Appendix B

Introduction to R and RStudio

This chapter provides a (brief) introduction to R and RStudio. R is a free, open-source software environment for statistical computing and graphics [116, 169]. RStudio is an open-source integrated development environment (IDE) for R that adds many features and productivity tools for R. This chapter includes a short history, installation information, a sample session, background on fundamental structures and actions, information about help and documentation, and other important topics.

R is a general purpose package that includes support for a wide variety of modern statistical and graphical methods (many of which have been contributed by users). It is available for Linux, Mac OS X, and Windows. The R Foundation for Statistical Computing holds and administers the copyright of the R software and documentation. R is available under the terms of the Free Software Foundation's GNU General Public License in source code form.

RStudio facilitates use of R by integrating R help and documentation, providing a workspace browser and data viewer, and supporting syntax highlighting, code completion, and smart indentation. It integrates reproducible analysis with `knitr` and R Markdown (see Appendix D), supports the creation of slide presentations, and includes a debugging environment. It facilitates the creation of dynamic Web applications using Shiny (see Section 11.3). It also provides support for multiple projects as well as an interface to source code control systems such as GitHub. It has become the default interface for many R users, and is our recommended environment for analysis.

RStudio is available as a client (standalone) for Windows, Mac OS X, and Linux. There is also a server version. Commercial products and support are available in addition to the open-source offerings (see `http://www.rstudio.com/ide` for details).

The first versions of R were written by Ross Ihaka and Robert Gentleman at the University of Auckland, New Zealand, while current development is coordinated by the R Development Core Team, a group of international volunteers.

R is similar to the S language, a flexible and extensible statistical environment originally developed in the 1980s at AT&T Bell Labs (now Alcatel–Lucent). Insightful Corporation has continued the development of S in their commercial software package S-PLUS^{TM}.

B.1 Installation

New users are encouraged to download and install R from the Comprehensive R Archive Network (CRAN, `http://www.r-project.org`) and install RStudio from `http://www.rstudio.com/download`. The sample session in the appendix of the *Introduction to R* document, also

available from CRAN (see B.2), is highly recommended reading.

The home page for the R project, located at `http://r-project.org`, is the best starting place for information about the software. It includes links to CRAN, which features precompiled binaries as well as source code for R, add-on packages, documentation (including manuals, frequently asked questions, and the R newsletter) as well as general background information. Mirrored CRAN sites with identical copies of these files exist all around the world. Updates to R and packages are regularly posted on CRAN.

B.1.1 Installation under Windows

Versions of R for Windows XP and later—including 64-bit versions—are available at CRAN. The distribution includes `Rgui.exe`, which launches a self-contained windowing system that includes a command-line interface, `Rterm.exe` for a command-line interface only, `Rscript.exe` for batch processing only, and `R.exe`, which is suitable for batch or command-line use. More information on Windows-specific issues can be found in the CRAN *R for Windows FAQ*.

B.1.2 Installation under Mac OS X

A version of R for Mac OS X 10.6 and higher is available at CRAN. This is distributed as a disk image containing the installer. In addition to the graphical interface version, a command line version (particularly useful for batch operations) can be run as the command `R`. More information on Macintosh-specific issues can be found in the CRAN *R for Mac OS X FAQ*.

B.1.3 Installation under Linux

R is available for most Linux distributions through your distribution's repositories. For example, R is provided on Debian-based distributions like Ubuntu by the `r-base` package. Many additional packages, such as `r-cran-rpart`, are provided at the maintainer's discretion. Installation on Ubuntu is as simple as:

```
sudo apt-get update
sudo apt-get install r-base r-base-dev
```

CRAN provides distribution-specific packages for the Debian, Red Hat, SuSE, and Ubuntu distributions at `https://cran.r-project.org/bin/linux`.

B.1.4 RStudio

RStudio for Mac OS X, Windows, or Linux can be downloaded from `http://www.rstudio.com/ide`. RStudio requires R to be installed on the local machine. A server version (accessible from Web browsers) is also available for download. Documentation of the advanced features in the system is available on the RStudio website.

B.2 Running RStudio and sample session

Once installation is complete, the recommended next step for a new user would be to start RStudio and run a sample session (see Figure B.1).

The "`>`" character is the command prompt, and commands are executed once the user presses the RETURN or ENTER key. R can be used as a calculator (as seen from the first

```
R version 3.3.2 (2016-10-31) -- "Sincere Pumpkin Patch"
Copyright (C) 2016 The R Foundation for Statistical Computing
Platform: x86_64-apple-darwin13.4.0 (64-bit)

R is free software and comes with ABSOLUTELY NO WARRANTY.
You are welcome to redistribute it under certain conditions.
Type 'license()' or 'licence()' for distribution details.

  Natural language support but running in an English locale

R is a collaborative project with many contributors.
Type 'contributors()' for more information and
'citation()' on how to cite R or R packages in publications.

Type 'demo()' for some demos, 'help()' for on-line help, or
'help.start()' for an HTML browser interface to help.
Type 'q()' to quit R.

> 3 + 6
[1] 9
> 2 * 3
[1] 6
> x <- c(4, 5, 3, 2)
> x
[1] 4 5 3 2
> y <- seq(1, 4)
> y
[1] 1 2 3 4
> mean(x)
[1] 3.5
> sd(y)
[1] 1.290994
> ds <- read.csv("http://nhorton.people.amherst.edu/r2/datasets/help.csv")
> mean(ds$age)
[1] 35.65342
> # mean(age)     # this will generate an error
> with(ds, mean(age))
[1] 35.65342
> ds$age[1:15]
 [1] 37 37 26 39 32 47 49 28 50 39 34 58 58 60 36
> q()
```

Figure B.1: Sample session in R.

two commands on lines 1 and 3). New variables can be created (e.g., `x` and `y`) using the assignment operator `<-`. If a command generates output, then it is printed on the screen, preceded by a number indicating place in the vector (this is particularly useful if output is longer than one line, e.g., as it is for `ds$age[1:25]`). Saved data (here assigned the name `ds`) is read into R on line 15, then summary statistics are calculated (e.g., using `mean()`) and individual observations are displayed. The `$` operator allows access to objects within a data frame. Alternatively, the `with()` function can be used to access objects within a data set.

As shown in the example below, it is important to remember that R is case-sensitive. A comprehensive sample session in R can be found in Appendix A of *An Introduction to R* [207].

```
x <- 1:3
X <- seq(2, 4)
x

[1] 1 2 3

X

[1] 2 3 4
```

B.3 Learning R

B.3.1 Getting help

R features extensive online documentation, though it can sometimes be challenging to comprehend. Each command has an associated help file that describes usage, lists arguments, provides details of actions, gives references, lists other related functions, and includes examples of its use. The help system is invoked using either the `?` or `help()` commands.

```
?function
help(function)
```

where `function` is the name of the function of interest (Alternatively, the `Help` tab in RStudio can be used to access the help system.)

For example, the help file for the `mean()` function is accessed by the command `help(mean)`. The output from this command is provided in Figure B.2. It describes the `mean()` function as a generic function for the (trimmed) arithmetic mean, with arguments `x` (an R object), `trim` (the fraction of observations to trim, having a default value of `0`—setting `trim` equal to `0.5` is equivalent to calculating the median), and `na.rm` (should missing values be present, the default behavior `na.rm` equals `FALSE`, which leaves missing values as they are).

Some commands (e.g., `if`) are reserved, so `?if` will not generate the desired documentation. Running `?"if"` will work (see also `?Reserved` and `?Control`). Other reserved words include `else`, `repeat`, `while`, `function`, `for`, `in`, `next`, `break`, `TRUE`, `FALSE`, `NULL`, `Inf`, `NaN`, and `NA`.

The `RSiteSearch()` function will search for key words or phrases in many places (including the search engine at `http://search.r-project.org`). The RSeek.org site can also be helpful in finding more information and examples. Examples of many functions are available using the `example()` function.

```
example(mean)
```

Other useful resources are `help.start()`, which provides a set of online manuals, and `help.search()`, which can be used to look up entries by description. The `apropos()` command returns any functions in the current search list that match a given pattern (which facilitates searching for a function based on what it does, as opposed to its name).

Other resources for help available from CRAN include the R-help mailing list. The StackOverflow site for R provides a series of questions and answers for common questions that are tagged as being related to R. New users are also encouraged to read the R FAQ (frequently asked questions) list. RStudio provides a curated guide to resources for learning R and its extensions.

```
mean                    package:base                    R Documentation

Arithmetic Mean

Description:
     Generic function for the (trimmed) arithmetic mean.

Usage:
     mean(x, ...)

     ## Default S3 method:
     mean(x, trim = 0, na.rm = FALSE, ...)

Arguments:
       x: An R object.  Currently there are methods for numeric/logical
          vectors and date, date-time and time interval objects.
          Complex vectors are allowed for 'trim = 0', only.

    trim: the fraction (0 to 0.5) of observations to be trimmed from
          each end of 'x' before the mean is computed. Values of trim
          outside that range are taken as the nearest endpoint.

   na.rm: a logical value indicating whether 'NA' values should be
          stripped before the computation proceeds.

     ...: further arguments passed to or from other methods.

Value:
     If 'trim' is zero (the default), the arithmetic mean of the values
     in 'x' is computed, as a numeric or complex vector of length one.
     If 'x' is not logical (coerced to numeric), numeric (including
     integer) or complex, 'NA_real_' is returned, with a warning.

     If 'trim' is non-zero, a symmetrically trimmed mean is computed
     with a fraction of 'trim' observations deleted from each end
     before the mean is computed.

References:
     Becker, R. A., Chambers, J. M. and Wilks, A. R. (1988) _The New S
     Language_. Wadsworth & Brooks/Cole.

See Also:
     'weighted.mean', 'mean.POSIXct', 'colMeans' for row and column
     means.

Examples:
     x <- c(0:10, 50)
     xm <- mean(x)
     c(xm, mean(x, trim = 0.10))
```

Figure B.2: Documentation on the mean() function.

B.3.2 swirl

The swirl system is a collection of interactive courses to teach R programming and data science within the R console. It requires the installation of the swirl package, then use of the install_from_swirl() function to download courses. Table B.1 displays some of the courses that were available as of 2016. A sample session is displayed below. After some preliminary introductions, the user is instructed to enter a series of commands and explore in the console. The swirl system detects whether the correct commands have been input.

COURSE	DESCRIPTION
R Programming (beginner)	The basics of programming in R
R Programming Alt (beginner)	Same as the original, but modified for in-class use
Data Analysis (beginner)	Basic ideas in statistics and data visualization
Mathematical Biostatistics Boot Camp (beginner)	One- and two-sample t-tests, power, and sample size
Open Intro (beginner)	A very basic introduction to statistics, data analysis, and data visualization
Regression Models (intermediate)	The basics of regression modeling in R
Getting and Cleaning Data (intermediate)	dplyr, tidyr, lubridate, oh my!

Table B.1: Some of the interactive courses available within swirl.

```
> library(swirl)

| Type swirl() when you are ready to begin.

> install_from_swirl("Getting and Cleaning Data")

| Course installed successfully!

> swirl()

| Welcome to swirl!

| Please sign in. If you've been here before, use the same name as you did
| then. If you are new, call yourself something unique.

What shall I call you? Nick

| Please choose a course, or type 0 to exit swirl.

1: Getting and Cleaning Data
2: R Programming
3: Regression Models
4: Take me to the swirl course repository!

Selection: 1

| Please choose a lesson, or type 0 to return to course menu.
```

```
1: Manipulating Data with dplyr
2: Grouping and Chaining with dplyr
3: Tidying Data with tidyr
4: Dates and Times with lubridate

Selection: 1

| Attempting to load lesson dependencies...

| Package dplyr loaded correctly!

| In this lesson, you'll learn how to manipulate data using dplyr. dplyr is
| a fast and powerful R package written by Hadley Wickham and Romain
| Francois that provides a consistent and concise grammer for manipulating
| tabular data.

...
```

B.4 Fundamental structures and objects

Here we provide a brief introduction to R data structures.

B.4.1 Objects and vectors

Almost everything in R is an object, which may be initially confusing to a new user. An object is simply something stored in R's memory. Common objects include vectors, matrices, arrays, factors, data frames (akin to data sets in other systems), lists, and functions.

The basic variable structure is a vector. Vectors (and other objects) are created using the `<-` or `=` assignment operators (which assign the evaluated expression on the right-hand side of the operator to the object name on the left-hand side).

```
x <- c(5, 7, 9, 13, -4, 8) # preferred
x =  c(5, 7, 9, 13, -4, 8)  # equivalent
```

The above code creates a vector of length 6 using the `c()` function to concatenate scalars. The `=` operator is used in other contexts for the specification of arguments to functions. Other assignment operators exist, as well as the `assign()` function (see `help("<-")` for more information). The `exists` function conveys whether an object exists in the workspace, and the `rm` command removes it. In RStudio, the "Environment" tab shows the names (and values) of all objects that exist in the current workspace.

Since vector operations are so fundamental in R, it is important to be able to access (or index) elements within these vectors. Many different ways of indexing vectors are available. Here, we introduce several of these using the `x` as created above. The command `x[2]` returns the second element of `x` (the scalar 7), and `x[c(2, 4)]` returns the vector $(7, 13)$. The expressions `x[c(TRUE, TRUE, TRUE, TRUE, TRUE, FALSE)]`, `x[1:5]` and `x[-6]` all return a vector consisting of the first five elements in `x` (the last specifies all elements except the 6th). Knowledge and basic comfort with these approaches to vector indexing are important to effective use of R, as they can help with computational efficiency.

```
x[2]

[1] 7

x[c(2, 4)]

[1]  7 13

x[c(TRUE, TRUE, TRUE, TRUE, TRUE, FALSE)]

[1]  5  7  9 13 -4

x[1:5]

[1]  5  7  9 13 -4

x[-6]

[1]  5  7  9 13 -4
```

Vectors are *recycled* if needed; for example, when comparing each of the elements of a vector to a scalar.

```
x > 8

[1] FALSE FALSE  TRUE  TRUE FALSE FALSE
```

The above expression demonstrates the use of comparison operators (see `?Comparison`). Only the third and fourth elements of x are greater than 8. The function returns a logical value of either `TRUE` or `FALSE` (see `?Logic`).

A count of elements meeting the condition can be generated using the `sum()` function.

Other comparison operators include `==` (equal), `>=` (greater than or equal), `<=` (less than or equal and `!=` (not equal). Care needs to be taken in the comparison using `==` if noninteger values are present (see `all.equal()`).

```
sum(x > 8)

[1] 2
```

B.4.2 Operators

There are many operators defined in R to carry out a variety of tasks. Many of these were demonstrated in the sample session (assignment, arithmetic) and previous examples (comparison). Arithmetic operations include `+`, `-`, `*`, `/`, `^` (exponentiation), `%%` (modulus), and `%/%` (integer division). More information about operators can be found using the help system (e.g., `?"+"`). Background information on other operators and precedence rules can be found using `help(Syntax)`.

R supports Boolean operations (OR, AND, NOT, and XOR) using the `|`, `||`, `&`, `!` operators and the `xor()` function. The `|` is an "or" operator that operates on each element of a vector, while the `||` is another "or" operator that stops evaluation the first time that the result is true (see `?Logic`).

B.4.3 Lists

Lists in R are very general objects that can contain other objects of arbitrary types. List members can be named, or referenced using numeric indices (using the `[[` operator).

```
newlist <- list(first = "hello", second = 42, Bob = TRUE)
is.list(newlist)

[1] TRUE

newlist

$first
[1] "hello"

$second
[1] 42

$Bob
[1] TRUE

newlist[[2]]

[1] 42

newlist$Bob

[1] TRUE
```

The `unlist()` function flattens (makes a vector out of) the elements in a list (see also `relist()`). Note that unlisted objects are coerced to a common type (in this case `character`).

```
unlisted <- unlist(newlist)
unlisted

  first  second     Bob
"hello"    "42"  "TRUE"
```

B.4.4 Matrices

Matrices are like two-dimensional vectors. Thus, they are rectangular objects where all entries have the same type. We can create a 2×3 matrix, display it, and test for its type.

```
A <- matrix(x, 2, 3)
A

     [,1] [,2] [,3]
[1,]    5    9   -4
[2,]    7   13    8

is.matrix(A)    # is A a matrix?
```

```
[1] TRUE

is.vector(A)

[1] FALSE

is.matrix(x)

[1] FALSE
```

Note that comments are supported within R (any input given after a # character is ignored).

Indexing for matrices is done in a similar fashion as for vectors, albeit with a second dimension (denoted by a comma).

```
A[2, 3]

[1] 8

A[, 1]

[1] 5 7

A[1, ]

[1]  5  9 -4
```

B.4.5 Dataframes

Data sets are often stored in a `data.frame`, which is a special type of `list` that is more general than a `matrix`. This rectangular object, similar to a data table in other systems, can be thought of as a two-dimensional array with columns of vectors of the same length, but of possibly different types (as opposed to a matrix, which consists of vectors of the *same* type; or a list, whose elements needn't be of the same length). The function `read_csv()` in the `readr` package returns a `data.frame` object.

A simple `data.frame` can be created using the `data.frame()` command. Variables can be accessed using the `$` operator, as shown below (see also `help(Extract)`). In addition, operations can be performed by column (e.g., calculation of sample statistics). We can check to see if an object is a `data.frame` with `is.data.frame()`.

```
y <- rep(11, length(x))
y

[1] 11 11 11 11 11 11

ds <- data.frame(x, y)
ds

   x  y
1  5 11
2  7 11
3  9 11
4 13 11
```

```
5 -4 11
6  8 11

ds$x[3]

[1] 9

is.data.frame(ds)

[1] TRUE
```

Note that the use of `data.frame()` differs from the use of `cbind()`, which yields a `matrix` object (unless it is given data frames as inputs).

```
newmat <- cbind(x, y)
newmat

      x  y
[1,]  5 11
[2,]  7 11
[3,]  9 11
[4,] 13 11
[5,] -4 11
[6,]  8 11

is.data.frame(newmat)

[1] FALSE

is.matrix(newmat)

[1] TRUE
```

Data frames are created from matrices using `as.data.frame()`, while matrices are constructed from data frames using `as.matrix()`.

Although we strongly discourage its use, data frames can be attached to the workspace using the `attach()` command. The Google R Style guide provides similar advice [90]. Name conflicts are a common problem with `attach()` (see `conflicts()`, which reports on objects that exist with the same name in two or more places on the search path).

The `search()` function lists attached packages and objects. To avoid cluttering the name-space, the command `detach()` should be used once a data frame or package is no longer needed.

A number of R functions include a `data` argument to specify a data frame as a local environment. For others, the `with()` and `within()` commands can be used to simplify reference to an object within a data frame without attaching.

B.4.6 Attributes and classes

Many objects have a set of associated attributes (such as names of variables, dimensions, or classes) that can be displayed or sometimes changed. For example, we can find the dimension of the matrix defined earlier.

```
attributes(A)

$dim
[1] 2 3
```

Other types of objects within R include `list`s (ordered objects that are not necessarily rectangular), regression models (objects of class `lm`), and formulae (e.g., `y ~ x1 + x2`).

R supports *object-oriented programming* (see `help(UseMethod)`). As a result, objects in R have an associated *class* attribute, which changes the default behavior for some operations on that object. Many functions (called *generics*) have special capabilities when applied to objects of a particular class. For example, when `summary()` is applied to an `lm` object, the `summary.lm()` function is called. Conversely, `summary.aov()` is called when an `aov` object is given as argument. These class-specific implementations of generic functions are called *methods*. The `class()` function returns the classes to which an object belongs, while the `methods()` function displays all of the classes supported by a generic function.

```
head(methods(summary))

[1] "summary,ANY-method"              "summary,DBIObject-method"
[3] "summary,diagonalMatrix-method"   "summary,MySQLConnection-method"
[5] "summary,MySQLDriver-method"      "summary,MySQLResult-method"
```

Objects in R can belong to multiple classes, although those classes need not be nested. As noted above, generic functions are *dispatched* according the class attribute of each object. Thus, in the example below we create the `tbl` object, which belongs to multiple classes. When the `print()` function is called on `tbl`, R looks for a method called `print.tbl_df()`. If no such method is found, R looks for a method called `print.tbl()`. If no such method is found, R looks for a method called `print.data.frame()`. This process continues until a suitable method is found. If there is none, then `print.default()` is called.

```
tbl <- as.tbl(ds)
class(tbl)

[1] "tbl_df"     "tbl"        "data.frame"

print(tbl)

# A tibble: 6  2
      x     y
  <dbl> <dbl>
1     5    11
2     7    11
3     9    11
4    13    11
5    -4    11
6     8    11

print.data.frame(tbl)

   x  y
1  5 11
2  7 11
```

```
3  9 11
4 13 11
5 -4 11
6  8 11

print.default(tbl)

$x
[1]  5  7  9 13 -4  8

$y
[1] 11 11 11 11 11 11

attr(,"class")
[1] "tbl_df"     "tbl"        "data.frame"
```

There are a number of functions that assist with learning about an object in R. The `attributes()` command displays the attributes associated with an object. The `typeof()` function provides information about the underlying data structure of objects (e.g., logical, integer, double, complex, character, and list). The `str()` function displays the structure of an object, and the `mode()` function displays its storage mode. For data frames, the `glimpse()` function provides a useful summary of each variable.

A few quick notes on specific types of objects are worth relating here:

- A vector is a one-dimensional array of items of the same data type. There are six basic data types that a vector can contain: `logical`, `character`, `integer`, `double`, `complex`, and `raw`. Vectors have a `length()` but not a `dim()`. Vectors can have—but needn't have—`names()`.
- A `factor` is a special type of vector for categorical data. A factor has `level()`s. We change the reference level of a factor with `relevel()`. Factors are stored internally as integers that correspond to the id's of the factor levels.

  Pro Tip: Factors can be problematic and their use is discouraged since they can complicate some aspects of data wrangling. A number of R developers have encouraged the use of the `stringsAsFactors = FALSE` option.

- A `matrix` is a two-dimensional array of items of the same data type. A matrix has a `length()` that is equal to `nrow()` times `ncol()`, or the product of `dim()`.
- A `data.frame` is a `list` of vectors of the same length. This is like a matrix, except that columns can be of different data types. Data frames always have `names()` and often have `row.names()`.

Pro Tip: Do not confuse a `factor` with a `character` vector.

Note that data sets typically have class `data.frame`, but are of type `list`. This is because, as noted above, R stores data frames as special types of lists—a list of several vectors having the same length, but possibly having different types.

```
class(mtcars)

[1] "data.frame"

typeof(mtcars)

[1] "list"
```

Pro Tip: If you ever get confused when working with data frames and matrices, remember that a `data.frame` is a `list`, whereas a `matrix` is more like a `vector`.

B.4.7 Options

The `options()` function in R can be used to change various default behaviors. For example, the `digits` argument controls the number of digits to display in output. The current options are returned when `options()` is called, to allow them to be restored. The command `help(options)` lists all of the settable options.

B.4.8 Functions

Fundamental actions within R are carried out by calling *functions* (either built-in or user defined—see Appendix C for guidance on the latter). Multiple *arguments* may be given, separated by commas. The function carries out operations using the provided arguments and returns values (an object such as a vector or list) that are displayed (by default) or which can be saved by assignment to an object.

As an example, the `quantile()` function takes a numeric vector and returns the minimum, 25th percentile, median, 75th percentile, and maximum of the values in that vector. However, if an optional vector of quantiles is given, those quantiles are calculated instead.

```
vals <- rnorm(1000) # generate 1000 standard normals
quantile(vals)

    0%    25%    50%    75%   100%
-3.206 -0.665  0.023  0.757  2.891

quantile(vals, c(.025, .975))

 2.5% 97.5%
-2.11  1.97

# Return values can be saved for later use.
res <- quantile(vals, c(.025, .975))
res[1]

 2.5%
-2.11
```

Arguments (usually named) are available for many functions. The documentation specifies the default action if named arguments are not specified. For the `quantile()` function, there is a `type` argument that allows specification of one of nine algorithms for calculating quantiles.

```
res <- quantile(vals, probs = c(.025, .975), type = 3)
res

 2.5% 97.5%
-2.13  1.97
```

Some functions allow a variable number of arguments. An example is the paste() function. The calling sequence is described in the documentation as follows.

```
paste(..., sep = " ", collapse = NULL)
```

To override the default behavior of a space being added between elements output by paste(), the user can specify a different value for sep.

B.5 Add-ons: Packages

B.5.1 Introduction to packages

Additional functionality in R is added through packages, which consist of functions, data sets, examples, vignettes, and help files that can be downloaded from CRAN. The function install.packages() can be used to download and install packages. Alternatively, RStudio provides an easy-to-use Packages tab to install and load packages.

In many cases, add-on packages (see Appendix A) need to be installed prior to running the examples in this book. Packages that are not on CRAN can be installed using the install_github() function in the devtools package.

```
install.packages("mdsr")     # CRAN version
devtools::install_github("beanumber/mdsr")     # dev version
```

The library() function will load an installed package . For example, to install and load Frank Harrell's Hmisc() package, two commands are needed:

```
install.packages("Hmisc")
library(Hmisc)
```

If a package is not installed, running the library() command will yield an error. Here we try to load the Zelig package (which has not been installed):

```
> library(Zelig)
Error in library(Zelig) : there is no package called 'Zelig'
```

To rectify the problem, we install the package from CRAN.

```
> install.packages("Zelig")
trying URL 'https://cran.rstudio.com/macosx/contrib/3.3/Zelig_5.0-12.tgz'
Content type 'application/x-gzip' length 1398050 bytes (1.3 MB)
==================================================
downloaded 1.3 Mb
```

```
library(Zelig)
```

Packages can be installed from other repositories (e.g., OmegaHat or Bioconductor) by specifying the repository using the `repos` argument, or in the case of GitHub, using the `install_github()` function from the `devtools` package.

The `require()` function will test whether a package is available—this will load the library if it is installed, and generate a warning message if it is not (as opposed to `library()`, which will return an error).

B.5.2 CRAN task views

The *Task Views* on CRAN (`http://cran.r-project.org/web/views`) are a very useful resource for finding packages. These are curated listings of relevant packages within a particular application area (such as multivariate statistics, psychometrics, or survival analysis). Table B.2 displays the task views available as of 2016.

B.5.3 Session information

The `sessionInfo()` function provides version information about R as well as details of loaded packages.

```
> sessionInfo()
R version 3.3.2 (2016-10-31)
Platform: x86_64-apple-darwin13.4.0 (64-bit)
Running under: macOS Sierra 10.12.1
locale:
[1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
attached base packages:
[1] methods   stats     graphics  grDevices utils     datasets  base
```

The `R.Version()` function provides access to components of the version and platform status.

```
str(R.Version())

List of 14
 $ platform      : chr "x86_64-apple-darwin13.4.0"
 $ arch          : chr "x86_64"
 $ os            : chr "darwin13.4.0"
 $ system        : chr "x86_64, darwin13.4.0"
 $ status        : chr ""
 $ major         : chr "3"
 $ minor         : chr "3.2"
 $ year          : chr "2016"
 $ month         : chr "10"
 $ day           : chr "31"
 $ svn rev       : chr "71607"
 $ language      : chr "R"
 $ version.string: chr "R version 3.3.2 (2016-10-31)"
 $ nickname      : chr "Sincere Pumpkin Patch"
```

Task View	Subject
Bayesian	Bayesian Inference
ChemPhys	Chemometrics and Computational Physics
ClinicalTrials	Clinical Trial Design, Monitoring, and Analysis
Cluster	Cluster Analysis and Finite Mixture Models
DifferentialEquations	Differential Equations
Distributions	Probability Distributions
Econometrics	Econometrics
Environmetrics	Analysis of Ecological and Environmental Data
ExperimentalDesign	Design of Experiments (DoE) and Analysis of Experimental Data
ExtremeValue	Extreme Value Analysis
Finance	Empirical Finance
Genetics	Statistical Genetics
gR	gRaphical Models in R
Graphics	Graphic Displays and Dynamic Graphics and Graphic Devices and Visualization
HighPerformanceComputing	High-Performance and Parallel Computing with R
MachineLearning	Machine Learning and Statistical Learning
MedicalImaging	Medical Image Analysis
MetaAnalysis	Meta-Analysis
Multivariate	Multivariate Statistics
NaturalLanguageProcessing	Natural Language Processing
NumericalMathematics	Numerical Mathematics
OfficialStatistics	Official Statistics and Survey Methodology
Optimization	Optimization and Mathematical Programming
Pharmacokinetics	Analysis of Pharmacokinetic Data
Phylogenetics	Phylogenetics, Especially Comparative Methods
Psychometrics	Psychometric Models and Methods
ReproducibleResearch	Reproducible Research
Robust	Robust Statistical Methods
SocialSciences	Statistics for the Social Sciences
Spatial	Analysis of Spatial Data
SpatioTemporal	Handling and Analyzing Spatio-Temporal Data
Survival	Survival Analysis
TimeSeries	Time Series Analysis
WebTechnologies	Web Technologies and Services

Table B.2: A complete list of CRAN task views.

Sometimes it is desirable to remove a package (B.5.1) from the workspace. For example, a package might define a function with the same name as an existing function. Packages can be detached using the syntax `detach(package:PKGNAME)`, where `PKGNAME` is the name of the package. Objects with the same name that appear in multiple places in the environment can be accessed using the `location::objectname` syntax. As an example, to access the `mean()` function from the `base` package, the user would specify `base::mean()` instead of `mean()`.

The names of all variables within a given data set (or more generally for sub-objects within an object) are provided by the `names()` command. The names of all objects defined

within an R session can be generated using the `objects()` and `ls()` commands, which return a vector of character strings. RStudio includes an `Environment` tab that lists all the objects in the current environment.

The `print()` and `summary()` functions return the object or summaries of that object, respectively. Running `print(object)` at the command line is equivalent to just entering the name of the object, i.e. `object`.

B.5.4 Packages and name conflicts

Different package authors may choose the same name for functions that exist within base R (or within other packages). This will cause the other function or object to be *masked*. This can sometimes lead to confusion, when the expected version of a function is not the one that is called. The `find()` function can be used to determine where in the environment (workspace) a given object can be found.

```
find("mean")

[1] "package:mosaic" "package:Matrix" "package:base"
```

As an example where this might be useful, there are functions in the `base` and `Hmisc` packages called `units()`. The find command would display both (in the order in which they would be accessed).

```
library(Hmisc)
find("units")

[1] "package:Hmisc" "package:base"
```

When the `Hmisc` package is loaded, the `units()` function from the `base` package is masked and would not be used by default. To specify that the version of the function from the `base` package should be used, prefix the function with the package name followed by two colons: `base::units()`. The `conflicts()` function reports on objects that exist with the same name in two or more places on the search path.

B.5.5 Maintaining packages

The `update.packages()` function should be run periodically to ensure that packages are up-to-date (see also `packageVersion()`). The `packrat` package provides a comprehensive dependency system for R. This functionality can be extremely helpful to support reproducible analysis (see Appendix D), as the exact set of packages used for an analysis can be identified and accessed in a project. Support for `packrat` is built into RStudio.

As of December 2016, there were nearly 10,000 packages available from CRAN. This represents a tremendous investment of time and code by many developers [78]. While each of these has met a minimal standard for inclusion, it is important to keep in mind that packages in R are created by individuals or small groups, and not endorsed by the R core group. As a result, they do not necessarily undergo the same level of testing and quality assurance that the core R system does.

B.5.6 Installed libraries and packages

Running the command `library(help="PKGNAME")` will display information about an installed package. Alternatively, the `Packages` tab in RStudio can be used to list, install, and

update packages. Entries in the book that utilize packages include a line specifying how to access that library (e.g., `library(mosaic)`). More information about packages used in this book can be found in Appendix A.

B.6 Further resources

Hadley Wickham's *Advanced R* book [220] (`http://adv-r.had.co.nz`) is probably the best source for learning more about how R works. Extensive resources and documentation can be found at the Comprehensive R Archive Network (CRAN).

B.7 Exercises

Exercise B.1

A user has typed the following commands into the RStudio console.

```
obj1 <- 2:10
obj2 <- c(2, 5)
obj3 <- c(TRUE, FALSE)
obj4 <- 42
```

What values are returned by the following commands?

```
obj1 * 10
obj1[2:4]
obj1[-3]
obj1 + obj2
obj1 * obj3
obj1 + obj4
obj2 + obj3
sum(obj2)
sum(obj3)
```

Exercise B.2

A user has typed the following commands into the RStudio console.

```
a <- c(10, 15)
b <- c(TRUE, FALSE)
c <- c("happy", "sad")
```

What do each of the following commands return? Describe the class of the object as well as its value.

```
data.frame(a, b, c)
cbind(a, b)
rbind(a, b)
cbind(a, b, c)
list(a, b, c)[[2]]
```

Exercise B.3

A user has typed the following commands into the RStudio console.

```
mylist <- list(x1="sally", x2=42, x3=FALSE, x4=1:5)
```

What values do each of the following commands return?

```
is.list(mylist)
names(mylist)
length(mylist)
mylist[[2]]
mylist[["x1"]]
mylist$x2
length(mylist[["x4"]])
class(mylist)
typeof(mylist)
class(mylist[[4]])
typeof(mylist[[3]])
```

Exercise B.4

The following code undertakes some data analysis using the HELP (Health Evaluation and Linkage to Primary Care) trial.

```
library(mosaic)
ds <-
   read.csv("http://nhorton.people.amherst.edu/r2/datasets/helpmiss.csv")
summarise(group_by(select(filter(mutate(ds,
  sex = ifelse(female==1, "F", "M")), !is.na(pcs)), age, pcs, sex),
  sex), meanage=mean(age), meanpcs=mean(pcs),n=n())
```

Describe in words what computations are being done. Using the "pipe" notation, translate this code into a more readable version.

Exercise B.5

The following concepts should have some meaning to you: package, function, command, argument, assignment, object, object name, data frame, named argument, quoted character string. Construct an example of R commands that make use of at least four of these. Label which part of your example R command corresponds to each.

Exercise B.6

Which of these kinds of names should be wrapped with quotation marks when used in R?

1. function name
2. file name
3. the name of an argument in a named argument
4. object name

Exercise B.7

What's wrong with this statement?

```
help(NHANES, package <- "NHANES")
```

Exercise B.8

Consult the documentation for `CPS85` in the `mosaicData` package to determine the meaning of CPS.

Exercise B.9

For each of the following assignment statements, describe the error (or note why it does not generate an error).

```
result1 <- sqrt 10
result2 <-- "Hello to you!"
3result <- "Hello to you"
result4 <- "Hello to you
result5 <- date()
```