2

DATA, VARIABLES, AND DATA MANAGEMENT

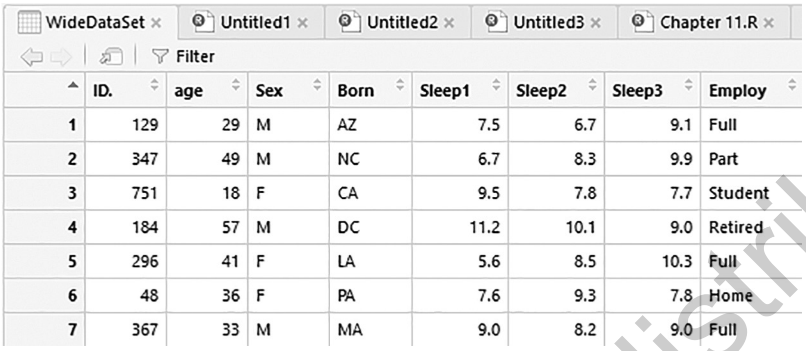
ABOUT THE DATA AND VARIABLES

The data used throughout this book come from a selection of variables in the 2016 *General Social Survey* (GSS) (Smith, Davern, Freese, & Morgan, 2019). The full GSS dataset consists of 2867 responses to 208 variables and are made available for public use. If you have questions regarding the variables or the coding, we recommend that you download the *General Social Survey* codebook as a supplemental resource to use as you work through this book's examples. The full codebook can be downloaded from the following website: http://gss.norc.org/documents/codebook/GSS_Codebook.pdf

STRUCTURE AND ORGANIZATION OF CLASSIC "WIDE" DATASETS

Now that we have learned the basics of R/RStudio, we will cover some of the important features of the data, including the structure of the dataset, which can be either wide or long. A wide dataset is organized in a way that a row of responses will be associated with a single participant (respondent). If there are repeated measures for the participant, they will be in separate columns. For example, if a participant were asked their age, gender, birth state, how many hours they slept for three nights in a row, and employment status, the responses would look something like Figure 2.1. Notice that the repeated measures of number of hours slept are in individual columns, and that the respondents—with a respondent identification number—each have their own row. The alternative structure of a dataset can be seen in a long dataset. This is when repeated measures are listed in the rows instead of across the columns. This results in multiple rows for a single participant, thus causing the physical structure of the dataset to be longer. The same data seen in Figure 2.1 as a wide dataset is shown in Figure 2.2; however, the figure is shortened to only show 10 of the 21 entries that existed.

FIGURE 2.1 ● SCREENSHOT OF DATA FROM A WIDE DATASET STRUCTURE

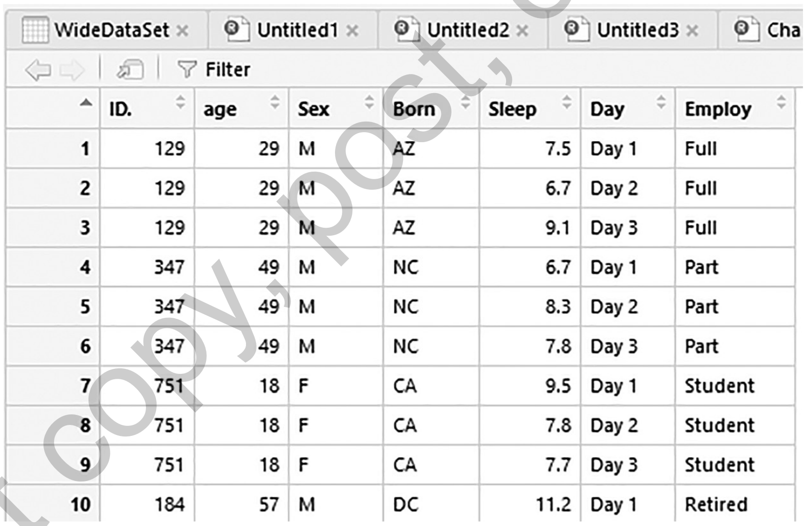


WideDataSet × | Untitled1 × | Untitled2 × | Untitled3 × | Chapter 11.R ×

Filter

	ID.	age	Sex	Born	Sleep1	Sleep2	Sleep3	Employ
1	129	29	M	AZ	7.5	6.7	9.1	Full
2	347	49	M	NC	6.7	8.3	9.9	Part
3	751	18	F	CA	9.5	7.8	7.7	Student
4	184	57	M	DC	11.2	10.1	9.0	Retired
5	296	41	F	LA	5.6	8.5	10.3	Full
6	48	36	F	PA	7.6	9.3	7.8	Home
7	367	33	M	MA	9.0	8.2	9.0	Full

FIGURE 2.2 ● SCREENSHOT OF PARTIAL DATA FROM A LONG DATASET STRUCTURE



WideDataSet × | Untitled1 × | Untitled2 × | Untitled3 × | Cha

Filter

	ID.	age	Sex	Born	Sleep	Day	Employ
1	129	29	M	AZ	7.5	Day 1	Full
2	129	29	M	AZ	6.7	Day 2	Full
3	129	29	M	AZ	9.1	Day 3	Full
4	347	49	M	NC	6.7	Day 1	Part
5	347	49	M	NC	8.3	Day 2	Part
6	347	49	M	NC	7.8	Day 3	Part
7	751	18	F	CA	9.5	Day 1	Student
8	751	18	F	CA	7.8	Day 2	Student
9	751	18	F	CA	7.7	Day 3	Student
10	184	57	M	DC	11.2	Day 1	Retired

Please note that the unit of analysis and distribution of the data change depending on the structure of the data. For instance, if we were to consider the distribution for hours of sleep in the wide dataset, each day would have its own column of values. However, in the long dataset, the distribution would display each measure of hours of sleep as separate rows repeated for the same individual.

We will be working with a wide dataset throughout this book. However, it is important to know what structure of dataset you are working on since statistical procedures will differ for each type. It is also important to have a clear understanding of your units of analysis, distributions, and how they are derived from the dataset with which you are working.

THE *GENERAL SOCIAL SURVEY*

The GSS (Smith et al., 2019) has typically been administered biennially (every two years) since 1972. The survey collects data on Americans' demographics, behaviors, and attitudes. These data can be used by social scientists to provide cross-sectional or "snapshot" observations of social trends that represent the population. In order to become familiar with and obtain data from the GSS, it is as easy as doing an Internet search for the "general social survey" or go to gss.norc.org.

The survey is conducted through face-to-face interviews with participants over 18 years old and takes roughly 90 minutes. Since 2006, the GSS has been conducted in Spanish as well as English. The sampling has changed from its earlier years of modified probability sampling to the more current full probability sampling (see the GSS Codebook). Additionally, the research team aims to reach a representative population by sampling in both rural and urban geographical units.

The full dataset for the GSS has over 950+ variables that are responses to questions about respondents' demographics, such as age, race, income, and marital status, as well as the respondents' feelings about social matters such as Internet use, national politics, and other personal preferences. It is possible to download all years or specific years, but for the purpose of this book, we will be working with the provided subset of 208 variables from 2016. This gives us more than enough data to become familiar with using R/RStudio and navigating the GSS for research purposes. However, at the website (gss.norc.org), click on the tab for "Get the Data" and you can acquire much more than the subset of data we use in this book.

Since we will be working with the data in R/RStudio, it is best for the data to be saved as a "comma separated values" or CSV file, which can be uploaded into R/RStudio pretty easily.

Tip: While working through this book, we recommend having a folder on your desktop titled *CSV* (or something else) for the storage of all your files. This will be convenient for loading your data into R/RStudio when you want to practice. You will be able to set the same working directory for your projects **(setwd('C:/users/username/Desktop/CSV')** because they will be in this folder. Just be sure the code has *your* filepath and working directory information to access the correct file.

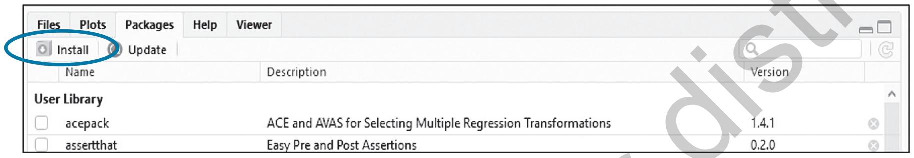
The first thing we want to do after saving the GSS data to a CSV file is to open our working directory, followed by uploading the CSV document using the **read.csv** command. After this is completed and the CSV file appears in the environment window, you can consider if you are ready to open any libraries, or just begin analysis and load any libraries as they are needed.

```
setwd('C:/Users/username/Desktop/CSV')# Setting the working
directory.
GSS2016 <- read.csv('GSS2016.csv', header=TRUE) # Uploading the
CSV file of the data.
```

If the packages for the libraries have already been installed in the RStudio program on the computer you are working on, then the libraries will open with the command **`library(libraryname)`**. If the packages have not already been installed, then they need to be installed, and the opening of the library re-run.

Tip: To install a package go to the lower right window of RStudio (the viewer window) and click on the Packages tab. Select Install and a new window will appear (Figure 2.3).

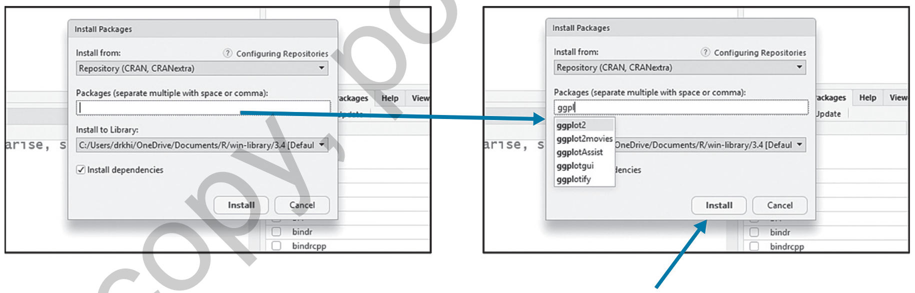
FIGURE 2.3 ● WHERE TO FIND "PACKAGES" AND "INSTALL"



Name	Description	Version
User Library		
acepack	ACE and AVAS for Selecting Multiple Regression Transformations	1.4.1
assertthat	Easy Pre and Post Assertions	0.2.0

The install packages window will open in the center of all four RStudio windows. Start typing the package you need in the empty packages bar and RStudio will start to generate possibilities for you to choose. Select the one you want and click on Install (Figure 2.4). Next, go back to the script window and re-run the library.

FIGURE 2.4 ● THE SCREENS FOR INSTALLING PACKAGES



Install from: Repository (CRAN, CRANextra)

Packages (separate multiple with space or comma):

Install to Library: C:/Users/drkhi/OneDrive/Documents/R/win-library/3.4 [Defaul

✓ Install dependencies

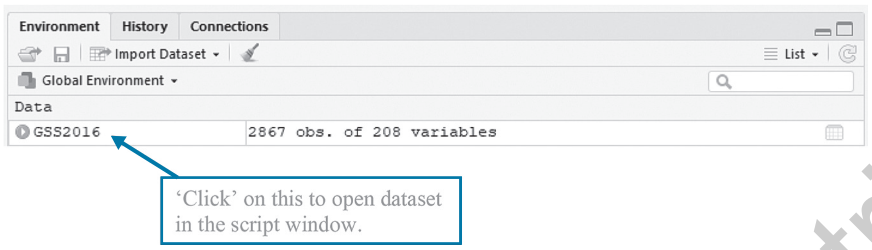
Packages (separate multiple with space or comma): ggpl

ggplot2
ggplot2movies
ggplotAssist
ggplotgui
ggplotify

After the working directory is set, the dataset is loaded, and the libraries installed, it is time to start becoming familiar with the data. By looking at the environment window, we can see that there are 208 variables and 2867 respondents. This is where we can double-check that our data has been loaded correctly into the program. After establishing that our data and our RStudio is set up properly, we can begin some data work. For the rest of this book, we will be performing all analyses on this subset of data from the GSS survey.

Tip: It is always a good idea to look at the dataset after you have loaded it into R/RStudio to make sure your values and labels are there. Double-check that you are about to work with the dataset you anticipated, and that it has loaded correctly. You can do this by clicking on the dataset in the environment window and it will appear in the console (Figures 2.5 and 2.6).

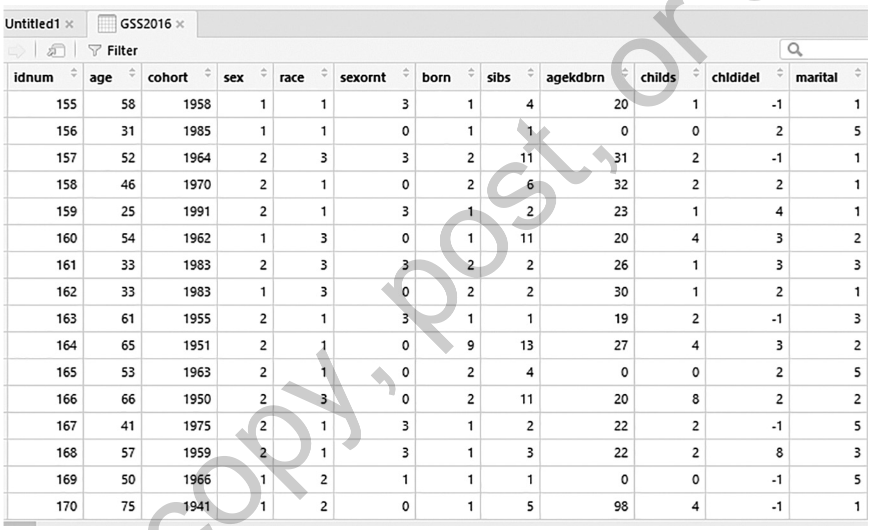
FIGURE 2.5 ● SCREENSHOT OF THE ENVIRONMENT WINDOW SHOWING THE DATASET HAS BEEN UPLOADED



```
Data
GSS2016                 2867 obs. of 208 variables
```

'Click' on this to open dataset in the script window.

FIGURE 2.6 ● SCREENSHOT OF THE SCRIPT WINDOW SHOWING THE ACTUAL VALUES (SPREADSHEET) OF THE DATASET



idnum	age	cohort	sex	race	sexornt	born	sibs	agekdbrn	childs	chldidel	marital
155	58	1958	1	1	3	1	4	20	1	-1	1
156	31	1985	1	1	0	1	1	0	0	2	5
157	52	1964	2	3	3	2	11	31	2	-1	1
158	46	1970	2	1	0	2	6	32	2	2	1
159	25	1991	2	1	3	1	2	23	1	4	1
160	54	1962	1	3	0	1	11	20	4	3	2
161	33	1983	2	3	3	2	2	26	1	3	3
162	33	1983	1	3	0	2	2	30	1	2	1
163	61	1955	2	1	3	1	1	19	2	-1	3
164	65	1951	2	1	0	9	13	27	4	3	2
165	53	1963	2	1	0	2	4	0	0	2	5
166	66	1950	2	3	0	2	11	20	8	2	2
167	41	1975	2	1	3	1	2	22	2	-1	5
168	57	1959	2	1	3	1	3	22	2	8	3
169	50	1966	1	2	1	1	1	0	0	-1	5
170	75	1941	1	2	0	1	5	98	4	-1	1

When using GSS data, keep in mind that there are different ways that it can be loaded into R (e.g., through Stata or SPSS) that may require you to alter a script slightly from what you have learned here. Also, be sure to pay attention to spacing and capitalization because R is sensitive to cases and spacing.

VARIABLES AND MEASUREMENT

In statistics, we are often looking to see if there is a relationship between variables beyond what we would see due to chance. These variables can be independent variables, dependent variables, or control variables depending on the statistical analysis chosen to address the research question.

An independent variable (IV), also known as the X variable, is the variable in which its responses are independent of the dependent variable. The IV can be manipulated (or controlled) by the researcher. Whereas, the dependent variable (DV), also known as the Y variable is presumed to be dependent upon the independent variable. We often hypothesize that a dependent variable will change in response to a change in the independent variable. In other words, Y relies on—or responds to—X. Control variables are the variables that we, the researchers, hold constant in order to attempt to isolate the relationship between the IV and the DV.

The level of measurement for variables can be nominal, ordinal, interval, or ratio, depending on how the data are collected for each variable. Nominal and ordinal measurements both have categorical (or discreet) data attributes, while interval and ratio measures are referred to as continuous.

Nominal data can only fit into one classification and the categories are not ranked, not ordered, and not equidistant from each other. Ordinal data fit into one category but the categories *can* be ranked or ordered. Interval and ratio data are both continuous data that have responses that have measurable, equal distances between the (numeric) attributes. Interval data does not have a meaningful zero, and ratio data does have a meaningful zero. A meaningful zero is such that if the response is zero, it represents an absence or lack of something.

Nominal data examples include individuals' political party, marital status, or yes/no options. Ordinal data examples include grade level (freshman, sophomore, junior, senior); 1st, 2nd, and 3rd places; or Likert scales (i.e., strongly agree, agree, neutral, disagree, strongly disagree). Interval data examples include degrees Celsius, year of birth, or IQ. Ratio data examples can include weight, length of time at a job, or number of children. For most of the analyses in this book, we will refer to interval and ratio variables together (i.e., interval/ratio variables).

It is extremely important to remember that the level of measurement is based on how the data are collected. For example, if we wanted to identify the level of measurement for `age` based on information from a survey, it would depend on how the question was asked. If respondents answered the question, "Are you over the age of 30?" with a yes or a no, the data reflect a dichotomous, or binary, nominal measure of age. If participants answered the question, "How old are you?" by picking from the following ordered options: 18.0–29.9 years old, 30.0–39.9 years old, 40.0–49.9 years old, 50.0–59.9 years old, and 60.0-plus years old, we would consider the data ordinal. If the participants were to respond to the question, "How many years old are you?" by filling in their age in years, that would be interval/ratio data.

When working with the GSS dataset, remember to use the codebook to help inform you of what each code (number) represents. Please do not let the size of the codebook intimidate; a search of any term (using a Control + F) can get you quickly to the variable you are interested in finding. For a small example of some of the variables we will be working with throughout the following chapters, see Table 2.1. We can also see that our subset of the GSS data contains a range of different levels of variable measurement.

TABLE 2.1 ⬢ LEVELS OF MEASUREMENT FOR VARIABLES IN GSS 2016 SAMPLE EXAMPLES

Nominal	Ordinal	Interval/Ratio
Race	Degree	Education
Gender	Happiness level	Number of siblings

RECODING VARIABLES

Categorical variables (i.e., nominal, ordinal) have attributes, or categories, that are often represented by a numeric code for statistical analyses. These include variables with only two options (e.g., yes/no), which are special cases known as binary, or dichotomous, variables. Continuous variables (i.e., interval/ratio) are already represented by numeric values (e.g., age).

LOGIC OF CODING

Coding data is the process by which we apply a (typically) numeric code to raw data so that we can enter it into a computer program and run statistical analysis. An example of this is the variable for gender (`sex`), which has the attributes for male and female. Because this book is using the GSS2016 dataset, the codes have already been established and can be found in the GSS codebook. The code for male is 1 and the code for female is 2. If you are collecting your own data, you, as the researcher, will be the one to establish the codes for your data and the code for male could just as easily be 13 and female 99—they are just numerical representations of the attributes male and female.

When running analyses, we work with the numeric representation of the variable attributes and apply the value labels which are character based (also known as factor or string data). However, when working with many other datasets, or a dataset you have created on your own, you will need to create your own codes for your variable attributes. Just as an example, we will add the value labels to the variable gender (`sex`) in order to create the output for both the codes and the labels.

```
table(GSS2016$sex) # Generate a table with raw frequencies for gender.
```

This will be the first (top) output below.

```
GSS2016$sex <- factor(GSS2016$sex,
                      levels = c(1,2),
                      labels = c("Male", "Female")) #Recode the
value labels for attributes in the variable gender.
```

```
table(GSS2016$sex)
```

# Generate a table with count for sex attributes after adding value labels and changing variable to factor.

Outputs: Raw Frequencies for **sex** with codes and with value labels.

```
 1      2       ← These are the codes for the attributes Male and Female
1276  1591
```

```
Male   Female ←  These are the value labels for the attributes Male and Female
1276    1591
```

Tip: During the process of adding the value labels, we convert the actual data to characters (factor/string) data, which may compromise future mathematical analysis we want to perform. It is best to do this after we have run our analysis.

INFORMATION BOX 2.1

Finding the Structure of the Data

If you are unsure of the structure of the data, you can always find out by running the command for structure (**str**). As an example, we can check the structure before we change the variable sex and after we change the variable sex and we can see the structure change from integer to factor.

```
str(GSS2016$sex)
int [1:2867] 1 1 1 2 2 2 1 2 1 1 . . .

str(GSS2016$sex)
Factor w/2 levels "Male" , "Female": 1 1 1 2 2 2 1 2 1 1 . . .
```

There are plenty of reasons why a researcher would want to recode some of their data, which is to change the attributes and/or numerical classifications of a variable. For example, they might want to change a continuous variable into a categorical variable, create a dichotomous (or binary) variable from a categorical variable, change original values, aggregate (collapse) data, remove unnecessary data, or handle with missing data, to name a few.

One common reason to recode a variable is that a researcher might want to combine a variable's multiple categories in a way that will result in two categories, also known as a dichotomous, or binary, variable. For instance, if we wanted to run an analysis with the variable `health`, which has seven categories (Excellent, Good, Fair, Poor, Don't know, No answer, and Not applicable), but only wanted two categories (Good or better and Less than good), we would need to recode the variable.

The first thing we need to do is create a new variable that is a duplicate of the original so we can work with the new variable but retain the original variable (Figure 2.7). This gives us the girth to make a mistake and not damage our original data. It is also important to keep the original variable whenever we collapse or aggregate data in order to retain the most detailed level of information in case we want to work with it again later.

In order to create a new variable, let's run a table on our old data to keep and use as a reference to make sure our new variable is created properly.

```
table(GSS2016$health)
```

Output: Raw frequencies of attributes for variable `health` before recoding.

```
  0   1   2   3   4 9
979 418 919 430 118 3
```

```
GSS2016$healthFact <-(GSS2016$health) # Create new (duplicate)
variable.
table(GSS2016$healthFact)
```

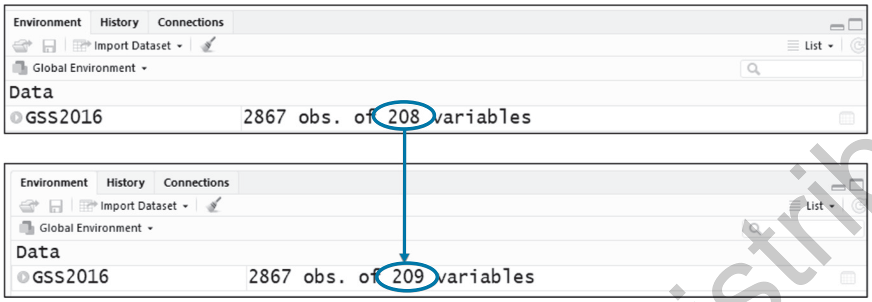
Tip: Always take a quick look at either the frequencies, or the variable itself, to double-check that your new variable was created properly. In this case, you would want to run the same command you did on the original to be able to compare information.

Output: Raw frequencies of attributes of our new variable `healthFact`.

```
  0   1   2   3   4 9
979 418 919 430 118 3
```

After creating the new variable, we can add the value labels to the variable and run a frequency table of the variable with the labels. However, you may have noticed that we named this variable `healthFact` for health *factor*. This is done to show what will happen if we label variables as factors too soon. If we do this, our data become factors (string or characters), which can sometimes create problems with strict numerical analysis. This is a

FIGURE 2.7 ⬢ NOTE THE CHANGE IN THE ENVIRONMENT WINDOW FROM 208 VARIABLES TO 209



```
Data
GSS2016          2867 obs. of 208 variables
```

```
Data
GSS2016          2867 obs. of 209 variables
```

compelling argument for always creating a *new* variable rather than making changes to the original variable.

Before recoding, we need to know what the codes actually represent. We can find this information by using the GSS codebook. This codebook is a very large PDF file that explains all 900+ variables and their attributes. We highly recommend the search function!

INFORMATION BOX 2.2

Searching for a Variable

Using the Control + F shortcut to produce a search window is a quick way to search for a variable. After searching for the health variable, which is what this was created from, you can see the variable attributes and their codes (refer to the screenshot below). As an example, *Fair* health responses were coded as 3.

```
GSS2016$healthFact <-factor(GSS2016$health,
                            levels=c(1,2,3,4,8,9,0),
                            labels=c("Excellent", "Good", "Fair",
                            "Poor", "Don't know", "No answer",
                            "Not applicable"))
table(GSS2016$healthFact)
```

Output: Raw frequencies of attributes of new variable `healthFact` with the labels created.

```
   Excellent         Good                 Fair     Poor
         418          919                  430      118
  Don't know    No answer       Not applicable
           0            3                  979
```

In addition to looking at the frequencies, we can become familiar with all of the raw values by simply running the name of the data and variable with the dollar sign operator discussed earlier:

```
GSS2016$health
```

Output: Screenshot of a portion of the individual responses to the variable `health`.

```
  [1] 2 0 2 2 1 0 4 2 2 2 0 4 0 2 0 1 1 0 0 2 0 1 0 1 3 0 1 0 0
 [30] 2 0 2 2 3 3 0 0 2 1 0 2 2 0 2 0 3 0 2 1 0 0 1 0 2 1 0 0 0
 [59] 1 2 1 0 3 0 2 3 0 3 3 3 3 2 0 0 1 0 1 2 1 2 2 1 2 2 2 2 0
 [88] 3 2 3 0 3 2 2 2 0 0 0 2 2 2 0 3 1 3 3 2 3 3 2 3 1 2 0 0 2
[117] 2 0 2 0 1 1 1 3 0 0 1 2 0 2 0 1 2 4 4 2 0 3 1 2 4 2 3 2 0
[146] 0 2 0 0 2 1 0 2 3 2 1 2 2 0 1 0 2 3 3 0 4 4 0 1 2 0 3 0 1
```

```
GSS2016$healthFact
```

Output: Screenshot of a portion of the individual responses to the variable `healthFact`.

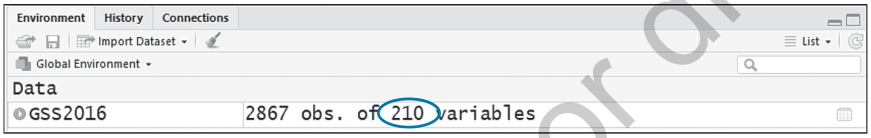
```
 [1]  Good           Not applicable Good           Good           Excellent
 [6]  Not applicable Poor           Good           Good           Good
[11]  Not applicable Poor           Not applicable Good           Not applicable
[16]  Excellent      Excellent      Not applicable Not applicable Good
[21]  Not applicable Excellent      Not applicable Excellent      Fair
[26]  Not applicable Excellent      Not applicable Not applicable Good
[31]  Not applicable Good           Good           Fair           Fair
[36]  Not applicable Not applicable Good           Excellent      Not applicable
```

We are able to see immediately that the labels have been successfully applied. Also noticeable, from either the frequency outputs or the data boxes, is the large amount of No answer and Not applicable cases. The GSS sometimes skips certain question models for certain individuals. When we recode the data, we will want to account for that and remove these from the calculations. We will be combining the Good and Excellent responses in a single category, "Good or better," which has a code of 1, and Fair and Poor responses into the category "Less than good" to which we will apply the code 2.

When we gave labels to the variable `healthFact` we created the new variable as a factor variable (Figure 2.8). However, we need to have a numeric variable in order to recode, so we will be creating a new variable (`healthBiNom`) that will be a binary nominal variable; meaning it has only two attributes (1 = Good or better, 2 = Less than good).

```
GSS2016$healthBiNom <-(GSS2016$health)  # Create new (duplicate) variable based on the original.
```

FIGURE 2.8 ● NOTE THE CHANGE IN THE ENVIRONMENT WINDOW FROM 209 TO 210



```
Data
GSS2016        2867 obs. of 210 variables
```

```
table(GSS2016$healthBiNom)
```

Output: Raw frequencies of attributes for variable `healthBiNom` before recoding.

```
  0   1   2   3   4  9
979 418 919 430 118  3
```

```
GSS2016$healthBiNom
```

Output: Screenshot of a portion of individual responses of `healthBiNom` before recoding.

```
  [1] 2 0 2 2 1 0 4 2 2 2 0 4 0 2 0 1 1 0 0 2 0 1 0 1 3 0 1 0 0
 [30] 2 0 2 2 3 3 0 0 2 1 0 2 2 0 2 0 3 0 2 1 0 0 1 0 2 1 0 0 0
 [59] 1 2 1 0 3 0 2 3 0 3 3 3 3 2 0 0 1 0 1 2 1 2 2 1 2 2 2 2 0
 [88] 3 2 3 0 3 2 2 2 0 0 0 2 2 2 0 3 1 3 3 2 3 3 2 3 1 2 0 0 2
[117] 2 0 2 0 1 1 1 3 0 0 1 2 0 2 0 1 2 4 4 2 0 3 1 2 4 2 3 2 0
[146] 0 2 0 0 2 1 0 2 3 2 1 2 2 0 1 0 2 3 3 0 4 4 0 1 2 0 3 0 1
[175] 0 2 1 3 2 3 3 4 3 0 2 0 1 2 3 0 2 0 2 3 2 0 2 2 2 1 2 0 0
```

Now we will run the scripts for recoding the variable from the original seven possible categories to create only two categories. Then we will examine frequencies and a snapshot of the data to make sure the recoding was successful. After determining that the 8, 9, and 0 codes are representing answers that need to be removed from the analysis, and that the strong positive responses of 1 and 2 (Excellent and Good) will be grouped and the neutral and negative responses of 3 and 4 (Fair and Poor) will be grouped, we will then recode the new variable using the script below.

```
GSS2016$healthBiNom[GSS2016$healthBiNom==0]=NA
GSS2016$healthBiNom[GSS2016$healthBiNom==8]=NA
GSS2016$healthBiNom[GSS2016$healthBiNom==9]=NA
GSS2016$healthBiNom[GSS2016$healthBiNom==1]=1
GSS2016$healthBiNom[GSS2016$healthBiNom==2]=1
GSS2016$healthBiNom[GSS2016$healthBiNom==3]=2
GSS2016$healthBiNom[GSS2016$healthBiNom==4]=2

GSS2016$healthBiNom
```

Output: Screenshot of a portion of responses to the variable `healthBiNom` after recoding.

```
  [1]    1  NA  1   1   1  NA  2   1   1   1  NA  2  NA  1  NA  1   1  NA  NA
 [20]    1  NA  1  NA  1   2  NA  1  NA  NA  1  NA  1   1   2   2  NA  NA  1
 [39]    1  NA  1   1  NA  1  NA  2  NA  1   1  NA  NA  1  NA  1   1  NA  NA
 [58]   NA   1  1   1  NA  2  NA  1   2  NA  2   2   2   2   1  NA  NA  1  NA
 [77]    1   1  1   1   1  1   1  1   1   1  NA  2   1   2  NA  2   1   1   1
 [96]   NA  NA NA   1   1  1  NA  2   1   2   2   1   2   2   1  2   1   1  NA
[115]   NA   1  1  NA   1  NA  1  1   1   2  NA  NA  1   1  NA  1  NA  1   1
[134]    2   2  1  NA   2  1   1  2   1   2   1  NA  NA  1  NA  NA  1   1  NA
[153]    1   2  1   1   1  1  NA  1  NA  1   2   2  NA  2   2  NA  1   1  NA
```

```
table(GSS2016$healthBiNom)
```

Output: Raw frequencies of attributes for variable `healthBiNom` after recoding.

```
   1     2
1337   548
```

It is also possible to look at the structure of a variable by using the "str" script with the variable to indicate how the data is being read by R.

```
str(GSS2016$health)
str(GSS2016$healthFact)
str(GSS2016$healthBiNom)
```

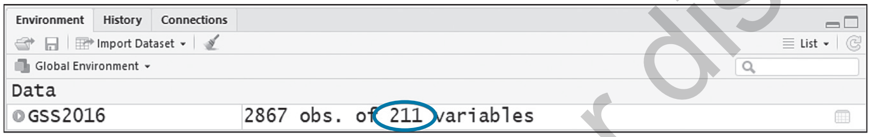
Outputs: Structures of the three variables just recoded.

```
 int [1:2867] 2 0 2 2 1 0 4 2 2 2 . . .
Factor w/7 levels "Excellent" , "Good": 2 7 2 2 1 7 4 2 2 2 . . .
 num [1:2867] 1 NA 1 1 1 NA 2 1 1 1 . . .
```

We can then generate a new factor variable that will be a factor variable with value labels for the binary health variable. We will add an *L* to the variable name to remind us that this health variable is the one with the NA and with the labels (Figure 2.9).

```
GSS2016$healthBiNomL <-factor(GSS2016$healthBiNom,
                               levels=c(1,2),
                               labels=c("Good or better", "Less
                               than good"))
```

FIGURE 2.9 ⬢ NOTE THE CHANGE IN THE ENVIRONMENT WINDOW



```
Data
GSS2016          2867 obs. of 211 variables
```

```
GSS2016$healthBiNomL
```

Output: Screenshot of a portion of responses to the variable `healthBiNomL` after recoding.

```
 [1] Good or better <NA>           Good or better Good or better Good or better
 [6] <NA>           Less than good Good or better Good or better Good or better
[11] <NA>           Less than good <NA>           Good or better <NA>
[16] Good or better Good or better <NA>           <NA>           Good or better
[21] <NA>           Good or better <NA>           Good or better Less than good
[26] <NA>           Good or better <NA>           <NA>           Good or better
[31] <NA>           Good or better Good or better Less than good Less than good
[36] <NA>           <NA>           Good or better Good or better <NA>
[41] Good or better Good or better <NA>           Good or better <NA>
```

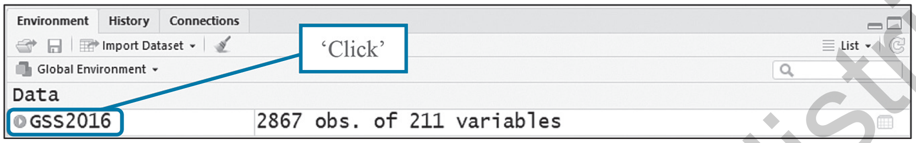
```
table(GSS2016$healthBiNomL)
```

Output: Raw frequencies of attributes of new variable `healthBiNomL` with the labels created.

```
Good or better    Less than good
     1337              548
```

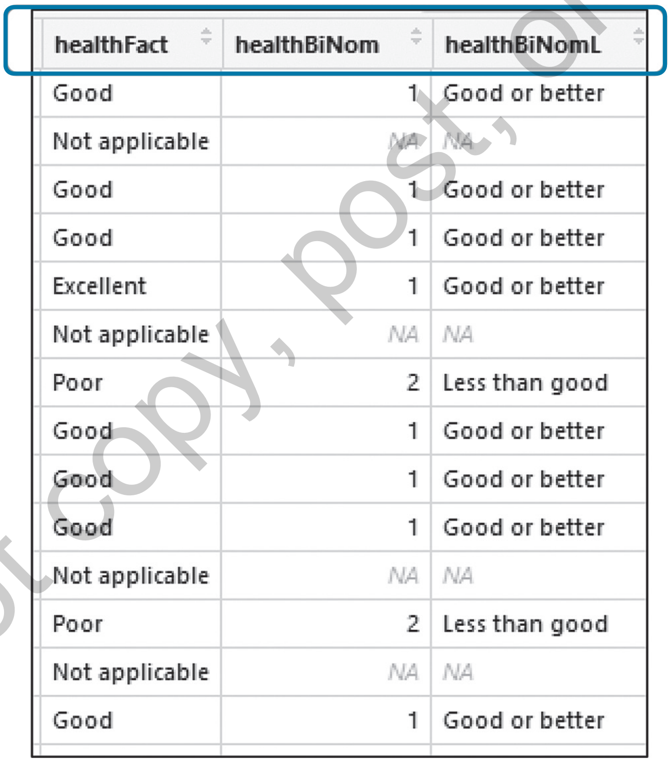
Tip: If you click on the data frame in our environment window (for this example it would be GSS2016), you can navigate to the end to see the new variables that have been created (Figures 2.10 and 2.11).

FIGURE 2.10 ● SCREENSHOT OF THE LOCATION OF DATA FRAME AREA IN THE GLOBAL ENVIRONMENT TO BE CLICKED



```
Data
GSS2016               2867 obs. of 211 variables
```

FIGURE 2.11 ● SCREENSHOT OF THE LAST FEW COLUMNS OF DATASET



healthFact	healthBiNom	healthBiNomL
Good	1	Good or better
Not applicable	NA	NA
Good	1	Good or better
Good	1	Good or better
Excellent	1	Good or better
Not applicable	NA	NA
Poor	2	Less than good
Good	1	Good or better
Good	1	Good or better
Good	1	Good or better
Not applicable	NA	NA
Poor	2	Less than good
Not applicable	NA	NA
Good	1	Good or better

We can also recode the variable for respondents' age at which first child was born (`agekdbrn`) using logical operations. First, we will need to run a table of the variable to see what responses exist.

```
table(GSS2016$agekdbrn)
```

Output: Frequencies of age at which first child was born `(agekdbrn)`.

```
  0    9  12  13  14  15  16  17   18   19   20   21   22   23   24   25   26
797    1   1   4  12  16  67  84  135  165  164  170  115  124  101  144  109
 27   28  29  30  31  32  33  34   35   36   37   38   39   40   41   42   43
 89   76  71  97  50  55  41  31   38   19   18   13   13   15    4    5    2
 44   45  46  47  98  99
  1    3   2   1  3   11
```

We then recode the values for Don't know (98), No Answer (99), and Not applicable (0); to make them all NA.

```
GSS2016$agekdbrnNEW<-GSS2016$agekdbrn
GSS2016$agekdbrnNEW[GSS2016$agekdbrn==0]=NA
GSS2016$agekdbrnNEW[GSS2016$agekdbrn==98]=NA
GSS2016$agekdbrnNEW[GSS2016$agekdbrn==99]=NA

table(GSS2016$agekdbrnNEW)
```

Output: Frequencies of age at which first child was born `(agekdbrnNEW)` after recoding for NA.

```
  9  12  13  14  15  16  17   18   19   20   21   22   23   24   25   26  27
  1   1   4  12  16  67  84  135  165  164  170  115  124  101  144  109  89
 28  29  30   31  32  33  34   35   36   37   38   39   40   41   42  43  44
 76  71  97   50  55  41  31   38   19   18   13   13   15    4    5   2   1
 45  46  47
  3   2  1
```

Now we can apply some logical operations to recode the variable into three age categories of less than 25, 25 to 35, and Over 35.

```
GSS2016$agekdbrnNEW[GSS2016$agekdbrn>35]= 1
GSS2016$agekdbrnNEW[GSS2016$agekdbrn>= 20 &
GSS2016$agekdbrn<=35]= 2
GSS2016$agekdbrnNEW[GSS2016$agekdbrn<20]= 3

table(GSS2016$agekdbrnNEW)
```

Output: Frequencies of age at which first child was born `(agekdbrnNEW)` after recoding.

```
  1     2      3
110   1475   1282
```

We can also create a new variable with value labels by writing the label directly into the script instead of having to write it independently. Here, we create another variable for the participant's age at the birth of their first child and treat it as a categorical/factor variable.

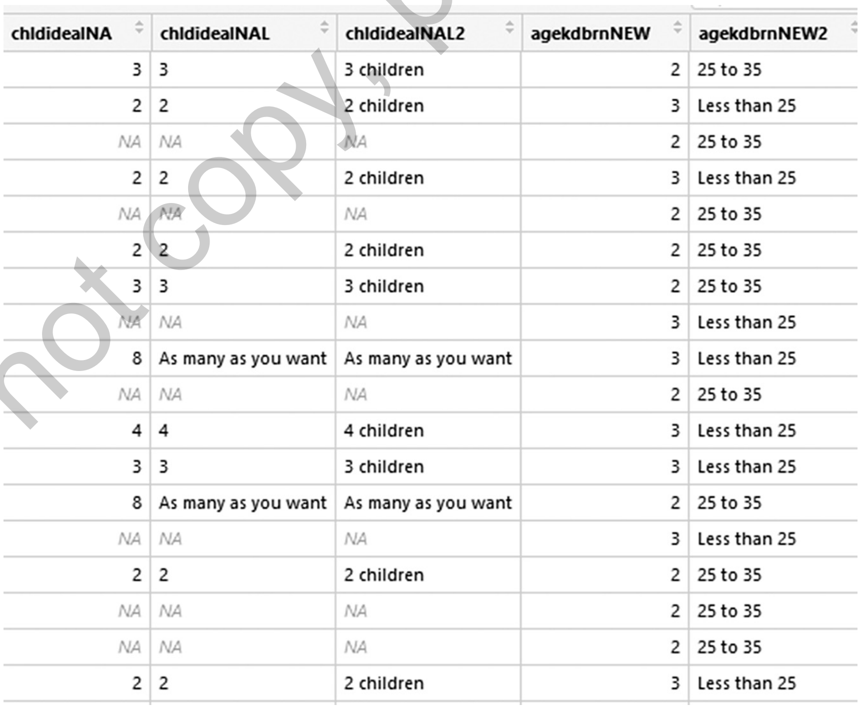
```
GSS2016$agekdbrnNEW2[GSS2016$agekdbrn>35]= "Over 35"
GSS2016$agekdbrnNEW2[GSS2016$agekdbrn>=20 &
GSS2016$agekdbrn<=35]= "25 to 35"
GSS2016$agekdbrnNEW2[GSS2016$agekdbrn<20]= "Less than 25"

table(GSS2016$agekdbrnNEW2)
```

Output: Frequencies of age at which first child was born (`agekdbrnNEW2`) after recoding.

```
  25 to 35   Less than 25      Over 35
    1475         1282            110
```

Tip: Similar to spreadsheets, it is also possible to see if a variable has been read by RStudio as a factor or as a number by looking at the actual data cases/observations in the script window. When the data in a column is right-side aligned, it is being read as numeric; however, if it is left-side aligned, it is being read as string information.

Output: Screenshot of left- and right-aligned data in the script window.



chldidealNA	chldidealNAL	chldidealNAL2	agekdbrnNEW	agekdbrnNEW2
3	3	3 children	2	25 to 35
2	2	2 children	3	Less than 25
NA	NA	NA	2	25 to 35
2	2	2 children	3	Less than 25
NA	NA	NA	2	25 to 35
2	2	2 children	2	25 to 35
3	3	3 children	2	25 to 35
NA	NA	NA	3	Less than 25
8	As many as you want	As many as you want	3	Less than 25
NA	NA	NA	2	25 to 35
4	4	4 children	3	Less than 25
3	3	3 children	3	Less than 25
8	As many as you want	As many as you want	2	25 to 35
NA	NA	NA	3	Less than 25
2	2	2 children	2	25 to 35
NA	NA	NA	2	25 to 35
NA	NA	NA	2	25 to 35
2	2	2 children	3	Less than 25

Another reason to recode a variable is to remove any negative numbers that may be included in the coding—for example, to change some missing values (e.g., -999) into truly missing values (i.e., NA). This can influence statistical analysis. As an example, we can look at the variable `chldidel`, which includes the responses to the question, "What do you think is the ideal number of children for a family to have?" Table 2.2 shows how the variable is originally coded. If we do not recode the value 9 to missing, then statistical analyses will be run as though those individuals would like to have nine children instead of their actual response, "No Answer/Don't Know." The following steps will take you through the recoding process.

TABLE 2.2 ● THE GSS CODES FOR VARIABLE **`chldidel`**

Original code	Label
0	None
1	One
2	Two
3	Three
4	Four
5	Five
6	Six
7	Seven or more
8	As many as you want
9	No answer, don't know
-1	Not applicable

```
table(GSS2016$chldidel)
```

Output: Raw frequencies of attributes of new variable `chldidel`.

```
 -1    0    1    2    3    4   5  6   7   8
980   13   36  844  477  194  35  6  11  271
```

```
GSS2016$chldidealNA <-(GSS2016$chldidel)
# Create new (duplicate) variable
table(GSS2016$chldidealNA)
```

Output: Raw frequencies of attributes of new variable `chldidealNA`.

```
 -1    0    1     2     3     4    5    6    7     8
980   13   36   844   477   194   35    6   11   271
```

```
GSS2016$chldidealNA[GSS2016$chldidealNA=="-1"]=NA
GSS2016$chldidealNA[GSS2016$chldidealNA==9]=NA

table(GSS2016$chldidealNA)
```

Output: Raw frequencies of attributes of new variable `chldidealNA` after recoding.

```
 0    1     2     3     4    5   6    7     8
13   36   844   477   194   35   6   11   271
```

```
GSS2016$chldidealNA
```

Output: Screenshot of a portion of responses to the variable `chldidealNA` after recoding.

```
 [1]   3  2  NA  2 NA 2  3  NA  8 NA  4  3   8 NA  2  NA NA  2  2  3  3NA 2
[24]  NA NA  4  3  3  2 NA  2  NA 3  NA  2   2  2  4  NA  3 NA 2   2 NA2 NA
[47]   2  2  NA  8  2  2  3  NA  4  3  3  2  NA 4  NA  5  NA  8  2 NA 3 5  2
[70]   2  3  4  3  5  8  2  2  3 NA NA  2  NA 2  NA NA  2  8  2 NA NA2 NA
[93]   2 NA NA  2  2  2 NA NA NA  2  2  NA NA 4  NA  3  2  NA 2  NA 3 3  8
[116] NA NA  3  NA 2  2 NA NA  2  2  3  3  NA 2  2  4  3  NA 2  NA 2 2 NA
```

We can then create a new variable that is a factor variable of the labels for the number of ideal children after the recoding of the missing data.

```
GSS2016$chldidealNAL <-factor(GSS2016$chldidealNA,
levels=c(0,1,2,3,4,5,6,7,8),
labels=c("No children", "1 child",
                  "2 children" , "3 children" , "4 children",
                  "5 children" , "6 children" , "7 or more",
                  "As many as you want"))
table(GSS2016$chldidealNAL)
```

Output: Raw frequencies of attributes of new variable `chldidealNAL` after recoding.

```
No children  1 child  2 children  3 children  4 children
    13          36        844         477         194
5 children   6 children  7 or more  As many as you want
    35           6           11              271
```

Table 2.3 highlights the changes made from the codes for the variable `chldidealNA` to the new codes for variable `chldidealNAL`, which includes the labels and has changed the no answer, don't know to a true missing response: `NA`. This was done in order to remove the -1 and 9 codes so that they are not included as meaningful answers in statistical calculations.

TABLE 2.3 ⬢ THE GSS CODES FOR VARIABLE **`chldidealNA`** OR **`chldidealNAL`**

Original Code	Label	New Code	New Label
0	None	0	No children
1	One	1	1 child
2	Two	2	2 children
3	Three	3	3 children
4	Four	4	4 children
5	Five	5	5 children
6	Six	6	6 children
7	Seven or more	7	Seven or more
8	As many as you want	8	As many as you want
9	No answer, Don't know	NA	Not applicable
-1	Not Applicable	NA	Not applicable

In Chapters 3 and 4, you will learn different ways to generate descriptive statistics for your data. As with many operations in R and RStudio, there are several different ways to do this. However, one way to validate that your data recoding was successful is to run some descriptive statistics before and after to ensure the recoding did in fact take place. The following example uses the library package **`pastecs`** to produce descriptive statistics for comparisons on continuous variables. In order to run the **`stat.desc`** command you need to first open the `library (pastecs)`.

```
library(pastecs)  # Be sure to open the library for pastecs.
stat.desc(GSS2016$chldidealNA)  # Running descriptive statistics.
```

Output: Descriptive statistics for variable `childidealNA`.

```
     nbr.val      nbr.null        nbr.na           min           max         range
1.887000e+03 1.300000e+01  9.800000e+02  0.000000e+00  8.000000e+00  8.000000e+00
         sum        median          mean       SE.mean  CI.mean.0.95           var
6.387000e+03  3.000000e+00  3.384738e+00  4.789750e-02  9.393765e-02  4.329099e+00
     std.dev      coef.var
2.080649e+00  6.147149e-01
```

RECODING MISSING VALUES

There are entire books written on the topic of handling missing data. For the purpose of this book, most of the values of "Don't know," "No answer," or "Not applicable" will be converted to the value NA, which is understood by R to mean "truly missing." Sometimes when working with large datasets, a researcher may decide to recode missing responses in the entire dataset prior to working with it, and not each time they run a calculation. There are a variety of other procedures for handling missing data but we will not discuss those in this book.

If we are using a dataset that has already been coded, such as the GSS, and we simply need to recode a value as missing, we can do so by recoding the value so that it is interpreted by R as "truly missing" (NA) and thus not included in any calculations. For example, the codes for the variable `zodiac` include 1 through 12, each of which corresponds to one of the twelve astrological signs of the zodiac—but also "98" and "99". Based on information in the codebook, we know that the latter two numbers represent "Don't know," and "No answer," respectively. Accordingly, we need to recode the "98" and "99" values to NA so that they are not calculated as meaningful responses in statistical analyses. First, create a new variable and name it `zodiac2`.

```
GSS2016$zodiac2<-(GSS2016$zodiac)  # Creating new variable.
table(GSS2016$zodiac2)  # Observing frequencies of categories.
```

Output: Raw frequencies of attributes of new variable `zodiac2` before recoding.

```
  1   2   3   4   5   6   7   8   9  10  11  12  98  99
225 240 224 257 235 274 227 219 214 197 236 234   5  80
```

```
GSS2016$zodiac2[GSS2016$zodiac2==98]=NA  # Recoding the missing
values as NA.
GSS2016$zodiac2[GSS2016$zodiac2==99]=NA
table(GSS2016$zodiac2)  # Observing frequencies after recoding.
```

Output: Raw frequencies of attributes of new variable `zodiac2` after recoding.

```
  1    2    3    4    5    6    7    8    9   10   11   12
225  240  224  257  235  274  227  219  214  197  236  234
```

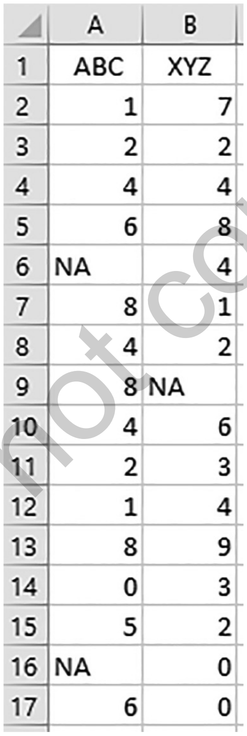
There are other ways to handle missing values that will depend on the data you are working with. In order to understand missing values a little better, we will start off with a simple example. Open a spreadsheet (e.g., Excel) and input the data exactly as it appears in Table 2.4. Title the spreadsheet "NotApp" and save it in your desktop folder.

Now we will open RStudio and load in the CSV file. Name the dataset "NOT".

If you loaded the "NOT" file into the same folder as the one you are currently working in, you do not need to set the working directory a second time.

```
setwd('C:/Users/username/Desktop/CSV') # Setting the working
directory (In this code, the CSV file is saved in a desktop folder called
CSV).
```

TABLE 2.4 ● DATA TO BE COPIED INTO A SPREADSHEET AND USED FOR THE NEXT EXAMPLE



	A	B
1	ABC	XYZ
2	1	7
3	2	2
4	4	4
5	6	8
6	NA	4
7	8	1
8	4	2
9	8	NA
10	4	6
11	2	3
12	1	4
13	8	9
14	0	3
15	5	2
16	NA	0
17	6	0

```
NOT <- read.csv('NotApp.csv',
header=TRUE) # Reading in the CSV file
NotApp and calling it NOT.
```

Next we can identify the cases in the variable ABC, and in the variable XYZ that are NA:

```
is.na(NOT$ABC)
is.na(NOT$XYZ)
```

Outputs: Showing where the NA is TRUE for `ABC` and `XYZ` independently.

```
> is.na(NOT$ABC)
 [1] FALSE FALSE FALSE FALSE TRUE
     FALSE FALSE FALSE FALSE FALSE
[11] FALSE FALSE FALSE FALSE TRUE
     FALSE
> is.na(NOT$XYZ)
 [1] FALSE FALSE FALSE FALSE FALSE
     FALSE FALSE TRUE FALSE FALSE
[11] FALSE FALSE FALSE FALSE FALSE
     FALSE
```

```
is.na(NOT)
```

Output: Showing where "is NA" is TRUE for the dataset NOT.

```
      ABC   XYZ
 [1,] FALSE FALSE
 [2,] FALSE FALSE
 [3,] FALSE FALSE
 [4,] FALSE FALSE
 [5,]  TRUE FALSE
 [6,] FALSE FALSE
 [7,] FALSE FALSE
 [8,] FALSE  TRUE
 [9,] FALSE FALSE
[10,] FALSE FALSE
[11,] FALSE FALSE
[12,] FALSE FALSE
[13,] FALSE FALSE
[14,] FALSE FALSE
[15,]  TRUE FALSE
[16,] FALSE FALSE
```

If we add an exclamation point (!), this is the equivalent of indicating "is not". The following script indicates that all the values that are not NA should be considered TRUE.

```
!is.na(NOT)
```

Output: Showing where "is not NA" is TRUE for the dataset NOT.

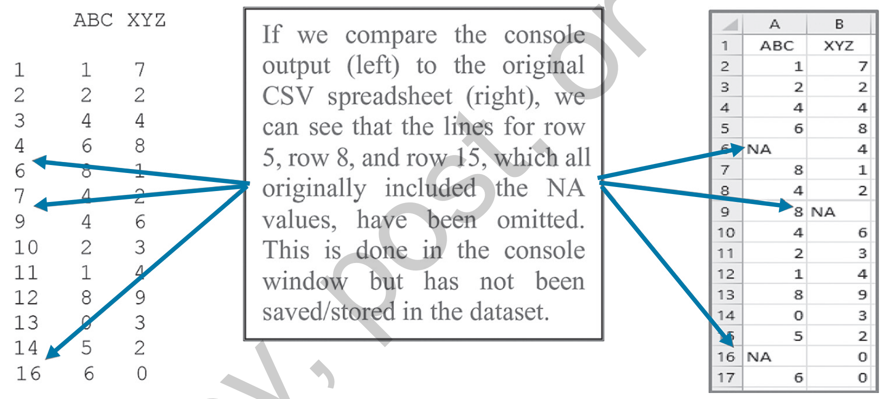
```
      ABC    XYZ
 [1,]  TRUE  TRUE
 [2,]  TRUE  TRUE
 [3,]  TRUE  TRUE
 [4,]  TRUE  TRUE
 [5,] FALSE  TRUE
 [6,]  TRUE  TRUE
 [7,]  TRUE  TRUE
 [8,]  TRUE FALSE
 [9,]  TRUE  TRUE
[10,]  TRUE  TRUE
[11,]  TRUE  TRUE
```

```
[12,]   TRUE   TRUE
[13,]   TRUE   TRUE
[14,]   TRUE   TRUE
[15,] FALSE   TRUE
[16,]   TRUE   TRUE
```

It is possible to remove the NAs from the dataset by using the **`na.omit`** command (Figure 2.12). This will remove the data for the entire row that contains the NA.

```
na.omit(NOT)
```

FIGURE 2.12 ● COMPARISON OF OUTPUT AFTER OMITTING NA DATA AND ORIGINAL DATA VALUES FOR NOT



```
     ABC  XYZ

1     1    7
2     2    2
3     4    4
4     6    8
6     8    1
7     4    2
9     4    6
10    2    3
11    1    4
12    8    9
13    0    3
14    5    2
16    6    0
```

If we compare the console output (left) to the original CSV spreadsheet (right), we can see that the lines for row 5, row 8, and row 15, which all originally included the NA values, have been omitted. This is done in the console window but has not been saved/stored in the dataset.

	A	B
1	ABC	XYZ
2	1	7
3	2	2
4	4	4
5	6	8
6	NA	4
7	8	1
8	4	2
9	8	NA
10	4	6
11	2	3
12	1	4
13	8	9
14	0	3
15	5	2
16	NA	0
17	6	0

We can create a new dataset with all of the rows with NA omitted (Figure 2.13).

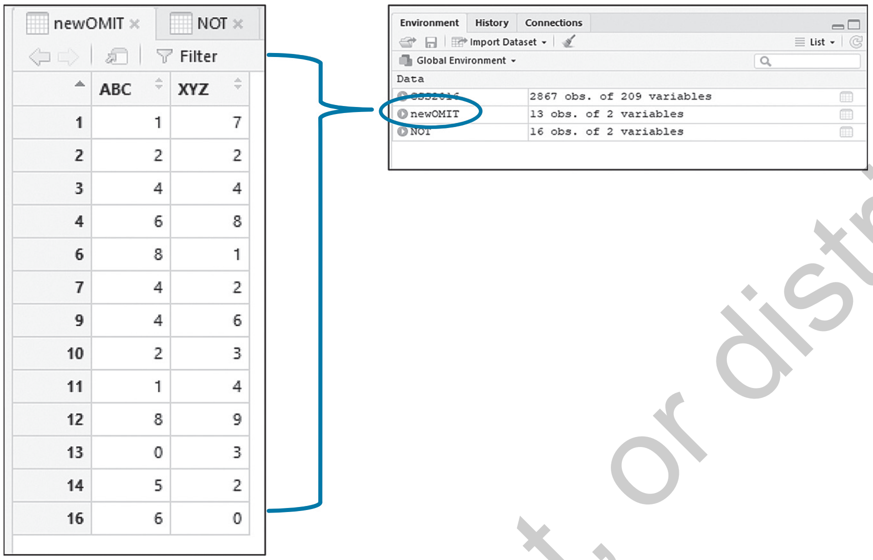
```
newOMIT<-na.omit(NOT)
```

Tip: Keep in mind that eliminating all rows with missing values often removes many cases (and a great deal of additional data) so this practice is generally not advisable.

If we keep the NA cases in the data, which is preferred, then we use the **`na.pass`** command to ignore the NA cases without removing them from the dataset entirely. For example, if we wanted to compute a new variable (`multiplied`) of ABC multiplied with XYZ, we can tell R to compute the variables while passing over the NA responses (Figure 2.14, p. 50).

```
NOT$multiplied<-na.pass(NOT$ABC*NOT$XYZ)
```

R will not run some calculations (e.g., finding the mean) unless it has been told to remove the NA cases from its calculations. In order to remove the missing values from the calculations, FIGURE 2.13 ● SCREENSHOTS OF THE NEW DATA FRAME WITHOUT THE NA CASES



	ABC	XYZ
1	1	7
2	2	2
3	4	4
4	6	8
6	8	1
7	4	2
9	4	6
10	2	3
11	1	4
12	8	9
13	0	3
14	5	2
16	6	0

```
Data
GSS2016   2867 obs. of 209 variables
newOMIT   13 obs. of 2 variables
NOT       16 obs. of 2 variables
```

use the `na.rm = TRUE` script. Let's try it both ways to compare the different output. It is clear that without removing the NA cases, R will not calculate the mean properly. The **`na.rm`** command can be used in other calculations as well.

```
mean(NOT$ABC)
mean(NOT$ABC, na.rm = TRUE)
```

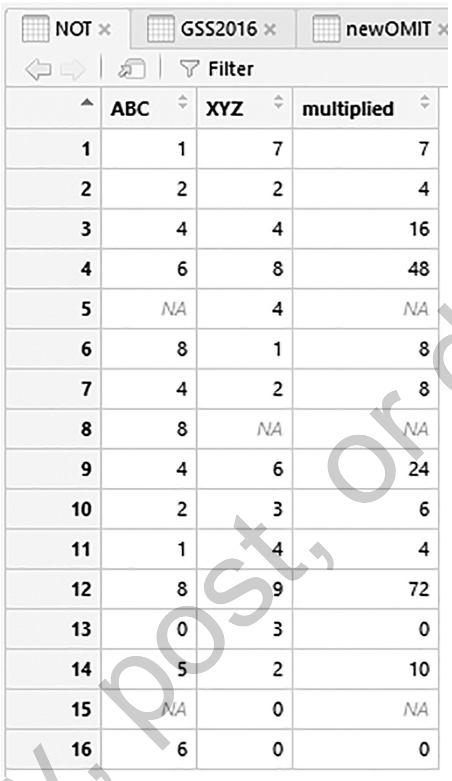
Outputs: Mean of the ABC variable with and without the NA cases.

```
> mean(NOT$ABC)
[1] NA
> mean(NOT$ABC, na.rm = TRUE)
[1] 4.214286
```

Imputation

If you are interested in imputation or inputting a value in place of the NA instead of removing the NA, that can be done, too. For instance, if you have the mean or median of the variable you can impute that instead of the missing value. Let's try imputation with the `multiplied` variable. There are three different ways to go about getting the mean and median. The first is by removing the NA values; the second is by calling only the data that is not an NA value, using the "is not NA" command (**!is.na**); and the third is by using the package **`pastecs`**.

FIGURE 2.14 ● SCREENSHOT OF THE NEW VARIABLE COMPUTED (`multiplied`) USING THE NA.PASS



	ABC	XYZ	multiplied
1	1	7	7
2	2	2	4
3	4	4	16
4	6	8	48
5	NA	4	NA
6	8	1	8
7	4	2	8
8	8	NA	NA
9	4	6	24
10	2	3	6
11	1	4	4
12	8	9	72
13	0	3	0
14	5	2	10
15	NA	0	NA
16	6	0	0

```
mean(NOT$multiplied, na.rm = TRUE)
median(NOT$multiplied, na.rm = TRUE)
sd(NOT$multiplied, na.rm = TRUE) # Whenever you are obtaining a
mean value, it is always a good idea to get the standard deviation as well.
```

Outputs: Mean, median, and standard deviation of the `multiplied` variable using `na.rm`.

```
> mean(NOT$multiplied, na.rm = TRUE)
[1] 15.92308
> median(NOT$multiplied, na.rm = TRUE)
[1] 8
> sd(NOT$multiplied, na.rm = TRUE)
[1] 21.15207
```

```
mean(NOT$multiplied[!is.na(NOT$multiplied)])
median(NOT$multiplied[!is.na(NOT$multiplied)])
sd(NOT$multiplied[!is.na(NOT$multiplied)])
```

Outputs: Mean, median, and standard deviation of the `multiplied` variable using `!is.na`.

```
> mean(NOT$multiplied[!is.na(NOT$multiplied)])
[1] 15.92308
> median(NOT$multiplied[!is.na(NOT$multiplied)])
[1] 8
> sd(NOT$multiplied[!is.na(NOT$multiplied)])
[1] 21.15207
```

And the third is by using a command from the **pastecs** library: **`stat.desc`**.

```
library(pastecs) # You need to make sure you opened the package
and loaded the library for "pastecs" (this can take a minute or two).
stat.desc(NOT$multiplied)
```

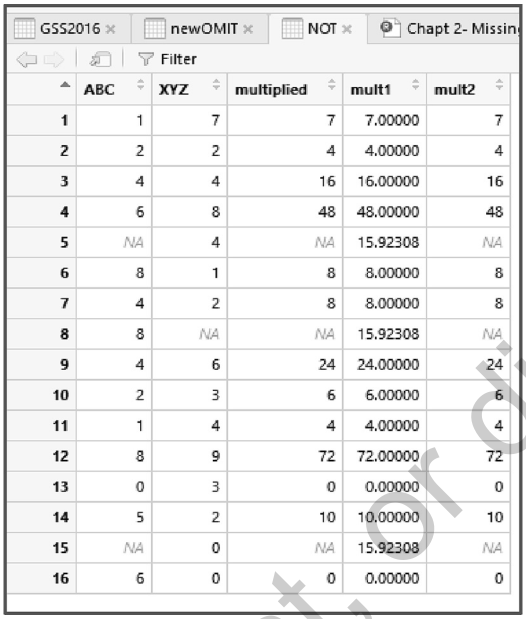
Outputs: Descriptive statistics for the multiplied variable using stat.desc.

```
 nbr.val      nbr.null     nbr.na        min          max
13.000000    2.000000    3.000000     0.000000    72.000000

range          sum        median        mean       SE.mean
72.000000   207.000000   8.000000    15.923077    5.866530

CI.mean.0.95        var        std.dev      coef.var
12.782071      447.410256    21.152075    1.328391
```

After obtaining the mean and median values for the variable, it is best to create a new second variable to work with. In this book, we have created two new variables `mult1` and `mult2` so that we can perform a mean imputation and a median imputation.

```
NOT$mult1<-NOT$multiplied
NOT$mult2<-NOT$multiplied

NOT$mult1[is.na(NOT$mult1)]<-mean(NOT$multiplied[!is.
na(NOT$multiplied)]) # Mean imputation
```

Output: The mean imputation for `mult1` using the`!is.na` command.



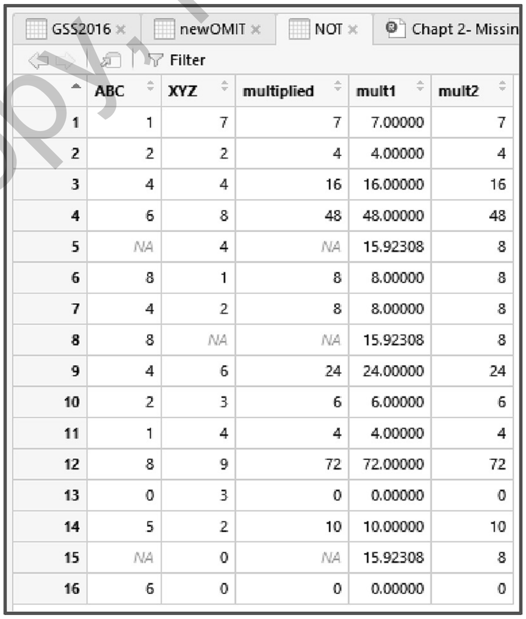
GSS2016 × | newOMIT × | NOT × | Chapt 2- Missin

Filter

	ABC	XYZ	multiplied	mult1	mult2
1	1	7	7	7.00000	7
2	2	2	4	4.00000	4
3	4	4	16	16.00000	16
4	6	8	48	48.00000	48
5	NA	4	NA	15.92308	NA
6	8	1	8	8.00000	8
7	4	2	8	8.00000	8
8	8	NA	NA	15.92308	NA
9	4	6	24	24.00000	24
10	2	3	6	6.00000	6
11	1	4	4	4.00000	4
12	8	9	72	72.00000	72
13	0	3	0	0.00000	0
14	5	2	10	10.00000	10
15	NA	0	NA	15.92308	NA
16	6	0	0	0.00000	0

```
NOT$mult2[is.na(NOT$mult2)]<-median(NOT$multiplied, na.rm =
TRUE)  # Median imputation
```

Output: The median imputation for `mult2` using the **`na.rm`** command.



GSS2016 × | newOMIT × | NOT × | Chapt 2- Missin

Filter

	ABC	XYZ	multiplied	mult1	mult2
1	1	7	7	7.00000	7
2	2	2	4	4.00000	4
3	4	4	16	16.00000	16
4	6	8	48	48.00000	48
5	NA	4	NA	15.92308	8
6	8	1	8	8.00000	8
7	4	2	8	8.00000	8
8	8	NA	NA	15.92308	8
9	4	6	24	24.00000	24
10	2	3	6	6.00000	6
11	1	4	4	4.00000	4
12	8	9	72	72.00000	72
13	0	3	0	0.00000	0
14	5	2	10	10.00000	10
15	NA	0	NA	15.92308	8
16	6	0	0	0.00000	0

```
stat.desc(NOT$mult1)
```

Outputs: Descriptive statistics for `mult1` after mean imputation.

```
  nbr.val      nbr.null      nbr.na        min          max
16.000000     2.000000     0.000000     0.000000     72.000000

  range         sum        median        mean       SE.mean
72.000000   254.769231    9.000000    15.923077     4.729748

CI.mean.0.95          var        std.dev        coef.var
10.081218         357.928205    18.918991       1.188149
```

If we compare the two descriptive statistics from before and after the mean imputation, we see that the number of observations has increased from 13 to 16, which makes sense because we imputed values where there used to be NAs for three cases. That means three additional observations are now in the data. The mean has remained the same as before the imputation because we used the mean as the value for imputation.

COMPUTING VARIABLES

Computing variables is the process in which we use (an) existing variable(s) with a mathematical operation that result(s) in a new variable.

For example, if we wanted to create a measure of the total years of education for a participant's mother (`maeduc`) and father (`paeduc`) combined, we would compute a new variable with this information. The first thing that needs to be done is to recode the data to remove the "not applicable" (97), "don't know" (98), "no answer" (99) cases for both variables.

```
table(GSS2016$maeduc)
```

Output: Frequencies of responses for `maeduc` before coding for NA.

```
   0    1    2    3    4    5    6    7    8    9    10   11
  59    5   11   27   16   32   77   29  182   61    98   86

  12   13   14   15   16   17   18   19   20   97    98   99
1055  100  236   53  302   20   89   10   33  101   180    5
```

```
table(GSS2016$paeduc)
```

Output: Frequencies of responses for `paeduc` before coding for NA.

```
  0    1    2    3    4    5   6    7    8    9   10   11   12   13   14   15
 43    8   11   40   26   24   80   32  157   60   74   65  774   72  147   35

 16    17   18   19   20   97   98   99
274   28   67   16   59   550  221   4
```

```
GSS2016$maeduc[GSS2016$maeduc==97]=NA
GSS2016$maeduc[GSS2016$maeduc==98]=NA
GSS2016$maeduc[GSS2016$maeduc==99]=NA

GSS2016$paeduc[GSS2016$paeduc==97]=NA
GSS2016$paeduc[GSS2016$paeduc==98]=NA
GSS2016$paeduc[GSS2016$paeduc==99]=NA

table(GSS2016$maeduc)
```

Output: Frequencies of responses for `maeduc` after coding for NA.

```
  0    1    2    3    4    5    6    7    8    9   10   11   12   13
 59    5    11   27   16   32   77   29  182   61   98   86  1055  100

 14    15   16   17   18   19   20
236    53   302  20   89   10   33
```

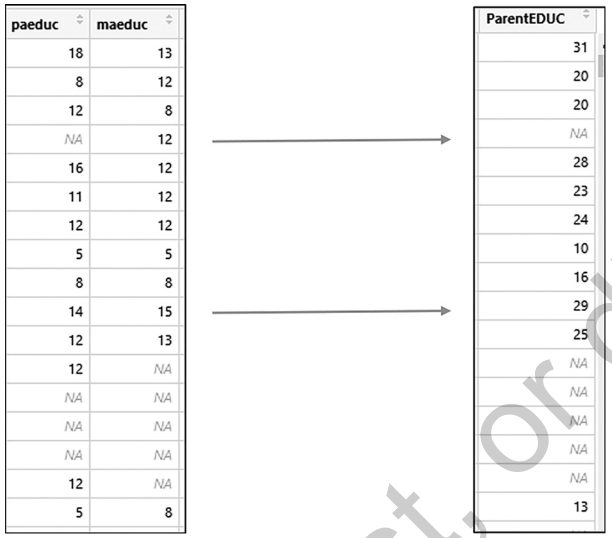
```
table(GSS2016$paeduc)
```

Output: Frequencies of responses for **`paeduc`** after coding for NA.

```
  0  1   2   3   4   5   6   7   8    9  10  11  12   13  14   15  16
 43  8   11  40  26  24  80  32  157  60  74  65  774  72  147  35  274

 17  18  19  20
 28  67  16  59
```

Next, we simultaneously create a new variable for parent education (`parentEDUC`) and compute the data for the variable by adding the data from highest year of school completed by the mother (`maeduc`) to the data from highest year of school completed by the father (`paeduc`) (Figure 2.15). Because both variables have a range of 20 with 0 as the minimum and 20 as the maximum, it is anticipated that the newly computed variable range cannot exceed 40 with a minimum of 0 and a maximum of 40. We can follow the computation of the variable with a frequency table to see if that is true.

FIGURE 2.15 ● SCREENSHOTS OF THE INDIVIDUAL VARIABLES (maeduc AND paeduc) THAT WERE ADDED TOGETHER TO COMPUTE THE NEW VARIABLE (ParentEDUC)



paeduc	maeduc
18	13
8	12
12	8
NA	12
16	12
11	12
12	12
5	5
8	8
14	15
12	13
12	NA
NA	NA
NA	NA
NA	NA
12	NA
5	8

ParentEDUC
31
20
20
NA
28
23
24
10
16
29
25
NA
NA
NA
NA
NA
13

paeduc + maeduc = ParentEDUC

```
GSS2016$ParentEDUC<-(GSS2016$paeduc+GSS2016$maeduc)
table(GSS2016$ParentEDUC)
```

Output: Frequencies of responses for combined highest years of education (parentEDUC).

```
0   1   2   3   4   5   6    7   8   9   10  11   12  13  14  15  16
23  2   6   8   5   3   15   4   13  10  12  13   42   9  31  18  68

17  18  19  20  21  22  23   24  25  26   27  28   29  30  31  32  33
24  44  24  87  45  82  61  464  75  141  53  157  42  92  18  115  22

34  35  36  37  38  39  40
43   7  39  10  15  3   8
```

REMOVING OUTLIERS

An outlier is an extreme or unusual value within a variable which typically exists in continuous (interval/ratio) data. The value at which a datapoint is considered an outlier is determined by the spread of the data for the variable and the researcher's choice on what they consider to be extreme. Seeking to find if you have outliers in your data can often uncover mistakes. For example, if we have a variable that has 5000 participants' ages and 4999 of them range between 18 and 97 but one is 333, logic suggests that this outlier is probably a mistake and we can mark it as missing. For the purpose of this section, we will use common practices to determine outliers for our variables.

Outliers have the ability to strongly influence the results of statistical analysis. For example, if a researcher wanted to know the average price of a home in a neighborhood with 10 homes and all 10 homes were somewhere between $250,000 and $350,000, they could expect the average to be somewhere between $250,000 and $350,000. But what if one house in the neighborhood was a newly built home worth $1.3 million? Or, what if one home had a kitchen fire and the owners abandoned it and let it become dilapidated and run down, so its value is low at $83,000? Could either of those unusual house prices influence the average cost of homes in the neighborhood enough to warrant a closer look at the data?

Although the scenarios above may seem exaggerated, it is indeed possible for an extreme value to skew an entire sample of observations. If we create a spreadsheet for the above three scenarios (Table 2.5), Scenario 2 (having one high outlier) increased the average home price by just over $100,000. Scenario 3, with the low-priced outlier, pulled down the average value of houses in the neighborhood by more than $20,000. However, if the outliers were not included in the calculation the average for the other nine homes was only roughly an $800 difference.

TABLE 2.5 ● AVERAGE HOME VALUE FOR THREE NEIGHBORHOOD SCENARIOS

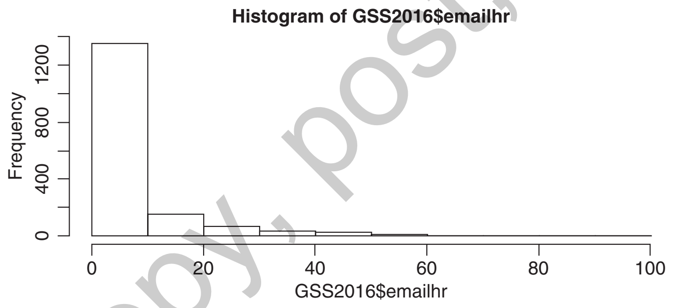
	Scenario 1	Scenario 2	Scenario 3
Home 1	256000	256000	256000
Home 2	342000	342000	342000
Home 3	267000	267000	267000
Home 4	296000	*1302000*	*83000*
Home 5	308000	308000	308000
Home 6	344000	344000	344000
Home 7	318000	318000	318000
Home 8	290000	290000	290000
Home 9	333000	333000	333000
Home 10	285000	285000	285000
Average (10)	**303900**	**404500**	**282600**
Average (9)		**304777**	**304777**

First, we need to determine if there are any outliers in the data for this variable. One common way to determine if a variable has any outliers is to create a boxplot or histogram figure representation of the variable which can help in determining which datapoints are visually unusual or extreme (these are discussed in more detail in Chapter 5). A frequency table can also be run to determine if there are data values that are extreme. After determining that there are outliers, we decide how to deal with them (e.g., whether we need to remove them or transform the variable measurement in some way).

If we determine that the outliers should be removed, the next step is to create a new variable where we will recode the outliers. For this example, we are going to use the variable `emailhr` in the GSS data. This variable is made up of responses to the question *about how many hours per week do you spend sending and answering electronic mail or e-mail?* First, we will remove the No answer, Don't know, and Not applicable responses by recoding them to NA.

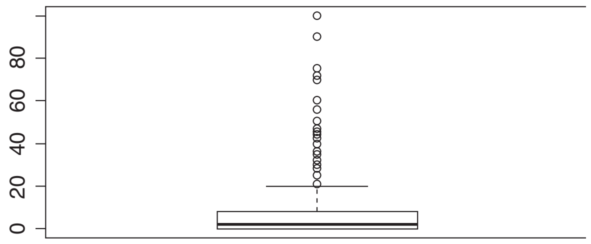
```
GSS2016$emailhr[GSS2016$emailhr=='-1']=NA
GSS2016$emailhr[GSS2016$emailhr==998]=NA
GSS2016$emailhr[GSS2016$emailhr==999]=NA

hist(GSS2016$emailhr)
```

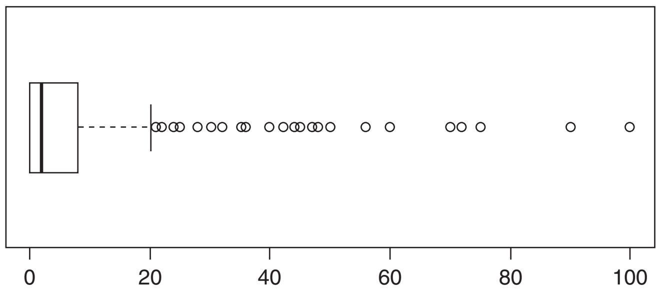
Output: Histogram for `emailhr` before removing outliers.



```
boxplot(GSS2016$emailhr, horizontal = FALSE)
boxplot(GSS2016$emailhr, horizontal = TRUE) # Horizontal =
FALSE will produce a vertical boxplot, while Horizontal = TRUE will
produce a horizontal boxplot.
```

Output: Vertical and horizontal boxplots for `emailhr` before removing outliers.





```
boxplot.stats(GSS2016$emailhr)
```

Output: Boxplot Statistics for `emailhr` before removing outliers.

```
$stats
[1]  0  0  2  8  20
# Lower whisker, lower hinge, median, upper hinge, upper whisker.

$n
[1]  1649
# Number of observations (not including NA) in the variable.

$conf
[1]  1.68873  2.31127
# Lower and upper extreme of the notch.

$out
   [1]  40  25  60  45  25  60  44  40  60  50  40  30  50  40  30
  [16]  25  42  90  75  25  22  50  50  40  25  24  32  21  30  30
  [31]  22  30  40  30  50  28  25  56  35  45  40  60  30  60  32
  [46]  30  30  22  21  45  22  70  50  30  40  50  25  25  30  30
  [61]  40  40  35  30  30  47  40  30  40  40  25  40  40  25  35
  [76]  25  35  70  48  28  25  40  30  30  40  40  50  50  25  30
  [91]  30  25  40  30  60  50  25  35  45  35  40  25  30  30  40
 [106]  25  30  40  22  30  35  25  35  36  25  50  25  72  30  21
 [121]  30  21  40  30  30  50  25  60  100  21  25  48  30  25  42
 [136]  50  30  25  30  30  40  30  40
# The suspected outliers—fall beyond the whiskers.
```

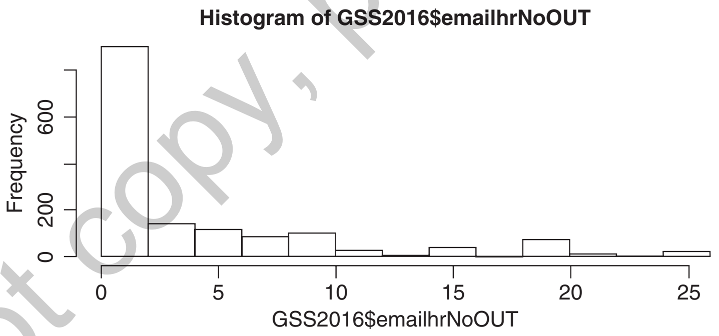
```
table(GSS2016$emailhr)
```

Output: Frequency table for `emailhr` before removing outliers

```
  0    1    2   3   4    5   6   7    8   9   10  12  14  15  16  17  18
416  306  183  77  62   92  27  44   42   4  100  27   9  36   3   1   1


 20   21   22  24  25   28  30  32   35  36   40  42  44  45  47  48  50
 76    5    5   1  24    2  33   2    8   1   25   2   1   4   1   2  13


 56   60   70  72  75   90 100
  1    7    2   1   1    1   1
```

`dataframe$NewVariable<-dataframe$OldVariable[set level or parameter]`
`GSS2016$emailhrNoOUT<-GSS2016$emailhr` # Creating the new variable emailhrNoOUT from the original variable emailhr.
`GSS2016$emailhrNoOUT[GSS2016$emailhrNoOUT>26]= NA` # Note that we chose to remove values larger than 26, instead of exact values.
`hist(GSS2016$emailhrNoOUT)` # A histogram of the new variable to compare to the original variable histogram.

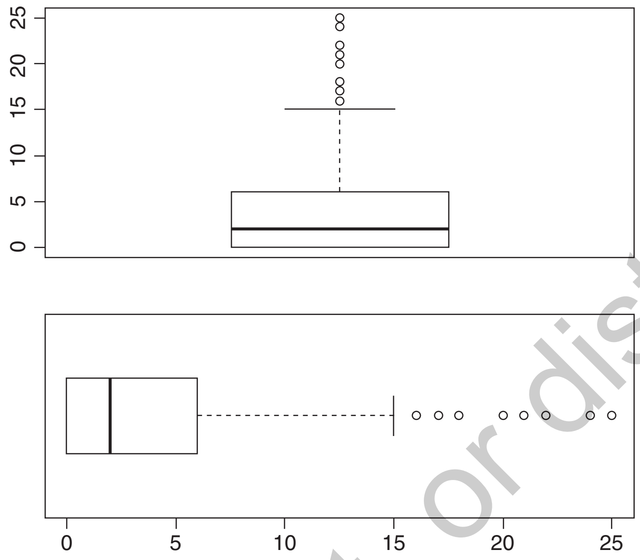
Output: Histogram for `emailhrNoOUT` after removing outliers from `emailhr`.



This second histogram gives a much more accurate depiction of the data spread for the majority of the responses.

`boxplot(GSS2016$emailhrNoOUT, horizontal = F)` # Note that the uppercase T or F can be used instead of writing out TRUE or FALSE.
`boxplot(GSS2016$emailhrNoOUT, horizontal = T)`

Output: Vertical and horizontal boxplots for `emailhrNoOUT` after removing outliers from `emailhr`.



```
boxplot.stats(GSS2016$emailhrNoOUT)
```

Output: Boxplot statistics for `emailhrNoOUT` after removing outliers.

```
$stats
[1]  0  0  2  6  15

$n
[1]  1541
$conf
[1]  1.758506  2.241494

$out
   [1]  25  20  25  20  20  20  20  25  20  20  25  20  22  20  20  25  24  21
        20  22  20
  [22]  20  25  20  20  20  20  20  20  22  21  20  20  22  20  20  20  20  20
        20  20  20
  [43]  25  25  20  20  20  20  17  20  25  20  25  20  20  20  25  20  20  25  20
        20  20
  [64]  20  20  16  20  20  25  25  20  20  25  25  20  20  20  25  20  22  20
        20  25  16
  [85]  20  20  25  20  20  25  20  20  20  21  20  21  20  25  20  20  20  21
        25  20  20
[106]  20  25  20  25  16  18  20  20  20  20  20
```

Tip: Keep in mind that when we remove outliers, our variable measure of central tendency and measure of variability will change. Then, there can be new outliers. So, be attentive when deciding what you choose to remove and why.

```
table(GSS2016$emailhrNoOUT)
```

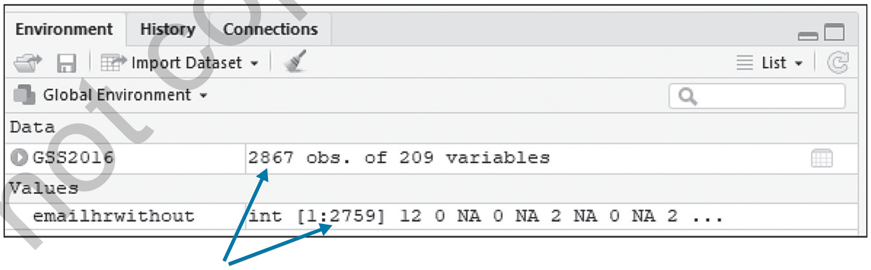
Output: Frequency table for `emailhrNoOUT` after removing outliers from email.

```
  0   1   2   3  4  5  6  7  8 9  10  12 14 15 16 17 18
416 306 183  77 62 92 27 44 42 4 100  27  9 36  3  1  1

 20  21 22  24  25
 76   5  5   1  24
```

If we did not want to make a new variable of the `emailhr` without the outliers, we could also create a new index of the `emailhr` without the outliers (Figure 2.16). This can be useful for immediate analysis of an individual variable, but will not be as helpful when conducting more advanced statistical analyses. The index cannot be saved "within" the dataset because it will have fewer rows. When the number of rows does not match, R cannot join them. All of the same univariate analyses discussed in later chapters, such as frequency tables, histograms, boxplots, and descriptive statistics can be run using this variable that we have created "outside" of the dataset as a separate object.

```
Emailhrwithout26<-(GSS2016$emailhr[GSS2016$emailhr <26])
```

FIGURE 2.16 ● SCREENSHOT OF NEW INDEX **emailhrwithout** AFTER REMOVING OUTLIERS



```
Data
GSS2016          2867 obs. of 209 variables
Values
emailhrwithout   int [1:2759] 12 0 NA 0 NA 2 NA 0 NA 2 ...
```

Looking at the environment window, we can observe a difference in observation numbers (rows). There are 108 responses that have been removed from the data. Those are the 108 outlier responses that were over 26 hours. A second way to deal with outliers is to "top code" the data and impute a "highest value". For example, a researcher might top code `emailhr` at 25 by imputing all values of 26 and over to 25 and referring to the top code simply as "25 or more hours".

Try the process of selecting out portions of the data. Remove or impute any responses above 15 hours. Then, create a histogram and table to double-check.

```
GSS2016$emailhrwithout15<-(GSS2016$emailhr)
GSS2016$emailhrwithout15[GSS2016$emailhr > 15]=NA

hist(GSS2016$emailhrwithout15)
table(GSS2016$emailhrwithout15)

boxplot(GSS2016$emailhrwithout15, horizontal = F)
boxplot(GSS2016$emailhrwithout15, horizontal = T)

boxplot.stats(GSS2016$emailhrwithout15)
Emailhrwithout15<-(GSS2016$emailhr[GSS2016$emailhr <15])
```

A third way to deal with outliers is to "transform" the data so that the outliers will not be lost, but the transformation can help get the data closer to a normal distribution. This is often done when the data are very skewed. Data can be transformed by performing a mathematical computation on the original data. The most common transformations involve polynomials and log transformations. See Chapter 12 to learn how to transform data in RStudio.

CONCLUSION

This chapter offered a review of types of datasets (long vs. wide structures) and levels of variable measurement. We introduced some background of the GSS dataset and codebook, which will be used throughout the rest of the book. We discussed some common procedures in R/RStudio for modifying data, including recoding variables and computing a single value based on responses across multiple variables. We also focused on handling missing data. Other, more advanced techniques for handling of missing data are beyond the scope of this chapter but are discussed more in Chapter 12.

References

Smith, T. W., Davern, M., Freese, J., & Morgan, S. L. (2019). *General Social Surveys, 1972–2018* [Machine-Readable Data File]. Chicago, IL: NORC at the University of Chicago. Retrieved from http://www.gss.norc.org/getthedata/Pages/Home.aspx

Exercises

1. Use the variable `grass` from the GSS2016 dataset.
   A. Identify what the variable represents.
   B. Run a table of the variable to get the frequencies for each of the attributes.
   C. Create a new variable `grassrecode` from the existing variable.
   D. Recode new variable to remove all the Don't knows, No answers, and Not applicable responses.
   E. Run an updated table of the variable.
2. Open an Excel spreadsheet. Use the picture below to create a csv.file and save as "sample1.csv."

Name	Age	Gender	Height
Daniel	25	M	72
Cammi	24	F	61
Tammy	35	F	68
Angela	33	F	64
Shannon	30	F	0
Ben	NA	M	74
Bill	42	M	69
Elizabeth	56	F	66
Susan	28	F	62
Rick	39	M	72
Becky	63	F	67

3. Now practice bringing the sample1 dataset into R using the **read.csv** command.
   A. First, set your working directory.
   B. Next, upload the newly created .csv file.
   C. Check the data by clicking on it in the environment window.
   **Note the missing data for Ben's age and the 0 for Shannon's height.
2. For the variable `age`, fill in (impute) the missing data with the mean of the variable.
   A. Calculate the mean of the variable.
   B. Then fill in the data using the **impute** command.

   C. Now check your data again to see if the missing data has been filled in. You should see the mean for the age of Ben.

5. Create a new variable named `height1` from the existing variable of height.

6. Remove the 0 response from your data by recoding it to be missing data, or NA.

   A. Calculate the median of the variable `height1`.

   B. Fill in the missing data using the **impute** command.

   C. Now check your data to make sure that the height for Shannon has been filled in using the median for the height variable.

Supplementary Digital Content

Download datasets and R code at the companion website at https://study.sagepub.com/researchmethods/statistics/gillespie-r-for-statistics