R for 1.1-1.4

A.S. Wagaman, CC BY-NC

Intro to The R Supplements

These R Supplements are designed to accompany a course in Probability using the Dobrow text. The text R code, along with some additional code and comments, has been provided for you. This should make it easier for you to engage with the R code in the text. Simply load up the .Rmd and run the code yourself. Compiled pdf files will also be available as a resource while you work on problems in class or for homework. I strongly encourage you to refer to these files during your reading from the text as well.

What Comes From the Text

R code from the text has been excerpted and entered here, indexed by book section. Modifications may be made as discussed below. Code chunks have the label from the text and relevant page number for your reference.

What is Different Compared to the Text?

You might notice a few things that are different in the code presented in these files than in the text versions. Here is a list of reasons for these modifications.

- The code in the text contains an error, such as a missing bracket or spelling issue. Such typos will mean the code will not run. Hence, typos were noted and corrected where found.
- The code in the text does not set seeds for reproducibility. Discussed in more detail below, I set seeds so that your code will give reproducible results, but you are still generating random values!
- The code in the text might suggest 100000 runs for a simulation where 1000 or 10000 is sufficient, so I used a smaller number. Most of the time this is not an issue. You can edit these values. Please do not ever run more than half a million simulations of anything on the R server.
- Sometimes the code in the text does not specify the number of runs before a simulation explicitly, but rather just uses a number like 10000 throughout the code. In these cases, the values in the code have been changed to a variable, and the variable value is specified before the simulation is run. This is to provide more consistency across the provided simulation code examples. It also means you only have to change the value once if you want to change it. Setting of other constants may also be done differently than in the text.
- Comments have been added to the code to help explain what the commands are doing. This is typically done the first few times a command is used.
- Function inputs may be set or altered for demonstration. For example, if the book example code does not set a value for an input c or uses c <- 2, you may see c <- 5 in these files.
- Additional code and examples are provided here that are not from the text. These supplemental examples may enhance a book code example or provide code for a book example without code.
- Spacing has been adjusted in the code to enhance readability.
- When learning code, you may not understand all the arguments/inputs for a function. The book does not "name" the arguments, but I have done so in many cases to assist you with learning the functions at hand.
- A few times, R scripts from the text were available from the author's website but no text reference could be found. These scripts have been evaluated and may be included in the appropriate chapter. We also make a note if a specified R script is listed in the text but was not found online.

R for 1.1-1.4 - Foundations of Probability

```
# NA; the textbook does not contain any R code in these four sections
```

Supplemental Material - Intro to R

R is a powerful programming language, but it can be used for basic operations, like a calculator. Briefly, we will take a look at a few topics: R and RMarkdown, R as a calculator for basic descriptive statistics, the basics of sampling values using *sample*, setting seeds, functions in R, and accessing the R help menu.

R and R Markdown

These files are RMarkdown documents, designed to include both R code and text/comments. R code is provided in these files in R chunks. If you are working in the .Rmd, you can run an entire chunk or run a particular line (or several lines), using keyboard commands (e.g. CTRL+Enter for a line) and the chunk menu (right-facing triangle for a chunk). You can add an R chunk using the insert option at the top of the .Rmd. When you are ready to compile, hit "Knit" (arrow next to it will let you swap to Knit to PDF if HTML happens). You can put text between R chunks to comment on what you are seeing in the chunks. If these files are new to you, or you have questions about how to work with them, please ask for assistance.

For our course, you will need to export compiled PDFs of the weekly R problem so that you can submit your solutions on Moodle. Again, if you have issues with this after the class walkthrough, please ask me, our TA, or the Stat Fellows for assistance with the RMarkdown file.

These supplement files though, are yours to work from as you read from the text, work on homework, etc.

R as a Calculator

R can be used as a calculator for basic operations. It can also save values and compute basic descriptive statistics. Your text will use the *sum*, *mean*, *sd* and *var* functions to compute a sum, mean, standard deviation, and variance, so we give examples of those here. If you aren't sure what standard deviation and variance are, don't worry, we'll see formal definitions later this semester. For now, note that standard deviation is a measure of spread, and variance is its square.

```
#R as a basic calculator
2+3
```

```
## [1] 5
```

```
4/6
```

```
## [1] 0.6666667
```

```
3*8
```

```
## [1] 24
```

```
4^2
```

```
## [1] 16
```

```
exp(6) #e^6
```

```
## [1] 403.4288
```

```
log(12) #log is natural log
```

```
## [1] 2.484907
```

We can also save values for use later.

```
#Saving values to variables
x <-5 #saves the value 5 to the variable x
y <- 1:10 #makes y the integer values from 1 to 10
z <- c(2, 4, 6, 7, 9, 12, 13, 16) #creates a vector and assigns it to z
```

Now that we have some variables containing multiple values, we can try out some of the basic functions.

```
#Using some basic functions: sum, mean, sd, var
sum(y)
```

```
## [1] 55
```

```
mean(y) #measure of center
```

```
## [1] 5.5
```

```
sd(z) #measure of spread
```

```
## [1] 4.779047
```

```
var(z) #note that var=sd^2
```

```
## [1] 22.83929
```

Sample

sample is an R command that can generate random integers from a set (you provide) under various conditions (with or without replacement, with equal probabilities or with some specified probabilities, etc.).

```
#Section 1.10 in your text introduces the *sample* command
sample(1:5, 3, replace = FALSE) #generates 3 numbers from 1:5 without replacement
```

```
## [1] 1 4 5
```

```
sample(1:5, 3, replace = TRUE) #generates 3 numbers from 1:5 with replacement
```

```
## [1] 5 2 1
```

```
sample(1:5, 3, replace = FALSE, prob = c(0.1, 0.2, 0.3, 0.2, 0.2)) #generates 3 numbers from 1:5 withou
```

```
## [1] 4 5 3
```

```
x <- sample(1:10, 2, replace = TRUE) #saves two values generated from 1:10 with replacement as x
```

What *sample* command would you use to simulate rolling 2 fair standard six-sided dice?

ANSWER

Setting Seeds for Reproducibility

The *sample* commands above all work, but they are missing a key aspect for reproducible work. Simply put, the results are not reproducible without knowing *where* the random number generator started when the command was executed. That means that anyone who runs the line can end up with different random results. It may sound odd, but we often actually want the option to reproduce the same random results for anyone who runs our code. This is helpful for asking for help with issues with code, as well as working with RMarkdown. RMarkdown compiles from scratch, so if you run the code above, the output may differ from what it gave when you ran it before compiling.

The way to get around this is to set a seed. That is basically telling the random number generator *where* to start when generating random numbers. Anyone else using the same seed with your code should get the same output you did. This makes our work reproducible. Setting a seed is easy.

```
sample(1:6, 2, replace = TRUE) #no seed is set; output will vary each time
```

```
## [1] 3 2
```

```
sample(1:6, 2, replace = TRUE)
```

```
## [1] 3 2
```

```
sample(1:6, 2, replace = TRUE)
```

```
## [1] 4 2
```

Every time you run the lines above, your sampled values may differ. That's not the case once we set a seed.

```
set.seed(50)
sample(1:6, 2, replace = TRUE) #will always give 5 3 as output when run with set.seed(50) right before
```

```
## [1] 5 3
```

You can change the integer seed if you want to see other different results. To make your results reproducible, you can pass the seed to someone wanting to run your code (or leave it in the code you give them!).

Functions in R

Over the course of the semester, you will learn to write functions in R. These will most often be useful to us in performing a simulation to verify a theoretical result (or answer). Let's start with the basics.

Functions in R all have a similar framework. You set a function name, a list of inputs/arguments (which appear after "function" but before the bracket), then you perform a series of operations on or using the inputs/arguments, and set an output (the last item before the end bracket). Let's look at a very simple function. Run the entire chunk to add the function to your workspace, as well as run the example call.

```
addtwo <- function(a, b){  #function name starts the line
  total = a+b #operation performed on the inputs
  total #output, can be more formally set using "return"
}
addtwo(3, 9) #an example function call
```

```
## [1] 12
```

What is the name of this function?

ANSWER

What are the inputs/arguments to this function?

ANSWER

What does the function do?

ANSWER

What is the output of the function?

ANSWER

Again, bearing in mind that this is a simple function, let's consider the pseudocode that would support you in building this function. According to Wikipedia's Pseudocode entry: "Pseudocode is an informal high-level description of the operating principle of a computer program or other algorithm. It uses the structural

conventions of a normal programming language, but is intended for human reading rather than machine reading."

You've figured out that "addtwo" takes two numbers and outputs their sum. Here is example pseudocode that could help you write the associated function: Determine 2 inputs. Add the inputs. Output the total.

This example is very simple and does not include much of the programming language. For example, it does not have any loops and there is no reference to a computer function. We will work up to more complicated functions as the semester progresses.

Accessing R Help

As you learn R, you may encounter functions that you aren't fully familiar with, or where you don't know all the available options. To learn about a function, you can access its help menu in R. There are three primary ways you can access the help menu for a specific function.

- Type help(functionname) in the console or in a chunk and execute the line. Note that you won't want this left in the chunk when you go to compile, since it opens an external window.
- Type ?functionname instead of help(functionname).
- In RStudio, in the lower right panel, go to the Help tab and use the search feature.

Here are some example help commands for the functions we have seen (commented out with a # to start the line so they don't try to run).

```
# help(sample)
# ?sum
```

The help menu will show you the list of inputs/arguments to the function, their default values (if any), and in the details, give you well, more details about the options.

If you ever want to see what the underlying code for a function looks like, you can access that with just *functionname* entered into the console or in a chunk. For example, here we can look at the code for the *sample* command.

```
sample
```

```
## function (x, size, replace = FALSE, ...)
## {
##     UseMethod("sample")
## }
## <environment: namespace:mosaic>
```

This actually shows a bit more detail when typed in the console versus when it is compiled.

When we write a function, like *addtwo* up above, there is no associated help file but you can still access the code itself.

```
# ?addtwo #running this will tell you there is no associated documentation
addtwo #you can still pull up what the code for the function is
```

```
## function(a, b){  #function name starts the line
##   total = a+b #operation performed on the inputs
##   total #output, can be more formally set using "return"
## }
```

If you write a series of useful functions and want to make them available to the general public, you would consider making an R package. R packages are basically collections of useful functions and data sets with documentation. We won't worry too much about them for our class, although we might use some during the course of the semester.

Textbook Citation

Dobrow, R. P. (2013). Probability: with applications and R. John Wiley & Sons.