',
                                `9` = 'Other')) %>%
  droplevels()

#check the work
prop.table(x = table(brfss.2014.small$TRNSGNDR))
##
##        Male to female        Female to male Gender non-conforming
##             0.3500000             0.5090909             0.1409091
```

To include PHYSHLTH in the table, they needed a variable label for it as well. Leslie added the variable label to the labeling code.

```
# add variable labels to print in table
labelled::var_label(x = brfss.2014.small) <- c("Transition status (n = 220)",
                                               "Age category",
                                               "Race/ethnicity",
                                               "Income category",
                                               "Education category",
                                               "Health insurance?",
                                               "Days/month poor physical health")

# check data frame for labels
str(object = brfss.2014.small)
## 'data.frame':    222 obs. of  7 variables:
##  $ TRNSGNDR : Factor w/ 3 levels "Male to female",..: 1 1 2 3 1 3 2 1 1 2 ...
## ..- attr(*, "label")= chr "Transition status (n = 220)"
## $ X_AGEG5YR: Factor w/ 7 levels "40-44","45-49",..: 4 7 2 6 3 4 1 2 4 1 ...
## ..- attr(*, "label")= chr "Age category"
## $ X_RACE   : Factor w/ 5 levels "White","Black",..: 1 1 5 5 1 1 1 1 1 5 ...
## ..- attr(*, "label")= chr "Race/ethnicity"
## $ X_INCOMG : Factor w/ 6 levels "Less than $15,000",..: 6 5 6 1 3 2 5 1
5 6 ...
## ..- attr(*, "label")= chr "Income category"
## $ X_EDUCAG : Factor w/ 4 levels "Did not graduate high school",..: 2 4 4
1 2 2 4 2 4 1 ...
```

```
## ..- attr(*, "label")= chr "Education category"
## $ HLTHPLN1 : Factor w/ 2 levels "Yes","No": 2 1 1 2 2 1 2 1 1 2 ...
## ..- attr(*, "label")= chr "Health insurance?"
## $ PHYSHLTH : num  30 5 4 30 0 NA 0 25 0 0 ...
## ..- attr(*, "label")= chr "Days/month poor physical health"
```

Then, Leslie copied the table code and ran it to see what would happen.

```
# create a basic table as an object
trans.hc.table <- CreateTableOne(data = brfss.2014.small)

# use print to show table with labels
print(x = trans.hc.table, varLabels = TRUE)
##
##                                          Overall
##   n                                       222
##   Transition status (n = 220) (%)
##      Male to female                        77 (35.0)
##      Female to male                       112 (50.9)
##      Gender non-conforming                 31 (14.1)
##   Age category (%)
##      40-44                                 27 (12.2)
##      45-49                                 27 (12.2)
##      50-54                                 32 (14.4)
##      55-59                                 44 (19.8)
##      60-64                                 44 (19.8)
##      65-69                                 24 (10.8)
##      70-74                                 24 (10.8)
##   Race/ethnicity (%)
##      White                                152 (68.5)
##      Black                                 31 (14.0)
##      Native American                        4 ( 1.8)
##      Asian/Pacific Islander                 6 ( 2.7)
##      Other                                 29 (13.1)
##   Income category (%)
##      Less than $15,000                     46 (20.7)
##      $15,000 to less than $25,000          44 (19.8)
##      $25,000 to less than $35,000          19 ( 8.6)
##      $35,000 to less than $50,000          26 (11.7)
##      $50,000 or more                       65 (29.3)
```

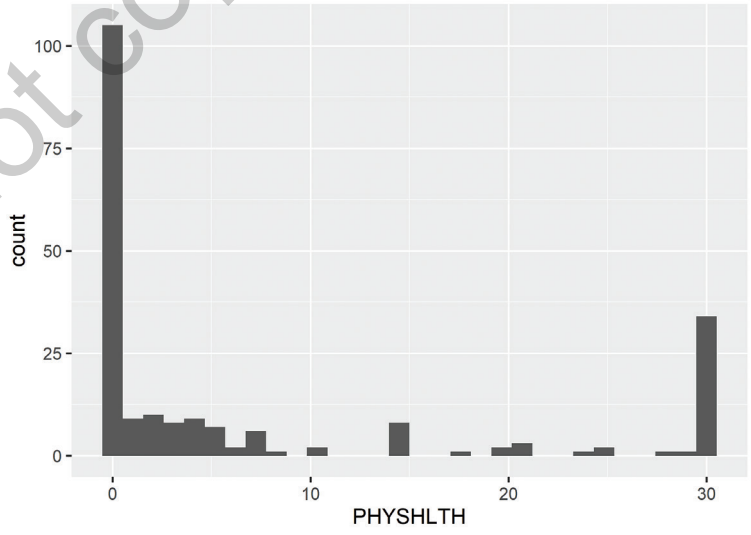
```
##      Don't know/not sure/missing                22 ( 9.9)
##   Education category (%)
##      Did not graduate high school               24 (10.8)
##      Graduated high school                      86 (38.7)
##      Attended college/technical school          68 (30.6)
##      Graduated from college/technical school    44 (19.8)
##   Health insurance? = No (%)                    24 (10.8)
##   Days/month poor physical health (mean (SD)) 7.53 (11.37)
```

The mean and standard deviation were added to the table! Kiara reminded Leslie that the mean and standard deviation were only good measures for the continuous variables when the variables were normally distributed. Leslie remembered that they had looked at the distribution of `PHYSHLTH` with a histogram. She took another look at Figure 2.16.

```
# make a histogram of PHYSHLTH (Figure 2.16)
brfss.2014.small %>%
  ggplot(aes(x = PHYSHLTH)) +
  geom_histogram()
```

The histogram confirmed that the variable is not normally distributed. Leslie remembered that median and IQR are good options for descriptive statistics when a variable is the numeric data type but not normally distributed. She looked in the documentation for the `print()` options for her table to see if there was any way to include the median and IQR instead of the mean and standard deviation. She found an argument for `nonnormal =` for adding the names of any numeric variables that are not normally distributed. Adding this to the print options, she got the following:

FIGURE 2.16 Distribution of the days of poor physical health variable



```
# use print to show table
print(x = trans.hc.table,
      varLabels = TRUE,
      nonnormal = 'PHYSHLTH')
##
##                                                  Overall
##   n                                               222
##   Transition status (n = 220) (%)
##      Male to female                                77 (35.0)
##      Female to male                               112 (50.9)
##      Gender non-conforming                         31 (14.1)
##   Age category (%)
##      40-44                                         27 (12.2)
##      45-49                                         27 (12.2)
##      50-54                                         32 (14.4)
##      55-59                                         44 (19.8)
##      60-64                                         44 (19.8)
##      65-69                                         24 (10.8)
##      70-74                                         24 (10.8)
##   Race/ethnicity (%)
##      White                                        152 (68.5)
##      Black                                         31 (14.0)
##      Native American                                4 ( 1.8)
##      Asian/Pacific Islander                         6 ( 2.7)
##      Other                                         29 (13.1)
##   Income category (%)
##      Less than $15,000                             46 (20.7)
##      $15,000 to less than $25,000                  44 (19.8)
##      $25,000 to less than $35,000                  19 ( 8.6)
##      $35,000 to less than $50,000                  26 (11.7)
##      $50,000 or more                               65 (29.3)
##      Don't know/not sure/missing                   22 ( 9.9)
##   Education category (%)
##      Did not graduate high school                  24 (10.8)
##      Graduated high school                         86 (38.7)
##      Attended college/technical school             68 (30.6)
##      Graduated from college/technical school       44 (19.8)
##   Health insurance? = No (%)                      24 (10.8)
##   Days/month poor physical health (median [IQR]) 1.00 [0.00, 11.00]
```

The R-Team was happy with the final product. Nancy explained that there is an R package called **kableExtra** that is more difficult to use, but offers more flexibility in formatting. She wanted to show Leslie the alternate way using **kableExtra**, but she knew that it might be a little overwhelming. Leslie suggested a quick coffee break first ☕. Kiara said that sounded perfect. They would caffeinate and then regroup to figure out a **kableExtra** table.

2.7.3 CREATING A TABLE FROM CLEAN DATA (ANOTHER WAY)

2.7.3.1 CREATING A NEW DATA FRAME WITH SUMMARY STATISTICS

Leslie joked that they had promised if she organized and annotated her code that she would not have to redo her work, and yet here she was, making the same table again. Nancy and Kiara smiled. Instead of continuing to work with the **tidyverse** formatting, Nancy suggested working in base R to get the summary statistics for this second way of building the table. She found this to be more straightforward for categorical variables and valuable to practice in general. Nancy explained that what they needed was a data frame with all the descriptive statistics they wanted listed in the table. Leslie started by creating a table of the percentages for the `TRNSGNDR` variable.

```
# get percents for TRNSGNDR
( trans.p <- prop.table(x = table(brfss.2014.small$TRNSGNDR)) )
##
##        Male to female        Female to male Gender non-conforming
##             0.3500000             0.5090909             0.1409091
```

These percentages needed some work. Leslie tried multiplying them by 100 and then rounding them to one decimal place, like the percentages in the table in the manuscript.

```
# get percents for TRNSGNDR
( trans.perc <- round(x = 100 * prop.table(x = table(brfss.2014.small$TRNSG
NDR)), 1) )
##
##        Male to female        Female to male Gender non-conforming
##                  35.0                  50.9                  14.1
```

Perfect! However, there were six more variables in the table. Nancy explained they could easily merge the percentages from all the variables in the `brfss.2014.small` data frame by converting each table of percentages into a data frame, giving the data frame a name, and using `rbind()` to merge the data frames, like this:

```
# get percents and assign a name for trans and race
# turn into data frames for easier merging
( trans.perc <- data.frame(round(x = prop.table(x = table(brfss.2014.small$
TRNSGNDR)) * 100, 1)) )
##                    Var1 Freq
## 1        Male to female 35.0
## 2        Female to male 50.9
## 3 Gender non-conforming 14.1
```

```
( race.perc <- data.frame(round(x = prop.table(x = table(brfss.2014.small
$X_RACE)) * 100, 1)) )
##                      Var1 Freq
## 1                   White 68.5
## 2                   Black 14.0
## 3        Native American  1.8
## 4 Asian/Pacific Islander   2.7
## 5                   Other 13.1
# merge together into one data frame
( table.perc <- rbind(trans.perc, race.perc) )
##                      Var1 Freq
## 1         Male to female 35.0
## 2         Female to male 50.9
## 3  Gender non-conforming 14.1
## 4                   White 68.5
## 5                   Black 14.0
## 6        Native American  1.8
## 7  Asian/Pacific Islander  2.7
## 8                   Other 13.1
```

Leslie thought Nancy must have had several cups of coffee on their break given how fast she typed the code.

2.7.3.2 USING A CUSTOM FUNCTION FOR REPEATED CODE

Leslie noted that `rbind()` worked great to create a data frame with each category and the percentage in that category. However, with seven variables to work with, Nancy suggested that they could be more efficient by writing their own ***function***, known as a ***custom function***. Kiara wondered if they could also use a ***for loop***. Nancy thought this would work, too, but they would try writing a function first. Leslie thought they should just use `rbind()` to do the whole thing, which Nancy said would also work, but she wanted to at least try one custom function. Nancy showed Leslie that she could put the code used to get the percentages into a function with the name of the object being replaced by `x`. She reminded Leslie that function names are in upper camel case (Section 1.6.3.3) and began writing the `TableFun()` function.

```
# finding percents and rounding to one decimal place
# putting all percents in one column with cbind
TableFun <- function(x){
    data.frame(round(x = prop.table(x = table(x)) * 100, 1))
}
```

This code creates a function called `TableFun()` that would apply the code inside the curly brackets `{}` to any variable. To use `TableFun()` on a single variable, Nancy showed Leslie she could just type `TableFun()` with the name of the variable in parentheses, like this:

```
# using the TableFun function for the TRNSGNDR variable
TableFun(x = brfss.2014.small$TRNSGNDR)
##                         x Freq
## 1         Male to female 35.0
## 2         Female to male 50.9
## 3 Gender non-conforming 14.1
```

2.7.3.3 USING THE APPLY FAMILY

However, said Nancy, it is complicated to use the function on all the variables in the `brfss.2014.small` data set at one time. One way to do it is to use the `lapply()` function, which is part of the `apply` family of functions (Box 2.2). The `lapply()` function is used to apply some code to each item in a list. In this case, use `lapply()` to apply `TableFun()` to each variable in the `brfss.2014.small` data frame. Nancy told Leslie that data frames are a special type of list where each variable is an element of the list.

2.2 Nancy's fancy code: *apply* and *lapply*

The *apply* family of functions can be used to do something to every row or column or both of a data frame or matrix or list. For example, if there were a small data frame that included the number of hot dogs several cousins ate each year for the last 2 years, the `apply()` function could be used to find the mean for each cousin (the rows) or each year (the columns).

```
#vectors for each variable
hotdogs.2016 <- c(6, 2, 0, 3, 9, 1)
hotdogs.2017 <- c(8, 3, 0, 2, 6, 2)
cousins <- c("Therese", "Geoff", "Nick", "John", "Jim", "Karen")

#make a data frame from vectors
#use cousins vector as row name rather than variable
cuz.hot.dogs <- data.frame(hotdogs.2016, hotdogs.2017)
row.names(x = cuz.hot.dogs) <- cousins
cuz.hot.dogs
##         hotdogs.2016 hotdogs.2017
## Therese            6            8
## Geoff              2            3
## Nick               0            0
## John               3            2
## Jim                9            6
## Karen              1            2
```

To find the mean for each cousin, use `apply()` for rows. The `apply()` function takes three arguments: data frame or matrix name (`X =`), rows or columns (`MARGIN =`), and function name (`FUN =`). For rows, the `MARGIN =` argument would be 1. For columns, the `MARGIN =` argument would be 2.

```
# mean by observation
apply(X = cuz.hot.dogs, MARGIN = 1, FUN = mean)
## Therese   Geoff    Nick    John     Jim     Karen
##     7.0     2.5     0.0     2.5     7.5       1.5
```

To find the mean for each year, use `apply()` for columns.

```
# mean by variable
apply(X = cuz.hot.dogs, MARGIN = 2, FUN = mean)
## hotdogs.2016 hotdogs.2017
##          3.5          3.5
```

Use `lapply()` instead if there is a list rather than a data frame. Perhaps it is a list of the pets the cousins have and also their favorite ice cream flavor.

```
# make a list
cuz.list <- list(pet = c('cat', 'dog', 'dog', 'cat', 'bird', 'cat'),
                 ice.cream = c('vanilla', 'chocolate',
                               'chocolate', 'chocolate',
                               'strawberry', 'strawberry'))
# print the list
cuz.list
## $pet
## [1] "cat"  "dog"  "dog"  "cat"  "bird" "cat"
##
## $ice.cream
## [1] "vanilla"  "chocolate"  "chocolate"  "chocolate"  "strawberry"
## [6] "strawberry"
```

To create a frequency table for each variable, use `lapply()` with two arguments. The first argument is the name of the list; the second is the function to use on each element of the list.

(Continued)

(Continued)

```
# make a table for each
# variable in cuzList list
lapply(X = cuz.list, FUN = table)
## $pet
##
## bird  cat  dog
##    1    3    2
##
## $ice.cream
##
##  chocolate strawberry    vanilla
##          3          2          1
```

Nancy went on to explain that using `lapply()` to apply `TableFun()` to the `brfss.2014.small` data frame would result in a list of tables of percentages, one table for each variable in the data frame. Once this list of tables is ready, she said, combine the tables using `rbind()` to merge objects by rows. Because there are seven tables to merge together, use the `do.call()` function to apply the `rbind()` function to each item in the list.

The `do.call()` function acts like a loop, conducting the same analyses for each element in the object, then starting over on the next element. So, Nancy summarized, `TableFun()` is applied to get tables of percentages from each variable in the `brfss.2014.small` data frame and then the `do.call()` function puts the table of percentages together using `rbind()` by going through and adding each table to the others one at a time. Leslie thought they should just have used `rbind()` and they would have been done by now.

Nancy explained that this process results in a new data frame containing each of the categories for each variable and the percentage of survey participants in the category. Leslie remembered that the `brfss.2014.small` data frame included the `PHYSHLTH` variable, which was not in the table. To exclude the `PHYSHLTH` variable from the calculations, they could take a subset of the variables by using the square brackets and subtracting the column where the variable was stored from the data frame. Leslie opened the data frame by clicking on it once under the Environment tab and counted to see that the `PHYSHLTH` variable was in column 7. They added –7 to the code to remove this column.

```
# use lapply to apply the TableFun function to
# all the variables in the data frame
# use the do.call function to call the rbind function
# to combine the list items into rows
( table.data <- do.call(rbind, (lapply(X = brfss.2014.small[ , -7],
                                       FUN = TableFun))) )
```

```
##                                                  x Freq
## TRNSGNDR.1                          Male to female 35.0
## TRNSGNDR.2                          Female to male 50.9
## TRNSGNDR.3                  Gender non-conforming 14.1
## X_AGEG5YR.1                                 40-44 12.2
## X_AGEG5YR.2                                 45-49 12.2
## X_AGEG5YR.3                                 50-54 14.4
## X_AGEG5YR.4                                 55-59 19.8
## X_AGEG5YR.5                                 60-64 19.8
## X_AGEG5YR.6                                 65-69 10.8
## X_AGEG5YR.7                                 70-74 10.8
## X_RACE.1                                    White 68.5
## X_RACE.2                                    Black 14.0
## X_RACE.3                           Native American  1.8
## X_RACE.4                    Asian/Pacific Islander  2.7
## X_RACE.5                                    Other 13.1
## X_INCOMG.1                     Less than $15,000 20.7
## X_INCOMG.2           $15,000 to less than $25,000 19.8
## X_INCOMG.3            $25,000 to less than $35,000 8.6
## X_INCOMG.4           $35,000 to less than $50,000 11.7
## X_INCOMG.5                     $50,000 or more 29.3
## X_INCOMG.6             Don't know/not sure/missing 9.9
## X_EDUCAG.1           Did not graduate high school 10.8
## X_EDUCAG.2                 Graduated high school 38.7
## X_EDUCAG.3       Attended college/technical school 30.6
## X_EDUCAG.4 Graduated from college/technical school 19.8
## HLTHPLN1.1                                    Yes 89.2
## HLTHPLN1.2                                     No 10.8
```

After she had looked at the `table.data` object, Leslie noticed that there were 27 observations (or rows) with row labels and two variables per row, `x` for the category name, and `Freq` for the percentage in the category. The last row was the No category for health care, which was not in the original table. She removed the last row by taking a subset of the `table.data` data frame so it included just rows 1 through 26.

```
# remove health care No category
table.data <- data.frame(table.data[c(1:26), ])
```

Finally, Leslie labeled the columns of the data frame to be consistent with the table columns.

```
# label the columns
colnames(x = table.data) <- c("Survey participant demographics (n = 220)", "Percent")
```

Nancy reassured Leslie that all of the functions used here would be used and explained multiple times throughout their meetings, so not to worry if `rbind()` or `do.call()` or anything else was not yet 100% clear. There would be additional opportunities to use these and all the other functions. Nancy suggested copying and pasting any new functions Leslie was trying to use and changing little things to see what would happen. In this way, the purpose of each part of the code might become clearer. Nancy also suggested that the help documentation in the lower left pane of RStudio and searching online can be useful for finding examples of code to play with and use.

Nancy also reminded Leslie that, if a function seems like too much at this point, she could always use the `rbind()` method they used earlier and wait until she was more comfortable with R to write her own functions. Kiara still thought for loops might be better, so she wrote some instructions about for loops for Leslie to review if she wanted yet another option (Box 2.3).

2.3 Kiara's reproducibility resource: Using for loops instead of functions

The function worked well to get the data ready for a **`kableExtra`** table. However, Kiara suggested that some people prefer to write functions only when they will be useful for more than one project. For loops are another option and are a fundamental aspect of computer programming. There are for loops in all programming languages, not just R. They are a way of doing the same thing multiple times, or doing something ***iteratively***. A for loop can be read as "for each item in (things you want to iterate over), do something." Kiara wrote the basic structure of a for loop.

```
# basic format of a for loop
for (i in 1:some number) {
   do something
}
```

The parts of the loop are as follows:

- The `1:some number` contains the elements you want to iterate over. For the **`kableExtra`** table, the same function would be used for the 7 variables from the `brfss.2014.small` data frame. This would be `1:7`. However, sometimes it is not clear how many iterations to do. For example, if you knew you wanted to do the same thing for each row in a data frame, but you didn't know off the top of your head how many rows there were, you could do `1:nrow(data.frame)`. The `nrow()` function counts the number of rows.
- The `i` stands for "each" and is the same type of `i` seen in equations. The top line then reads: "For each element in 1 through some number."
- The curly brackets `{ }` are around the body of the for loop. Whatever it is that should be done a lot of times, put that in the `{ }`.
- Most of the time the output would be stored, which requires one extra step of making an *empty* object that will store the output. This is called *initializing*.

For example, to use a for loop to square some numbers, here is how it might look:

```
# basic example of a for loop
squared.numbers <- NULL # initialize an empty vector that will
                        # contain the output
for (i in 1:10) {
  squared.numbers[i] <- i^2 # body of the for loop
}
# print out the result
print(squared.numbers)
##  [1]   1   4   9   16   25   36   49   64   81 100
```

Kiara wrote out the meaning in text.

> First, make an empty vector called `squared.numbers`. For each item in numbers 1 through 10, take that number or `i` and square it (`i^2`). Whatever the result of squaring `i` is, store that result in the `squared.numbers` vector.

She thought it might be necessary to explain why there is an `[i]` next to `squared.numbers`. The `i` is the element. So on the first iteration, `i = 1`. The `squared.numbers[i]` bit is saying to store the result of the 1st iteration in the 1st location of `squared.numbers`. On the 7th iteration, `i = 7`, make sure to store that result in the 7th location of the output `squared.numbers`.

Kiara wrote the initial for loop to use this structure to make the data frame for the **`kableExtra`** table that contains percentage per category of all 7 factors in the `brfss.2014.small` data set. As she wrote, she thought, "For each variable (or column) in `brfss.2014.small`, get the percentages via the `prop.table` function, and then combine the results into a single data frame."

```
# initialize an empty data frame
table.data <- data.frame()

# make the for loop
for (i in 1:ncol(brfss.2014.small)) {

  # get the percents for a variable and put them in table.each
  table.each <-
    data.frame(round(x = prop.table(x = table(brfss.2014.small[,
i])) * 100, 1))
```

(Continued)

(Continued)

```
  # combine table.each with whatever is in table.data already
  table.data <- rbind(table.data, table.each)
}

# print table.data
table.data
##                                       Var1 Freq
## 1                           Male to female 35.0
## 2                           Female to male 50.9
## 3                   Gender non-conforming 14.1
## 4                                   40-44 12.2
## 5                                   45-49 12.2
## 6                                   50-54 14.4
## 7                                   55-59 19.8
## 8                                   60-64 19.8
## 9                                   65-69 10.8
## 10                                  70-74 10.8
## 11                                  White 68.5
## 12                                  Black 14.0
## 13                         Native American  1.8
## 14                 Asian/Pacific Islander  2.7
## 15                                  Other 13.1
## 16                      Less than $15,000 20.7
## 17           $15,000 to less than $25,000 19.8
## 18           $25,000 to less than $35,000  8.6
## 19           $35,000 to less than $50,000 11.7
## 20                      $50,000 or more 29.3
## 21            Don't know/not sure/missing  9.9
## 22          Did not graduate high school 10.8
## 23                 Graduated high school 38.7
## 24       Attended college/technical school 30.6
## 25 Graduated from college/technical school 19.8
## 26                                    Yes 89.2
## 27                                     No 10.8
## 28                                      0 49.5
## 29                                      1 4.2
## 30                                      2 4.7
```

```
## 31                                    3  3.8
## 32                                    4  4.2
## 33                                    5  3.3
## 34                                    6  0.9
## 35                                    7  2.8
## 36                                    8  0.5
## 37                                   10  0.9
## 38                                   14  0.9
## 39                                   15  2.8
## 40                                   18  0.5
## 41                                   20  0.9
## 42                                   21  1.4
## 43                                   24  0.5
## 44                                   25  0.9
## 45                                   28  0.5
## 46                                   29  0.5
## 47                                   30 16.0
```

After this, there were still a couple details to work out. First, it was hard to tell which variable each of the categories was from in the first column, and the PHYSHLTH variable should have been excluded. Kiara edited the code to account for this by labeling the new data frame and removing the PHYSHLTH column from the loop.

```
# initialize an empty data frame
table.data <- data.frame()

# write the for loop
for (i in 1:(ncol(brfss.2014.small) - 1)) {
  # first, get the table
  table.each <-
    data.frame(round(x = prop.table(x = table(brfss.2014.small[,
i])) * 100, 1))

  # store the column name of that iteration for labels
  c.name <- colnames(brfss.2014.small[i])

  # make a new data frame that just contains the labels
  label.names <- data.frame(Variable = rep(c.name, times =
nrow(table.each)))
```

(Continued)

(Continued)

```
  # combine the label.names data frame and table.each data frame
via columns
  table.each.labelled <- cbind(label.names, table.each)

  # combine this with the table.data via rbind
  table.data <- rbind(table.data, table.each.labelled)
}

# print the new data frame
table.data
##     Variable                                    Var1 Freq
## 1   TRNSGNDR                        Male to female 35.0
## 2   TRNSGNDR                        Female to male 50.9
## 3   TRNSGNDR                 Gender non-conforming 14.1
## 4  X_AGEG5YR                                 40-44 12.2
## 5  X_AGEG5YR                                 45-49 12.2
## 6  X_AGEG5YR                                 50-54 14.4
## 7  X_AGEG5YR                                 55-59 19.8
## 8  X_AGEG5YR                                 60-64 19.8
## 9  X_AGEG5YR                                 65-69 10.8
## 10 X_AGEG5YR                                 70-74 10.8
## 11    X_RACE                                 White 68.5
## 12    X_RACE                                 Black 14.0
## 13    X_RACE                      Native American  1.8
## 14    X_RACE                Asian/Pacific Islander  2.7
## 15    X_RACE                                 Other 13.1
## 16 X_INCOMG                    Less than $15,000 20.7
## 17 X_INCOMG          $15,000 to less than $25,000 19.8
## 18 X_INCOMG          $25,000 to less than $35,000  8.6
## 19 X_INCOMG          $35,000 to less than $50,000 11.7
## 20 X_INCOMG                      $50,000 or more 29.3
## 21 X_INCOMG           Don't know/not sure/missing 9.9
## 22 X_EDUCAG          Did not graduate high school 10.8
## 23 X_EDUCAG                Graduated high school 38.7
## 24 X_EDUCAG       Attended college/technical school 30.6
## 25 X_EDUCAG Graduated from college/technical school 19.8
## 26 HLTHPLN1                                   Yes 89.2
## 27 HLTHPLN1                                    No 10.8
```

Two more adjustments were needed, making a subset of the `table.data` to exclude the last row and labeling the columns of the data frame to be consistent with the table columns.

```
# subset and add labels
table.data <- table.data[c(1:26), c(2:3)]
colnames(table.data) <- c("Survey participant demographics (n = 220)",
                          "Percent")

# print the new data frame
table.data
##              Survey participant demographics (n = 220) Percent
## 1                                        Male to female    35.0
## 2                                        Female to male    50.9
## 3                                Gender non-conforming     14.1
## 4                                                40-44     12.2
## 5                                                45-49     12.2
## 6                                                50-54     14.4
## 7                                                55-59     19.8
## 8                                                60-64     19.8
## 9                                                65-69     10.8
## 10                                               70-74     10.8
## 11                                               White     68.5
## 12                                               Black     14.0
## 13                                     Native American      1.8
## 14                                Asian/Pacific Islander     2.7
## 15                                               Other     13.1
## 16                                   Less than $15,000     20.7
## 17                     $15,000 to less than $25,000        19.8
## 18                     $25,000 to less than $35,000         8.6
## 19                     $35,000 to less than $50,000        11.7
## 20                                  $50,000 or more        29.3
## 21                      Don't know/not sure/missing         9.9
## 22                     Did not graduate high school        10.8
## 23                             Graduated high school       38.7
## 24                  Attended college/technical school      30.6
## 25            Graduated from college/technical school      19.8
## 26                                               Yes       89.2
```

The `table.data` data frame was now ready to go into a **`kableExtra`** table in Section 2.7.3.4.

2.7.3.4 FORMATTING TABLES WITH `KABLE()`

Finally, with a clean data frame containing all the information for the table, Nancy showed Leslie that the pipe structure can be used to send the data frame to `kable()` for formatting. Two packages are used in creating well-formatted `kable` tables, **`knitr`** and **`kableExtra`**. The **`knitr`** package is used to get the basic table, while the **`kableExtra`** package is used for some of the formatting options like adding sections.

TABLE 2.2 Transgender Survey Participant Demographics

Variable	Survey participant demographics (n = 220)	Percent
TRNSGNDR.1	Male to female	35.0
TRNSGNDR.2	Female to male	50.9
TRNSGNDR.3	Gender non-conforming	14.1
X_AGEG5YR.1	40–44	12.2
X_AGEG5YR.2	45–49	12.2
X_AGEG5YR.3	50–54	14.4
X_AGEG5YR.4	55–59	19.8
X_AGEG5YR.5	60–64	19.8
X_AGEG5YR.6	65–69	10.8
X_AGEG5YR.7	70–74	10.8
X_RACE.1	White	68.5
X_RACE.2	Black	14.0
X_RACE.3	Native American	1.8
X_RACE.4	Asian/Pacific Islander	2.7
X_RACE.5	Other	13.1
X_INCOMG.1	Less than $15,000	20.7
X_INCOMG.2	$15,000 to less than $25,000	19.8
X_INCOMG.3	$25,000 to less than $35,000	8.6
X_INCOMG.4	$35,000 to less than $50,000	11.7
X_INCOMG.5	$50,000 or more	29.3
X_INCOMG.6	Don't know/not sure/missing	9.9
X_EDUCAG.1	Did not graduate high school	10.8
X_EDUCAG.2	Graduated high school	38.7
X_EDUCAG.3	Attended college/technical school	30.6
X_EDUCAG.4	Graduated from college/technical school	19.8
HLTHPLN1.1	Yes	89.2

```
# open libraries
library(package = "knitr")
library(package = "kableExtra")

# send the table.data to kable and add a title (Table 2.2)
table.data %>%
kable(format = "html",
      caption = "Transgender Survey Participant Demographics") %>%
kable_styling()
```

"Well, Table 2.2 is a start," thought Leslie after she finally found the table in the viewer pane. For the final formatting, Nancy showed Leslie how to add labels by using the pipe structure. For each variable, use the `group_rows()` argument with the name of the variable and the rows the variable categories are in. For example, rows 1 to 3 are the transition status, so the first `group_rows()` argument shows: `group_rows("Transition status", 1, 3)`.

Unfortunately, `group_rows()` is one of those functions that is in more than one R package. It is in the **dplyr** package, which is part of the **tidyverse**, and it is in the **kableExtra** package. There are a couple of options to avoid the conflict: remove the **dplyr** package while creating the table, or use the `::` format to specify that the `group_rows()` function should be from the **kableExtra** package. Since **dplyr** is used for most of the data management tasks, Leslie decided to go with the `::` option and created Table 2.3.

```
# add the section names (Table 2.3)
table.data %>%
  kable(format = "html",
  caption = "Transgender Survey Participant Demographics",
      row.names = FALSE) %>%
kableExtra::group_rows(group_label = "Transition status",
           start_row = 1, end_row = 3) %>%
kableExtra::group_rows(group_label = "Age category",
           start_row = 4, end_row = 10) %>%
kableExtra::group_rows(group_label = "Race/ethnicity",
           start_row = 11, end_row = 15) %>%
kableExtra::group_rows(group_label = "Income category",
           start_row = 16, end_row = 21) %>%
kableExtra::group_rows(group_label = "Education category",
           start_row = 22, end_row = 25) %>%
kableExtra::group_rows(group_label = "Health insurance?",
           start_row = 26, end_row = 26)
```

Now that she had the manuscript table reproduced, Leslie was interested in making the bigger table with frequencies, and the median and IQR of PHYSHLTH. The first step, said Nancy, is to create a function (or

TABLE 2.3 Transgender Survey Participant Demographics

Variable Survey participant demographics (n = 220)	Percent
Transition status	
Male to female	35.0
Female to male	50.9
Gender non-conforming	14.1
Age category	
40–44	12.2
45–49	12.2
50–54	14.4
55–59	19.8
60–64	19.8
65–69	10.8
70–74	10.8
Race/ethnicity	
White	68.5
Black	14.0
Native American	1.8
Asian/Pacific Islander	2.7
Other	13.1
Income category	
Less than $15,000	20.7
$15,000 to less than $25,000	19.8
$25,000 to less than $35,000	8.6
$35,000 to less than $50,000	11.7
$50,000 or more	29.3
Don't know/not sure/missing	9.9
Education category	
Did not graduate high school	10.8
Graduated high school	38.7
Attended college/technical school	30.6
Graduated from college/technical school	19.8
Health insurance?	
Yes	89.2

expand `TableFun()`) to compute both frequencies and percentages. The `table()` function computes frequencies, so add it to `TableFun` and rename `TableFun` something logical like `TableFreqPerc`. Nancy wrote the code, but Leslie did not see the difference between `TableFun()` and `TableFreqPerc()`. Nancy showed her there was now a `table()` function before the `prop.table()` part.

```
# revise the TableFun function to compute both
# frequencies and percentages for Table 2.4
TableFreqPerc<- function(x){
    data.frame(table(x), round(x = prop.table(x = table(x)) * 100, 1))
}
# apply new function to brfss.2014.small data frame
bigger.table <- do.call(rbind, (lapply(X = brfss.2014.small, FUN =
TableFreqPerc)))
# click on the bigger.table object in the Environment
# pane to see the resulting table
# note that the categories show up twice in the data frame
# delete the second occurrence by making a subset of the data
bigger.table <- bigger.table[-3]
# remove Health insurance No category and PHYSHLTH numbers
bigger.table <- data.frame(bigger.table[c(1:26), ])
#add variable names
names(x = bigger.table) <- c("Survey participant demographics (n = 220)",
"Frequency", "Percent")
```

Now with a clean data frame, Leslie used the code from the smaller table—replacing `"table.data"` with `"bigger.table"`—to create a table with both frequencies and percentages (Table 2.4).

TABLE 2.4 Transgender Survey Participant Demographics

Survey participant demographics (*n* = 220)	Frequency	Percent
Transition status		
Male to female	77	35.0
Female to male	112	50.9
Gender non-conforming	31	14.1
Age category		
40–44	27	12.2
45–49	27	12.2
50–54	32	14.4
55–59	44	19.8

(Continued)

TABLE 2.4 (Continued)

Survey participant demographics (n = 220)	Frequency	Percent
60–64	44	19.8
65–69	24	10.8
70–74	24	10.8
Race/ethnicity		
White	152	68.5
Black	31	14.0
Native American	4	1.8
Asian/Pacific Islander	6	2.7
Other	29	13.1
Income category		
Less than $15,000	46	20.7
$15,000 to less than $25,000	44	19.8
$25,000 to less than $35,000	19	8.6
$35,000 to less than $50,000	26	11.7
$50,000 or more	65	29.3
Don't know/not sure/missing	22	9.9
Education category		
Did not graduate high school	24	10.8
Graduated high school	86	38.7
Attended college/technical school	68	30.6
Graduated from college/technical school	44	19.8
Health insurance?		
Yes	198	89.2

Although the original table creation was tedious, Leslie found that adding frequencies and rerunning the table code was not nearly as time-consuming.

The last thing, said Nancy, is to add the continuous `PHYSHLTH` variable to the table. Leslie reminded herself of the median and IQR for `PHYSHLTH` first.

```
# descriptive statistics for PHYSHLTH
median(x = brfss.2014.small$PHYSHLTH, na.rm = TRUE)
## [1] 1
IQR(x = brfss.2014.small$PHYSHLTH, na.rm = TRUE)
## [1] 11
```

The median days of poor physical health per month in this group was 1, and the IQR shows that there is an 11-day spread for the middle half of the people in this group. She hadn't been paying attention to the

values in the `CreateTableOne` process and noticed that this was quite different from the median of 0 and the IQR of 3 for the full data set. This group of transgender survey participants had a higher median number of days of poor health and more variation than the full sample. The other thing she noticed was the spread of people across income categories. Large percentages were in the lowest income category and the highest income category, with fewer people in between. Leslie wondered how this compared to all people in this age group, not just people who are transgender.

Now that Leslie was aware of what she was looking for in the table, Nancy told her that adding the code to the table required a few steps. She wrote the code with lots of comments and then walked Leslie through it. She decided to remove the "($n = 220$)" from the column header now that they were expanding the table beyond reproducing Table 2.1.

```
# revise the TableFreqPerc function to compute
# frequencies and percents for factors
# median and IQR for numeric data types for Table 2.5
TableFreqPerc<- function(x){
  if(is.factor(x))
    data.frame(table(x), round(x = prop.table(x = table(x)) * 100, 1))
}

# apply new function to brfss.2014.small data frame
bigger.table <- do.call(rbind, (lapply(X = brfss.2014.small, FUN =
TableFreqPerc)))

# note that the categories show up twice in the data frame
# delete the second occurrence by making a subset of the data
bigger.table <- bigger.table[-3]

# remove Health insurance No category
bigger.table <- data.frame(bigger.table[c(1:26), ])

# add the age summary data
# make a small data frame for the age information
bigger.table <- rbind(bigger.table, data.frame(x="Poor health (days/mo)",
                                  Freq = median(x = brfss.2014.small$PHYSHLTH,
                                         na.rm = TRUE),
                                  Freq.1 = IQR(x = brfss.2014.small$PHYSHLTH,
                                         na.rm = TRUE)))
# add variable names
names(bigger.table) <- c("Survey participant demographics", "Frequency",
"Percent")
```

Once the values were all together in the new data frame, Nancy told Leslie that she could reuse code from earlier to create a formatted Table 2.5. Leslie left the "($n = 220$)" out of the "Survey participant demographics" column title and added the overall sample size and the sample size for the transition status variable since the table was no longer truly a reproduced version of Table 2.1. Finally, she added one more `kableExtra::` section for row 27 to add the `group_label = "Days poor physical health per month (median, IQR)"` to the last section.

TABLE 2.5 Transgender Survey Participant Demographics (n = 222)

Survey participant demographics	Frequency	Percent
Transition status (n = 220)		
Male to female	77	35.0
Female to male	112	50.9
Gender non-conforming	31	14.1
Age category		
40–44	27	12.2
45–49	27	12.2
50–54	32	14.4
55–59	44	19.8
60–64	44	19.8
65–69	24	10.8
70–74	24	10.8
Race/ethnicity		
White	152	68.5
Black	31	14.0
Native American	4	1.8
Asian/Pacific Islander	6	2.7
Other	29	13.1
Income category		
Less than $15,000	46	20.7
$15,000 to less than $25,000	44	19.8
$25,000 to less than $35,000	19	8.6
$35,000 to less than $50,000	26	11.7
$50,000 or more	65	29.3
Don't know/not sure/missing	22	9.9
Education category		
Did not graduate high school	24	10.8
Graduated high school	86	38.7
Attended college/technical school	68	30.6
Graduated from college/technical school	44	19.8
Health insurance?		
Yes	198	89.2
Days poor physical health per month (median, IQR)		
Poor health (days/mo)	1	11.0

Leslie took a few minutes to write an interpretation of the results and the information reported in Table 2.5.

> Of the participants in the 2014 BRFSS, 222 people identified as transgender, were 40–74 years old (the age range when mammograms are recommended), and were asked about having a mammogram. Of these, 77 were male-to-female (35.0%), 112 were female-to-male (50.9%), and 31 were gender non-conforming (14.1%). Most were White ($n = 152$; 68.5%) or Black ($n = 31$; 14.0%). The most common age groups were 50–54 ($n = 44$; 19.8%) and 55–59 ($n = 44$; 19.8%).

Kiara stopped Leslie before she finished to let her know that, when there is a table in a report or manuscript, it is usually not necessary to interpret every number in the table. In fact, she said, this can be seen as repetition and a poor use of space. Kiara recommended that Leslie report information from the table that is important to the study at hand or that is unusual in some way. Leslie thought that made sense, since the entire purpose of a table is to report results in a clear way. Kiara commented that the interpretation she started to write was correct and clear, just that it had more detail than needed.

Everyone was exhausted, but before quitting, they went back over what they had learned about R and about transgender health care. Leslie felt like she learned a lot about descriptive statistics but that they did not get too much into the issue of transgender health care and cancer screening other than learning that this group has a high median number of days of poor health per month and a strange distribution of income, which were both things that might be important to explore further. Nancy and Kiara agreed and thought they might have some meetings where this happens just due to what they had learned that day. The team made a plan to meet soon to learn about graphing, which, as Nancy reminded everyone, is one of the biggest strengths of R.

It had been so long since they walked into the café that Leslie was hungry for another snack. Nancy and Kiara were chatting while they walked toward the door when they noticed that Leslie was not with them. They turned around to see her back in the line for food. Leslie waved. "See you next time."

Kiara and Nancy paused for a second but decided not to wait. Nancy waved back. "Not if I see you first!" she joked.

Kiara just rolled her eyes, ready to head home and relax after such an intense meeting.

/// 2.8 CHAPTER SUMMARY

Congratulations! Like Leslie, you've learned and practiced the following in this chapter.

2.8.1 Achievements unlocked in this chapter: Recap

2.8.1.1 Achievement 1 recap: Understanding variable types and data types

The main variable types in social science are continuous and categorical, which are *numeric* and *factor* data types in R. True continuous variables can take any value along a continuum (e.g., height), while categorical variables are measured in categories (e.g., marital status).

2.8.1.2 Achievement 2 recap: Choosing and conducting descriptive analyses for categorical (factor) variables

Descriptive statistics most appropriate for factor-type variables are frequencies, which show the number of observations for each category, and percentages, which show the percentage of observations for each category.

2.8.1.3 Achievement 3 recap: Choosing and conducting descriptive analyses for continuous (numeric) variables

If a numeric variable is normally distributed, the appropriate descriptive statistics to report are the mean and

standard deviation. The mean is a measure of central tendency that provides some idea of where the middle of the data is. The standard deviation is a measure of spread that indicates how spread out the values are. If a numeric variable is not normally distributed, the median and range or median and IQR are reported. The median is the middle value, while the IQR is the boundaries around the middle 50% of values.

2.8.1.4 Achievement 4 recap: Developing clear tables for reporting descriptive statistics

Computing the correct statistics is a great start, but organizing statistics in formats that are easy to understand is a key part of being an effective analyst. Creating tables that follow good formatting practices is one way to do this. Good formatting practices include number alignment, use of descriptive titles, choosing and reporting a consistent number of decimal places, and other formatting options.

2.8.2 Chapter exercises

The coder and hacker exercises are an opportunity to apply the skills from this chapter to a new scenario or a new data set. The coder edition evaluates the application of the concepts and functions learned in this R-Team meeting to new scenarios similar to those in the meeting. The hacker edition evaluates the use of the concepts and functions from this R-Team meeting in new scenarios, often going a step beyond what was explicitly explained.

The coder edition might be best for those who found some or all of the Check Your Understanding activities to be challenging or if they needed review before picking the correct responses to the multiple-choice questions. The hacker edition might be best if the Check Your Understanding activities were not too challenging and the multiple-choice questions were a breeze.

The multiple-choice questions and materials for the exercises are online at **edge.sagepub.com/harris1e**.

Q1: What is a measure of spread that is appropriate for each central tendency?

a. Mean
b. Median
c. Mode

Q2: Which of the following measures would be most appropriate for describing the central tendency of a variable that is continuous and normally distributed?

a. Mean
b. Variance
c. Median
d. Mode

Q3: Which of the following measures would be most appropriate for describing the spread of a variable that is extremely right-skewed?

a. Standard deviation
b. Range
c. IQR
d. Mode

Q4: In R, categorical variables are best represented by the _____ data type and continuous variables are best represented by the _____ data type.

Q5: Custom functions are useful when doing which of the following?

a. Loading a library
b. Visualizing the distribution of one variable
c. Working with continuous variables
d. Doing the same thing multiple times

2.8.2.1 Chapter exercises: Coder edition

Use the BRFSS data in the **transgender_hc_ch2.csv** file at **edge.sagepub.com/harris1e** to create a table of appropriate descriptive statistics for *all* transgender participants in the 2014 BRFSS. Spend a few minutes looking through the BRFSS website before beginning. Include variables representing transition status, days of poor physical health (`PHYSHLTH`), race/ethnicity, income, education, age, and age category. Write a paragraph using the numbers in the table to describe the characteristics of the transgender participants of the 2014 BRFSS.

1) Open the **transgender_hc_ch2.csv** 2014 BRFSS data file.
2) Select the data including only transgender participants.
3) Select the data including only the variables of interest.
4) Check the data types of all the variables and fix any that seem incorrect (Achievement 1).

5) Based on the BRFSS codebook, code missing values and add category labels appropriately.

6) Choose and conduct appropriate descriptive statistics for all variables in the small data set (Achievements 2 and 3).

7) Develop a well-formatted table of results including all variables in the small data set (Achievement 4).

8) Add a prolog and comments to your code.

9) Summarize the characteristics of transgender survey participants in the 2014 BRFSS.

2.8.2.2 Chapter exercises: Hacker edition

Complete #1–#5 from the coder edition; then do the following:

6) Find the mean or median value of `PHYSHLTH` (whichever is most appropriate) and recode to create a new factor variable with values above the mean or median labeled as "poor physical health" and values below the mean labeled as "good physical health"; check your coding (Achievement 3).

7) Choose and conduct appropriate descriptive statistics for all variables in the small data set (Achievements 2 and 3).

8) Develop a well-formatted table of results including all variables in the small data set (Achievement 4).

9) Add a prolog and comments to your code.

10) Summarize the characteristics of transgender survey participants in the 2014 BRFSS.

2.8.2.3 Instructor note

Solutions to exercises can be found on the website for this book, along with ideas for gamification for those who want to take it further.

SAGE edge™

Visit **edge.sagepub.com/harris1e** to download the datasets, complete the chapter exercises, and watch R tutorial videos.