3 CREATING DATA AND GRAPHS IN YOUR ESSAYS

3.1 INTRODUCTION

In the previous chapter, you learned how you can use R Markdown to integrate text and graphs together into an essay, "Netflix and Binge-watching." In that essay, Graph 1 compares Netflix subscriptions among four different generations, and Graph 2 compares the amount of binge-watching between men and women during weekdays and weekends. Both graphs employ bar plots, which are a popular and useful chart type for comparison between individuals or groups. Oftentimes, though, you need to visualize data to help answer other questions. For example, what is the distribution and composition of a variable? How does a variable change over time? What is the relationship between two variables? You could use other chart types to help answer all these questions.

In this chapter, you will learn how to create a bar plot for comparison between individuals, a pie chart to demonstrate composition, a histogram to reveal distribution, a scatter plot to explore the relationship between two variables, and a line plot to illustrate the temporal change. These, of course, do not exhaust the chart types you could create in R, but they provide a basis from which you could explore further.

Just as a reminder, once you create graphs in R, you can then place their R code in the .Rmd or R Markdown file as in the previous chapter. With the addition of graphs as visual aids, your writing will become a lot more interesting and a lot easier to read.

Now, there is one unique and important feature with the data used in this chapter. They are not drawn from existing datasets but are, instead, created from day-to-day life experiences. For example,

- Results of a made-up hot dog eating contest among friends
- Test scores in a hypothetical Law School Admission Test (LSAT) review class of 26 students

- Winning numbers in the Texas Pick-3 lottery
- Daily plays of the top 5 songs on Spotify
- Mileages from the 10 best-selling cars in 2017 from the Kelley Blue Book
- Number of views of Alfie Deyes channel on YouTube.

Using these types of data benefits your learning for two important reasons. First, for most first-time R users, learning how to import large, real-world datasets is often frustrating. The learning process involves datasets of various formats, demands attention to a variety of details, and at the same time, causes frustration because it can't produce any immediately gratifying output such as a graph or a reproducible report. When learning data importing, students often run into various simple coding errors that are not self-evident (e.g., missing a comma or parenthesis, a spelling error), and then become intimidated and quickly lose interest. Thus, it could be counterproductive to teach new R users first how to import existing datasets and then how to create data visualizations from that dataset; reversing the order of these two topics helps address this challenge for new users, but it does require that you create your own data.

Second, many students often have a misperception that data only exist in datasets, and the idea of working with datasets is incredibly intimidating. Through the data examples used in this chapter, you should realize that data can be found in everyday life. As a matter of fact, numerical information in everyday life could be visualized in many instances. For example, the years in which wars occurred in the U.S. history and the numbers of casualties, the results of your chemistry class lab experiment, the box office numbers of Oscar-winning movies, the batting averages and numbers of home runs of baseball players, the numbers of touchdowns of different football teams, and the list goes on.

Through this chapter, you as first-time R users will accomplish the following objectives:

1. Learn to create code chunks to be inserted into an .Rmd file
2. Learn to convert numerical information in daily life into data
3. Learn to construct and interpret five different types of charts to obtain different types of information
    - bar plot (compare individuals)
    - histogram (show distribution of variable)
    - pie chart (reveal composition of variable)
    - scatter plot (discover relationship between two variables)
    - time series line plot (uncover changes over time)
4. Learn to apply those skills in various exercises

3.2 BAR PLOT I: GRAPHING THE WINNERS OF A HOT DOG EATING CONTEST

3.2.1 When to Use a Bar Plot

How can you visualize and compare data for individuals through a plot?

The easiest and most practical way would be to create a bar plot. In this section, you will learn how you can use a bar plot to demonstrate the winner of a hot dog eating contest among a bunch of friends.

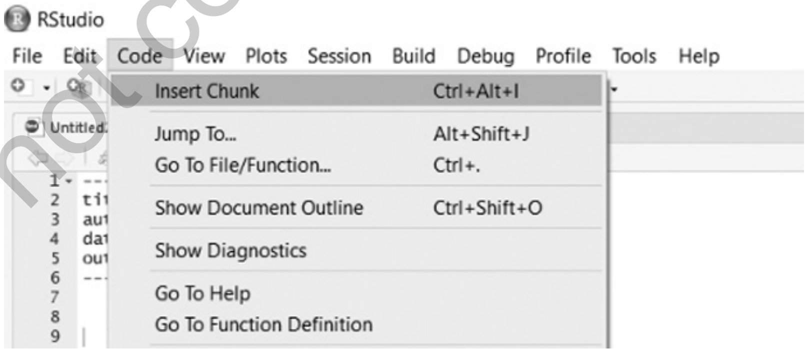
In this exercise, you will learn how to enter numbers into R as data R can recognize, and then how to use that data to create a bar plot that compares the number of hot dogs eaten by each contestant. By the end of this section, you should be comfortable with creating simple bar plots for comparison among individuals.

Four friends—Katharine, Doug, Jennifer, and Ryan—are competing to see who can eat the most hot dogs in 5 minutes. The hot dogs are plain, without ketchup or mustard—just a hot dog in a bun. The records for the four contestants are as follows:

1.	Ryan	9 hot dogs
2.	Jennifer	6 hot dogs
3.	Katharine	11 hot dogs
4.	Doug	14 hot dogs

3.2.2 Creating and Interpreting a Bar Plot

Step 1: The first step in creating a bar plot for your R Markdown report is to create an R code chunk in your .Rmd file. To do so, click **Code**, **Insert Chunk**, as shown below.


RStudio
File Edit Code View Plots Session Build Debug Profile Tools Help
Insert Chunk Ctrl+Alt+I
Jump To... Alt+Shift+J
Go To File/Function... Ctrl+.
Show Document Outline Ctrl+Shift+O
Show Diagnostics
Go To Help
Go To Function Definition

You can also press down three keys on your keyboard—**Ctrl**, **Alt**, and **I**—together to accomplish the same task.

After doing so, as described in the previous chapter, you should see the following in your .Rmd file.

~~~
```{r}

```
~~~

Inside this code chunk, place your R code from the next couple of steps to produce a bar plot, which will be integrated seamlessly within the text in the output document. The output document will look like the homework essay "Netflix and Binge-watching" in the previous chapter.

Note: To refresh your memory, remember that you can control what your code and output look like by adding options to `{r}`.

- If you don't want the R code to be shown in your final document, simply replace `{r}` with `{r, echo=FALSE}`.
- If you don't want the `#` to appear in your output, simply replace `{r}` with `{r comment=NA}`.
- If you don't want any warnings or messages to show up in your output, simply replace `{r}` with `{r, warning=FALSE, message=FALSE}`.

Step 2: The second step in creating a bar plot is writing the R code to create your variables. For this example, you will want to create a variable that will hold the contestants' names, and then a variable that will hold the number of hot dogs that each contestant ate. To accomplish this, you can use the following code, which provides a template for your assignment as well.

~~~
```{r}
# create variable for contestant names
contestants <- c("Ryan", "Jennifer", Katharine", "Doug")

# create variable for numbers of hot dogs eaten
hot_dogs <- c(9, 6, 11, 14)
```
~~~

Note: First, take a look at the two lines of code starting with hashtags or pound signs (`#`). Lines of code that start with `#` are called comment lines. Comment lines are ignored by R, so they are a perfect way to keep certain lines of code in your file without having them be executed, or to write notes to yourself. Comment lines are also an important tool to not only keep track of what your code is doing for yourself but also help others reproduce your code in the future.

The second thing to note in the code above is that the variables `contestants` and `hot_dogs` are being created through the c() function and the assignment symbol <-.

- The letter `c` and its parentheses, that is, the c() function, takes and combines whatever is inside the parentheses as input and exports it as output, with commas being used to separate the different elements of each variable.
- The expression `<-` is an assignment symbol that takes whatever comes out of a function in R and assigns it to the variable on the left.
- Since the contestants' names are characters, the elements of the variable `contestants` should be surrounded by quotation marks.
- The variable `hot_dogs` is similarly created but without quotation marks because the values are numeric. If you add quotation marks around the numeric values, they will be treated as characters, and there will be an error message when you try to plot these characters as numbers.
- Make sure the corresponding values of the two variables match. For example, the first observation of `contestants` is "Ryan," which should match with the value 9 in the first observation of `hot_dogs`, since that is how many hot dogs that Ryan ate. If a value is missing, always make sure to insert `NA` in its place because `NA` tells R that there is a missing value for that place.
- To keep track of, and show others, what you are doing, you place a comment line above each line of code. These comment lines are defined by the hashtag or pound sign `#`, which tells R to ignore these lines.

Step 3: The third step is to write the R code that will create a bar plot for the number of hot dogs eaten, with contestant names as labels, using the barplot() function.

~~~
```{r}
# create barplot
barplot(hot_dogs,
        main = "Hot Dog Eating Contest",
        names.arg =  contestants,
        ylab = "Number of Hot Dogs")
```
~~~

Note:

- Inside the parentheses of the barplot() function, list `hot_dogs` first because the variable provides the values for the different bars in the plot. It is followed by a comma if there is any additional argument to be listed.
- The `main =` option spells out the title of the plot, surrounded by quotation marks.

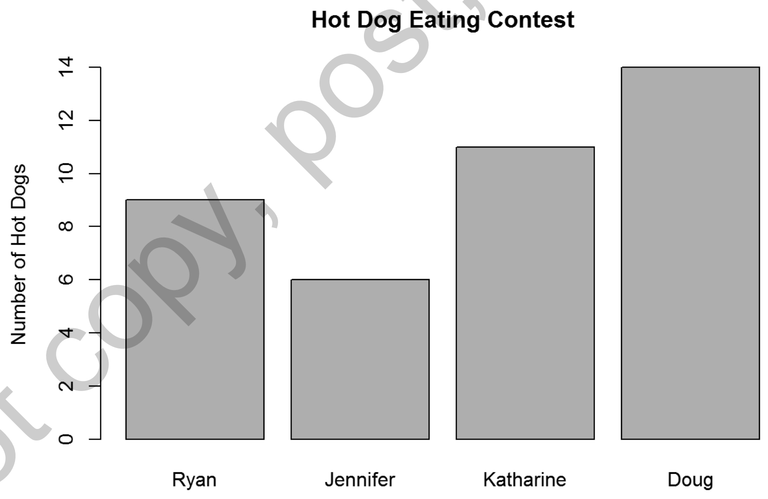
- The `names.arg =` option uses the values of the `contestants` variable as labels for the different bars.
- The `ylab =` option adds the label for the *y*-axis of the plot.

Step 4: The fourth step is to run the completed code chunk and obtain the plot. In this example, the complete code chunk and its plot output will look like this:

```
# create variable for contestant names
contestants <- c("Ryan", "Jennifer", "Katharine", "Doug")

# create variable for numbers of hot dogs eaten
hot_dogs <- c(9, 6, 11, 14)

# create barplot
barplot(hot_dogs,
        main = "Hot Dog Eating Contest",
        names.arg = contestants,
        ylab = "Number of Hot Dogs")
```



Step 5: The last step in producing a plot is to explain it! It is your job, not the reader's, to define what your plot does and then interpret what pattern the plot reveals.

For this example, you may draw the following conclusion:

The bar plot compares graphically the numbers of hot dogs eaten by four contestants—Ryan, Jennifer, Katharine, and Doug—in a hot dog eating contest, to identify a winner. From this plot, it's clear that Doug consumed the most, Jennifer had the least, while Katharine took second place and Ryan took third.

3.2.3 Exercise

In the example above, you examined data on how many hot dogs four different contestants were able to consume in a hot dog eating contest. Using this example as a template, you can create bar plots for similar comparisons, like who won the most votes in an election, and so on. All you need is one character variable that contains names or labels and one numeric variable that contains numerical data.

Try creating a bar plot on your own! To start, do the following.

- Google restaurants near you on the internet. In this search, you should find a list of restaurants near you and a certain number of reviews for each restaurant.
- Create a variable that contains the names of the restaurants (pick 3–5 restaurants).
- Create a variable that contains the number of positive (say, four-star) reviews per restaurant.
- Create your bar plot.
- Define and interpret your plot.

You can use the following example code as a template for your bar plot. Remember to replace number1, number2, and number3 with actual numerical values for the numbers of four-star reviews for the three restaurants.

```
restaurant <- c('restaurant1', 'restaurant2', 'restaurant3')

reviews <- c(number1, number2, number3)

barplot(reviews,
   main = "Most Highly Recommended Restaurants Near Me",
   names.arg = restaurant,
   ylab = "Number of Four Star Reviews")
```

Once the code runs and creates a figure, please be sure to add a comment for each line of code.

3.3 BAR PLOT II: GRAPHING WINNING LOTTERY NUMBERS IN TEXAS PICK-3

3.3.1 Another Instance Where a Bar Plot Is Useful

You learned earlier how to use a bar plot to display who won a hot dog eating contest. In that example, you observed the final number of hot dogs eaten for each player and then plotted the numbers.

Yet, in many other situations, you may have a collection or stream of numbers that have occurred over a period of time. In these cases, the best course of action would be to display the frequency count of each unique number using a bar plot. How do you do that? Will it be in the same way as before? This time you may use winning lottery numbers as an example.

Ok, let's be real, the chance of winning a lottery is slim to none. But what if you were able to determine which numbers were going to be picked the most? The easiest way to display this type of information would be to use the barplot function together with the table function in order to count the frequency of each number.

As an example, here are the morning winning numbers during a 10-day period (5/21/2018 to 4/27/2018) from the Texas Pick-3 lottery. The data are available at the following website: https://www.txlottery.org/export/sites/lottery/Games/Pick_3/Winning_Numbers/index.html_8783066.html. (Also as a disclaimer, if you visit this site, the numbers will likely be different, but that's okay! You can still create the same graph using the steps below for the new numbers.)

Below are the raw data, and let's find out which numbers were picked the most.

5, 9, 4, 4, 6, 1, 4, 7, 9, 9, 0, 8, 4, 1, 4, 1, 9, 0, 7, 1, 1, 3, 0, 8, 2, 4, 3, 2, 7, 9

3.3.2 Creating and Interpreting a Bar Plot

Step 1: Again, the first step is to create an R code chunk by clicking **Code**, and **Insert Chunk**. Once you have done that you will see the following in your .Rmd file:

~~~
```{r}

```
~~~

Again, inside this code chunk, place your R code from the steps below to produce the bar plot to be merged with the text in the .Rmd file.

Step 2: The second step is to create a variable for the winning lottery numbers, like in the hot dog example. Again, to do so, you use the c() function and the assignment symbol <-. Like before, you should add a comment line to describe what is being done.

~~~
```{r}
# create a variable of the winning numbers
winning_numbers <- c(5, 9, 4, 4, 6, 1, 4, 7, 9, 9, 0, 8, 4, 1, 4, 1, 9, 0, 7, 1, 1, 3, 0,
                     8, 2, 4, 3, 2, 7, 9)
```
~~~

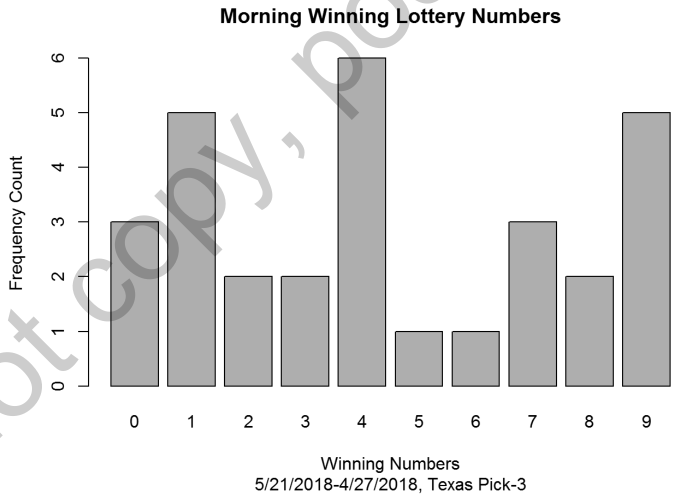
Step 3: The third step is to plot the frequencies of the winning numbers using the following line of code. Inside the parentheses of the barplot function is the table() function, inside which is the variable `winning_numbers`. The table() function counts the frequency of each unique number in the variable, and then the barplot() function displays each frequency in bar form.

~~~
```{r}
# plot the frequency of each winning number
barplot(table(winning_numbers))
```
~~~

Step 4: After you add a title, a subtitle, a label for the *y*-axis, and a label for the *x*-axis, the completed code chunk and its output will look as follows:

```
#Create the data set
winning_numbers <- c(5, 9, 4, 4, 6, 1, 4, 7, 9, 9, 0, 8, 4, 1, 4, 1, 9, 0, 7, 1, 1, 3, 0,
                    8, 2, 4, 3, 2, 7, 9)

# Plot the frequency of each winning number
barplot(table(winning_numbers),
        main = "Morning Winning Lottery Numbers",
        sub = "5/21/2018-4/27/2018, Texas Pick-3",
        ylab = "Frequency Count",
        xlab = "Winning Numbers")
```



Step 5: The fifth step is to define and interpret the plot produced. The bar plot displays the number of times each number has appeared as a winning lottery number in the Texas Pick-3 lottery during a 10-day period. As shown in the plot, the number 4 appears most frequently, with 1 and 9 being the second most frequent; and the least frequent numbers are 5 and 6.

3.4 PIE CHART: GRAPHING THE COMPOSITION OF DAILY PLAYS AMONG TOP 5 SONGS ON SPOTIFY

3.4.1 When to Use a Pie Chart

Another commonly used plot is the pie chart, which is often used to show comparison or composition. While this is a plot that everyone should know how to make, you should note that it is a somewhat controversial chart type. Many analysts argue against pie charts because they can be difficult to interpret when there aren't clear disparities between variable values.

In this example, you will look at the composition of the daily plays of the top 5 songs in the United States on Spotify on February 28, 2018. One way to think about this example is how the whole pie that consists of all the daily plays for the top 5 spots is divided up among the top 5 songs. The data are collected from the Spotify Charts Top 200 for February 28, 2018 (https://spotifycharts.com/regional/us/daily/2018-02-28). Obviously, you can do the same exercise for any other day, and for more than just the top 5 spots.

The data for February 28, 2018, are as follows:

Rank	TRACK	STREAMS (daily plays)
1	God's Plan by Drake	3,246,141
2	Psycho (feat. Ty Dolla $ign) by Post Malone	3,003,919
3	Look Alive (feat. Drake) by BlocBoy JB	1,697,529
4	All the Stars (with SZA) by Kendrick Lamar	1,695,184
5	Stir Fry by Migos	1,261,703

3.4.2 Creating and Interpreting a Pie Chart

Step 1: As with the bar plot, the first step in creating a pie chart is to create an R code chunk for the R Markdown file. As before, to do this, click **Code** and **Insert Chunk**. Once you have done this, the following will appear in the .Rmd file:

~~~
```{r}

```
~~~

Inside this code chunk, you will insert the R code from the next few steps to produce the pie chart to be merged with the text in the .Rmd file.

Step 2: The second step is to turn the records above into two variables—`songs` and `dailyPlays`. Once again, we're going to use the c() function and the assignment symbol <- to create these two variables. We'll also add comment lines to help explain what each line of code does. (This can make it a lot easier for others to review and re-create your code later on.)

~~~
```{r}
# create the songs variable
songs <-  c('God's Plan',
            'Psycho (feat. Ty Dolla $ign)',
            'Look Alive (feat. Drake)',
            'All the Stars (with SZA)',
            'Stir Fry')

# create the number of daily plays for each song
dailyPlays <- c(3246141, 3003919, 1697529, 1695184, 1261703)
```
~~~

Step 3: The third step in creating a pie chart is to use the pie() function. The code for creating the chart is below:

~~~
```{r}
# create pie chart
pie(dailyPlays,
    labels = songs,
    main = "Top 5 Songs on Spotify on February 28, 2018, in the U.S.")
```
~~~

Note:

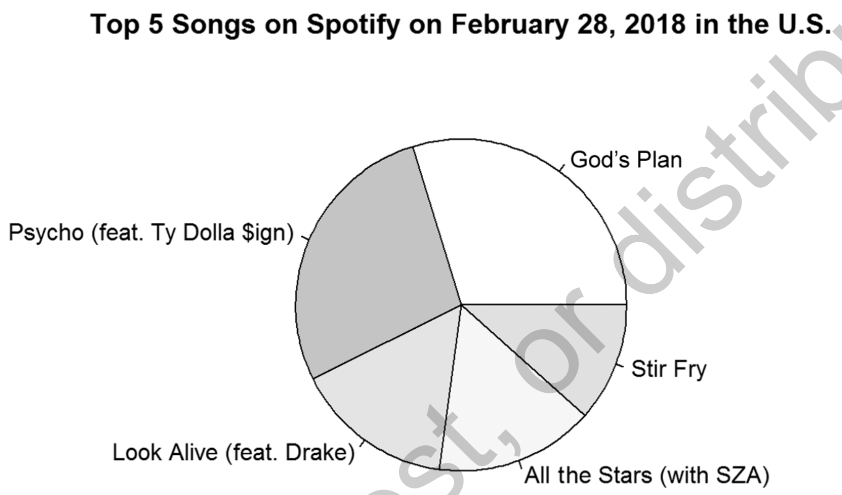
- Inside the parentheses of the pie() function, list the variable `dailyPlays` first since each value of the variable determines the size of each slice in the pie.
- The `labels = songs` option tells R to label each slice with the name of the corresponding song.
- The `main =` option provides the title of the chart.

Step 4: The completed code chunk and its pie chart output look as follows:

```
# create the songs variable
songs <- c('God's Plan',
           'Psycho (feat. Ty Dolla $ign)',
           'Look Alive (feat. Drake)',
           'All the Stars (with SZA)',
           'Stir Fry')

# create the number of daily plays for each song
dailyPlays <- c(3246141, 3003919, 1697529, 1695184, 1261703)
```

```
# create pie chart
pie(dailyPlays,
    labels = songs,
    main = "Top 5 Songs on Spotify on February 28, 2018, in the U.S.")
```



Step 5: The pie chart above displays the composition of the daily plays for the top 5 spots on Spotify in the United States played on February 28, 2018. According to the pie chart, `God's Plan` and `Psycho` received the most daily plays, `Look Alive` and `All the Stars` came in the third and fourth place, and `Stir Fry` ranked in the fifth place among the top 5.

3.4.3 Exercise

While you used popular Spotify songs for data, you can use various other kinds of data to make a pie chart—all you need is one variable that contains names and the other variable that contains numerical values.

This time, try creating a pie chart on your own! To start, do the following:

- Create a variable that contains categories (i.e., song titles, movie titles, food titles, etc.).
- Create a variable that contains numbers (i.e., number of times people watched certain shows, the number of people who live in each of the most populated cities in the United States, etc.).
- Create your pie chart.

You can use the following example code chunk as a template.

```
words <- c('thing1', 'thing2', 'thing3')

numbers <- c(number1, number2, number3)

pie(numbers, labels = words, main = "Title for your graph")
```

Again, add a comment line to each line of your code. Doing so makes it easier to understand your code and can make it easier for others to reproduce your analysis.

3.5 HISTOGRAM: GRAPHING THE DISTRIBUTION OF LSAT SCORES IN A REVIEW CLASS

3.5.1 When to Use a Histogram

A histogram shows the distribution of a single numeric or integer variable. Similar to a bar plot, it tells you the number of times that something occurs. However, while a bar plot graphs a discrete categorical variable, a histogram can graph any numerical variable (integers or not). Moreover, a histogram plots the frequencies of observations in certain ranges or bins of values, thus showing where in the distribution an event occurs the most (i.e., the central tendency of the distribution). In addition to revealing the central tendency, a histogram also shows whether the distribution is widespread or narrowly peaked, and whether the distribution is symmetric around the central tendency or skewed toward some outliers. These are the distributional features of a histogram you ought to pay attention to.

In this example, you will learn about the distribution of the LSAT scores in a hypothetical review class of 26 students, using a histogram. Just to be clear, the LSAT is the Law School Admission Test, in which scores range between 120 and 180 points.

3.5.2 Creating and Interpreting a Histogram

Step 1: The first step in creating a histogram in a R Markdown document is to create an R code chunk. As before, click **Code**, **Insert Chunk**, and then you will see the following in your .Rmd file.

~~~
```{r}
```
~~~

Inside this code chunk, you should place the R code from the next few steps to produce the chart to be merged with the text in the .Rmd file.

Step 2: The second step is to create the variable for LSAT scores—`lsat_scores`. In the previous sections, you learned how to use the c() function and the assignment symbol `<-` when creating a variable. To do so, use the code below. Unlike the bar plot, a histogram is based on one variable and the focus is on the distribution of the values, rather than individual observations. Thus, the values do not have to be in any specific order to match with those of another variable.

Also, remember that it is a great habit to write a comment line above each line of code. It really does make your code look cleaner and much easier to read.

~~~
```{r}
# create the LSAT score variable
lsat_scores <- c(151, 174, 141, 148, 144, 137, 149, 179, 139, 128, 164, 151, 164, 152,
                 149, 155, 157, 151, 138, 123, 150, 168, 148, 154, 153, 162)
```
~~~

Step 3: The third step is to create a histogram plot for the variable, using the hist() function.

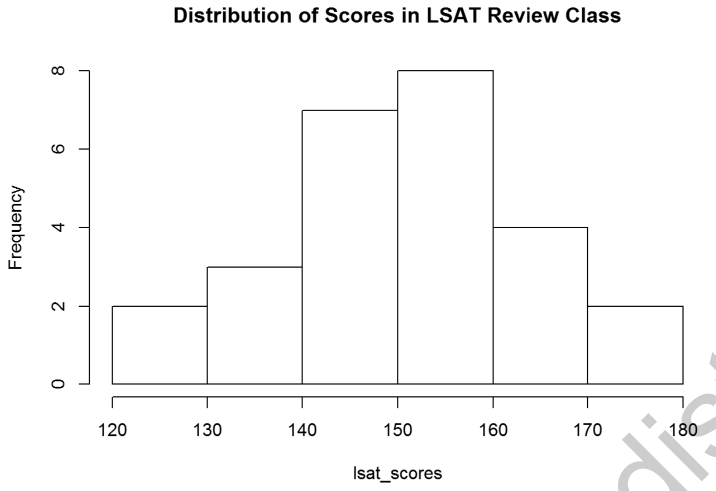
~~~
```{r}
# create the histogram
hist(lsat_scores, main = "Distribution of Scores in LSAT Review Class")
```
~~~

Inside the parentheses of the hist() function, list `lsat_scores` first because the variable provides the scores of different students. Like in the previous sections, the `main =` option spells out the title of the plot, surrounded by quotation marks.

Step 4: The completed code chunk and its plot output look as follows:

```
# create the LSAT score variable
lsat_scores <- c(151, 174, 141, 148, 144, 137, 149, 179, 139, 128, 164, 151, 164, 152,
                 149, 155, 157, 151, 138, 123, 150, 168, 148, 154, 153, 162)

# create the histogram
hist(lsat_scores, main = "Distribution of Scores in LSAT Review Class")
```



Step 5: The histogram above displays the distribution of LSAT scores in a class. To interpret the histogram, you should identify its peak, spread, and symmetry. In the figure, the most frequent LSAT scores are between 150 and 160, which is likely the center of the distribution. The second most frequent values are between 140 and 150. The spread of the distribution ranges between 120 and 180, not surprisingly. The distribution is slightly skewed to the left, and it appears that more LSAT scores are clustered on the left side of the peak of the histogram.

3.5.3 Exercise

You can practice creating histogram plots for a variety of different variables so long as the variables are numeric. These plots will help you better understand the distribution of your chosen variables.

This time, try creating a histogram on your own! To start, do the following:

- Find data for a numeric variable (daily temperatures in a city during a year, GDP per capita of different countries in a year, etc.).
- Use the **hist()** function to create your histogram plot.

You can use the following code as a template.

```
# find data for a variable
variable <- c(value1, value2, etc.)

# create a histogram
hist(variable, main = "Title of histogram")
```

Again, make sure to add a comment line to each line of code.

3.6 SCATTER PLOT: GRAPHING THE RELATIONSHIP BETWEEN TWO VARIABLES—GAS MILEAGES IN THE CITY AND ON THE HIGHWAY

3.6.1 When to Use a Scatter Plot

In this section, you will learn how to create a basic scatter plot for two variables. Scatter plots are useful for uncovering and illustrating the relationship between two numeric variables. By the end of this section, you should be comfortable not only with creating scatter plots but also with labeling your data points.

As an example, try using the data for two variables—miles per gallon in the city and miles per gallon on the highway—for the 10 best-selling cars in 2017. You may find the data from the following Kelley Blue Book site: **“10 Best-Selling Cars of 2017 . . . So Far”** (https://www.kbb.com/car-reviews-and-news/top-10/best-selling-cars-2017/2100004451/). Simply click on the **Start List>** button below the page title, and you can find the data starting with the car rated number 10 first, and so on.

The question of interest could be “Do cars that have high gas mileages in the city also have high mileages on the highway?” If the answer is yes, you should find the two variables be positively associated with each other.

The data collected from the website are as follows:

1.	Ford F-Series	18.0 mpg city	24.0 mpg highway
2.	Chevy Silverado	18.0 mpg city	24.0 mpg highway
3.	Ram Truck	15.0 mpg city	22.0 mpg highway
4.	Nissan Rogue	26.0 mpg city	33.0 mpg highway
5.	Honda CR-V	25.0 mpg city	31.0 mpg highway
6.	Toyota Rav4	22.0 mpg city	28.0 mpg highway
7.	Toyota Camry	24.0 mpg city	33.0 mpg highway
8.	Toyota Corolla	28.0 mpg city	36.0 mpg highway
9.	Honda Civic	30.0 mpg city	39.0 mpg highway
10.	Honda Accord	27.0 mpg city	36.0 mpg highway

3.6.2 Creating and Interpreting a Scatter Plot

Step 1: As before, the first step is to create an R code chunk in the R Markdown file. To do this, click **Code**, and **Insert Chunk**. Once you have done this, this will appear in the .Rmd file:

~~~
```{r}
```
~~~

Step 2: The second step is to create your data. Inside the code chunk, create three variables—city mileage, highway mileage, and vehicle brand—using the c() function and the assignment symbol <-.

~~~
```{r}
# create variables
city <- c(18.0, 18.0, 15.0, 26.0, 25.0, 22.0, 24.0, 28.0, 30.0, 27.0)
highway <- c(24.0, 24.0, 22.0, 33.0, 31.0, 28.0, 33.0, 36.0, 39.0, 36.0)
names <- c("Ford F-Series", "Chevy Silverado", "Ram Truck", "Nissan Rogue", "Honda CR-V",
           "Toyota Rav4", "Toyota Camry", "Toyota Corolla", "Honda Civic", "Honda Accord")
```
~~~

The three variables created with the raw data are called `city`, `highway`, and `names`, respectively. The `city` variable contains the data on miles per gallon in the city for different vehicle brands; the `highway` variable contains the data on miles per gallon on the highway for different vehicle brands; the `names` variable contains the names of the ten car models.

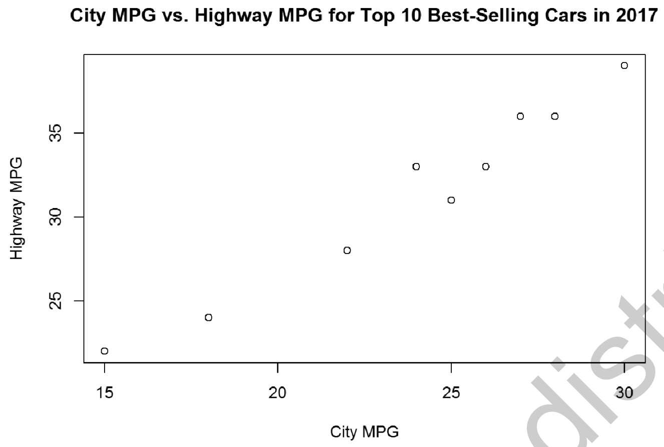
Step 3: The third step is to enter the R code for the scatter plot using the plot() function.

~~~
```{r}
# scatter plot for highway and city gas mileages
plot(city, highway,
     main = "City MPG vs. Highway MPG for Top 10 Best-Selling Cars in 2017",
     xlab = "City MPG",
     ylab = "Highway MPG")
```
~~~

Note:

- Inside the plot() function, list the variable `city` first so that it is represented on the *x*-axis, and then list `highway` so that it is represented on the *y*-axis.
- Add a title using the `main =` option, which is "City MPG vs. Highway MPG for Top 10 Best-Selling Cars in 2017."
- Add two more options: `xlab =` and `ylab =`, to define the labels for the *x* and *y* axes; in this case, the *x*-axis is labeled as "City MPG" and the *y*-axis as "Highway MPG."

Running the code above gives you the following figure:



The scatter plot shows a clear positive correlation between the two variables. Higher city mileages are associated with higher highway mileages.

Note two interesting things about the figure. First, there are 10 cars but only 9 data point symbols. This is because "Ford F-Series" and "Chevy Silverado" have identical values on both variables and overlay on the same symbol.

Second, what isn't immediately clear from looking at the figure is which data points correspond to which vehicle models. To help fix this problem, you can add text labels to each of the data points. To do this, you can use the text() function as shown below.

~~~
```{r}
# add labels to data points
text(city, highway, labels = names, pos = 1, cex = 0.4, offset=0.25)
```
~~~

Note:

- Inside the text() function, first list the *x*-axis variable `city` and the *y*-axis variable `highway`, which provide the coordinates for the data points in the scatter plot.
- Then list the `labels =` option, which takes the text values of the variable entered (in this case, it is `names`) and places those text values near corresponding data point coordinates.
- The `pos =` option specifies the position of the text value near a data point, with 1 = below, 2 = left, 3 = above, and 4 = right.
- The `cex =` option specifies how the text value should be scaled relative to the default value of 1, with 1.2 being 20% larger, 0.4 being 40% smaller, and so on.

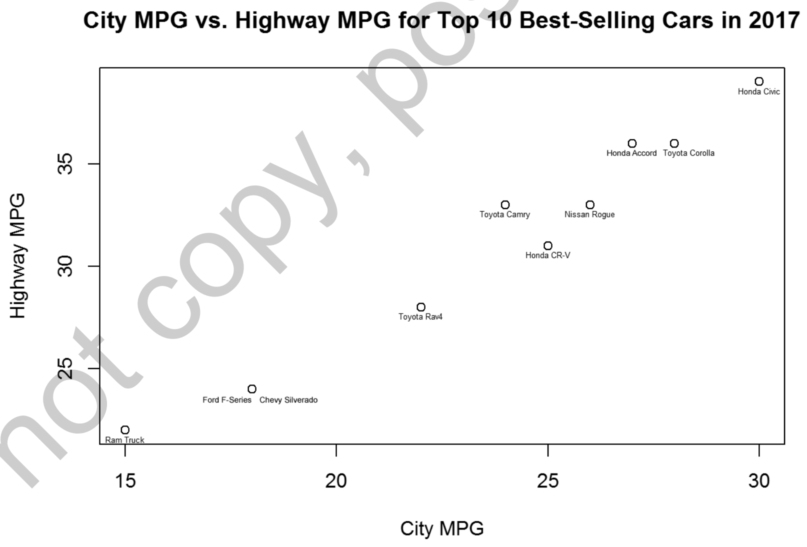
- The `offset =` option, when `pos =` is also used, determines the distance of the text label from the specified data point coordinate in fractions of a character width. The smaller the value, the closer they are.

Step 4: The completed code chunk and the scatter plot output look as follows:

```
# create variables
city <- c(18.0, 18.0, 15.0, 26.0, 25.0, 22.0, 24.0, 28.0, 30.0, 27.0)
highway <- c(24.0, 24.0, 22.0, 33.0, 31.0, 28.0, 33.0, 36.0, 39.0, 36.0)
names <- c("Ford F-Series", "Chevy Silverado", "Ram Truck", "Nissan Rogue", "Honda CR-V",
          "Toyota Rav4", "Toyota Camry", "Toyota Corolla", "Honda Civic", "Honda Accord")

# scatter plot for highway and city mileages
plot(city, highway,
     main = "City MPG vs. Highway MPG for Top 10 Best-Selling Cars in 2017",
     xlab = "City MPG",
     ylab = "Highway MPG")

# add labels to data points
text(city, highway, labels = names, pos = 1, cex = 0.4, offset=0.25)
```



Step 5: The scatter plot demonstrates a positive correlation between the two variables: `city` and `highway`. The higher a vehicle's miles per gallon in the city, the higher that vehicle's miles per gallon is on the highway. This is not a surprising result.

With text labels for the data points, the scatter plot provides a much easier way to compare the 10 models. Honda Civic is far ahead of the other nine in both city and highway MPG. In contrast, Ram Truck has the lowest gas efficiency in terms of both city and highway MPG.

3.6.3 Exercise

Any two numerical variables can be used to create a scatter plot to demonstrate the relationship between the two variables.

Now try creating a scatter plot on your own! To start, do the following:

- Create two variables that contain numeric values (i.e., the height and weight of people, the amount of rain fall and crop harvest, etc.).
- Create your scatter plot.

You can use the following example code as a template.

```
x <- c('number1', 'number2', 'number3', etc.)
y <- c('number1', 'number2', 'number3', etc.)

plot(x, y,
      main = "Whatever you want to name your scatter plot",
      xlab = "name of x-axis",
      ylab = "name of y-axis")
```

3.7 TIME SERIES PLOT: GRAPHING THE CHANGING PATTERN OF YOUTUBE VIDEO VIEWS

3.7.1 When to Use a Time Series Plot

Many phenomena involve changes over time. The purpose of graphing these changes over time is to show whether the variable increases or decreases over time, and whether there are a lot of ups and downs or just a plain linear trend in the variable. In this section, you will learn how to create a basic time series line plot.

In this example, you will use data on the number of views that Alfie Deyes, of PointlessBlogVlogs, received on his YouTube channel. The question of interest is, "What is the changing pattern in the daily views of Alfie Deyes on YouTube?"

Data were obtained from each of Alfie Deyes's daily vlogs between February 16 and February 25, 2018, at the following link: https://www.youtube.com/user/PointlessBlogTv/videos.

The data are as follows:

1. 2-16-2018 ("Getting My First Tattoo!"); 828,000 views
2. 2-17-2018 ("Our Last Day in Amsterdam"); 441,000 views
3. 2-18-2018 ("Opening 2 Mystery Boxes!"); 344,000 views
4. 2-20-2018 ("Getting an Audi R8 & Seeing Ed Sheeran Live!"); 412,000 views
5. 2-22-2018 ("Why Zoe's Not in My Books . . . "); 506,000 views
6. 2-24-2018 ("3 New Rules in Our House . . . "); 412,000 views
7. 2-25-2018 ("Double Date = Pizza & Games Evening!"); 471,000 views

3.7.2 Creating and Interpreting a Time Series Plot

Step 1: As before, the first step is to create an R code chunk in the R Markdown file. As before, to do this, click **Code**, and **Insert Chunk**. Once you have done this, this will appear in the .Rmd file:

~~~
```{r}
```
~~~

Step 2: The second step is to turn the records above into two variables—`date` and `views`. The `date` variable is a calendar date variable. The `views` variable stores the number of views for each of the seven YouTube videos. Once again, use the c() function and the assignment symbol <- to create these two variables. And add a comment line to explain what the code does. The R code is presented below.

~~~
```{r}
# create date and views variables
date <- c("2/16/2018", "2/17/2018", "2/18/2018", "2/20/2018",
"2/22/2018", "2/24/2018", "2/25/2018")

date <- as.Date(date, format = "%m/%d/%y")

views <- c(828000, 441000, 344000, 412000, 506000, 412000, 471000)/1000
```
~~~

Two things, however, make the R code above different from the examples in the earlier sections. First, you need to apply a special function as.Date() to let R specify the `date` variable as a calendar date variable. After the first line of code that generates a character `date` variable using the c() function, you need to add a second line of code that converts it into a calendar date variable and then overwrites the old character variable.

In this line of code, apply the as.Date() function to convert the old character variable into a calendar date variable, in the format of month–day–year (`"%m/%d/%y"`). Note how the date format is specified via quotation marks, the percentage symbol, forward slash, m for month, d for day, and y for year.

Second, all observations in the number of views variable `views` are more than 100,000. To make the labeling of the variable values more legible on the *y*-axis, convert the variable value into number of views in thousands. You may do so by dividing the output of the c() function by 1,000.

Step 3: The third step is essentially the same as using the plot() function for scatter plot.

~~~
```{r}
# create a time series line plot
plot(date, views, type = "b",
     main = "Numbers of Views of Videos by Alfie Deyes on YouTube",
     ylab="number of views in thousands",
     xlab = "February 16-25, 2018")
```
~~~

Note:

- Inside the plot() function, list the variable `date` first because it falls on the *x*-axis, and then list `views` so that it falls on the *y*-axis.
- Add a title using the `main =` option, which is "Numbers of Views of Videos by Alfie Deyes on YouTube".
- Add two more options: `xlab =` and `ylab =`, to define the labels for the *x* and *y* axes, respectively. In this case, the *x*-axis is labeled as "February 16–25, 2018" and the *y*-axis as "number of views in thousands."
- The only thing different from the scatter plot code is the new option `type =`. This option controls the type of plot you will have. Basically, `l` gives a plot with lines between data points, `p` gives a plot of only points, `b` gives a plot of both lines and points, `o` gives overplotted points and lines, `s` or `S` connects data points by stair steps, and `h` draws each data point as a histogram line vertical line. In this example, the `b` type of plot is chosen. You should experiment with the other different plot types.

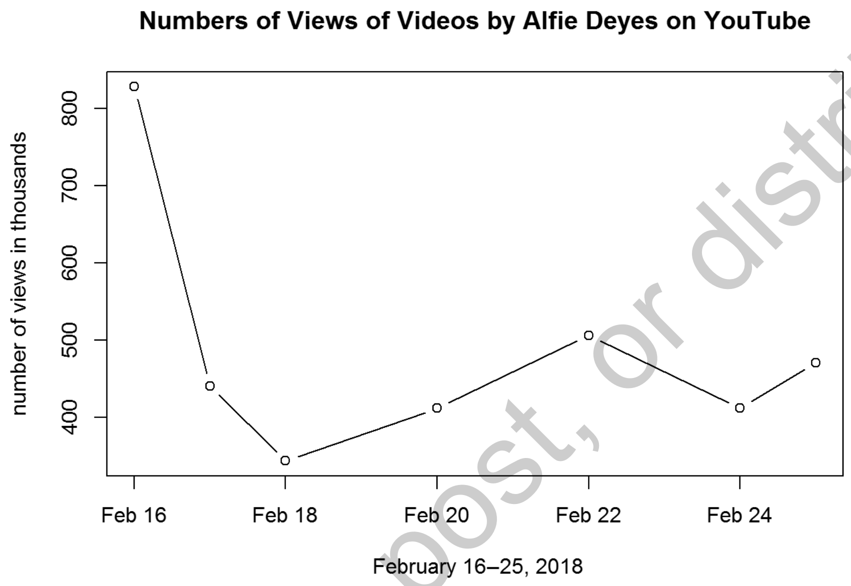
Step 4: The completed code chunk and its plot output look as follows:

```
date <- c("2/16/2018", "2/17/2018", "2/18/2018", "2/20/2018",
"2/22/2018", "2/24/2018", "2/25/2018")

date <-as.Date(date, format = "%m/%d/%y")

views <- c(828000, 441000, 344000, 412000, 506000, 412000, 471000)/1000
```

```
# create a time series line plot
plot(date, views, type = "b",
     main = "Numbers of Views of Videos by Alfie Deyes on YouTube",
     ylab="number of views in thousands",
     xlab = "February 16-25, 2018")
```



Step 5: The graph above demonstrates the changing pattern in the popularity of the seven chosen videos by Alfie Deyes from the PointlessBlogVlogs YouTube channel for the dates February 16–25, 2018. The first video, "Getting My First Tattoo!," uploaded on February 16, received the most views, exceeding 800,000. Every video after that received far fewer views, although the February 22 video appears to be the second most popular, with slightly over 500,000 views.

3.7.3 Exercise

Any numerical variable that is time series in nature can be used to create a time series plot. The plot reflects the changes over time.

Try creating a time series plot on your own! To start, do the following:

- Create two variables, one for calendar date or time and the other a numeric variable reflecting observation values at different time points (say, amount of rainfall in different years, level of pollution over time).
- Create your time series plot.

You can use the following example code as a template:

```
# create relevant variables
date <- c("date1", "date2", etc)

date <-as.Date(date, format = "%m/%d/%y")

y <- c(value1, value2, etc)

# create a time series line plot
plot(date, y, type = "b",
     main = "title",
     ylab="y-axis label",
     xlab = "x-axis label")
```

3.8 USEFUL TIPS: POLISHING AND EXPORTING GRAPHS

3.8.1 How to Polish Your Graph

You can polish your graphs further by adding more options into your code. For example, `col =` allows you to change the color of your bars and points. Say, to apply the white color, specify `col=1` or `col=white`.

What if you want to use a color other than white? To choose a color by name, you can use the palette() function to find a short list of eight color names as follows:

```
palette()
```

```
[1] "black"   "red"     "green3"  "blue"    "cyan"    "magenta" "yellow"
[8] "gray"
```

If you want the full list of 657 colors, run the colors() function as follows. A list of 657 color choices will be displayed, and you may simply choose the code for the color of your choice. The long output is omitted here to save space.

~~~
```{r}
colors()
```
~~~

You can also download the full color index chart by Earl F. Glynn from this site: http://research.stowers.org/mcm/efg/R/Color/Chart/.

What is even more interesting is that you can have multiple colors for different groups of data points based on another variable. For example, in the scatter plot, you could create a domestic or foreign variable for domestic or foreign cars, and then give them each a different color in the scatter plot with the following option, `col = as.integer(foreign)`.

Yet, color isn't the only thing you can change about a graph. You could also easily change a data point's shape using the `pch =` option. You can find the choices here on this site: http://www.sthda.com/english/wiki/r-plot-pch-symbols-the-different-point-shapes-available-in-r.

In addition, you can divide different data points into different groups of different shapes by using this line of code, for example, `pch = as.integer(foreign)`.

A list of different options for how to change the look of your data figures can be found at this site: https://www.statmethods.net/advgraphs/parameters.html.

In your free time, try playing with the different options and customizing your graphs to make them more detailed and unique.

3.8.2 How to Export a Graph

In this chapter, you learned how to create many different kinds of graphs. While graphs are often created and integrated into the output document using R Markdown, sometimes you might also want to export your graph and use it in a PowerPoint presentation, Word Document, or to simply print it out to hang on your wall—you can do that!

To export your graph, simply follow the steps below:

- Knit your .Rmd file and produce an html document
- Right click on the graph you want to save as a separate file
- Select "Save image as . . . " and complete the process

3.9 SUMMARY

In this chapter, you learned how to strengthen an essay like the one in Chapter 2 with data and graphs. More concretely, you learned the following:

- How to convert numerical information in everyday life into data
  - — Results of a fake hot dog eating contest
  - — Test scores of a hypothetical LSAT review class
  - — Winning lottery numbers in Texas Pick-3
  - — Daily plays of top 5 songs on Spotify
  - — Mileages for top 10 best-selling cars in 2017 from Kelley Blue Book
  - — YouTube video views of Alfie Deyes

- How to create different types of graphs for different purposes
  - — Comparing a variable among individuals
  - — Illustrating the distribution of a variable
  - — Demonstrating the composition of a variable
  - — Exploring the relationship between two variables
  - — Capturing the changing pattern of a variable over time

As noted at the beginning of the chapter, you can now easily incorporate these into your essay assignments like the one in Chapter 2. The data and graphs in your essay will help you explore and display the patterns that would have otherwise been less apparent or vivid.

As a caveat, like you will often encounter in your own experiments, not every graph is a good fit for the question or data at hand. You should bear in mind one important takeaway from this chapter. There is not a one-size-fits-all graph that works perfectly every time. You should always experiment, try out different graphs, and make and then correct mistakes. A lot of data visualization is about trial and error, and in the process, you will find out what works best for what you are trying to do.

Finally, here is a heads up for the more ambitious readers. The graphing functions you learned in this chapter are from base R. A popular graphing package is ggplot2, which will be introduced in Subsection 6.2.3.1 of Chapter 6.

3.10 REFERENCES

The data in this chapter draw on various sources.

Alfie Deyes Vlogs. (2018, February 28). YouTube Channel. Retrieved from https://www.youtube.com/user/PointlessBlogTv/videos

Kelley Blue Book. (2017, July 28). *10 Best-selling cars of 2017 . . . so far.* Retrieved from https://www.kbb.com/car-reviews-and-news/top-10/best-selling-cars-2017/2100004451/.

Spotify. (2018, February 28). *Spotify charts top 200.* Retrieved from https://spotifycharts.com/regional/us/daily/2018-02-28

Texas Lottery. (2018). *Pick 3™ Past Winning Numbers*. Retrieved from https://www.txlottery.org/export/sites/lottery/Games/Pick_3/Winning_Numbers/index.html_8783066.html

The discussion related to R Markdown largely draws on the same sources for Chapter 2.

The discussion related to the four types of charts—bar plot, pie chart, histogram, scatter plot, and time series line plot—largely draws on the following sources:

DataCamp. (2017). *Quick-R.* Retrieved from https://www.statmethods.net/graphs/index.html

Help files on base R graph functions such as barplot(), pie(), hist(), and plot().

Kabacoff, R. I. (2011). *R in Action.* Shelter Island, NY: Manning.

Verzani, J. (2004). *Using R for introductory statistics.* Boca Raton, FL: Chapman & Hall.

Several other websites provide intriguing materials on visualization in R beyond the scope of this chapter. It is useful to check them out.

Holtz, Y. (2018). *The R graph gallery.* Retrieved from https://www.r-graph-gallery.com/

Piwek, L. (n.d.). *Tufte in R.* Retrieved from http://motioninsocial.com/tufte/