1

Getting Started

The R statistical software environment is a powerful and flexible platform for performing statistical analyses. The basic package contains thousands of functions for computing statistics, and user-contributed packages for this open-source software provide thousands more. Advanced users can write their own functions to implement new methods for statistical analyses.

Best of all, the base R package and all user-contributed packages are available free of charge to anyone with an internet connection.

This chapter tells you how to obtain R software and contributed packages and introduces you to some basic R functions. It also shows you how to read data sets into R and save output and graphics produced while you are using the package.

Conventions used in this book. This book is intended to be read in conjunction with *Sampling: Design and Analysis, Third Edition* by Sharon L. Lohr, henceforth referred to as SDA. Many of the examples in this book refer to figures, tables, examples, or exercises in SDA. To avoid confusion, we refer to figures in SDA as "Figure x.x in SDA." We refer to figures in *this* book as "Figure x.x" with no qualifier.

The names of external data files and programs, such as `agsrs.csv` and `ch02.R`, are in `typewriter font`, as are the names of R packages and code we type. Variable names, function names, and internal R data set names are in *italic type.*

Much of this book consists of R commands and output, set in light shaded boxes such as the following:

```
# This is a comment
# Enter data values into vector 'myvec'
myvec <- c(1,2,3,9,14,27,5,21,pi)
# Print the vector 'myvec'
myvec
## [1]  1.000000  2.000000  3.000000  9.000000 14.000000 27.000000  5.000000
## [8] 21.000000  3.141593
# Calculate summary statistics
summary(myvec)
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##    1.00    3.00    5.00    9.46   14.00   27.00
```

To distinguish between the code and the output that is produced, a command that we typed into R is displayed flush against the left margin. The output from that command is preceded by `##`. Our comments are preceded by a single `#`. You can obtain the commands and comments (without the output) for all code in files `ch01.R`, `ch02.R`, etc., on the book website (see the Preface for the website address).

DOI: 10.1201/9781003228196-1

1.1 Obtaining the Software

The R system is available for FREE download from the Comprehensive R Archive Network at

`https://cran.r-project.org/`.

If R is not already installed on your system, download the package now. Click on the link for the operating system of your choice: Windows, Mac, or Linux. Install the package (you can do this for Windows or Mac by double-clicking on the icon after the download is finished). You should be able to click "Next" for all dialogs to finish the installation.

Good! Now you can use R directly or through the integrated development environment RStudio®. RStudio adds a larger number of statistical packages to R and provides a user-friendly graphical interface. You can obtain RStudio free of charge for Windows, Mac, or Linux from `https://rstudio.com`. As with R, you can install RStudio by double-clicking on the icon after download and can click on "Next" for all dialogs. If you have any trouble, numerous videos are available online that demonstrate the installation of R and RStudio.

Figure 1.1 is a screenshot of RStudio. It shows four windows:

1. Upper-left: Program editor window, where you will write code and programs. You can save the contents of this window so you have a record of what you have done.
2. Lower-left: R console, where you will run R commands.
3. Upper-right: The Workspace window tells you what you have created so far. Environment lists the variables in memory. History lists the commands you have submitted in the R console. The Tutorial window provides a tutorial for using R and RStudio.
4. Lower-right: Displays the files in the current working directory, plots you have produced, packages installed and whether they are loaded, and help for commands.

How to get things done in RStudio

1. Work out commands in the program editor window, and then either copy/paste the commands into the console or use Ctrl-Enter or click the icon "Run" to submit the current line or selection. You can open a new program editor window by clicking "R Script" under the "New File" icon in the upper left of the main RStudio toolbar (this can also be accessed through the File menu).
2. The program editor will keep a history of what you have done.
3. Make comments using `#` to help your future self recall what you did and why you did it.

Updating R and RStudio. If you already have R and RStudio installed, you may update them to the latest versions. You can either reinstall the latest version or (only for Windows) use the *updateR* function from contributed package `installr`. To update RStudio, select "Check for Updates" from the Help menu within the program.

1.2 Installing R Packages

The base R software contains few functions for analyzing survey data. In this book we rely on user-contributed packages that provide functions for selecting probability samples and for graphing and analyzing survey data.

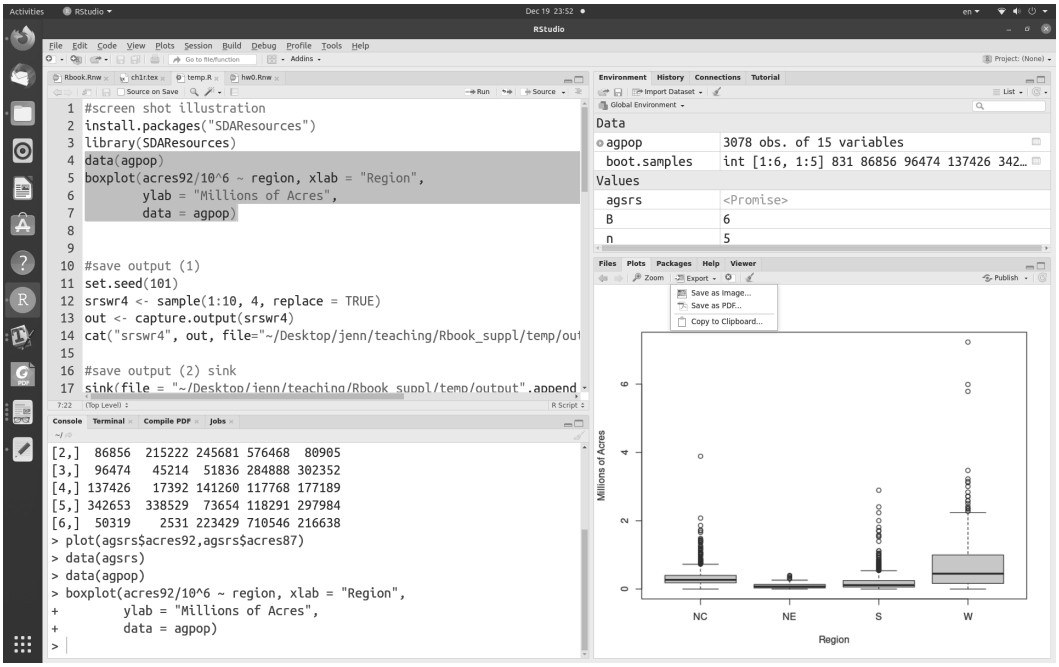

FIGURE 1.1: Screenshot of RStudio.

The main contributed packages used in this book are:

- `survey` (Lumley, 2020): Used to calculate statistics from complex surveys with functions such as *svydesign*, *svymean*, and *svytotal*.
- `sampling` (Tillé and Matei, 2021): Used to select probability samples. The `sampling` package will also analyze data from complex samples, although in this book we primarily use the `survey` package for data analysis.
- `SDAResources` (Lu and Lohr, 2021): Includes all the data sets of SDA in R format and additional functions for analyzing and graphing probability samples.

Install these packages now. You only need to install each package once for a particular version of R (you may need to reinstall packages if you update to a new R version).

Copy and paste the following code into the R Console window and press [Enter]. Each package will be downloaded (along with package dependencies) and installed.

```
install.packages("survey")
install.packages("sampling")
install.packages("SDAResources")
```

You must load a package into the program with the *library* function each time you open a new R session. When they are needed, run (in this order):

```
library(survey)
library(sampling)
library(SDAResources)
```

1.3 R Basics

Getting help. There are many sources for help about R and RStudio. The web pages `https://www.r-project.org/` and `https://education.rstudio.com/` contain links to online tutorials and resources. For those who prefer a printed book, Horton and Kleinman (2015), Horton et al. (2018), Ismay and Kennedy (2019), and Kabacoff (2021) provide useful introductions to getting started with R and RStudio. The book by Wickham (2019) is an advanced guide to R.

The *help* function in R provides access to the documentation pages for R functions, data sets, and other objects. For example, if you want to get help with the function *mean*, type `help("mean")`, or type `help.search("mean")` into the R console to search the help system for documentation matching the given character string "mean". You can also display the code of a function (for some packages) by typing its name. For example, typing `ls` in the R console window displays the code for the function *ls*, which lists the objects in the directory; to list the objects, type `ls()`.

For another example, suppose that you have not installed package `survey` yet, and you want to get help with the function *svymean*. Typing `help("svymean")` will show that "No documentation for 'svymean' in specified packages and libraries". Typing `help.search("svymean")` will direct you to the page with information that function *svymean* is under package `survey`.

To find help for a package, for example, package `survey`, type `help(package = "survey")`. To find help for data *agpop*, type `help("agpop")`.

In RStudio, you can find help on a function by typing the function name into the search window of the Help menu.

Frequently used commands. Tables 1.1 and 1.2 list frequently used operators, functions, and commands on vectors, matrices, and data frames. Note that R is case-sensitive: *myvar*, *MYVAR*, and *Myvar* are all different variables.

The R functions we use to analyze survey data operate on data frames. A data frame is like a matrix, where each row contains the data values for an observation, and each column contains the data values for one variable. You can turn a matrix A into a data frame with the *as.data.frame* function.

```
A<-matrix(0,2,2)  # create a 2*2 matrix with all 0s
A[1,1]<-2
A[1,2]<-5
A[2,1]<-1
A[2,2]<-4
colnames(A)<-c("y","x")
A
##      y x
## [1,] 2 5
## [2,] 1 4
A.data <- as.data.frame(A) # turn matrix A into a data frame A.data
A.data
##   y x
## 1 2 5
## 2 1 4
```

TABLE 1.1
Frequently used commands in R: Vectors, matrices, and mathematical functions.

Command	Purpose	Example	Output
Assignment and Comment			
<-	Assign value	a <- 1	1
=	Assign value	b = 1 + 2	3
#	Comment	# My comment	
Mathematical Functions			
+	Addition	1 + 1	2
−	Subtraction	1 − 1	0
*	Scalar multiplication	1*2	2
/	Division	1/2	0.5
^	Exponentiation	2^3	8
abs	Absolute value	abs(−3)	3
exp	Exponential function	exp(1)	2.718282
log	Natural log	log(2.718282)	1
sqrt	Square root	sqrt(4)	2
Vector/Matrix Operations			
c	Combine values into vector	vec1<-c(1,2,3) vec2<-c(4,5,6)	1 2 3 4 5 6
seq	Sequence or, when by=1	seq(from=1,to=10,by=2) 1:5	1 3 5 7 9 1 2 3 4 5
sequence	Vector of sequences	sequence(vec1)	1 1 2 1 2 3
rep	Replicate	rep(1,4) vec3<-rep(vec1,2)	1 1 1 1 1 2 3 1 2 3
cbind	Column bind	mat1<-cbind(vec1,vec2)	1 4 2 5 3 6
rbind	Row bind	mat2<-rbind(vec1,vec2)	1 2 3 4 5 6
length	Length of vector	length(vec1)	3
sort	Sort a vector	sort(vec3)	1 1 2 2 3 3
order	Indices to sort a vector	order(vec3)	1 4 2 5 3 6
unique	List unique objects	unique(vec3)	1 2 3
sum	Summation	sum(c(1,2,4))	7
prod	Product	prod(c(1,2,4))	8
%*%	Matrix multiplication	mat1 %*% mat2	17 22 27 22 29 36 27 36 45
t	Transpose	t(mat1)	1 2 3 4 5 6

1.4 Reading Data into R

The first step for using R to analyze data from a survey is to read the data into the system. There are four basic ways to do this.

Enter the data directly into R. You can use the *c* function to enter a vector directly from the R console. Type

```
myvec <- c(5,2,8,4)
```

TABLE 1.2
Frequently used commands in R: Extracting data elements and computing statistics.

Command	Purpose	Example	Output
Working with Data			
as.data.frame	Create data frame	A<-as.data.frame(mat1)	vec1 vec2 1 4 2 5 3 6
names(data)	Extract variable names	names(A)	"vec1" "vec2"
$	Extract a column by name	A$vec1	1 2 3
[, j]	Extract a column by number	A[,1]	1 2 3
[i ,]	Extract a row by number	A[1,]	1 4
nrow(data)	# of rows in *data*	nrow(A)	3
ncol(data)	# of columns in *data*	ncol(A)	2
head(data,n=i)	First i rows	head(A,n=2)	1 4 2 5
tail(data, n=i)	Last i rows	tail(A,n=2)	2 5 3 6
Statistics			
max/min	Maximum/Minimum of a vector or matrix	max(vec1) min(mat1)	3 1
quantile	Calculate quantiles	quantile(1:101,probs=c(0,0.25,0.5))	0% 25% 50% 1 26 51
mean	Sample mean	mean(vec1)	2
var	Sample variance	var(vec1)	1
cor	Correlation of two vectors	cor(vec1,vec2)	1

to store the vector of values (5, 2, 8, 4) in variable *myvec*. You can also use the *data.frame* function to enter data into a data frame from the R console:

```
# read matrix into data frame "mydata"
mydata <- data.frame(x = c(1,4,3,2), y = c(2,4,2,4), z = c(3,7,2,3))
# show the data
mydata
##   x y z
## 1 1 2 3
## 2 4 4 7
## 3 3 2 2
## 4 2 4 3
```

Read data from a comma-delimited (.csv) or text file. With a longer data set, it is often more convenient to store the data in an external file and then read it in through the data step. The following code will read data in a text file from a web page.

```
ex.data <- read.table(file="https://math.unm.edu/**/**.txt", header=T)
# Use header=T if there is a header
ex.data   # print the data
```

Suppose the folder on your computer containing the data sets is `C:\MyFilePath\`. To read a `csv` file, use `read.csv`. Note that to specify a file path in R, you should replace each backslash '\' with either a forward slash '/' or a double backslash '\\'.

```
ex.data <- read.csv(file="C:/MyFilePath/exampledata.csv",header=F)
```

```
# header=F is the default
colnames(ex.data) <- c("y","x1","x2") #add column names
ex.data
```

Import data with RStudio. Use the "Import Dataset" dropdown from the "Environment" window (top right panel in RStudio). The import formats are grouped into 3 categories: text data, spreadsheet data, and statistical data.

- import data from webpage

  Select "From Text (readr)", enter URL address, and click "Import".

- import data from local file

  Select "From Text (base)" or "From Excel", select a local file and click "Import"

Luraschi (2021) gives a detailed description of how to import data in RStudio.

Read R data sets. This is the easiest method of all—provided someone else has already saved the file as an R data set. All the data sets in SDA are loaded in R package `SDAResources`. Here is an example to read data *agpop*.

```
library(SDAResources)
data(agpop)  # Load the agpop data set
N <- nrow(agpop)
N
## [1] 3078
head(agpop)
##                  county state acres92 acres87 acres82 farms92 farms87 farms82
## 1 ALEUTIAN ISLANDS AREA    AK  683533  726596  764514      26      27      28
## 2        ANCHORAGE AREA    AK   47146   59297  256709     217     245     223
## 3        FAIRBANKS AREA    AK  141338  154913  204568     168     175     170
## 4           JUNEAU AREA    AK     210     214     127       8       8      12
## 5  KENAI PENINSULA AREA    AK   50810   85712   98035      93     119     137
## 6        AUTAUGA COUNTY    AL  107259  116050  145044     322     388     453
##   largef92 largef87 largef82 smallf92 smallf87 smallf82 region
## 1       14       16       20        6        4        1      W
## 2        9       10       11       41       52       38      W
## 3       25       28       21       12       18       25      W
## 4        0        0        0        5        4        8      W
## 5        9       18       17       12       18       19      W
## 6       25       32       32        8       19       17      S
```

1.5 Saving Output

Section 1.4 gave four methods for reading data into R. How do you save the output from the program? Here are several methods that will allow you to save the output or paste it into another document. Section 1.6 will show you how to incorporate R output directly into a LaTeX document.

Use the *sink* function to save console input and output. First, use *sink* to create a file in local drive. Next, run the R program. Last, close *sink*. Below is an example.

```
sink(file = "C:/MyOutputPath/output.txt",append = TRUE,
      type = c("output","message"))
```

```
cat("boot.samples", sep="\n")  #add name of the R output
#R program
set.seed (244)
B <- 6
n <- 5
boot.samples <- matrix(sample(agpop$acres92, size=B*n, replace=TRUE),B,n)
boot.samples
sink(file = NULL) #close sink
```

The file `C:\MyOutputPath\output.txt` in the local drive appears as follows:

```
boot.samples
       [,1]    [,2]   [,3]   [,4]   [,5]
[1,]    831 1449976  98142 161724 138986
[2,]  86856  215222 245681 576468  80905
[3,]  96474   45214  51836 284888 302352
[4,] 137426   17392 141260 117768 177189
[5,] 342653  338529  73654 118291 297984
[6,]  50319    2531 223429 710546 216638
```

In some cases you may also want to save the R commands, so you can look at them later. You may use the function *source* to read in an R program with option `echo=TRUE` to include the R code, so that R output together with R code will be saved in `output.txt`.

```
sink(file ="MyOutputPath/output.txt",append = TRUE,
      type = c("output","message"))
source("MyInputFilePath/homework3.R", echo=TRUE)
sink(file = NULL)
```

Convert a matrix to LaTeX code in table format. In case you want to include a matrix as part of a LaTeX document, there is an easy way to convert the matrix to LaTeX code in table format. The following code, using the `stargazer` package, converts the matrix *boot.samples* to a LaTeX table.

```
# save output as latex
library(stargazer)
set.seed (244)
B <- 6
n <- 5
boot.samples <- matrix(sample(agpop$acres92, size=B*n, replace=TRUE),B,n)
# Now use stargazer to produce Latex commands for table
# Add header=FALSE to omit the initial comments in the Latex code output.
stargazer(boot.samples, digits = 2,header=FALSE)
##
## \begin{table}[!htbp] \centering
##   \caption{}
##   \label{}
## \begin{tabular}{@{\extracolsep{5pt}} ccccc}
## \\[-1.8ex]\hline
## \hline \\[-1.8ex]
## $831$ & $1,449,976$ & $98,142$ & $161,724$ & $138,986$ \\
## $86,856$ & $215,222$ & $245,681$ & $576,468$ & $80,905$ \\
## $96,474$ & $45,214$ & $51,836$ & $284,888$ & $302,352$ \\
## $137,426$ & $17,392$ & $141,260$ & $117,768$ & $177,189$ \\
## $342,653$ & $338,529$ & $73,654$ & $118,291$ & $297,984$ \\
## $50,319$ & $2,531$ & $223,429$ & $710,546$ & $216,638$ \\
```

```
## \hline \\[-1.8ex]
## \end{tabular}
## \end{table}
```

Saving graphs. R and RStudio both allow you to save a graph using menu options. In standalone R, you can save a graph by choosing "Save as" from the File menu. In RStudio, select the Export dropdown from the plot panel (lower right-panel):

Plots→ Export → Save as Image or Save as PDF.

Alternatively, you can specify files to save the image using functions such as *jpeg*, *png*, and *pdf*, which save the plot in an external file with the designated format. All of these functions have multiple options for sizing and formatting the graph. To save a graph as a pdf file, for example, first open the pdf file, then create the plot, and then close the file:

```
# Open a pdf file
pdf("~/MyOutputPath/rplot.pdf")
# Create a plot
boxplot(acres92/10^6 ~ region, xlab = "Region", ylab = "Millions of Acres",
        data = agpop)
# Close the pdf file
dev.off()
```

Or, to save as a jpeg file,

```
# save as jpeg file
jpeg("~/MyOutputPath/rplot.jpg",width=350,height=350)
boxplot(acres92/10^6 ~ region, xlab = "Region", ylab = "Millions of Acres",
        data = agpop)
dev.off()
```

Saving a data set. You can use function *write.table* or *write.csv* to save a data set to an external file. In the following, we will use data *classes* to create a new dataset in a long format called *classeslong*, and use *write.csv* to save the data in file `classeslong.csv`.

```
# read in data classes
data(classes)
# change to a long format by creating a record for each student
# create new data frame with each row repeated as many times as number of students
classeslong<-classes[rep(1:nrow(classes),times=classes$class_size),]
# add column of student ids, goes from 1 to number of students in each class
classeslong$studentid <- sequence(classes$class_size)
nrow(classeslong)
## [1] 647
head(classeslong)
##     class class_size studentid
## 1       1         44         1
## 1.1     1         44         2
## 1.2     1         44         3
## 1.3     1         44         4
## 1.4     1         44         5
## 1.5     1         44         6
# you can save classeslong in your local drive (or other file path) using write.csv
write.csv(classeslong, file="~/classeslong.csv", row.names = FALSE)
```

1.6 Integrating R Output into LaTeX Documents

The `Sweave` system within R allows you to embed R code and output within LaTeX documents to generate a `pdf` file that includes narrative, graphics, code (if needed), and the results of the computations in R. The R package `knitr` serves as an engine for `Sweave` and allows you to generate dynamic reports with reproducible research (that is, someone else can produce the same results when given access to the data and code) using R. Help for `knitr` is available on the web (`https://yihui.org/knitr/`) and in the book by Xie (2015). We used the `knitr` package to produce this book.

Let's start with a simple example, using RStudio.

Step 1. Open a new `Rnw` script by clicking the "R Sweave" icon under the "New File" icon in the upper left of the main RStudio toolbar. Your first Rnw file will appear as in Figure 1.2. Save the file with extension `Rnw`—for example, you might name this file `test.Rnw`.

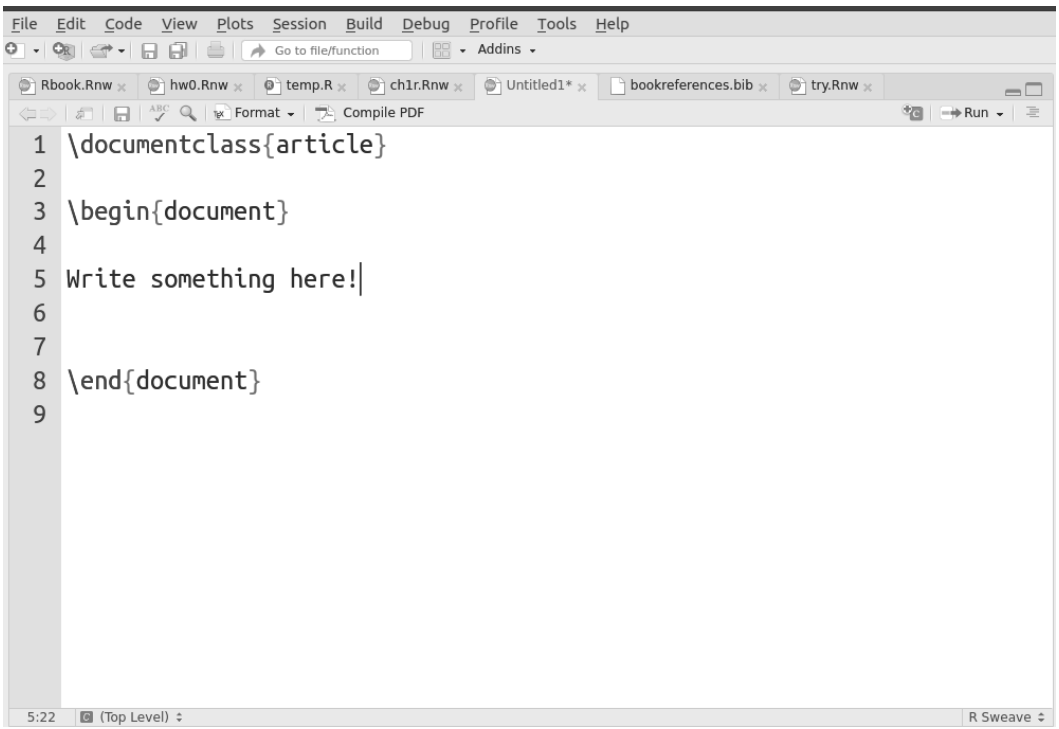

FIGURE 1.2: Example of a `Rnw` file.

Step 2. Weave the `Rnw` file using `knitr`. You will be asked to install package `knitr` if this is the first time you have used it.
Click Tools → Global options → Sweave → Weave Rnw files using → knitr.

Step 3. Click "Compile PDF" to generate a pdf file. The file `test.pdf` will appear in the same directory as `test.Rnw`, along with the LaTeX file, log file, and other auxiliary files.

Embedding R code within LaTeX. `Sweave` allows you to embed R code within LaTeX documents so you can incorporate statistical results in your pdf document. Type your LaTeX document in the program editor window in RStudio. To embed a "chunk" of R code in

`knitr`, start with << ·· >>= (all on one line) and close with @. Below is an illustration (we had to break the first line so it fit on the page, but you should leave it unbroken):

```
<<exercise0, echo=TRUE, size='footnotesize', include=TRUE, fig.width=6,
     fig.height=5, out.width='0.80\\textwidth'>>=
# Insert R code here!
x <- 1:10
y <- seq(from=20,to=2,by=-2)
x
y
x*y
cor(x,y)
# End the chunk with an @
@
```

This example, which has label *exercise0*, includes the following options to customize the R output.

echo = TRUE (print source code in the document) or FALSE (do not print source code in the document).

size = font size of the code text, if echoed. Options include 'normalsize,' 'footnotesize,' 'large,' and other font sizes in LaTeX.

include = TRUE if it is desired to include the chunk output in the document. If FALSE, the output is not produced, but the R code is still executed and the plots are generated.

fig.width = figure width in inches

fig.height = figure height in inches

out.width = scaling figures to fit on page (can be inches or cm, or relative to page size). Here, the width is scaled to be 0.8 times the value of the *textwidth* in the LaTeX document.

To learn more about chunk options, please refer to `https://yihui.org/knitr/options/#chunk_options`.

Here is what is displayed when we include the code for *exercise0* in the document:

```
# Insert R code here!
x <- 1:10
y <- seq(from=20,to=2,by=-2)
x
##  [1]  1  2  3  4  5  6  7  8  9 10
y
##  [1] 20 18 16 14 12 10  8  6  4  2
x*y
##  [1] 20 36 48 56 60 60 56 48 36 20
cor(x,y)
## [1] -1
# End the chunk with an @
```

Now, try a slightly more complicated example in `hw0.Rnw` (included on the book webpage) to see how to embed R chunks (including graphs) within latex, and compile it. You can see the pdf document that is produced by the example in file `hw0.pdf`.

Many excellent books are available for learning and using LaTeX. The books by Talbot (2012), Oetiker et al. (2021), and Wikibooks contributors (2021) may be downloaded free of charge.

1.7 Missing Data

Many survey data sets have observations that are missing. For example, the data file `agpop.csv`, from which the sample used in Chapter 2 is drawn, has missing data. As is common in data sets intended to be readable by multiple programs, a designated number is used to indicate that the data value is missing. In this data set, the value "−99" indicates that the data value is missing. This value must be recoded to the R symbol that indicates missing data before performing calculations. Otherwise, if, say, you want to calculate the mean of a variable, a function will treat all the "−99"s as if they were observations with the value −99 instead of missing values—this could lead to embarrassing results such as computing a negative value for the average number of acres per farm.

Missing values in R are coded by the symbol NA. If your data set has missing values that are coded as a number (such as −99), you should recode those to NA before starting your analysis. Missing data in the R data sets in package `SDAResources` (Lu and Lohr, 2021) have been recoded to NA, but the data sets for SDA that are in `.csv` format use other codes, which are described in Appendix A. If you want to read the data set `agpop.csv` into R from the `.csv` file (instead of loading it from `SDAResources`), you should recode the missing data as follows:

```
agpop1 <- read.csv(file="~/agpop.csv",header=TRUE)
# some variables have missing values, coded as -99
sum(agpop1==-99)  # 59 missing values altogether
## [1] 59
# look at a row with missing data
agpop1[200,]
##                   county state acres92 acres87 acres82 farms92 farms87 farms82
## 200 SAN FRANCISCO COUNTY    CA       7     -99      19       6       5       7
##     largef92 largef87 largef82 smallf92 smallf87 smallf82 region
## 200        0        0        0        6        5        7      W
agpop2 <- agpop1
# recode missing values to NA
agpop2[agpop1==-99] <- NA
# look at row with missing data again
agpop2[200,]
##                   county state acres92 acres87 acres82 farms92 farms87 farms82
## 200 SAN FRANCISCO COUNTY    CA       7      NA      19       6       5       7
##     largef92 largef87 largef82 smallf92 smallf87 smallf82 region
## 200        0        0        0        6        5        7      W
# count missing values in recoded data
sum(is.na(agpop2))
## [1] 59
```

Different functions in R treat missing data in different ways. For many functions, the treatment of missing values can be specified using the *na.rm* argument. The *sum* function, for example, will compute the sum of the non-missing values in a vector if you include `na.rm=TRUE`; otherwise, it returns NA if there are missing values.

```
sum(agpop2$acres87)
## [1] NA
sum(agpop2$acres87,na.rm=TRUE)
## [1] 963466689
```

1.8 Summary, Tips, and Warnings

Tables 1.1 and 1.2 describe commonly used R functions for working with data. Table 1.3 lists other functions used in this chapter to read and write data. The `base`, `stats`, and `utils` packages are automatically included when you install R on your system.

TABLE 1.3
Functions used for Chapter 1.

Function	Package	Usage
library	base	Load an R package that you have installed on your system. You need to load a package each time you start a new R session.
sink	base	Send R output to an external file
source	base	Read and execute R commands from an external file
install.packages	utils	Install an R package
data	utils	Load a specified data set
read.table	utils	Read data from an external file into a data frame. You can specify the character that separates fields, row and column names, and whether the first line of the file gives the variable names.
read.csv	utils	This function is like read.table for an external file that is in comma-delimited (`.csv`) format.
write.table	utils	Write an R data set to an external file
write.csv	utils	Write an R data set to an external file in `.csv` format

Tips and Warnings

- R commands can be tricky, and sometimes the result of using functions on vectors and matrices may not be what you were expecting. When creating new variables or matrices, print out the first few rows or compute summary statistics to check that you have created the object you want.
- Use vector and matrix commands whenever possible when performing calculations with R. Although we occasionally use *for* loops in this book where this will make the code easier to understand, in general, it is much more efficient to perform operations on vectors.
- Many surveys contain missing data. Check how these are coded in the data set, and recode missing values to NA before starting your analysis.
- Sometimes multiple packages contain functions or data sets with the same names. For example, the `sampling` and `survival` packages (the `survival` package is loaded when you load the `survey` package) both have functions named *strata* and *cluster*. If you load `sampling` and then load `survival`, you will get the function from the `survival` package when you call *strata*—the function *strata* from the `sampling` package is masked. To avoid this problem, load the `sampling` package after loading the `survival` package or access the function from `sampling` by typing `sampling::strata`. For this book, we suggest loading the packages in the order given in Section 1.2.