Chapter 10

Supplement 1: Tutorial: An Introduction to R

S10.1 INTRODUCTION TO R

R is a computer language for statistical computing that is widely used by statisticians, researchers, and students [1]. Not only is it flexible and powerful, it is free! R has many more commands than we will need to use. This tutorial is a starting point for learning the language; it is intended to give a minimal yet sufficient set of R commands for this chapter. If you are interested in more details, there are many textbooks and references for R, as well as online help. The book *Using R for Introductory Statistics* by John Verzani is a particularly good resource. [2]

This introduction to R is not intended to be a statistics textbook; it only describes *how* to do a computation, not *why* or under what conditions.

If you would like to download R to your own computer, go to http://www.r-project.org/. Appendix A, at the end of this tutorial, has more information on installing R.

Open R on your computer.

The first thing you should do in every session with R is to change to the directory (possibly a storage device) where you are keeping your data. Click on the File menu on the upper-left corner of the screen, select "Change dir. . .," and select your folder. If you are working at an Apple computer, use the Misc menu and select "change working directory."

You will find the following quick reference list for R syntax and commands helpful:
http://cran.r-project.org/doc/contrib/Short-refcard.pdf

S10.1.1 Calculations in R

We communicate commands to R by typing them at the ">" prompt and pressing the Enter key. Try these out:

```
> 2/6*12
> 3^2
```

You may add comments if they are preceded by a # symbol. This is a good way to make notes about what you are doing. Try out

```
> 100/10^2     #exponentiation takes precedence over division
```

Try

```
> 3 + 7 *
```

Instead of the usual prompt, R returns a + prompt to indicate that the command is incomplete; R waits for the rest of the command to return the result of the computation:

Algebraic and Discrete Mathematical Methods for Modern Biology. http://dx.doi.org/10.1016/B978-0-12-801213-0.09991-3

```
> 3 + 7 *
+ 5
[1] 38
```

S10.1.2 Assigning Values to Variables

In R, acceptable variable names use letters, numbers, a dot, or an underscore. A variable name must start with a letter or a dot. R is case-sensitive, so for example, `A` and `a` are not the same variable. A few strings are not allowed as variable names because they are reserved for specific uses in R. In particular, lowercase `c` cannot be a variable because it serves another purpose, which we will soon be using. Lowercase `q` is another reserved name: the function `q()` ends the session and closes the window. Some examples of acceptable variable names are `x3`, `yr12`, `yr.birth`, `birth_month`, `R2D2`.

Variables are assigned values using an equal sign on the right of the variable. Try out the following commands in R:

```
> x=12
> x^2
> year = 2011
> year=year+1
> year
> e                 # the letter e is not a constant in R.
> e=exp(1)          # this assigns the "usual" value to e
> pi                # "pi" cannot be used as a variable name, since it
                        represents a defined constant in R
> 3=x               # this will generate an error message
> wt<-12.63         # " <- " is another assignment operator
> wt
```

Only the most recently assigned value of a variable is retained. Variables are saved in the workspace. For more information, see Section S10.1.5.

S10.1.3 Data

Suppose we want to use R to find the mean and standard deviation of this data on the number of books college students were required to purchase for one semester: 9, 8, 5, 12, 4, 17

We need to store the data in a data vector using the `c()` function, as follows:

```
> books = c(9, 8, 5, 12, 4, 17)
```

To see the contents of the vector, enter

```
> books
```

The index `[1]` denotes the first entry of the vector. If the vector were long enough to continue onto the next line, there would be an index at the beginning of the next line to indicate the position in the vector for the next data value. We can inspect any position in a vector using its index. To demonstrate, enter

```
> books[4]
```

You can add data to a vector in a variety of ways. Try these:

```
> books[7]=9

> books=c(books,c(5,8,11))
> books
```

Note: Changing the position of the cursor on a line *cannot* be accomplished in R with a mouse click; the left and right arrow keys must be used. If you want to move left to correct a typing error, you must either backspace or use the left arrow key; move back to the end of the line using the right arrow key or the "End" key.

We can apply mathematical operations and functions to data vectors. Investigate the following, inspecting the variable after each operation.

```
> books -9  # This operation does not alter the entries of the variable books.

> 1/books

> sqrt(books)

> x=c(1,5,3,5)

> x2=x + 10

> x2

> sum(books)

> mean(books)

> sum(books)/length(books)

> max(books)

> sd(books)        # sd() is the sample standard deviation function.

> sort(books)    #Does this change the order of the vector entries?
```

It is frequently helpful to remind yourself what objects you may have defined and saved in your workspace. To see which variables are currently in your workspace, use `ls()` or `objects()`. To delete an object, use the "remove" operator, `rm()`. For example, try

```
> rm(x)
> ls()
```

If you discover that you have mistyped a command, of course you can retype it and reenter it. But it may be more efficient to backtrack using arrow keys, and then edit. The up-arrow and down-arrow keys let you scroll through previous commands. Try this out:

You can edit the current line and re-execute it. For example, scroll up to

```
> sort(books)
```

Use the left arrow to move to the > prompt and change the line to

```
> y=sort(books)
```

and enter.

Now the variables *y* and *books* contain the same data but in different orders.

S10.1.4 Data Entry Made Easier

There are several ways to import data from other sources into your R workspace. We will return to this topic when we encounter more complex data structures. Here are some of the simpler methods:

- *Copy and paste from a text document.* To see how this works, open a Word document or Notepad, and type this data, including commas: 5,3,2,6,7. Use CTRL/C to copy and CTRL/V to paste into

```
> x=c(
```

then close parentheses and enter.

- *Use the* `data.entry()` *function*. This function allows you to edit values of an existing vector but it does not create a new vector. Try

```
> data.entry(x)
```

This opens up a data entry window that looks like a spreadsheet. You can edit existing entries of `x` and add new values. Once this window is open, you can also add new variables. Experiment with this.

Close the window by clicking on the X in the upper-right corner of the window. Check that your new data is saved.

If the data.entry window is closed and you want to use it to enter a new variable, first establish the variable with an assignment statement, for example,

```
> my.new.data=c(1)
```

Now you can apply `data.entry(my.new.data)`.

- *Use the* `scan()` *function*. Using the `c()` function requires you to format the data with comma separation. The `scan()` function will read what you paste in, and it requires separation by spaces. To demonstrate, in Word or Notepad, type 4 5 6 7 8. Use CTRL/C to copy. Now in R, enter

```
> y=scan()
```

R will provide a prompt for the data. Use CTRL/V to paste in the data and press Enter. R gives another prompt. Press Enter to terminate the scan. The interaction should look like this:

```
> y=scan()
1: 4 5 6 7 8
6:
Read 5 items
>
```

Warning: The scan function does not accept commas or tabs as separators.

The scan function will read data from a file when given the file name. For example, suppose you have a set of ages saved in the file named *age.txt* in your working directory. The data can be listed in a column or across the same line, but should be separated by spaces, not commas. You can access the data with this command:

```
> age=scan(file="age.txt")
```

- Also see `read.table()` in the next section.

S10.1.5 Saving and Retrieving the Workspace

An R workspace is a portion of computer memory where objects such as data vectors are kept while R is running. Before you close an R session, you should explicitly save the workspace (which contains all your variables, data vectors, etc.) so that you can retrieve it later. Click on the floppy disk icon in the toolbar above the R console. On a Mac, use the menu named "workspace" and the option to "save workspace file." Locate the folder where you want to save the workspace, and assign a file name to the workspace. Use the extension *.RData*. For example, save the workspace as *Chapter1Data.RData*.

It is a good practice to save the workspace periodically even if you aren't exiting the program, because in the event of a system crash or power outage, what hasn't been saved is lost.

Using separate workspaces for different data analysis projects can help reduce clutter and improve organization.

When you're ready to end your session with R, type `q()` to quit.

When you want to access the workspace later, open R, click on the "load workspace" icon, and click on the workspace file. On a Mac, you can choose "load workspace file" under the "workspace" menu. You may have to wait while the workspace loads—it may seem slow.

Exercise 10.1. Here is data on the length of stay (days) in the hospital for a sample of 20 patients who have digestive system disorders:

1	4	3	2	7	5	2	2	4	5
3	3	4	5	1	1	2	3	4	1

1.1 Enter this data into a vector named `stay`. There is a function named `length` in R. Find the result of

```
> length(stay)
```

1.2 Use R to find the mean length of hospital stay.

1.3 The deviation of a value x from the sample mean is $x - \bar{x}$. Calculate a vector of the deviations from the sample mean of the lengths of stays using a vector operation.

1.4 Find the sum of the deviations from the mean. In theory, this sum should be 0. Why would the result from R be different from 0?

1.5 Find the sum of the squared deviations from the mean.

1.6 Add two new observations to the dataset, 1 and 12. Find out how the mean and the sum of squared deviations from the mean are affected by adding these new data values. □

S10.2 DESCRIPTIVE STATISTICS AND GRAPHICAL DISPLAYS FOR UNIVARIATE DATA

Triglycerides are a type of fat found in your blood. The body uses them for energy and they are necessary for good health, but high triglycerides can raise your risk for heart disease. Normal levels are less than 150 mg/dL, while 200 mg/dL or above is considered high. The blood triglyceride levels of 90 adults are listed in Appendix B. What is a typical, or average, value in this dataset? How much variability does the data have? What is the shape of the distribution? Are there any unusual values? These questions are easy to answer with R.

S10.2.1 Graphical Displays: Stem-and-Leaf, Histogram, Boxplot

A good starting point for investigating the data is to create a graphical display. The triglycerides dataset is also available as a file named *triglyceride.txt* on this volume's website. Read the data into a vector called `trigly` using the `scan()` function, and copy and paste.

The `stem()` function produces a stem-and-leaf plot. Try it out:

```
> stem(trigly)
```

The result is

```
The decimal point is 2 digit(s) to the right of the |

0 | 34444
0 | 55555556666666777777777777888899999999
1 | 0000111122222333444
1 | 5555566788888
2 | 0112
2 | 5666
3 | 0
3 | 67
4 | 1
```

We immediately see that the distribution is not symmetric, that it is skewed to the right. We can also see that more than half of the measurements are in the "normal" range, under 150 mg/dL.

We might prefer to make a histogram, particularly if the display will be used in a report. The histogram function in R has many options, but the simplest use of the command is brief:

```
> hist(trigly)
```

See Figure S10.1 for the resulting graph. Use the File Menu to see options for printing and saving a graph.

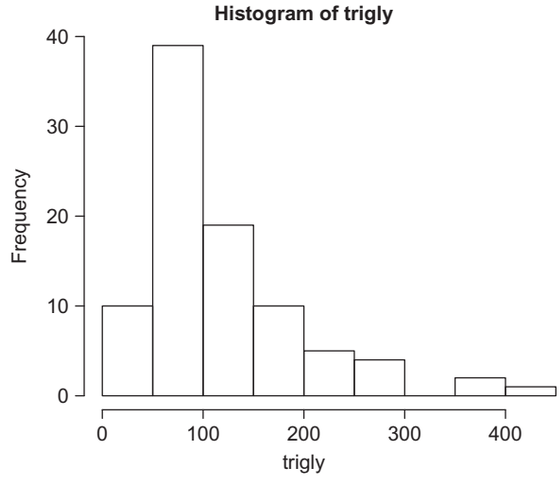

FIGURE S10.1 Histogram of triglyceride levels.

If you prefer to have a relative frequency histogram rather than a frequency histogram, use

```
> hist(trigly, prob=TRUE)
```

We can change the number of bins in the graph by specifying the number of breaks. See, for example, the result of

```
> hist(trigly, breaks=15)
```

You may prefer to specify a title and/or axis labels. Try this example:

```
> hist(trigly, main = "Histogram of triglyceride levels",
xlab="triglycerides")
```

You may use breaks, main, prob, xlab, and other options not explored here, in any combination you choose. For further options, see

```
> help(hist)
```

One useful option is to scale the labeling in the histogram larger or smaller using the cex (character expansion) specification. An example of its usage is

```
> hist(trigly, cex.lab=1.5)
```

This makes the axis labels 1.5 times as large as the default setting. Use `cex` value less than one to make the labels smaller.

Warning: If you copy and paste commands from Word into R, R will not recognize left-quotation and right-quotation marks from the formatting of Word. It might be easier to use Notepad instead of Word.

A boxplot can help you visualize both the median and the distribution of the dataset. Create a boxplot of the triglycerides data using

```
> boxplot(trigly, ylab="triglyceride measurements")
```

S10.2.2 Summary Statistics

A quick way to calculate several statistics at once is the summary() function. For example, the command

```
> summary(trigly)
```

produces the following

```
Min.  1st Qu.  Median  Mean   3rd Qu.  Max.
31.0  66.0     92.0    118.5  149.0    411.0
```

Table S10.1 gives functions for additional statistics.

TABLE S10.1 Some Useful Functions for Calculating Summary Statistics, Covariance, and Correlation in R

Statistic	R Function
Five-number summary	`fivenum()`
Median	`median()`
Mean	`mean()`
Variance	`var()`
Standard deviation	`sd()`
90th percentile of *x* Works for any percentile $0 < p < 1$	`quantile(x, .90)`
Interquartile range	`IQR()`
Trimmed mean of *x* `trim` is the proportion of data deleted from each end of the distribution	`mean(x, trim=.1)`
Covariance of *x* and *y*	`cov(x,y)`
Correlation between *x* and *y*	`cor(x,y)`

S10.2.3 Categorical Data

Suppose you want to create a categorical data vector named `vote` to record the responses to a survey question, and you use the values Y, N, U for "yes," "no," and "undecided." If the vector is short enough, you may want to simply type in the values

```
> vote=c("Y", "N", "N", "Y", "U", "Y", "N")
```

The quotation marks are necessary to distinguish strings of characters from variable names. Typing even a short list like the one above is a tedious task. You don't have to type the quotation marks if you use `data.entry(vote)`.

Unfortunately, the `scan()` function needs quotation marks around character strings.

S10.2.4 Creating a Data Frame With `read.table()`

There is another way to import data, which involves creating an object called a data frame. A data frame is used to store a rectangular grid of data. Each column of the data frame is a data vector and (usually) each row contains the values of the variables for one experimental unit or subject. To see how this works, consider the dataset called "adults" in Appendix B. Copy and paste this data into a separate text file, and call it *adults.txt*. You may also download the *adults.txt* data file from this volume's website. Save the file in your working directory for R.

Import this data into R using the following command:

```
> adults=read.table(file=file.choose(), header=TRUE)
```

The function `file.choose()` allows you to locate your data file on your computer. The option `header=TRUE` alerts the software that the first line of the text file contains variable names.

To verify the variable names, use

```
> names(adults)
```

Check the number of rows and columns using

```
> dim(adults)
```

We see that there is data for 198 people, and there are six columns.

Each variable is referenced by its name *preceded by the name of the data frame and a $ symbol*. For example, the line

```
> adults$race
```

displays all the values in the column for race.

To shorten a name, you can create a new data vector for that variable. For example,

```
> race=adults$race
```

S10.2.5 Frequency Table

Clearly, race is a categorical variable. You can create a frequency table for the variable as follows:

```
> table(race)      #  or use > table(adults$race)
```

S10.2.6 Barplot

The `barplot()` function is not as user-friendly as the `histogram()` function. For one thing, it needs summarized data rather than raw data. See what results from these two commands:

```
> barplot(race)    #this produces an error message
> barplot(table(race))
```

We can add a title:

```
> barplot(table(race),main="race distribution")
```

Additional options include `ylab` (*y*-axis label) and `xlab` (*x*-axis label).

S10.2.7 Logical Valued Vectors

In R, `TRUE` and `FALSE` are logical constants, and a vector is a logical valued vector when its entries are logical constants. Usually, such a vector is created using a logical expression like `x > 8`. For example, if we have

`x = c(18,14,3,9,7)` and assign
`> y = (x>8)`

then the values of `y` are

```
> y
[1]  TRUE  TRUE FALSE  TRUE FALSE
```

Logical comparison operators for use in R are listed in Table S10.2.

TABLE S10.2 Logical Comparison Operators in R

<	Less than
<=	Less than or equal to
>	Greater than
>=	Greater than or equal to
==	Equal to
!=	Not equal to

Notice in particular that "is equal to" used for comparisons is denoted by == , which is different from the single = used for assigning values. As one example of the difference,
`> y = c(18, 14, 3, 9, 7)` makes y a numerical vector, while
`> y == c(18, 12, 5, 9, 10)` produces a sequence of TRUEs and FALSEs.
Logical vectors may be combined with the operators `&` (and), | (or), and `!` (not). For example,

```
> (x>8)&(x<13)
[1] FALSE FALSE FALSE  TRUE FALSE
```

A quick way to count the TRUEs in a logical vector is by using the `sum()` function. When the input to `sum()` is a logical vector, the function is evaluated by giving each TRUE the value of 1 and each FALSE the value of 0. Examples:

```
> sum( x >=10)
[1] 2
```

To find how many of the adults in our dataset are age 50 or over, use

```
> sum(adults$age >=50)
[1] 17
```

To find how many females in the dataset have BMI less than 25,

```
> sum((adults$BMI < 25)& (adults$gender=="f"))
[1] 26
```

Additional uses of logical valued vectors are illustrated in the exercises.

Exercise 10.2.

2.1 In the "adults" dataset, use R to find the proportion of the subjects who have normal levels of triglycerides (under 150 mg/dL). What proportion have high levels (200 or higher)?

2.2 Compare the 10% trimmed mean, the median, and the mean of the triglyceride levels. How is the shape of the distribution related to the differences (or similarities) in these values?

2.3 In the "adults" dataset, one of the variables is BMI, or body mass index. Do a quick Internet search to find out what BMI is, if you are not familiar with it.

a. What would you expect the shape of the distribution of BMI to be? Test your conjecture by making a histogram of the BMI data.

b. Compare the histogram of BMI data using 20 breaks and the histogram with 10 breaks. Does the shape of the distribution appear to change?

2.4 Create a boxplot of the BMI data. The boxplot function identifies outliers. You can get the values of the outliers using

```
> boxplot(adults$BMI)$out
```

List the outliers.

2.5 The "adults" dataset includes the variable `gender`. Here is a command that will create comparative boxplots of BMI for males and females:

```
> boxplot(adults$BMI~adults$gender)
```

How do the BMI distributions for males and females compare?

2.6 The BMI values for males and females can be stored in two separate variables with the help of a logical variable. The command line

```
> BMI.male=adults$BMI[adults$gender=="m"]
```

assigns to the vector `BMI.male` only those entries of `adults$BMI` for which the logical statement `adults$gender=="m"` is true.

a. Create `BMI.male` and use a similar construction to create a variable `BMI.female`.

b. Find descriptive statistics for BMI for males and females. Does a comparison of these statistics agree with your comparison based on boxplots?

Note: Now that you have separated BMI data for males and females into two separate vectors, you can also create comparative boxplots using

```
> boxplot(BMI.male, BMI.female, names=c("men","women"))
```

2.7 In the "adults" dataset, the variable `smoke` is categorical, coded 0 for a nonsmoker, 1 for a smoker, and 2 for someone who chews tobacco. Make a frequency table and a barplot for this variable.

2.8 Are males in the "adults" dataset more likely to smoke than females? Use the following command to create a two-way table to investigate this question:

```
> table(adults$smoke,adults$gender)
```

□

S10.3 SIMPLE LINEAR REGRESSION

S10.3.1 Scatterplot and Fitted Line Plot

Here are the heights of eleven female students along with their "midparent" height (the average of the heights of the mother and father):

midparent	daughter
66.0	64.0
65.5	63.0
71.5	69.0
68.0	69.0
70.0	69.0
65.5	65.0
67.0	63.0
70.5	68.5
69.5	69.0
64.5	64.0
67.5	67.0

Read the data into a table from the text file *heights.txt*, available on this volume's companion website:

```
> heights=read.table(file=file.choose(),header=TRUE)
> heights
   midparent daughter
1       66.0    64.0
2       65.5    63.0
3       71.5    69.0
4       68.0    69.0
5       70.0    69.0
6       65.5    65.0
7       67.0    63.0
8       70.5    68.5
9       69.5    69.0
10      64.5    64.0
11      67.5    67.0

> midparent=heights$midparent
> daughter=heights$daughter
```

To see a scatterplot of the data, we use

```
> plot(midparent, daughter)
```

In the `plot()` function, the order of the arguments is (explanatory variable, response variable).
We can provide a title:

```
> plot(midparent, daughter, main = "heights in inches")
```

We can find the least-squares regression line with the linear model function, `lm()`. Input to `lm()` is a model formula in the form "response variable ~ explanatory variable."

```
> lm(daughter~midparent)

Call:
lm(formula = daughter ~ midparent)

Coefficients:
(Intercept)  midparent
     1.6497     0.9555
```

We see that the least-squares regression line is

```
daughter = 1.6497 + 0.9555 midparent
```

To add the least-squares line to the scatterplot, use

```
> abline(1.6497,0.9555)
```

The name "abline" is pronounced "$a - b$ line." The intercept is a, the slope is b, and the linear equation is expressed as $y = a + bx$.

S10.3.2 Correlation

You can calculate the Pearson correlation coefficient with the cor() function:

```
> cor(midparent,daughter)
[1] 0.8503884
```

S10.3.3 Test of Significance

When the linear model function estimates the least-squares regression equation, it also computes a great deal of additional information that is accessible if we store the results; simply assign a name to the output of the function. Then apply the `summary()` function.

```
> Ht=lm(daughter~midparent)
> Ht

Call:
lm(formula = daughter ~ midparent)

Coefficients:
(Intercept)   midparent
    1.6497       0.9555

> summary(Ht)

Call:
lm(formula = daughter ~ midparent)

Residuals:
    Min      1Q   Median      3Q     Max
-2.6707 -0.8429   0.4627  0.8071  2.3737

Coefficients:
            Estimate   Std. Error  t value Pr(>|t|)
(Intercept) 1.6497   13.3632   0.123  0.90446
midparent   0.9555    0.1971   4.849  0.00091 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 1.451 on 9 degrees of freedom
Multiple R-squared: 0.7232,     Adjusted R-squared: 0.6924
F-statistic: 23.51 on 1 and 9 DF,  p-value: 0.00091
```

In the output from `summary()` above, the model is restated, followed by a five-number summary of the residuals.

Under the heading "Coefficients," R gives the least-squares estimate of the slope and intercept, the standard error of each estimate, a t-statistic used to test for statistical significance, and the two-sided p-value for testing the null hypothesis that the coefficient is zero. We see that the p-value for the coefficient of midparent is 0.00091, so we conclude that the linear relation is statistically significant.

Additional statistics include the Multiple R-squared, Adjusted R-squared, and the Analysis of Variance F-statistic.

S10.3.4 Linear Regression with Two Predictors

Linear regression techniques are readily extended to models with multiple predictor variables. The simplest of these has two predictors, call them X_1 and X_2, and the model is

$$Y = \beta_0 + \beta_1 X_1 + \beta_2 X_2 + \in$$

In order to use data to estimate the coefficients in this model, it is necessary to specify the model in the linear models function of R. The model statement in R is a code that tells how the given predictors are used in the

model. The model statement for the model above is

$$\text{formula} = Y \sim X_1 + X_2$$

For example, in the "adults" dataset, let's investigate whether triglyceride levels are related as a linear function of a person's age and BMI. Save the output from the `lm()` function as `trigly.model1`:

```
> trigly.model1=lm(formula=adults$trigly~adults$age + adults$BMI)
```

To view the results, apply the `summary()` function:

```
> summary(trigly.model1)

Call:
lm(formula = adults$trigly ~ adults$age + adults$BMI)

Residuals:
    Min      1Q  Median      3Q     Max
-156.75  -47.14  -20.11   23.58  689.75

Coefficients:
             Estimate  Std. Error   t value   Pr(>|t|)
(Intercept) -190.5044   53.6368      -3.552    0.00048 ***
adults$age     3.0651    0.8056       3.805    0.00019 ***
adults$BMI     7.3523    1.6998       4.325    2.43e-05 ***
---
Signif. codes: 0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

Residual standard error: 95.97 on 195 degrees of freedom
Multiple R-squared: 0.1551, Adjusted R-squared: 0.1465
F-statistic: 17.9 on 2 and 195 DF, p-value: 7.284e-08
```

From this output, the estimated relationship is

$$\widehat{\text{triglyceride}} = -190.5044 + 3.0651\ \text{age} + 7.3523\ \text{BMI}$$

S10.4 MISSING VALUES

In multivariable datasets, it sometimes happens that values of some variables are missing for some observations. R denotes missing values by NA. Different sources of data will use different conventions for indicating missing values. For example, missing values in Minitab are represented by an asterisk.

Leaving a blank space where a value is missing will create an error message when the data is scanned into R. For example, suppose the data file contains this:

```
height weight
66       123
68
72       185
65       139
```

Here's what happens when we attempt to import this:

```
> example=read.table(file=file.choose(),header=T)
Error in scan(file, what, nmax, sep, dec, quote, skip, nlines, na.strings, :
  line 2 did not have 2 elements
```

Let's try putting in an asterisk for the missing value:

```
height   weight
66          123
68          *
72          185
65          139
```

Now the data gets read in without an error message, but the second column is interpreted as nonnumeric. This won't be obvious until a calculation is attempted:

```
> example
   height   weight
1     66       123
2     68         *
3     72       185
4     65       139

> example$weight/example$height
[1] NA NA NA NA
Warning message:
In Ops.factor(example$weight, example$height) :
  / not meaningful for factors
```

So let's try using NA for the missing value in the text file. The data is imported without an error message. The component-wise calculation attempted before now works wherever a data point is not missing:

```
> example
  height weight
1     66    123
2     68     NA
3     72    185
4     65    139
> example$weight/example$height
[1] 1.863636       NA 2.569444 2.138462
```

NA is produced whenever a calculation is attempted with the *weight* vector:

```
> sum(example$weight)
[1] NA
```

We can work around this handicap with the option `na.rm=TRUE`, which removes cases with missing values from the calculation:

```
> sum(example$weight,na.rm=TRUE)
[1] 447
```

Many, but not all, R functions have the argument na.rm, and in some, `na.rm=TRUE` is the default argument. Here are some examples:

The `plot`() function has `na.rm=TRUE` as the default:

```
> plot(example$height,example$weight) #produces a plot with 3 points
```

However, the cor() function doesn't accept the na.rm argument:

```
> cor(example$height,example$weight)
[1] NA
> cor(example$height,example$weight,na.rm=TRUE)
Error in cor(example$height, example$weight, na.rm = TRUE) :
unused argument(s) (na.rm = TRUE)
```

The covariance function as well as the correlation function allow the option of using only observations that have values for both the variables involved:

```
> cov(x,y, use="complete.obs")
```

In summary, missing values should be represented in data by NA before the data is imported into R. But for some purposes, it may also be necessary to simply delete observations with missing values from the dataset.

S10.5 MATRICES, COVARIANCE MATRICES, AND MATRIX MULTIPLICATION, BRIEFLY

There are various ways to build matrices in R. For the sake of brevity, only one method will be described here. It consists of binding equal-length vectors of data together as columns. If, for example, we want to use the three vectors `x1`, `x2`, and `x3` to form a matrix called G, the appropriate command is

```
> G=cbind(x1, x2, x3)
```

The vectors `x1`, `x2`, and `x3` are still distinct objects in the workspace, and G is a new object.
Try it out:

```
> x1=c(1,2,3,4,5)
> x2=c(3,1,3,1,3)
> x3=c(2,3,4,4,3)
> G=cbind(x1,x2,x3)
> G
     x1 x2 x3
[1,]  1  3  2
[2,]  2  1  3
[3,]  3  3  4
[4,]  4  1  4
[5,]  5  3  3
```

The function `dim()` returns the number of rows and number of columns of the input.

```
> dim(G)

[1] 5 3
```

A quick way to calculate the covariance matrix of a set of variables is to bind the variables into a matrix and then apply the `cov()` function. For the example above, we get

```
> cov(G)
     x1      x2     x3
x1   2.50    0.0    0.75
x2   0.00    1.2   -0.30
x3   0.75   -0.3    0.70
```

The symbol for matrix multiplication in R is `%*%`. If the operation is used to multiply a matrix times a vector, R will treat the vector as either a column or a row vector—whichever is conformable for the multiplication.

For example, let `z = c(10, 5, 15)`. Our matrix *G* above is 5 × 3, so we can multiply in the order `Gz` but not in the order `zG`. On the other hand, `cov(G)` is 3 × 3, so both `zG` and `Gz` can be calculated.

```
> z = c(10, 5, 15)
> G %*% z
       [,1]
```

```
[1,]     55
[2,]     70
[3,]    105
[4,]    105
[5,]    110
> z %*% G
Error in z %*% G : non-conformable arguments

> cov(G) %*% z
        [,1]
x1     36.25
x2      1.50
x3     16.50
> z %*% cov(G)
        x1  x2   x3
[1,] 36.25 1.5 16.5
```

Usually, matrix multiplication gives different results if order is reversed; that is, $AB \neq BA$. However, because `cov(G)` is a symmetric matrix, `z %*% cov(G)` and `cov(G) %*% z` are equal component-wise, although one is a row vector and the other is a column vector.

Matrix inversion: The inverse of a matrix M is `solve(M)`.

For example, to find the inverse of `cov(G)`, use `solve(cov(G))`.

```
> cov(G)
         x1   x2    x3
x1     2.50  0.0  0.75
x2     0.00  1.2 -0.30
x3     0.75 -0.3  0.70
> solve(cov(G))
         x1         x2     x3
x1   0.6250 -0.1875000 -0.750
x2  -0.1875  0.9895833  0.625
x3  -0.7500  0.6250000  2.500
```

Solving $Y = MX$

If X and Y are vectors of length k and M is a k-by-k matrix, the matrix equation $Y = MX$ represents a system of linear equations, which has a unique solution $X = M^{-1}Y$ if M is invertible. The quickest way to find the solution in R is with the function `solve(M,Y)`, which returns the solution X. An alternative method is to invert M using the `solve()` function and use matrix multiplication. Both methods are demonstrated below.

```
> M=cov(G)
> M
     x1   x2   x3
x1 2.50  0.0  0.75
x2 0.00  1.2 -0.30
x3 0.75 -0.3  0.70
> Y=c(2,5,3)
> solve(M,Y)
[1] -1.937500 6.447917 9.125000
> solve(M) %*% Y
        [,1]
x1 -1.937500
x2  6.447917
x3  9.125000
```

APPENDIX A DOWNLOADING R TO YOUR COMPUTER

The main page for the R Project is http://www.r-project.org/.

Locate the box titled "Getting Started" and click on the link to Cran Mirrors. The download will be faster if you choose a location closer to you.

From the box titled "Download and Install R," click on the version (Linux, MacOS X, or Windows) that you need. If you are using Windows, the next step is to click on the link to the "base" subdirectories. "Base" is the essential part of the R software; many additional packages are available for specialized computations and functions. One or two more clicks will download the software.

APPENDIX B DATASETS

The datasets can be downloaded from the volume's companion website. The names of the files are given in parentheses below.

TRIGLYCERIDE LEVELS OF 90 ADULTS (TRIGLYCERIDE.TXT)

```
55 177 57 41 38 61 57 86 95 248 117 178 264 70 65 115 355 122 64 70 68 79 139 153 73
74 150 75 114 39 57 259 151 71 66 164 182 75 65 31 176 66 411 133 88 86 42 166 264 68
68 223 49 47 99 93 131 53 67 210 88 115 118 54 210 48 130 91 57 299 145 83 106 155 87
144 49 106 99 140 176 69 201 47 104 114 66 372 146 85
```

ADULTS (ADULTS.TXT)

age	gender	race	BMI	smoke	triglyceride
49	f	b	28.1889	0	97
46	m	b	26.8507	0	161
42	m	b	20.6721	1	90
35	f	b	24.5826	0	78
28	f	b	29.5205	1	73
48	m	w	21.8603	0	171
36	f	b	25.8599	1	52
42	m	w	26.5418	1	72
26	m	b	31.3969	0	90
29	f	b	21.1316	0	62
28	m	b	25.1967	1	42
34	m	b	26.6853	0	88
37	m	b	26.3176	0	56
50	m	b	26.1201	1	134
41	f	b	24.2658	0	57
42	f	b	20.8198	1	57
49	m	w	25.3286	0	118
34	m	b	32.9980	0	82
28	f	b	25.1019	0	83
46	f	b	31.4084	0	98
41	f	b	26.5180	1	45
29	f	b	27.3167	0	78
44	f	w	24.8535	0	85
25	m	w	26.2549	1	57
41	m	bh	24.1385	0	70
24	m	b	24.2635	1	40
48	f	w	28.8710	0	103
55	m	w	27.3167	0	179

40	f	w	26.8837	0	144
50	f	b	28.6299	0	92
27	m	w	26.8739	1	375
56	m	w	26.8088	0	187
35	m	w	52.6034	0	340
40	f	b	22.3120	0	49
39	m	b	22.7230	1	74
34	f	b	26.9553	0	46
36	f	b	27.2495	1	83
29	m	b	25.0647	0	87
42	f	w	20.3587	0	76
36	f	b	27.9854	1	43
37	m	b	30.7129	0	81
23	m	w	22.9570	0	63
30	f	w	28.0715	0	90
41	m	w	26.1355	0	158
46	m	b	36.3509	1	61
43	m	b	29.8341	1	169
29	f	b	20.1333	0	87
30	m	w	24.0683	0	114
50	f	b	24.9586	0	43
28	m	w	24.0212	1	103
40	m	b	33.2231	0	103
44	f	b	29.1903	0	59
32	m	w	24.4097	0	132
38	m	w	32.7723	1	257
52	m	b	21.2335	1	166
34	m	b	24.8066	0	53
50	m	w	24.9637	0	355
43	m	w	27.5179	1	411
34	m	w	27.7585	0	88
41	f	b	31.5972	1	50
40	m	b	24.4773	0	99
37	m	b	21.7681	1	98
51	m	b	24.3253	0	105
34	m	b	27.6014	1	65
42	m	w	33.1414	0	297
28	f	w	22.1288	1	259
35	m	b	26.7795	1	62
34	m	b	28.8848	0	201
26	f	b	20.7820	0	80
39	m	b	25.9882	0	80
43	m	w	27.9740	0	80
25	m	b	30.7106	0	38
48	m	b	32.7615	0	123
31	m	b	28.7933	1	77
51	m	w	26.7756	0	80
23	f	b	25.8398	0	95
41	m	w	29.8416	0	113
38	m	w	30.7373	0	98
28	f	w	22.1100	0	96
26	m	b	25.8398	0	199
32	m	w	31.2382	1	199
27	m	w	31.2382	0	96

```
45      f       w       30.9965       0       96
36      m       w       33.1216       1       154
45      m       w       24.5371       0       100
44      f       b       20.7754       0       53
44      f       b       26.6028       2       100
34      m       w       29.1464       1       117
29      m       w       26.6362       0       228
50      m       w       26.8311       0       67
36      m       b       23.6474       1       70
30      m       b       23.6474       1       62
39      m       w       27.3617       0       220
43      m       b       27.5712       0       195
44      m       w       31.4614       1       191
32      m       b       25.9680       1       69
42      m       b       31.6284       0       236
30      m       w       27.0435       1       98
50      m       w       26.7795       0       88
33      m       b       30.2973       0       149
28      m       w       32.0946       0       130
42      m       w       27.7191       2       182
31      m       w       24.8166       1       115
30      m       w       33.2952       0       115
32      m       b       28.4980       0       94
23      m       w       24.6486       0       108
32      m       b       26.5795       0       114
29      m       b       22.2400       1       64
49      m       w       27.3446       0       123
47      m       w       28.7933       0       855
38      f       b       20.5983       1       105
37      f       w       22.6291       0       93
26      m       w       26.6288       1       125
25      f       b       22.3120       0       126
38      m       w       23.6208       0       69
39      m       b       32.2230       1       120
32      m       b       26.4537       0       68
26      f       b       22.6774       0       63
33      m       w       32.8480       0       131
52      m       w       26.1114       0       96
36      m       b       28.1200       1       209
45      m       w       28.3325       0       54
44      m       w       29.6554       1       171
38      f       b       24.1450       1       140
24      m       w       29.2859       0       79
30      m       w       25.4503       0       36
37      m       b       37.8562       0       210
40      m       w       26.9151       0       57
42      m       b       28.6136       0       159
37      m       b       31.3694       0       98
45      f       w       24.5432       0       194
39      m       b       25.6832       0       65
44      m       b       29.8341       0       210
33      m       b       30.2248       0       73
23      m       b       22.2964       0       84
36      m       b       28.8299       0       61
```

49	m	w	29.6464	0	153
48	f	b	32.4462	0	316
41	m	b	27.5461	0	109
28	m	b	28.3424	1	83
49	m	b	32.5842	0	176
55	m	w	28.7856	0	137
29	f	b	28.7856	0	65
30	m	b	28.6939	0	65
35	m	w	29.6292	0	80
35	f	b	26.6058	0	80
33	m	b	33.8267	0	147
48	m	b	31.5988	0	413
40	m	w	28.6266	1	183
43	f	b	20.9851	0	97
31	m	h	26.1530	0	85
30	m	w	28.2427	0	93
45	m	w	26.0370	1	80
30	f	b	33.4675	1	84
34	m	w	27.3074	0	123
23	f	b	22.7955	0	56
30	m	w	20.6721	1	151
39	m	b	24.6767	1	121
31	m	b	28.9719	0	60
30	f	b	28.9719	0	65
28	f	w	23.0432	0	70
22	m	w	34.2567	0	112
23	f	b	27.6311	1	123
58	f	b	25.0568	1	103
37	m	b	17.4134	0	75
49	m	w	29.0886	1	506
54	m	b	35.0065	0	874
35	m	b	29.5648	0	87
35	m	w	27.9074	0	100
22	f	w	26.8152	0	49
39	m	b	25.1071	1	71
45	m	w	26.3083	1	101
34	m	w	29.8437	0	142
35	m	w	29.6982	0	125
26	m	w	29.5337	0	80
44	m	b	29.1464	0	151
37	m	b	30.3814	0	127
45	m	b	25.7243	0	101
29	f	b	21.1416	0	72
23	m	w	38.1021	0	115
39	m	w	24.9637	0	57
52	f	w	25.8402	0	71
26	m	w	32.9980	0	161
34	f	b	20.6575	0	104
24	m	h	27.4027	0	150
48	m	b	28.6457	0	71
37	m	w	25.2914	0	139
30	f	b	26.4537	0	91
34	f	w	22.9985	0	72
42	m	b	29.1243	1	103

```
54      m       b       29.2964       0       147
29      m       w       23.3248       0       55
48      m       w       28.1200       0       49
37      f       b       26.6225       1       162
34      m       b       24.1266       0       49
30      m       b       27.4730       0       181
22      f       b       24.3716       0       52
24      m       w       25.5448       1       93
```

APPENDIX C ANSWERS TO EXERCISES

1.1
```
> stay=c(1,4,3,2,5,7,5,2,2,4,3,3,4,5,1,1,2,3,1,4)
> length(stay)
  [1] 20
```
1.2
```
> mean(stay)
  [1] 3.1
```
1.3
```
> deviation=c(stay-3.1)
> deviation
  [1] -2.1 0.9 -0.1 -1.1 1.9 3.9 1.9 -1.1 -1.1 0.9 -0.1 -0.1 0.9 1.9 -2.1 -2.1
  -1.1 -0.1 -2.1 0.9
```
1.4
```
> sum(deviation)
  [1] -1.776357e-15
```
The result is extremely close to 0, but not exactly 0, due to rounding errors that generate a very tiny deviation from 0.

1.5
```
> deviationsq=c(deviation^2)
> deviationsq
  [1] 4.41 0.81 0.01 1.21 3.61 15.21 3.61 1.21 1.21 0.81 0.01 0.01 0.81 3.61 4.41
4.41 1.21 0.01 4.41 0.81
> sum(deviationsq)
  [1] 51.8
```
1.6
```
> newstay=c(stay,c(1,12))
> mean(newstay)
  [1] 3.409091
> newdeviationsq=c((newstay-3.409091)^2)
> newdeviationsq
 [1] 5.8037194 0.3491734 0.1673554  1.9855374 2.5309914 12.8946274
 [7] 2.5309914 1.9855374 1.9855374  0.3491734 0.1673554  0.1673554
[13] 0.3491734 2.5309914 5.8037194  5.8037194 1.9855374  0.1673554
[19] 5.8037194 0.3491734 5.8037194 73.8037174}
> sum(newdeviationsq)
[1] 133.3182
```

The new values increase the mean a little, but increase the sum of the squared deviation from the mean considerably.

2.1
```
> sum(adults$triglyceride<150)
[1] 154
> sum(adults$triglyceride>=200)
[1] 19
```
2.2
```
> mean(adults$triglyceride, trim=0.1)
[1] 104.5563
```

```
> median(adults$triglyceride)
[1] 96
> mean(adults$triglyceride)
[1] 123.2273
```

Comparison of the three summary statistics reveals that the distribution has some large outliers to the right, which makes the mean greater than the median and the 10% trimmed mean.

2.3 **a.** `> hist(adults$BMI)`

b. `> hist(adults$BMI, breaks=20)`

Making a histogram with 20 breaks generates a picture of the distribution that looks closer to normal (see Figures S10.2 and S10.3).

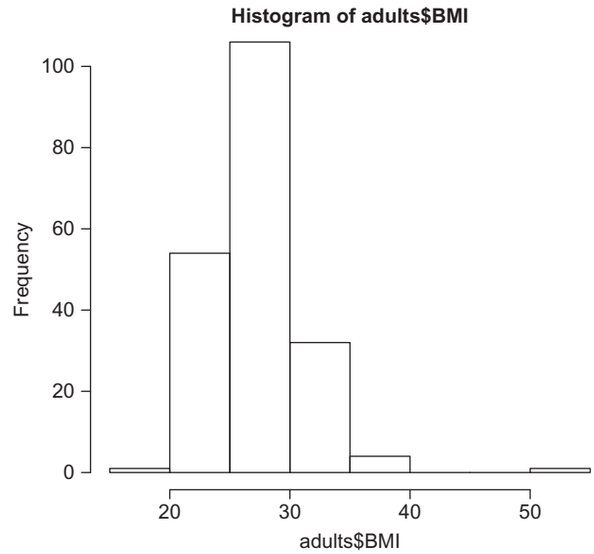

FIGURE S10.2 Histogram of BMI; answer to Exercise 2.3.a.

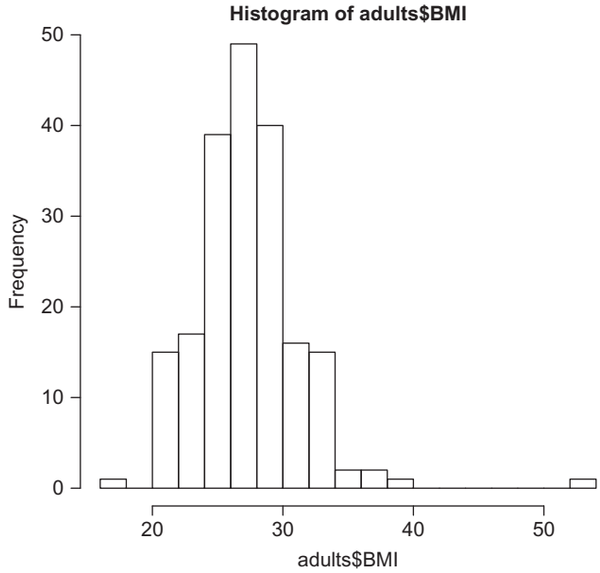

FIGURE S10.3 Histogram of BMI with 20 breaks; answer to Exercise 2.3.b.

2.4 `> boxplot(adults$BMI, ylab="BMI")`
See Figure S10.4.

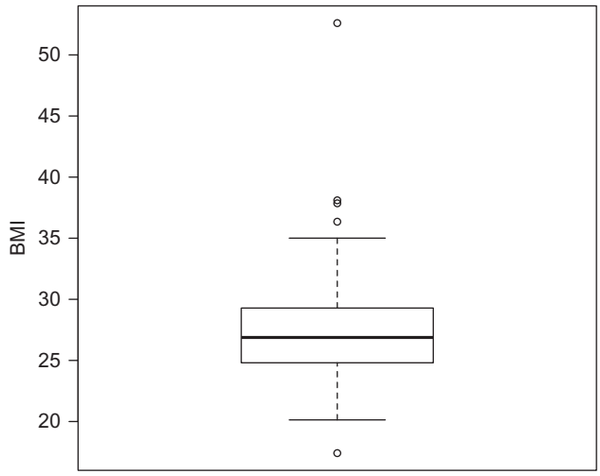

FIGURE S10.4 Boxplot of BMI; answer to Exercise 2.4.

```
> boxplot(adults$BMI)$out
[1] 52.6034 36.3509 37.8562 17.4134 38.1021
```

2.5 See Figure S10.5.

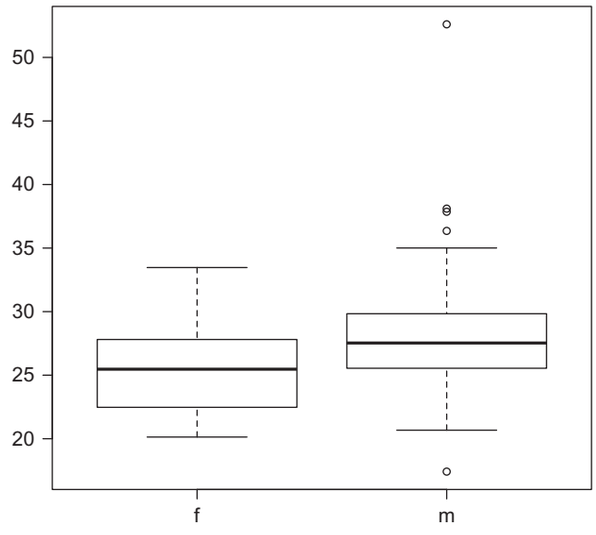

FIGURE S10.5 Boxplot of BMI by gender; answer to Exercise 2.5.

Women have lower triglyceride levels on average than men, and all of the outliers were men.

2.6 **a.** `> BMI.female=adults$BMI[adults$gender=="f"]`

b. `> summary(BMI.male)`

```
Min. 1st Qu. Median Mean 3rd Qu. Max.
17.41 25.58 27.53 27.98 29.80 52.60
```

```
> summary(BMI.female)
Min. 1st Qu. Median Mean 3rd Qu. Max.
20.13 22.55 25.47 25.42 27.72 33.47
```

Yes, women have a lower median and mean BMI, and the min and the max are less extreme in women.

2.7
```
> table(adults$smoke)

  0   1   2
140  56   2
> barplot(table(adults$smoke), ylab="Number of people")
```

See Figure S10.6.

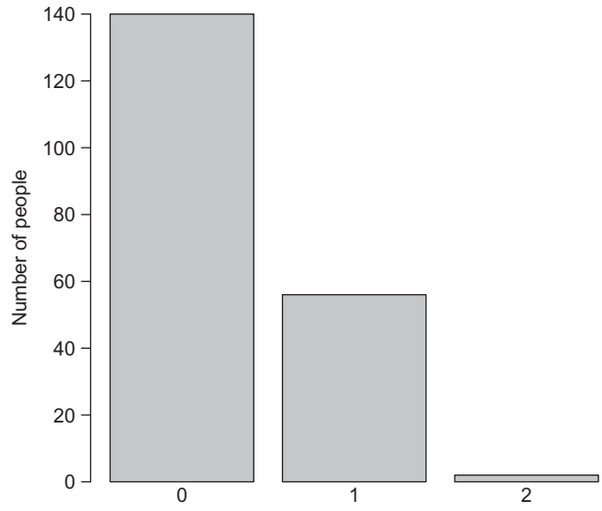

FIGURE S10.6 Barplot of the number of tobacco users; answer to Exercise 2.7.

2.8
```
   f  m
0 41 99
1 14 42
2  1  1
```

A slightly smaller proportion of females were smokers; females are a little bit less likely to smoke.

REFERENCES

[1] R Development Core Team. R: A Language and Environment for Statistical Computing. Vienna, Austria; R Foundation for Statistical Computing. http://www.R-project.org/.

[2] Verzani J. Using R for introductory statistics. Boca Raton, FL: CRC Press; 2014.