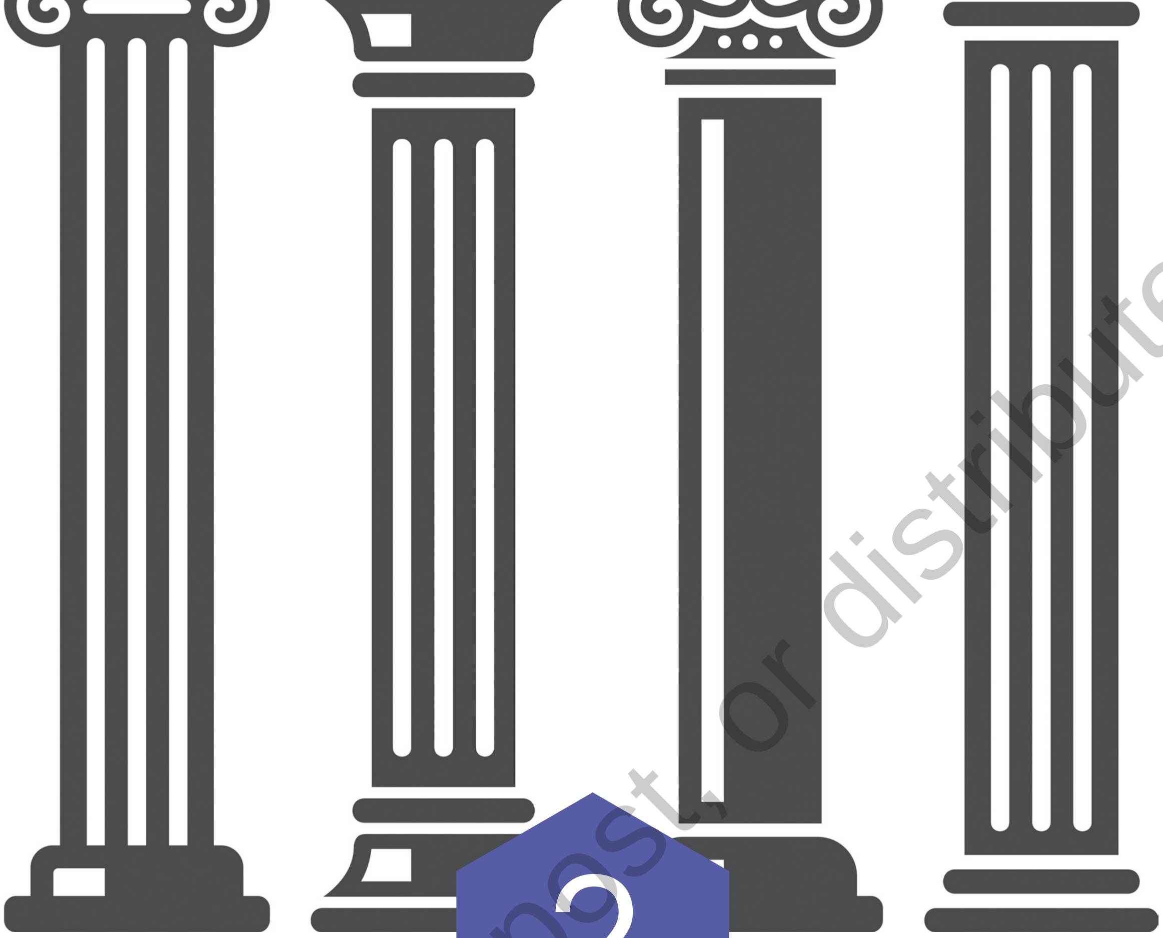

2 ROWS AND COLUMNS

LEARNING OBJECTIVES

Explain what a dataframe is and how data are organized in a dataframe.

Create and use dataframes in R.

Access rows and columns in a dataframe.

Gain experience using the following R functions: c, data.frame, str, summary, head, tail.

Although we live in a three-dimensional world, where a box of cereal has height, width, and depth, it is a sad fact of modern life that pieces of paper, chalkboards, whiteboards, and computer screens are still only two dimensional. As a result, most of the statisticians, accountants, computer scientists, and engineers who work with lots of numbers tend to organize them in rows and columns. There's really no good reason for this other than that it makes it easy to fill a rectangular display with numbers. Rows and columns can be organized any way that you want, but the most common way is to have the rows be cases or instances, and the columns be attributes or variables. Take a look at this nice, two-dimensional representation of rows and columns in Table 2.1:

TABLE 2.1 ● MYFAMILY DATAFRAME

Name	Age	Gender	Weight
Dad	43	Male	188
Mom	42	Female	136
Sis	12	Female	83
Bro	8	Male	61
Dog	5	Female	44

Pretty obvious what's going on, right? The top line, in bold, is not really part of the data. Instead, the top line contains the attribute or variable names. Note that computer scientists tend to call them attributes, whereas statisticians call them variables. Either term is OK. For example, age is an attribute that every living thing has, and you could count it in minutes, hours, days, months, years, or other units of time. Here we have the age attribute calibrated in years. Technically speaking, the variable names in the top line are metadata, or what you could think of as data about data. Imagine how much more difficult it would be to understand what was going on in that table without the metadata. There's lot of different kinds of metadata: variable names are just one simple type of metadata.

So if you ignore the top row, which contains the variable names, each of the remaining rows is an instance or a case. Again, computer scientists might call them instances, and statisticians might call them cases, but either term is fine. The important thing is that each row refers to an actual thing. In this case, all of our things are living creatures in a family. You could think of the Name column as case labels, in that each of these labels refers to one and only one row in our data. Most of the time when you are working with a large dataset, there is a number used for the case label, and that number is unique for each case (i.e., the same number would never appear in more than one row). Computer scientists sometimes refer to this column of unique numbers as a key. A key is very useful, particularly for matching things up from different data sources, and we will run into this idea again later. For now, though, just take note that the Dad row can be distinguished from the Bro row, even though they are both Males. Even if we added an Uncle row that had the same Age, Gender, and Weight as Dad, we would still be able to tell the two rows apart because one would have the name Dad and the other would have the name Uncle.

One other important note: look at how each column contains the same kind of data all the way down. For example, the Age column is all numbers. There's nothing in the Age column like Old or Young. This is a really valuable way of keeping things organized. After all, we could not run the mean() function on the Age column if it contained a little piece of text, like Old or Young. On a related note, every cell (i.e., an intersection of a row and a column, such as Sis's Age) contains just one piece of information. Although a spreadsheet or a word processing program might allow us to put more than one thing in a cell, a real data handling program will not. Finally, see that every column has the same number of entries, so that the whole forms a nice rectangle. When statisticians and other people who work with databases work with a data set, they expect this rectangular arrangement.

CREATING DATAFRAMES

Now let's figure out how to get these rows and columns into R. One thing you will quickly learn about R is that there is almost always more than one way to accomplish a goal. Sometimes the quickest or most efficient way is not the easiest to understand. In this case, we will build each column one by one and then join them together. This is somewhat labor intensive, and not the usual way that we would work with a dataset, but it is easy to understand. First, we run this command to make the column of names:

```
myFamilyNames <- c("Dad","Mom","Sis","Bro","Dog")
```

One thing you might notice is that every name is placed within double quotes. This is how you signal to R that you want it to treat something as a string of characters rather than the name of a storage location. If we had asked R to use Dad instead of "Dad" it would have looked for a storage location (a data object or variable) named Dad. Another thing to notice is that the commas separating the different values are outside of the double quotes. If you were writing a regular sentence this is not how things would look, but for computer programming the comma can only do its job of separating the different values if it is not included inside the quotes. Once you have typed the line above, remember that you can check the contents of myFamilyNames by typing it on the next command line:

```
myFamilyNames
```

The output should look like this:

```
[1] "Dad" "Mom" "Sis" "Bro" "Dog"
```

Next, you can create a vector of the ages of the family members, like this:

```
myFamilyAges <- c(43, 42, 12, 8, 5)
```

Note that this is exactly the same command we used in Chapter 1, so if you have kept R running between then and now you would not even have to retype this command because myFamilyAges would still be there. Actually, if you closed R since working the examples from Chapter 1, you will have been prompted to save the workspace. If you did so, then R restored all of the data objects you were using in the last session. You can always check by typing myFamilyAges on a blank command line. The output should look like this:

```
[1]  43  42  12  8  5
```

Hey, now you have used the c() function and the assignment arrow to make myFamilyNames and myFamilyAges. If you look at the data table earlier in the chapter, you should be able to figure out the commands for creating myFamilyGenders and myFamilyWeights. In case you run into trouble, these commands also appear soon, but you should try to figure them out for yourself. After you type the command to create the new data object, you should also type the name of the data object at the command line to make sure that it looks the way it should. There are four variables, each with five values in it. Two of the variables are character data and the other two are integer data.

Before we show you the R code to create myFamilyGenders and myFamilyWeights, let's explore myFamilyAges some more. We now know that myFamilyAges is a variable, and that is a vector, which means it is a list of numbers. We can access each number individually, using square brackets. For example, if we want to output just the second element in myFamilyAges, we could do the following:

```
myFamilyAges[2]
[1]  42
```

Here are those two extra commands, to define myFamilyGenders and myFamilyWeights in case you need them:

```
myFamilyGenders <-
        c("Male","Female","Female","Male","Female")

myFamilyWeights <- c(188,136,83,61,44)
```

Look out! We're starting to get commands that are long enough that they break onto more than one line. In your code window you can type the code just as you see it above.

R knows what to do when it gets to the end of a line, but the command is not yet finished. If you type the first line above at the console and press enter, R will respond with a + to show that it is still expecting more input—you can type the second line and press enter again to run the whole command.

Now we are ready to tackle the dataframe. In R, a dataframe is a list (of columns), where each element in the list is a vector. Each vector is the same length, which is how we get our nice rectangular row and column setup, and generally each vector also has its own name. The command to make a dataframe is very simple:

```
myFamily <- data.frame(myFamilyNames, myFamilyAges,
        myFamilyGenders, myFamilyWeights)
```

The data.frame() function makes a dataframe from the four vectors that we listed. Notice that we have also used the assignment arrow to make a new stored location where R puts the dataframe. This new data object, called myFamily, is our dataframe. Once you have gotten that command to work, type myFamily at the command line to get a report back of what the dataframe contains.

```
myFamily
```

Here's the output you should see:

```
  myFamilyNames myFamilyAges myFamilyGenders
1           Dad           43            Male
2           Mom           42          Female
3           Sis           12          Female
4           Bro            8            Male
5           Dog            5          Female

  myFamilyWeights
1            188
2            136
3             83
4             61
5             44
```

This looks great. Notice that R has put row numbers in front of each row of our data. These are different from the output line numbers we saw in square brackets before, because these are actual indices into the dataframe. Later on, we will use row numbers like these to extract single rows or groups of rows from dataframes.

EXPLORING DATAFRAMES

With a small data set like this one, only five rows, it is pretty easy just to take a look at all of the data. But when we get to a bigger data set, this won't be practical. We need to have other ways of summarizing what we have. The first method reveals the type of structure that R has used to store a data object.

```
str(myFamily)

'data.frame': 5 obs. of 4 variables:
$ myFamilyNames : Factor w/ 5 levels
                "Bro","Dad","Dog",..: 2 4 5 1 3
$ myFamilyAges : num 43 42 12 8 5
$ myFamilyGenders: Factor w/ 2 levels
               "Female","Male": 2 1 1 2 1
$ myFamilyWeights: num 188 136 83 61 44
```

OK, so the function str() reveals the structure of the data object that you name between the parentheses. In this case, we pretty well knew that myFamily was a dataframe because we just set that up in a previous command. In the future, however, we will run into many situations where we are not sure how R has created a data object, so it is important to know str() so that you can ask R to report what an object is at any time.

In the first line of output, we have the confirmation that myFamily is a dataframe as well as an indication that there are five observations (obs., which is another word that statisticians use instead of cases or instances) and four variables. After that first line of output, we have four sections that each begin with $. For each of the four variables, these sections describe the component columns of the myFamily dataframe object.

Each of the four variables has a mode or type that is reported by R right after the colon on the line that names the variable:

```
$ myFamilyGenders: Factor w/ 2 levels
```

For example, myFamilyGenders is shown as Factor. In the terminology that R uses, Factor refers to a special type of label that can be used to identify and organize groups of cases. R has organized these labels alphabetically and then listed out the first few cases (because our dataframe is so small it actually is showing us all of the cases). For myFamilyGenders we see that there are two levels, meaning that there are two different options: female and male. R assigns a number, starting with 1, to each of these levels, so every case that is Female gets assigned a 1 and every case that is Male gets assigned a 2 (Female comes before Male in the alphabet, so Female is the first Factor label, and gets a 1). If you have your thinking cap on, you might be wondering why we started out by typing in small strings of text, like Male, but then R has gone ahead and converted these small pieces of text into numbers that it calls Factors. The reason for this lies in the statistical origins of R. For years, researchers have done things like calling an experimental group Exp and a control group Ctl without intending to use these small strings of text for anything other than labels. So R assumes, unless you tell it otherwise, that when you type in a short string like Male that you are referring to the label of a group, and that R should prepare for the use of Male as a Level of a Factor. When you don't want this to happen you can instruct R to stop doing this by using the option on the data.frame() function stringsAsFactors=FALSE. We will look with more detail at options and defaults a little later on.

Phew, that was complicated! By contrast, our two numeric variables, myFamilyAges and myFamilyWeights, are very simple. You can see that after the colon the mode is shown as num (which stands for numeric) and that the first few values are reported:

```
$ myFamilyAges : num 43 42 12 8 5
```

Putting it all together, we have pretty complete information about the myFamily dataframe and we are just about ready to do some more work with it. We have seen firsthand that R has some kind of cryptic labels for things as well as some obscure strategies for converting this to that. R was designed for experts, rather than novices, so we will just have to take our lumps so that one day we can be experts, too.

Next, we will examine another very useful function called summary(). The summary command provides some overlapping information to the str command but also goes a little further, particularly with numeric variables. Here's what we get:

```
summary(myFamily)

 myFamilyNames  myFamilyAges
 Bro: 1         Min.     : 5
 Dad: 1         1st Qu.  : 8
 Dog: 1         Median   : 12
 Mom: 1         Mean     : 22
 Sis: 1         3rd Qu.  : 42
                Max.     : 43

 myFamilyGenders   myFamilyWeights
 Female: 3         Min.    : 44
 Male  : 2         1st Qu. : 61.0
                   Median  : 83.0
                   Mean    : 102.4
                   3rd Qu. : 136.0
                   Max     : 188.0
```

In order to fit on the page properly, these columns have been reorganized somewhat. The name of a column/variable sits up above the information that pertains to it, and each block of information is independent of the others (so it is meaningless, for instance, that Bro: 1 and Min. happen to be on the same line of output). Notice, as with str(), that the output is quite different depending on whether we are talking about a factor, like myFamilyNames or myFamilyGenders, versus a numeric variable like myFamilyAges and myFamilyWeights. The columns for the Factors list out a few of the factor names along with the number of occurrences of cases that are coded with that factor. So, for instance, under myFamilyGenders it shows three females and two males. In contrast, for the numeric variables we get five different calculated quantities that help to summarize the variable. There's no time like the present to start to learn about what these are, so here goes:

- Min. refers to the minimum or lowest value among all the cases. For this dataframe, five is the age of Dog and it is the lowest age of all of the family members.

- 1st Qu. refers to the dividing line at the top of the first quartile. If we took all the cases and lined them up side by side in order of age (or weight) we could then divide up the whole into four groups, where each group had the same number of observations. Just like a number line, the smallest cases would be on the left with the largest on the right. If we're looking at myFamilyAges, the leftmost group, which contains one quarter of all the cases, would start with five on the low end (Dog) and would have eight on the high end (Bro). So the first quartile is the value of age (or another variable) that divides the first quarter of the cases from the other three quarters. Note that if we don't have a number of cases that divides evenly by four, the value is an approximation.
- Median refers to the value of the case that splits the whole group in half, with half of the cases having higher values and half having lower values. If you think about it, the median is also the dividing line that separates the second quartile from the third quartile.
- Mean, as we have learned before, is the numeric average of all of the values. For instance, the average age in the family is reported as 22.
- 3rd Qu. is the third quartile. If you remember back to the first quartile and the median, this is the third and final dividing line that splits up all of the cases into four equal-sized parts. You might be wondering about these quartiles and what they are useful for. Statisticians like them because they give a quick sense of the shape of the distribution. Everyone has the experience of sorting and dividing things up—pieces of pizza, playing cards into hands, a bunch of players into teams—and it is easy for most people to visualize four equal-sized groups and useful to know how high you need to go in age or weight (or another variable) to get to the next dividing line between the groups.
- Finally, Max is the maximum value and, as you might expect, displays the highest value among all of the available cases. For example, in this dataframe Dad has the highest weight: 188. Seems like a pretty trim guy.

Wow, that was a lot of info! Taking a step back, in statistics, central tendency is a key concept that is used to explain the center of a probability distribution. The most common measures of central tendency are the mean, median, and mode. Another key concept in statistics is dispersion (also called variability, scatter, or spread), which denotes how stretched or squeezed a distribution is. Common examples of measures of dispersion are the variance, standard deviation, and quartiles. As you just saw, in R we can use the summary function to get the first two measures of central tendency (mean and median) and the quartiles for a measure of dispersion. Mode refers to the most common occurring element in the dataset. We will explore mode, variance, and standard deviation in more detail later in this book.

While both the str and summary functions are very useful, sometimes we just want to look at a couple of rows in the dataframe. Previously, we typed myFamily at the command line and saw all the rows in the dataframe. However, if the dataframe has many rows, a better way is to use head or tail.

```
head(myFamily, 2)
myFamilyNames myFamilyAges myFamilyGenders myFamilyWeights
1         Dad           43            Male             188
2         Mom           42          Female             136

tail(myFamily, 2)
myFamilyNames myFamilyAges myFamilyGenders myFamilyWeights
4         Bro            8            Male              61
5         Dog            5          Female              44
```

You can see in the code that head() lists the first rows in the dataframe and tail() lists the last rows in the dataframe. The number of rows to display is the second argument to both head() and tail(). In our case, we used head() to show the first two rows and tail() to show the last two rows in the myFamily dataframe.

ACCESSING COLUMNS IN A DATAFRAME

Just one more topic to pack in before ending this chapter: how to access the stored variables in our new dataframe? R stores the dataframe as a list of vectors and we can use the name of the dataframe together with the name of a vector to refer to each one using the $ to connect the two labels like this:

```
myFamily$myFamilyAges
[1] 43 42 12 8 5
```

If you're alert, you might wonder why we went to the trouble of typing out that big long thing with the $ in the middle, when we could have just referred to myFamilyAges as we did earlier when we were setting up the data. Well, this is a very important point. When we created the myFamily dataframe, we *copied* all the information from the individual vectors that we had before into a brand-new storage space. So now that we have created the myFamily dataframe, myFamily$myFamilyAges actually refers to a completely separate (but so far identical) vector of values. You can prove this to yourself very easily, and you should, by adding some data to the original vector, myFamilyAges:

```
myFamilyAges <- c(myFamilyAges, 11)
myFamilyAges
[1] 43 42 12 8 5 11

myFamily$myFamilyAges
[1] 43 42 12 8 5
```

Look very closely at the five lines above. In the first line, we use the c() command to add the value 11 to the original list of ages that we had stored in myFamilyAges (perhaps we have adopted an older cat into the family). In the second line, we ask R to report what the vector myFamilyAges now contains. Dutifully, on the third line above, R reports that myFamilyAges now contains the original five values and the new value of 11 on the end of the list. When we ask R to report myFamily$myFamilyAges, however, we still have the original list of five values only. This shows that the dataframe and its component columns/vectors is now a completely independent piece of data. We must be very careful, if we established a dataframe that we want to use for subsequent analysis, that we don't make a mistake and keep using some of the original data from which we assembled the dataframe.

Here's a puzzle that follows on from this question. We have a nice dataframe with five observations and four variables. This is a rectangular dataset, as we discussed at the beginning of the chapter. What if we tried to add on a new piece of data on the end of one of the variables? In other words, what if we tried something like the following command?

```
myFamily$myFamilyAges<-c(myFamily$myFamilyAges, 11)
```

If this worked, we would have a pretty weird situation: the variable in the dataframe that contained the family members' ages would all of a sudden have one more observation than the other variables: no more perfect rectangle! Try it out and see what happens. The error message you will receive shows how R approaches situations like this.

So what new skills and knowledge do we have at this point? Here are a few of the key points from this chapter:

- In R, as in other programs, a vector is a list of elements/things that are all of the same kind, or what R refers to as a mode. For example, a vector of mode numeric would contain only numbers.
- Statisticians, database experts, and others like to work with rectangular datasets where the rows are cases or instances and the columns are variables or attributes.
- In R, one of the typical ways of storing these rectangular structures is in an object known as a dataframe. Technically speaking, a dataframe is a list of vectors where each vector has the exact same number of elements as the others (making a nice rectangle).
- In R, the data.frame() function organizes a set of vectors into a dataframe. A dataframe is a conventional, rectangular data object where each column is a vector of uniform mode and having the same number of elements as the other columns in the dataframe. Data are copied from the original source vectors into a new storage area. The variables/columns of the dataframe can be accessed using $ to connect the name of the dataframe to the name of the variable/column.
- The str() and summary() functions can be used to reveal the structure and contents of a dataframe (as well as of other data objects stored by R). The str() function shows the structure of a data object, whereas summary() provides numerical summaries of numeric variables and overviews of non-numeric variables.
- The head() and tail() functions can be used to reveal the first or last rows in a dataframe.
- A factor is a labeling system often used to organize groups of cases or observations. In R, as well as in many other software programs, a factor is represented internally with a numeric ID number, but factors also typically have

labels such as Male and Female or Experiment and Control. Factors always have levels, and these are the different groups that the factor signifies. For example, if a factor variable called Gender codes all cases as either Male or Female, then that factor has exactly two levels.

- Min and max are often used as abbreviations for minimum and maximum; these are the terms used for the highest and lowest values in a vector. Bonus: the range of a set of numbers is the maximum minus the minimum.
- The mean is the same thing that most people think of as the average. Bonus: the mean and the median are both measures of what statisticians call central tendency.
- Quartiles are a division of a sorted vector into four evenly sized groups. The first quartile contains the lowest-valued elements, for example, the lightest weights, whereas the fourth quartile contains the highest-valued items. Because there are four groups, there are three dividing lines that separate them. The middle dividing line that splits the vector exactly in half is the median. The term "first quartile" often refers to the dividing line to the left of the median that splits up the lower two quarters, and the value of the first quartile is the value of the element of the vector that sits right at that dividing line. Third quartile is the same idea, but to the right of the median and splitting up the two higher quarters. Bonus: quartiles is a measure of dispersion.

CASE STUDY: CALCULATING NPS USING A DATAFRAME

Let's practice working with dataframes by setting up a small number of survey responses. Specifically, six surveys with likelihood to recommend (LTR) values of 9,9,7,6,8,7 and the type of travel also defined as follows: "Business travel", "Business travel", "Business travel", "Mileage tickets", "Personal Travel", "Personal Travel". Given this, is there a difference in net promoter score (NPS), comparing all the survey responses to just the business travel tickets?

In order to do this analysis, we first need to create a dataframe that represents the six surveys. Then, we can calculate and compare the overall NPS, with the NPS value for business travel tickets. Here is the code:

```
# Six surveys
ltr <- c(9,9,7,6,8,7)
TypeOfTravel <- c("Business travel",
     "Business travel", "Business travel", "Mileage",
     "Personal Travel", "Personal Travel")
survey <- data.frame(ltr, TypeOfTravel)

# look at the newly created dataframe
str(survey)

# Calculate number of promoters and detractors
numP <- sum(survey$ltr > 8)
numD <- sum(survey$ltr < 7)

# Now calculate NPS
total <- nrow(survey)
nps <- (numP/total - numD/total) * 100
nps

# do same analysis, but for the business travel tickets
busTravelDF <-
       survey[survey$TypeOfTravel=="Business travel",]

# Calculate number of promoters and detractors
numP <- sum(busTravelDF$ltr > 8)
numD <- sum(busTravelDF$ltr < 7)

# calculate NPS
total <- nrow(busTravelDF)
bus.nps <- (numP/total - numD/total) * 100
bus.nps
```

One thing to see is that, when we created the dataframe, R turned the type of travel into a factor. The code below shows this explicitly, and also shows the specific levels for the factor. A factor is a categorical variable with a defined set of choices. R stores a list of the categories and makes sure that every element of a factor variable fits one of those defined categories.

```
str(survey$TypeOfTravel)
 Factor w/ 3 levels "Business travel",..: 1 1 1 2 3 3

levels(survey$TypeOfTravel)
[1] "Business travel" "Mileage" "Personal Travel"
```

Finally, as we can see from the output (below), with this dataset, people flying with business class tickets have a higher NPS, as compared to all their passengers. Note that we would need to talk to SMEs to understand if this was expected, and if so, why.

```
nps
[1] 16.66667

bus.nps
[1] 66.66667
```

Chapter Challenges

1. Use the c() command to create a new variable containing the favorite food of each family member. For example, your list could contain the entry "Pizza." Make sure that your new variable includes exactly five values. Call the new variable myFoods. Use str() on your new variable to show what kind of variable it is.
2. Add your new variable to the myFamily dataframe. If you were running the code while reading this chapter, you will have myFamilyNames, myFamilyAges, myFamilyGenders, and myFamilyWeights already available. Otherwise, you will need to type in the data for those variables as shown in this chapter.
3. Rerun the summary() function on myFamily to get descriptive information on all of the variables including your new variable. Take note of the data type for your new variable and report it in a comment.
4. Create an expression that shows a list of TRUE and FALSE values based on the age of each family member. The variable should be TRUE if myFamily$myFamilyAges is less than 40. In other words, your index will be TRUE for kids and FALSE for adults. Assign the results of your expression to a new variable called myIndex.
5. Use myIndex from the previous problem to show the favorite foods for each kid in the family. If you used the variable name suggested in problem 1, the expression would be myFamily$myFoods[myIndex].
6. The ! character is used to invert a set of Boolean values by changing each TRUE to FALSE and vice versa. Adapt the expression from the previous problem to show the favorite foods for each kid in the family.

Sources

http://en.wikipedia.org/wiki/Central_tendency
http://en.wikipedia.org/wiki/Median
http://en.wikipedia.org/wiki/Relational_model
http://stat.ethz.ch/R-manual/R-devel/library/base/html/data.frame.html
http://www.burns-stat.com/pages/Tutor/hints_R_begin.html
http://www.khanacademy.org/math/statistics/v/mean-median-and-mode

R Functions Used in This Chapter

c()	Combines data elements together.
<-	Indicates an assignment arrow.
data.frame()	Makes a dataframe from separate vectors.
head()	Lists the first rows in a dataframe.
levels()	Shows the levels for a factor variable.
nrow()	Reports the number of rows for a dataframe.
str()	Reports the structure of a data object.
sum()	Calculates the sum of a set of values.
summary()	Reports data modes/types and a data overview.
tail()	Lists the last rows in a dataframe.

This text includes access to datasets and select student resources. To learn more, visit sagepub.com