# Contents at a Glance

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>Preface</td>
<td>xii</td>
</tr>
<tr>
<td><strong>HOUR 1</strong></td>
<td>The R Community 1</td>
</tr>
<tr>
<td>2</td>
<td>The R Environment 11</td>
</tr>
<tr>
<td>3</td>
<td>Single-Mode Data Structures 33</td>
</tr>
<tr>
<td>4</td>
<td>Multi-Mode Data Structures 67</td>
</tr>
<tr>
<td>5</td>
<td>Dates, Times and Factors 103</td>
</tr>
<tr>
<td>6</td>
<td>Common R Utility Functions 115</td>
</tr>
<tr>
<td>7</td>
<td>Writing Functions: Part I 129</td>
</tr>
<tr>
<td>8</td>
<td>Writing Functions: Part II 151</td>
</tr>
<tr>
<td>9</td>
<td>Loops and Summaries 173</td>
</tr>
<tr>
<td>10</td>
<td>Importing and Exporting 217</td>
</tr>
<tr>
<td>11</td>
<td>Data Manipulation and Transformation 235</td>
</tr>
<tr>
<td>12</td>
<td>Efficient Data Handling in R 261</td>
</tr>
<tr>
<td>13</td>
<td>Graphics 287</td>
</tr>
<tr>
<td>14</td>
<td>The ggplot2 Package for Graphics 313</td>
</tr>
<tr>
<td>15</td>
<td>Lattice Graphics 345</td>
</tr>
<tr>
<td>16</td>
<td>Introduction to R Models and Object Orientation 379</td>
</tr>
<tr>
<td>17</td>
<td>Common R Models 411</td>
</tr>
<tr>
<td>18</td>
<td>Code Efficiency 455</td>
</tr>
<tr>
<td>19</td>
<td>Package Building 471</td>
</tr>
<tr>
<td>20</td>
<td>Advanced Package Building 489</td>
</tr>
<tr>
<td>21</td>
<td>Writing R Classes 505</td>
</tr>
<tr>
<td>22</td>
<td>Formal Class Systems 523</td>
</tr>
<tr>
<td>23</td>
<td>Dynamic Reporting 547</td>
</tr>
<tr>
<td>24</td>
<td>Building Web Applications with Shiny 561</td>
</tr>
<tr>
<td><strong>APPENDIX</strong></td>
<td>Installation 573</td>
</tr>
<tr>
<td>Index</td>
<td>579</td>
</tr>
</tbody>
</table>
# Table of Contents

**Preface** ................................................................. xii

**HOUR 1: The R Community** ........................................... 1
  A Concise History of R .............................................. 1
  The R Community .................................................... 3
  R Development ....................................................... 7
  Summary ............................................................... 8
  Q&A ................................................................. 8
  Workshop ............................................................. 9
  Activities ............................................................. 9

**HOUR 2: The R Environment** ......................................... 11
  Integrated Development Environments .............................. 11
  R Syntax ............................................................ 14
  R Objects ........................................................... 16
  Using R Packages .................................................. 23
  Internal Help ....................................................... 28
  Summary ............................................................ 29
  Q&A ................................................................. 30
  Workshop ............................................................. 30
  Activities ............................................................. 32

**HOUR 3: Single-Mode Data Structures** ................................ 33
  The R Data Types .................................................. 33
  Vectors, Matrices, and Arrays .................................... 34
  Vectors ............................................................. 35
  Matrices ............................................................ 49
  Arrays .............................................................. 58
  Relationship Between Single-Mode Data Objects ................. 60
  Summary ............................................................ 62
  Q&A ................................................................. 62
  Workshop ............................................................. 63
  Activities ............................................................. 64
**HOUR 4: Multi-Mode Data Structures** .................................................. 67
  Multi-Mode Structures ......................................................... 67
  Lists ........................................................................ 68
  Data Frames ................................................................. 86
  Exploring Your Data ....................................................... 93
  Summary ..................................................................... 98
  Q&A ........................................................................ 98
  Workshop ................................................................. 100
  Activities ............................................................... 101

**HOUR 5: Dates, Times, and Factors** ........................................... 103
  Working with Dates and Times .......................................... 103
  The lubridate Package .................................................... 107
  Working with Categorical Data ........................................ 108
  Summary ..................................................................... 112
  Q&A ........................................................................ 112
  Workshop ................................................................. 113
  Activities ............................................................... 114

**HOUR 6: Common R Utility Functions** ...................................... 115
  Using R Functions ............................................................ 115
  Functions for Numeric Data ............................................ 117
  Logical Data ................................................................. 121
  Missing Data ................................................................ 122
  Character Data ............................................................. 123
  Summary ..................................................................... 125
  Q&A ........................................................................ 126
  Workshop ................................................................. 126
  Activities ............................................................... 127

**HOUR 7: Writing Functions: Part I** .......................................... 129
  The Motivation for Functions ............................................ 129
  Creating a Simple Function ............................................ 130
  The If/Else Structure ..................................................... 136
  Summary ..................................................................... 146
## HOUR 8: Writing Functions: Part II

Errors and Warnings ........................................... 151
Checking Inputs ............................................... 155
The Ellipsis ..................................................... 157
Checking Multivalue Inputs ................................. 162
Using Input Definition ....................................... 164
Summary ......................................................... 168
Q&A ............................................................. 168
Workshop ......................................................... 170
Activities ......................................................... 171

## HOUR 9: Loops and Summaries

Repetitive Tasks ................................................ 173
The “apply” Family of Functions ............................ 181
The `apply` Function ......................................... 183
The `lapply` Function ....................................... 195
The `sapply` Function ....................................... 204
The `tapply` Function ....................................... 208
Summary ......................................................... 213
Q&A ............................................................. 213
Workshop ......................................................... 214
Activities ......................................................... 216

## HOUR 10: Importing and Exporting

Working with Text Files ...................................... 217
Relational Databases ........................................... 223
Working with Microsoft Excel ............................... 226
Summary ......................................................... 231
Q&A ............................................................. 232
Workshop ......................................................... 232
Activities ......................................................... 233
# Contents

## HOUR 11: Data Manipulation and Transformation
- Sorting ........................................ 236
- Appending ..................................... 237
- Merging ........................................ 238
- Duplicate Values ............................. 241
- Restructuring ................................ 242
- Data Aggregation ............................. 249
- Summary ..................................... 258
- Q&A ........................................... 258
- Workshop .................................... 259
- Activities ................................... 259

## HOUR 12: Efficient Data Handling in R
- dplyr: A New Way of Handling Data ........ 261
- Efficient Data Handling with data.table ... 273
- Summary ..................................... 282
- Q&A ........................................... 283
- Workshop .................................... 283
- Activities ................................... 284

## HOUR 13: Graphics
- Graphics Devices and Colors ............... 287
- High-Level Graphics Functions .............. 289
- Low-Level Graphics Functions .............. 298
- Graphical Parameters ........................ 304
- Controlling the Layout ...................... 305
- Summary ..................................... 308
- Q&A ........................................... 309
- Workshop .................................... 309
- Activities ................................... 311

## HOUR 14: The ggplot2 Package for Graphics
- The Philosophy of ggplot2 ................. 313
- Quick Plots and Basic Control ............. 314
- Changing Plot Types ......................... 317
Aesthetics .......................................................... 320
Paneling (a.k.a Faceting) ............................................ 328
Custom Plots .......................................................... 333
Themes and Layout .................................................. 338
The ggvis Evolution .................................................. 342
Summary .............................................................. 342
Q&A ................................................................. 343
Workshop ............................................................ 343
Activities .............................................................. 344

**HOUR 15: Lattice Graphics** ........................................... 345

The History of Trellis Graphics ................................. 345
The Lattice Package ................................................ 346
Creating a Simple Lattice Graph ............................... 346
Graph Options ....................................................... 356
Multiple Variables .................................................. 358
Groups of Data ....................................................... 360
Using Panels ......................................................... 362
Controlling Styles .................................................... 372
Summary .............................................................. 376
Q&A ................................................................. 377
Workshop ............................................................ 378
Activities .............................................................. 378

**HOUR 16: Introduction to R Models and Object Orientation** ............................................. 379

Statistical Models in R ............................................. 379
Simple Linear Models ............................................. 380
Assessing a Model in R ........................................... 382
Multiple Linear Regression .................................... 391
Interaction Terms .................................................... 396
Factor Independent Variables .................................. 398
Variable Transformations ....................................... 402
R and Object Orientation ........................................ 405
Summary .............................................................. 407
# Contents

- Q&A ........................................................................................................ 408
- Workshop .................................................................................................. 408
- Activities .................................................................................................... 409

**HOUR 17: Common R Models** .................................................................. 411
- Generalized Linear Models ....................................................................... 411
- Nonlinear Models ..................................................................................... 423
- Survival Analysis ...................................................................................... 430
- Time Series Analysis ................................................................................ 441
- Summary ................................................................................................... 452
- Q&A .......................................................................................................... 452
- Workshop .................................................................................................. 452
- Activities .................................................................................................... 453

**HOUR 18: Code Efficiency** ....................................................................... 455
- Determining Efficiency ............................................................................ 455
- Initialization .............................................................................................. 458
- Vectorization ............................................................................................. 459
- Using Alternative Functions .................................................................... 462
- Managing Memory Usage ....................................................................... 463
- Integrating with C++ ............................................................................... 464
- Summary ................................................................................................... 468
- Q&A .......................................................................................................... 469
- Workshop .................................................................................................. 469
- Activities .................................................................................................... 470

**HOUR 19: Package Building** ................................................................... 471
- Why Build an R Package? ........................................................................ 471
- The Structure of an R Package ................................................................. 472
- Code Quality ............................................................................................. 476
- Automated Documentation with roxygen2 ............................................. 477
- Building a Package with devtools .......................................................... 482
- Summary ................................................................................................... 485
- Q&A .......................................................................................................... 485
- Workshop .................................................................................................. 486
- Activities .................................................................................................... 487
HOUR 23: Dynamic Reporting ........................................ 547
  What Is Dynamic Reporting? ........................................ 547
  An Introduction to knitr ........................................... 548
  Simple Reports with RMarkdown .................................. 548
  Reporting with LaTeX ............................................. 553
  Summary ............................................................ 557
  Q&A ................................................................. 558
  Workshop ............................................................ 558
  Activities ............................................................ 559

HOUR 24: Building Web Applications with Shiny ..................... 561
  A Simple Shiny Application ........................................ 561
  Reactive Functions ................................................. 566
  Interactive Documents ............................................. 569
  Sharing Shiny Applications ....................................... 570
  Summary ............................................................ 571
  Q&A ................................................................. 571
  Workshop ............................................................ 571
  Activities ............................................................ 572

APPENDIX: Installation .................................................. 573
  Installing R .......................................................... 573
  Installing Rtools for Windows .................................... 575
  Installing the RStudio IDE ....................................... 577

Index ................................................................. 579
Preface

Mango Solutions has been teaching face-to-face R training courses to business professionals and academics alike for over 13 years. In this time, we've seen R grow from its early days as a low cost alternative to S-PLUS and SAS to become the leading analytical programming language in the world today, with several thousand contributors and somewhere upward of a million users. R is widely used throughout academia and is commercially supported by the likes of Microsoft, Google, HP, and Oracle.

In Mango’s face-to-face training program we teach R to statisticians, data scientists, physicists, biologists, chemists, geographers, and psychologists among others. All are looking to R to help improve the way they analyze their data in a professional environment. Our aim with this book was to take tried and tested training material and turn it into a lasting resource for anyone looking to learn R for analysis.

Who Should Read This Book?

This book is designed for professional statisticians, data scientists, and analysts looking to widen the scope of analytical tools available to them by learning R. Although it is expected that you might have some programming knowledge in another analytical application or language for data analysis, such as SAS, Python, or Excel/VBA, this is not a prerequisite. This book is suitable for complete novices in programming. From the start, we do not assume any prior knowledge of R; however, those familiar with the basics may find that they can jump straight to later chapters.

What Should You Expect from This Book?

This book is designed to take you from the basics of the R language through common tasks in data science, including data manipulation, visualization, and modeling, to elements of the language that will allow you to produce high-quality, production-ready code. As with our face-to-face training, this book is structured around simple and easy-to-follow examples, all of which are available to download from the book’s website (http://www.mango-solutions.com/wp/teach-yourself-r-in-24-hours-book). Throughout, we introduce good practices for writing code as well as provide tips and tricks from our combined experience in R development.
By the end of this book, you should have a good understanding of the fundamentals of R as well as many of the most commonly used packages. You should have a good understanding of what makes well written R code and how to implement this yourself.

**How Is This Book Organized?**

This book is designed to guide you through everything you need to know to get started with the R language and then introduce additional elements of the language for specific tasks.

The following is an outline of each of the hours and what to expect:

**Hour 1, “The R Community”**—In this hour, we start by looking at how R evolved from the S language to become the all-purpose data science programming language that it is today. The R community offers a plethora of help and support options for users. We look at some of the better-known options during this hour.

**Hour 2, “The R Environment”**—In this hour, we start a new R session via RStudio, type some basic commands, and explore the idea of an R “object.” You will be more formally introduced to the concept of an R package.

**Hour 3, “Single-Mode Data Structures”**—In this hour, we describe the standard types of data found in R and introduce three key structures that can be used to store these data types: vectors, matrices, and arrays. We illustrate the ways in which these structures can be created and manage these data structures with a focus on how we can extract data from them.

**Hour 4, “Multi-Mode Data Structures”**—The majority of data sources contain a mixture of data types, which we need to store together in a simple, effective format. In this hour, we focus on two key data structures that allow us to store “multi-mode” data: lists and data frames. We illustrate the ways in which these structures can be created and manage these data structures with a focus on how we can extract data from them. We also look at how these two data structures can be effectively used in our day-to-day work.

**Hour 5, “Dates, Times, and Factors”**—In this hour, you learn more about some of the special data types in R that enable us to work with dates and times and with categorical data.

**Hour 6, “Common R Utility Functions”**—In this hour, we introduce you to some of the most common utility functions in R that you will find yourself using every day.

**Hour 7, “Writing Functions: Part I”**—One of the strengths of R is that we can extend it by writing our own functions, allowing us to create utilities that can perform a variety of tasks. In this hour, we look at ways in which we can create our own functions, specify
inputs, and return results to the user. We also discuss the “if/else” structure in R and use it to control the flow of code within a function.

**Hour 8, “Writing Functions: Part II”**—This hour looks at a range of advanced function-writing topics, such as returning error messaging, checking whether inputs are appropriate to our functions, and the use of function “ellipses.”

**Hour 9, “Loops and Summaries”**—In this hour, you see how we can apply simple functions and code in a more “applied” fashion. This allows us to perform tasks repeatedly over sections of our data without the need to produce verbose, repetitive code.

**Hour 10, “Importing and Exporting”**—In this hour, we introduce common methods for importing and exporting data. By the end of the hour you will have seen how R can be used to read and write flat files and connect to database management systems (DBMSs) as well as Microsoft Excel.

**Hour 11, “Data Manipulation and Transformation”**—As data scientists and statisticians, we rarely get to control the structure and format of our data. Now we will look a little closer at the structure of our data. Several approaches to data manipulation in R have evolved over time. In this hour, we start by looking at what could be called “traditional” approaches to the data manipulation tasks of sorting, setting, and merging. We then look at the popular packages `reshape`, `reshape2`, and `tidyr` for data restructuring.

**Hour 12, “Efficient Data Handling in R”**—We begin the hour by looking at the incredibly popular `dplyr` package. The `data.table` package is a standalone package for data manipulation that offers greater efficiency for very large data.

**Hour 13, “Graphics”**—After all the manipulations to our data, we want to be able to start to do something with it. In this hour, we look at how we can create graphics using the base graphics functionality, including how to send your graphics to devices such as a PDF and the standard graphics functions. We finally look at how to control the layout of graphics on the page.

**Hour 14, “The ggplot2 Package for Graphics”**—In this hour, we look at the hugely popular `ggplot2` package, developed by Hadley Wickham for creating high-quality graphics.

**Hour 15, “Lattice Graphics”**—Here we will look at a third way of creating graphics: using the `lattice` package. This graphic system is well suited to graphing highly grouped data, with the code designed to closely resemble the modeling capabilities of R.

**Hour 16, “Introduction to R Models and Object Orientation”**—In this hour, we see how to fit a simple linear model and assess its performance using a range of textual and
graphical methods. Beyond this, we introduce “object orientation” and see how the R statistical modeling framework is built on this concept.

Hour 17, “Common R Models”—In this hour, we extend the ideas of the previous hour to other modeling approaches. Specifically, we look at Generalized Linear Models, nonlinear models, time series models, and survival models.

Hour 18, “Code Efficiency”—In this hour, we look at some of the techniques we can use to improve the efficiency and, importantly, the professionalism of our R code.

Hour 19, “Package Building”—When we put our code into a package, it forces us to ensure that our code is of a high standard and we are adhering to good practices, such as documenting our code. We focus here on making sure our code is well written and documented, the starting point for high-quality, professional code that is easy to share and reuse.

Hour 20, “Advanced Package Building”—There are a number of ways we can extend a package to make it more robust to changes and easier for users to get started with. You learn the most common of these extra components in this hour.

Hour 21, “Writing R Classes”—In this hour, we take a general look at some key features of object-oriented programming before focusing in on R’s S3 implementation.

Hour 22, “Formal Class Systems”—During this hour, we look at the more formal S4 and Reference Class systems in R. Along the way, you will be introduced to concepts such as validity checking, multiple dispatch, message-passing object orientation, and mutable objects.

Hour 23, “Dynamic Reporting”—Up to this point we have seen the fundamentals of the R language as well as the aspects of R that allow us to ensure that we write high-quality, well-documented, and easily shareable code. In this hour, we take a look at one of the ways you can extend your use of R, specifically for simplifying the generation of reports that rely heavily on R-generated output.

Hour 24, “Building Web Applications with Shiny”—Although you may initially be put off by the idea of building a web application, we introduce a package that allows you to generate web applications entirely in R, writing only R code. This is currently one of the most popular packages available in R, with more and more packages being added to CRAN that use this framework.

About the Sample Code

Throughout this book, we have included examples of the concepts that are being introduced. You may notice that the code is prefixed with the symbols “>” and “+”. These are the
R prompt and continuation characters and do not need to be entered when writing code. We have used the formatting conventions of function for a function name and package for a package name.


NOTE

Code-Continuation Arrows and Listing Line NumbersYou might see code-continuation arrows (►) occasionally in this book to indicate when a line of code is too long to fit on the printed page. Also, some listings have line numbers and some do not. The listings that have line numbers have them so that we can reference code by line; the listings that do not have line numbers are not referenced by line.

Contacting the Authors

If you have any comments or questions about this book, please drop us an email at rin24hours@mango-solutions.com.
About the Authors

Andy Nicholls has a Master of Mathematics degree from the University of Bath and Master of Science in Statistics with Applications in Medicine from the University of Southampton. Andy worked as a Senior Statistician in the pharmaceutical industry for a number of years before joining Mango Solutions as an R consultant in 2011. Since joining Mango, Andy has taught more than 50 on-site R training courses and has been involved in the development of more than 30 R packages. Today, he manages Mango Solution’s R consultancy team and continues to be a regular contributor to the quarterly LondonR events, by far the largest R user group in the UK, with over 1,000 meet-up members. Andy lives near the historical city of Bath, UK with his wonderful, tolerant wife and son.

Richard Pugh has a first-class Mathematics degree from the University of Bath. Richard worked as a statistician in the pharmaceutical industry before joining Insightful, the developers of S-PLUS, joining the pre-sales consulting team. Richard’s role at Insightful included a variety of activities, providing a range of training and consulting services to blue-chip customers across many sectors. In 2002, Richard co-founded Mango Solutions, developing the company and leading technical efforts around R and other analytic software. Richard is now Mango’s Chief Data Scientist and speaks regularly at data science and R events. Richard lives in Bradford on Avon, UK with his wife and two kids, and spends most of his “spare” (ha!) time renovating his house.

Aimee Gott has a PhD in Statistics from Lancaster University where she also completed her undergraduate and master’s degrees. As Training Lead, Aimee has delivered over 200 days of training for Mango. She has delivered on-site training courses in Europe and the U.S. in all aspects of R, as well as shorter workshops and online webinars. Aimee oversees Mango’s training course development across the data science pipeline, and regularly attends R user groups and meet-ups. In her spare time, Aimee enjoys learning European languages and documenting her travels through photography.
Dedications

This book is dedicated to my wife, for her love and support and for putting up with losing our summer to all the late nights, and to my baby boy who learnt to sit up, eat, crawl, and walk whilst this book was being written! —Andy Nicholls

This book is dedicated to my family for having to put up with me writing the book at weekends. —Richard Pugh

To Stephen, Carol, Richard, and Kirstie. —Aimee Gott

Acknowledgments

Throughout the process of writing, many people have taken time to assist us, guide us, and shape this book. First of all, thanks go to Andy Miskell, Jeff Stagg, Mike K. Smith, and Susan Duke, who took the time to review our initial outline. Thanks also go to the consultancy team at Mango and Stephen Kaluzny of TIBCO who were all able to answer our questions while writing.

We would also like to thank all those who have been involved in the production of this book. In particular we would like to thank Elaine Wiley (production editor), Trina MacDonald (acquisitions editor), Songlin Qiu (development editor), Olivia Basegio (publisher assistant), Stephanie Locke (technical editor), Bart Reed (copyeditor), and Katie Matejka (proofreader).
We Want to Hear from You!

As the reader of this book, you are our most important critic and commentator. We value your opinion and want to know what we’re doing right, what we could do better, what areas you’d like to see us publish in, and any other words of wisdom you’re willing to pass our way.

We welcome your comments. You can email or write to let us know what you did or didn’t like about this book—as well as what we can do to make our books better.

Please note that we cannot help you with technical problems related to the topic of this book.

When you write, please be sure to include this book’s title and author as well as your name and email address. We will carefully review your comments and share them with the author and editors who worked on the book.

Email: errata@informit.com

Mail: Addison-Wesley/Prentice Hall Publishing
      ATTN: Reader Feedback
      330 Hudson Street
      7th Floor
      New York, New York, 10013

Reader Services

Register your copy of Teach Yourself R in 24 Hours at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account.* Enter the product ISBN, 9780672338489, and click Submit. Once the process is complete, you will find any available bonus content under Registered Products.

*Be sure to check the box that you would like to hear from us in order to receive exclusive discounts on future editions of this product.
What You’ll Learn in This Hour:

▶ What a list object is
▶ How to create and manipulate a data frame
▶ How to perform an initial investigation in the structure of our data

The majority of data sources contain a mixture of data types, which we need to store together in a simple, effective format. The “single-mode” structures introduced in the last hour are useful basic data objects, but are not sufficiently sophisticated to store data containing multiple “modes.” In this hour, we focus on two key data structures that allow us to store “multi-mode” data: lists and data frames. We will illustrate the ways in which these structures can be created and managed, with a focus on how to extract data from them. We also look at how these two data structures can be effectively used in our day-to-day work.

Multi-Mode Structures

In the last hour, we examined the three structures designed to hold data in R:

▶ Vectors—Series of values
▶ Matrices—Rectangular structures with rows and columns
▶ Arrays—Higher dimension structures (for example, 3D and 4D arrays)

Although these objects provide us with a range of useful functionality, they are restricted in that they can only hold a single “mode” of data. This is illustrated in the following example:

```r
> c(1, 2, 3, "Hello")                # Multiple modes
[1] "1" "2" "3" "Hello"
> c(1, 2, 3, TRUE, FALSE)            # Multiple modes
[1] 1 2 3 1 0
> c(1, 2, 3, TRUE, FALSE, "Hello")   # Multiple modes
[1] "1" "2" "3" "TRUE" "FALSE" "Hello"
```
As you can see, when we attempt to store more than one mode of data in a single-mode structure, the object (and its contents) will be converted to a single mode.

The preceding example uses a vector to illustrate this behavior, but let’s suppose we want to store a rectangular “dataset” using a matrix. For example, we might attempt to create a matrix that contains the forecast temperatures for New York over the next five days:

```r
> weather <- cbind(
+   Day = c("Saturday", "Sunday", "Monday", "Tuesday", "Wednesday"),
+   Date = c("Jul 4", "Jul 5", "Jul 6", "Jul 7", "Jul 8"),
+   TempF = c(75, 86, 83, 83, 87)
+ )
> weather
Day      Date    TempF
[1,] "Saturday"  "Jul 4" "75"
[2,] "Sunday"    "Jul 5" "86"
[3,] "Monday"    "Jul 6" "83"
[4,] "Tuesday"   "Jul 7" "83"
[5,] "Wednesday" "Jul 8" "87"
```

From the quotation marks, it is clear that R has converted all the data to character values, which can be confirmed by looking at the mode of this matrix structure:

```r
> mode(weather)    # The mode of the matrix
[1] "character"
```

This reinforces the need for data structures that allow us to store data of multiple modes. R provides two “multi-mode” data structures:

- Lists—Containers for any objects
- Data frames—Rectangular structures with rows and columns

**Lists**

The list is considered perhaps the most complex data object in R, and many R programmers will go to great lengths to avoid the use of lists in their structures. This perceived complexity, perhaps, stems from a lack of clarity over what a list “looks like.” Other structures, such as vectors and matrices, are relatively easy to visualize, and are therefore easier to adopt and manage.

Despite this, lists are simple structures that can be used to perform a number of complex operations.
What Is a List?

Lists are simply containers for other objects. The objects stored in a list can be of any type (for example, “matrix” or “vector”) and any mode. Therefore, you can create a list containing the following, for example:

▶ A character vector
▶ A numeric matrix
▶ A logical array
▶ Another list

When discussing lists, some people use the analogy of a box. For example, you might do the following:

▶ Create an empty box.
▶ Put some “things” into the box.
▶ Look into the box to see what things are in there.
▶ Take things back out of the box.

In a similar way, in this section, we will look at how to do the following:

▶ Create an empty list.
▶ Put objects into the list.
▶ Look at the number (and names) of objects in the list.
▶ Extract elements from the list.

Creating an Empty List

You create a list using the list function. The simplest list you can create is an empty list, like this:

```r
> emptyList <- list()
> emptyList
list()
```

Later, you will see how to add elements to this empty list.
Creating a Non-Empty List

More commonly, you’ll create a list and add initial elements to it at the same time. You achieve this by specifying a comma-separated set of objects within the `list` function:

```r
> aVector <- c(5, 7, 8, 2, 4, 3, 9, 0, 1, 2)
> aMatrix <- matrix(LETTERS[1:6], nrow = 3)
> unnamedList <- list(aVector, aMatrix)
> unnamedList
```

```
[[1]]
[1] 5 7 8 2 4 3 9 0 1 2

[[2]]
  [,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"
```

In this example, we created two objects (`aVector` and `aMatrix`) and then created a list (`unnamedList`) containing copies of these objects.

**NOTE**

**Original Objects**

When you create lists in this way, you take copies of the objects (`aVector` and `aMatrix` in this example). The original objects are not impacted by this action (that is, they are not edited, moved, changed, or deleted).

If you only need the objects within the list, you could create the objects as you specify the list, like this:

```r
> unnamedList <- list(c(5, 7, 8, 2, 4, 3, 9, 0, 1, 2),
                        matrix(LETTERS[1:6], nrow = 3))
> unnamedList
```

```
[[1]]
[1] 5 7 8 2 4 3 9 0 1 2

[[2]]
  [,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"
```
**Creating a List with Element Names**

When you create a list, you can optionally assign names to the elements. This helps you when you’re referencing elements in the list later.

```r
> namedList <- list(VEC = aVector, MAT = aMatrix)
> namedList
$VEC
[1] 5 7 8 2 4 3 9 0 1 2

$MAT
[,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"
```

As before, you can also create the (named) objects as you’re creating the list:

```r
> namedList <- list(VEC = c(5, 7, 8, 2, 4, 3, 9, 0, 1, 2),
+                   MAT = matrix( LETTERS[1:6], nrow = 3))
> namedList
$VEC
[1] 5 7 8 2 4 3 9 0 1 2

$MAT
[,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"
```

**Creating a List: A Summary**

You have now seen a few different ways of creating a list. It is worth recappping the ways in which we created the lists with some code examples:

```r
> # Create an empty list
> emptyList <- list()

> # 2 Ways of Creating an unnamed list containing a vector and a matrix
> unnamedList <- list(aVector, aMatrix)
> unnamedList <- list(c(5, 7, 8, 2, 4, 3, 9, 0, 1, 2),
+                      matrix( LETTERS[1:6], nrow = 3))

> # 2 Ways of Creating a named list containing a vector and a matrix
> namedList <- list(VEC = aVector, MAT = aMatrix)
> namedList <- list(VEC = c(5, 7, 8, 2, 4, 3, 9, 0, 1, 2),
+                   MAT = matrix( LETTERS[1:6], nrow = 3))
```
In these examples, we created three lists that we will use as examples over the next few sections:

```r
> emptyList       # An empty list
list()

> unnamedList     # A list with unnamed elements
[[1]]
  [1] 5 7 8 2 4 3 9 0 1 2

[[2]]
  [,1] [,2]
[1,] "A"  "D"
[2,] "B"  "E"
[3,] "C"  "F"

> namedList       # A list with element names
$VEC
  [1] 5 7 8 2 4 3 9 0 1 2

$MAT

[,1] [,2]
[1,] "A"  "D"
[2,] "B"  "E"
[3,] "C"  "F"
```

**NOTE**

**Printing Style**

Notice the difference in printing when a list has element names versus when there are no element names: Elements are indexed with double square brackets (for example, `[[1]]`) for "unnamed" lists, and with dollar symbols (for example, `$VEC`) for "named" lists. This gives you a hint as to how you'll be able to reference the elements of a list later.

**List Attributes**

As with single-mode structures, a set of functions allows you to query some of the list attributes. Specifically, you can use the `length` function to query the number of elements in the list, and the `names` function to return the element names.

The `length` function returns the number of elements in the list, as shown here:

```r
> length(emptyList)
[1] 0
> length(unnamedList)
[1] 2
> length(namedList)
[1] 2
```
The `names` function returns the names of the elements in the list, or `NULL` if there are no elements or no element names assigned:

```r
> names(emptyList)
NULL
> names(unnamedList)
NULL
> names(namedList)
[1] "VEC" "MAT"
```

With single-mode data structures, we additionally used the `mode` function to return the type of data they held. Because lists are multi-mode structures, there is no longer a single mode of data being stored, so the word “list” is returned:

```r
> mode(emptyList)
[1] "list"
> mode(unnamedList)
[1] "list"
> mode(namedList)
[1] "list"
```

## Subscripting Lists

Two types of list subscripting can be performed:

- You can create a subset of the list, returning a shorter list.
- You can reference a single element within the list.

### Subsetting the List

You can use square brackets to select a subset of an existing list. The return object will itself be a list.

```
LIST [ Input specifying the subset of list to return ]
```

As with vectors, you can put one of five input types in the square brackets, as shown in Table 4.1.

<table>
<thead>
<tr>
<th>Input</th>
<th>Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blank</td>
<td>All values of the list are returned.</td>
</tr>
<tr>
<td>A vector of positive integers</td>
<td>Used as an index of list elements to return.</td>
</tr>
<tr>
<td>A vector of negative integers</td>
<td>Used as an index of list elements to omit.</td>
</tr>
<tr>
<td>Input</td>
<td>Effect</td>
</tr>
<tr>
<td>-------------------------------------------</td>
<td>--------------------------------------------------</td>
</tr>
<tr>
<td>A vector of logical values</td>
<td>Only corresponding TRUE elements are returned.</td>
</tr>
<tr>
<td>A vector of character values</td>
<td>Refers to the names of elements to return.</td>
</tr>
</tbody>
</table>

To illustrate the subsetting of lists, we will use the `namedList` object created earlier.

**Blank Subscripts**

If you use a blank subscript, the whole of the list is returned:

```r
> namedList [ ]  # Blank subscript
$VEC
  [,1] [1,2]
[1,] 5 7 8 2 4 3 9 0 1 2

$MAT
  [,1] [,2]
[1,] "A"  "D"
[2,] "B"  "E"
[3,] "C"  "F"
```

**Positive Integer Subscripts**

If you use a vector of positive integers, it is used as an index of elements to return:

```r
> subList <- namedList [ 1 ]  # Return first element
> subList                      # Print the new object
$VEC
  [,1] [1,2]
[1,] 5 7 8 2 4 3 9 0 1 2

> length(subList)              # Number of elements in the list
[1] 1
> class(subList)               # Check the "class" of the object
[1] "list"
```

As you can see from this example, the return object (saved as `subList` here) is itself a list. You can also use the `class` function to check the type of object, and it confirms `subList` is a list object.

**NOTE**

**An Object’s Class**

This is the first time in this book you’ve seen the `class` function used. It returns the type of objects, whereas the `mode` function returns the type of data held in an object. Let’s illustrate this distinction with a numeric matrix:
> aMatrix <- matrix(1:6, nrow = 2)  # Create a numeric matrix
> aMatrix  # Print the matrix
     [,1] [,2] [,3]
[1,]    1    3    5
[2,]    2    4    6

> mode(aMatrix)  # Mode of data held in this object
[1] "numeric"

> class(aMatrix)  # Type (or "class") of object
[1] "matrix"

### Negative Integer Subscripts

You can provide a vector of negative integers to specify the index of list elements to omit:

```r
> namedList
$VEC
[1]  5  7  8  2  4  3  9  0  1  2

$MAT
 [,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"

> namedList [-1]  # Return all but the first element
$MAT
 [,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"
```

### Logical Value Subscripts

You can provide a vector of logical integers to specify the list elements to return and omit:

```r
> namedList
$VEC
[1]  5  7  8  2  4  3  9  0  1  2

$MAT
 [,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"

> namedList [ c(T, F) ]  # Vector of logical values
$VEC
[1]  5  7  8  2  4  3  9  0  1  2
```
Character Value Subscripts

If your list has element names, you can provide a vector of character values to identify the (named) elements you wish to return:

```r
> namedList
$VEC
[1] 5 7 8 2 4 3 9 0 1 2

$MAT
[,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"

> namedList [ "MAT" ]        # Vector of Character values
$MAT
[,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"
```

Reference List Elements

In the last section, you saw that you can reference a list using square brackets to “subset” the list (that is, return a list containing only a subset of the original elements). More commonly, you’ll want to reference a specific element within your list.

You can reference elements of a list in two ways:

- You can use “double” square brackets.
- If there are element names, you can use the $ symbol.

Double Square Bracket Referencing

You can directly reference an element of a list using double square brackets. Although there are a number of uses of the double square brackets, the most common use is to supply a single integer index to refer to the element to extract:

```r
> namedList              # The original list
$VEC
[1] 5 7 8 2 4 3 9 0 1 2

$MAT
[,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"
```
When you use double square brackets in this way, you are directly referencing the objects contained within the list, as supported by the result of the `mode` function call. This is in contrast to the use of the single square bracket earlier, where we extracted a subset of the list itself:

```r
> namedList[[1]]  # The first element
[1] 5 7 8 2 4 3 9 0 1 2
> namedList[[2]]  # The second element
 [,1] [,2]
[1,] "A" "D"
[2,] "B" "E"
[3,] "C" "F"
> mode(namedList[[2]])  # The mode of the second element
[1] "character"
```

**Referencing Named Elements with $**

If the elements of your list are named, you can use the `$` symbol to directly reference them. As such, the following lines of code are equivalent ways of referencing the first (the "VEC") element of our `namedList` object:

```r
> namedList  # Print the original list
$VEC
[1] 5 7 8 2 4 3 9 0 1 2
> namedList[[1]]  # Return the first element
[1] 5 7 8 2 4 3 9 0 1 2
> namedList$VEC  # Return the "VEC" element
[1] 5 7 8 2 4 3 9 0 1 2
```
Double Square Brackets versus $ 

The $ symbol provides a more intuitive way of referencing named list elements, which is also more aesthetically pleasing than the use of double square brackets. We tend to use double square brackets when there are no element names assigned, and use $ when names exist. Here’s an example:

```r
> unnamedList       # List with no element names
[[1]]
[1] 5 7 8 2 4 3 9 0 1 2

[[2]]
[,1] [,2]
[1,] "A"  "D"
[2,] "B"  "E"
[3,] "C"  "F"

> unnamedList[[1]]  # First element
[1] 5 7 8 2 4 3 9 0 1 2

> namedList          # List with element names
$VEC
[1] 5 7 8 2 4 3 9 0 1 2

$MAT
[,1] [,2]
[1,] "A"  "D"
[2,] "B"  "E"
[3,] "C"  "F"

> namedList$VEC      # The "VEC" element
[1] 5 7 8 2 4 3 9 0 1 2

TIP

Shortened $ Referencing

When you use the $ symbol, you only need to provide enough of the name so that R understands which element you are referring to. This is illustrated in the following example:

```r
> aList <- list( first = 1, second = 2, third = 3, fourth = 4 )
> aList$s     # Returns the second
[1] 2
> aList$s$f  # Returns the first
[1] 1
Although it is possible to use shortened referencing in this way, it can lead to less maintainable and readable code, and should be avoided where possible when creating scripts.

**Adding List Elements**

You can add elements to a list in one of two ways:

- By directly adding an element with a specific name or in a specific position
- By combing lists together

### Directly Adding a List Element

You can add a single element to a list by assigning it into a specific index or name. The syntax mirrors that of the “Double Square Brackets versus $” section earlier. For example, let’s add a single element to our empty list:

```r
> emptyList <- list()  # Empty list
> emptyList[[1]] <- LETTERS[1:5]  # Add an element
> emptyList             # Updated (non)empty list

[1] "A" "B" "C" "D" "E"
```

Instead of using the double square brackets, we can use the $ symbol to add a “named” element to a list:

```r
> emptyList$ABC <- LETTERS[1:5]  # Add an element
> emptyList$ABC                # Updated (non)empty list

$ABC

[1] "A" "B" "C" "D" "E"
```
NOTE

Adding Nonconsecutive Elements
The preceding examples uses either square brackets or the $ symbol to add elements to the “first” position of an empty list. If we add an element to a later index, R interpolates a number of NULL elements to fill any gaps in the list:

```r
> emptyList <- list()                # Recreate the empty list
> emptyList                          # Empty list
  list()
> emptyList[[3]] <- "Hello"          # Assign to third element
> emptyList                           # Name the list
[[1]]
  NULL
[[2]]
  NULL
[[3]]
  [1] "Hello"
```

Combining Lists
You can grow lists by combining them together using the `c` function, as shown here:

```r
> list1 <- list(A = 1, B = 2)   # Create list1
> list2 <- list(C = 3, D = 4)   # Create list2
> c(list1, list2)               # Combine the lists
$A
[1] 1

$B
[1] 2

$C
[1] 3

$D
[1] 4
```

A Summary of List Syntax
As you have seen so far in this hour, the way we use lists varies slightly based on whether the elements of the list are named. At this point, it is worth reviewing the syntax to create and manage “unnamed” and “named” list structures.
Overview of Unnamed Lists

An overview of the key syntax covered is shown here, using a list without named elements as an example. First, let’s create a list and look at the list attributes:

```r
> unnamedList <- list(aVector, aMatrix) # Create the list
> unnamedList # Print the list
[[1]]
[1] 5 7 8 2 4 3 9 0 1 2

[[2]]
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

> length(unnamedList) # Number of elements
[1] 2

> names(unnamedList) # No element names
NULL
```

We can subset the list or extract list elements using single/double square brackets:

```r
> unnamedList[1] # Subset the list
[[1]]
[1] 5 7 8 2 4 3 9 0 1 2

> unnamedList[[1]] # Return the first element
[1] 5 7 8 2 4 3 9 0 1 2

> unnamedList[[3]] <- 1:5 # Add a new element

> unnamedList
[[1]]
[1] 5 7 8 2 4 3 9 0 1 2

[[2]]
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

[[3]]
[1] 1 2 3 4 5
```

Overview of Named Lists

Let’s look at a similar example using a list with element names. First, let’s create the list and view the list attributes:

```r
> namedList <- list(a = aVector, b = aMatrix)
> namedList
$a
 [1] 5 7 8 2 4 3 9 0 1 2

$b
[,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

> names(namedList)
[1] "a" "b"
```

```
> namedList$a # Access element by name
[1] 5 7 8 2 4 3 9 0 1 2

> namedList$c # Element does not exist
NULL
```
HOUR 4: Multi-Mode Data Structures

> namedList <- list(VRC = aVector, MAT = aMatrix)   # Create the list
> namedList                                         # Print the list
$VEC
[1] 5 7 8 2 4 3 9 0 1 2

$MAT
 [,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

> length(namedList)                     # Number of elements
[1] 2

> names(namedList)                      # Element names
[1] "VEC" "MAT"

We can subset the list using single square brackets, or reference elements directly with the $ symbol:

> namedList[1]                          # Subset the list
$VEC
[1] 5 7 8 2 4 3 9 0 1 2

> namedList$VEC                         # Return the first element
[1] 5 7 8 2 4 3 9 0 1 2

> namedList$NEW <- 1:5                  # Add a new element

> namedList
$VEC
[1] 5 7 8 2 4 3 9 0 1 2

$MAT
 [,1] [,2] [,3]
[1,] 1 3 5
[2,] 2 4 6

$NEW
[1] 1 2 3 4 5

Motivation for Lists

A good understanding of lists helps you to accomplish a number of useful tasks in R. To illustrate this, we will briefly look at two use cases that rely on list structures. Note that this section includes syntax that will be covered later in this book, but we include it here to illustrate “the art of the possible” at this stage.
**Flexible Simulation**

Consider a situation where we want to simulate a number of extreme values (for example, large financial losses by day, or particularly high values of some measure for each patient in a drug study). For each iteration, we may simulate any number of numeric values from a given distribution.

A list provides a flexible structure to hold all the simulated data. Consider the following code example:

```r
> nExtremes <- rpois(100, 3)                # Simulate number of extreme values by day from a Poisson distribution
> nExtremes[1:5]                            # First 5 numbers
[1] 0 3 5 7 3

> # Define function that simulates "N" extreme values
> exFun <- function(N) round(rweibull(N, shape = 5, scale = 1000))
> extremeValues <- lapply(nExtremes, exFun) # Apply the function to our simulated numbers

> extremeValues[1:5]                        # First 5 simulated outputs
[[1]
 numeric(0)

[[2]]
[1] 1305 948 1077

[[3]]
[1] 676 516 865 614 970

[[4]]
[1] 618 1217 818 1173 1205 1105 519

[[5]]
[1] 1026 933 657
```

From this example, note that the first simulated output generated no “extreme” values, resulting in the output containing an empty numeric vector (signified by `numeric(0)`). The “unnamed” list structure allows us to hold, in the same structure:

- This empty vector (indicating no “extreme values” for a particular day)
- Large vectors holding a number of simulated outputs (for days where many “extreme values” were simulated)

Given that we have stored this information in a list, we can query it to summarize the average number and average of extreme values:
TIP

The apply Functions

In the preceding examples, we used functions such as `lapply` (which applies a function to each element of a list) and `sapply` (which performs the same action but simplifies the outputs). We cover the apply family of functions later in Hour 9, “Loops and Summaries.”

Extracting Elements from Named Lists

In R, most objects are, fundamentally, lists. For example, let’s use the `t.test` function to perform a simple T-test. We will take the example straight from the `t.test` help file:

```r
> theTest <- t.test(1:10, y = c(7:20))  # Perform a T-Test
> theTest                                # Print the output
Welch Two Sample t-test
data:  1:10 and c(7:20)
t = -5.4349, df = 21.982, p-value = 1.855e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:
  -11.052802  -4.947198
sample estimates:
mean of x mean of y
  5.5      13.5
```

The output is printed as a nicely formatted text summary informing us of the significant T-test. But what if we wanted to use one of the elements of this output in further work (for example, the p-value). Consulting the help file, we see the return value is described as follows:

Value

A list with class `htest` containing the following components:

- `statistic` The value of the t-statistic.
- `parameter` The degrees of freedom for the t-statistic.
- `p.value` The p-value for the test.
- `conf.int` A confidence interval for the mean appropriate to the specified alternative hypothesis.
- **estimate** The estimated mean or difference in means, depending on whether it was a one-sample test or a two-sample test.

- **null.value** The specified hypothesized value of the mean or mean difference, depending on whether it was a one-sample test or a two-sample test.

- **alternative** A character string describing the alternative hypothesis.

- **method** A character string indicating what type of t-test was performed.

- **data.name** A character string giving the name(s) of the data.

The key thing to note here is that the return object is “a list.” Given that the output is a list, we can query the named elements of this list and see that the result matches the description of elements in the help file:

```r
> names(theTest)        # Names of list elements
[1] "statistic"  "parameter"  "p.value"  "conf.int"  "estimate"
[6] "null.value"  "alternative"  "method"  "data.name"
```

Given that this is a named list, and we know the names of the elements, we can use the $ symbol to directly reference the information we need:

```r
> theTest$p.value       # Reference the p-value
[1] 1.855282e-05
```

Using this approach, we can reference a wide range of elements from R outputs.

**NOTE**

**Print Methods**

In the preceding example, we created a complex object (fundamentally, a named list) that printed in a neat manner:

```r
> theTest               # Print the output

Welch Two Sample t-test

data:  1:10 and c(7:20)
t = -5.4349, df = 21.982, p-value = 1.855e-05
alternative hypothesis: true difference in means is not equal to 0
95 percent confidence interval:  
-11.052802  -4.947198
sample estimates: 
mean of x mean of y
5.5      13.5
```
The neat printout is generated by a print “method” associated with outputs from `t.test`. If we want to see the “raw” underlying structure, we can use the `print.default` function, which confirms that the structure is list based:

```r
> print.default(theTest)
$statistic
-5.43493

$parameter
df
21.98221

$p.value
[1] 1.855282e-05
...
```

## Data Frames

In the last section, we introduced the “list” structure, which allows you to store a set of objects of any mode. A data frame is, like many R objects, a named list. However, a data frame enforces a number of constraints on this named list structure. In particular, a data frame is constrained to be a named list that can only hold vectors of the same length.

### Creating a Data Frame

We create a data frame by specifying a set of named vectors to the `data.frame`. For example, let's create a data frame containing New York temperature forecasts over the next five days:

```r
> weather <- data.frame(                  # Create a data frame
+   Day   = c("Saturday", "Sunday", "Monday", "Tuesday", "Wednesday"),
+   Date  = c("Jul 4", "Jul 5", "Jul 6", "Jul 7", "Jul 8"),
+   TempF = c(75, 86, 83, 83, 87)
+ )
> weather                                 # Print the data frame
Day  Date TempF
1  Saturday Jul 4    75
2    Sunday Jul 5    86
3    Monday Jul 6    83
4   Tuesday Jul 7    83
5 Wednesday Jul 8    87
```
NOTE

Print Methods
As discussed earlier, the neat printing of this object is caused by a print “method” for data frames. We can see the raw structure using print.default, which again confirms that a data frame is fundamentally a named list of vectors:

```r
> print.default(weather)
$Day
 [1] Saturday Sunday Monday Tuesday Wednesday
Levels: Monday Saturday Sunday Tuesday Wednesday

$Date
Levels: Jul 4 Jul 5 Jul 6 Jul 7 Jul 8

$TempF
 [1] 75 86 83 83 87

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

CAUTION

Nonmatching Vector Lengths
If we try to create a data frame using vectors with nonmatching lengths, we get an error message:

```r
> data.frame(X = 1:5, Y = 1:2)
Error in data.frame(X = 1:5, Y = 1:2) :
  arguments imply differing number of rows: 5, 2
```

Querying Data Frame Attributes
Because a data frame is simply a named list, the functions we used to query list attributes will work the same way:

- The length function returns the number of elements of the list (that is, the number of columns).
- The names function returns the element (column) names.

The following example illustrates the use of these functions:

```r
> length(weather) # Number of columns
[1] 3
> names(weather)  # Column names
[1] "Day"  "Date"  "TempF"
```
Selecting Columns from the Data Frame

As with lists, we can reference a single element (vector) from our data frame using either double
squared brackets or the $ symbol:

```r
> weather # The whole data frame
   Day          Date TempF
1 Saturday Jul 4  75
2      Sunday Jul 5  86
3    Monday Jul 6  83
4  Tuesday Jul 7  83
5 Wednesday Jul 8  87

> weather[[3]] # The "third" column
[1] 75 86 83 83 87
> weather$TempF # The "TempF" column
[1] 75 86 83 83 87
```

Selecting Columns from the Data Frame

Because we can reference columns in this way, we can also use these approaches to add new
columns. For example, let’s add a new column called TempC to our data containing the tempera-
ture in degrees Celsius:

```r
> weather$TempC <- round( (weather$TempF - 32) * 5/9 )
> weather
   Day          Date TempF TempC
1 Saturday Jul 4  75   24
2      Sunday Jul 5  86   30
3    Monday Jul 6  83   28
4  Tuesday Jul 7  83   28
5 Wednesday Jul 8  87   31
```

Subscripting Columns

Because the columns of data frames are vectors, we can subscript them using the approaches
from Hour 3, “Single-Mode Data Structures.” Specifically, we can subscript the columns using
square brackets:

```r
DATA$COLUMN [ Input specifying the subset to return ]
```

As before, we can reference using blank, positive, negative, or logical inputs. Character inputs do
not make sense for referencing columns because the individual elements within columns are not
associated with element names.
Blank, Positive, and Negative Subscripts

If we use a blank subscript, all the values of the vector are returned:

```r
> weather
    Day    Date TempF TempC
 1  Saturday Jul 4 75   24
 2     Sunday Jul 5 86   30
 3     Monday Jul 6 83   28
 4    Tuesday Jul 7 83   28
 5  Wednesday Jul 8 87   31

> weather$TempF [ ]  # All values of TempF column
[1] 75 86 83 83 87
```

If we use a vector of positive integers, it refers to the elements of the column (vector) to return:

```r
> weather$TempF [ 1:3 ]  # First 3 values of the TempF column
[1] 75 86 83
```

If we use a vector of negative integers, it refers to the elements of the column (vector) to omit:

```r
> weather$TempF [ -(1:3) ]  # Omit the first 3 values of the TempF column
[1] 83 87
```

Logical Subscripts

As you saw in the last hour, we can provide a vector of logical values to reference a vector, and only the corresponding TRUE values are returned. Here’s an example:

```r
> weather$TempF
[1] 75 86 83 83 87
> weather$TempF [ c(F, T, F, F, T) ]  # Logical subscript
[1] 86 87
```

Of course, we usually generate the logical vector with a logical statement involving a vector. For example, we could return all the TempF values greater than 85 using this statement:

```r
> weather$TempF [ weather$TempF > 85 ]  # Logical subscript
[1] 86 87
```

Instead, we could reference a column of a data frame based on logical statements involving one or more other columns (because all columns are constrained to be the same length):

```r
> weather$Day [ weather$TempF > 85 ]  # Logical subscript
[1] Sunday  Wednesday
Levels: Monday Saturday Sunday Tuesday Wednesday
```
NOTE

Factor Levels
In the last example, you can see that the days where the forecast is greater than 85°F are Sunday and Wednesday. However, you should note two things about the output:

- There are no quotation marks around the returned values (Sunday and Wednesday).
- Additional “Levels” information has been printed.

This strange output is produced because, when you create a data frame using character columns, those columns are converted to “factors,” which are “category” columns that are automatically derived from character vectors when used in a data frame. You’ll see more on factors later in Hour 5, “Dates, Times, and Factors.”

Referencing as a Matrix
Although a data frame is structured as a named list, its rectangular output is more similar to the matrix structure you saw earlier. As such, R allows us to reference the data frame as if it was a matrix.

Matrix Dimensions
Because we can treat a data frame as a matrix, we can use the `nrow` and `ncol` functions to return the number of rows and columns:

```r
> nrow(weather)  # Number of rows
[1] 5
> ncol(weather)  # Number of columns
[1] 4
```

Subscripting as a Matrix
In Hour 3, you saw that you can subscript a matrix using square brackets and two inputs (one for the rows, one for the columns). We can use the same approach to subscript a data frame, where each input can be one of the standard five input types:

```r
DATA.FRAME [ Rows to return , Columns to return]
```

Blanks, Positives, and Negatives
We can use blank subscripts to return all rows and columns from a data frame:

```r
> weather[, ]       # Blank, Blank
Day  Date  TempF  TempC
1  Saturday Jul 4 75   24
2    Sunday Jul 5 86   30
3   Monday Jul 6 83   28
4  Tuesday Jul 7 83   28
5 Wednesday Jul 8 87   31
```
If we use vectors of positive integers, they are used to provide an index of the rows/columns to return. This example uses positive integers to return the first four rows and the first three columns:

```r
> weather[ 1:4, 1:3 ] # +ive, +ive
  Day  Date TempF
1 Saturday Jul 4  75
2   Sunday Jul 5  86
3  Monday Jul 6  83
4  Tuesday Jul 7  83
```

We can use vectors of negative integers to indicate the rows and columns to omit in the return result, as shown in this example:

```r
> weather[ -1, -3 ] # -ive, -ive
  Day  Date TempC
2   Sunday Jul 5  30
3    Monday Jul 6  28
4   Tuesday Jul 7  28
5 Wednesday Jul 8  31
```

In the preceding examples, we have used the same input type for both rows and columns. However, we can mix up the input types, as illustrated in this example, where we select the first four rows and all the columns:

```r
> weather[ 1:4, ] # +ive, Blank
  Day  Date TempF TempC
1 Saturday Jul 4  75    24
2   Sunday Jul 5  86    30
3  Monday Jul 6  83    28
4  Tuesday Jul 7  83    28
```

**Logical Subscripts**

We often use logical subscripts to reference specific rows of the data to return. To perform this action, we need to provide a logical value for each row of the data:

```r
> weather                       # The original data
  Day  Date TempF TempC
1  Saturday Jul 4  75    24
2    Sunday Jul 5  86    30
3    Monday Jul 6  83    28
4   Tuesday Jul 7  83    28
5 Wednesday Jul 8  87    31

> weather[ c(F, T, F, F, T), ] # Logical, Blank
  Day  Date TempF TempC
2    Sunday Jul 5  86    30
5 Wednesday Jul 8  87    31
```
As before, we more commonly apply a logical statement to a column (vector) contained in the data frame to generate the logical vector:

```r
> weather[ weather$TempF > 85, ]  # Logical, Blank
Day  Date TempF TempC
  2 Sunday Jul 5  86  30
  5 Wednesday Jul 8  87  31

> weather[ weather$Day != "Sunday", ]  # Logical, Blank
Day  Date TempF TempC
  1 Saturday Jul 4  75  24
  3 Monday Jul 6  83  28
  4 Tuesday Jul 7  83  28
  5 Wednesday Jul 8  87  31
```

**Character Subscripts**

We often use vectors of character strings to specify the columns we wish to return. Although a data frame has “row names,” we tend not to reference rows using character strings. This example selects the `Day` and `TempC` columns from the data, filtering so that only rows with temperatures greater than 85°F are returned:

```r
> weather[ weather$TempF > 85, c("Day", "TempC") ]  # Logical, Character
Day TempC
  2 Sunday  30
  5 Wednesday  31
```

**Summary of Subscripting Data Frames**

At this point, it is worth a quick review of some of the key syntax used to select subsets of a data frame. In particular, consider the following lines of code:

```r
> weather$Day[ weather$TempF > 85 ]  # Days where TempF > 85
[1] Sunday Wednesday
Levels: Monday Saturday Sunday Tuesday Wednesday

> weather[ weather$TempF > 85 , ]  # All data where TempF > 85
Day  Date TempF TempC
  2 Sunday Jul 5  86  30
  5 Wednesday Jul 8  87  31

> weather[ weather$TempF > 85 , c("Day", "TempF") ]  # 2 columns where TempF > 85
Day TempF
  2 Sunday  86
  5 Wednesday  87
```
In the first example, we are subscripting `weather$Day`. This is a vector, so we provide a single input (a logical vector in this case). It returns the two values of the `Day` column where the corresponding `TempF` column is greater than 85.

In the second example, we are now referencing data from the whole weather dataset. As such, we need two subscripts (one for rows, one for columns). In this example, we use a logical vector for the rows and blank for the columns, returning all columns but only rows where `TempF` is greater than 85. Attention should be paid to the use of the comma in the first example versus the second example, driven by the fact that we are referencing data from a vector (first example) versus the whole data frame (second example).

The third example extends the second example to pick only columns `Day` and `TempF` using a character vector for the column input.

**Exploring Your Data**

Later in this book, you'll see a range of functionality for manipulating data frames. For now, it is useful for you to look at a few simple functions that will help you to quickly understand the data stored in a data frame.

**The Top and Bottom of Your Data**

A function called `head` allows you to return the first few rows of the data. This is particularly useful when you have a large data frame and only want to get a high-level understanding of the structure of the data frame. The `head` function accepts any data frame and will return (by default) only the first six rows. For this example, we use the built-in `iris` data frame (for more information, open the help file for the `iris` data frame using the `?iris` command):

```r
> nrow(iris) # Number of rows in iris
[1] 150
> head(iris) # Return only the first 6 rows
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1     5.1       3.5       1.4       0.2      setosa
2     4.9       3.0       1.4       0.2      setosa
3     4.7       3.2       1.3       0.2      setosa
4     4.6       3.1       1.5       0.2      setosa
5     5.0       3.6       1.4       0.2      setosa
6     5.4       3.9       1.7       0.4      setosa
```

This immediately gives us a view on the structure of the data. We can see that the `iris` data frame has five columns: `Sepal.Length`, `Sepal.Width`, `Petal.Length`, `Petal.Width`, and `Species`. All columns seem to be numeric, except the `Species` column, which appears to be character (or a “factor,” as briefly discussed earlier).
The second argument to the `head` function is the number of rows to return. Therefore, we could look at more or fewer rows if we wish:

```r
> head(iris, 2)  # Return only the first 3 rows
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
  1          5.1         3.5          1.4         0.2      setosa
  2          4.9         3.0          1.4         0.2      setosa
```

If instead we wanted to look at the last few rows, we could use the `tail` function. This works in the same way as the `head` function, with the data frame as the first input and (optionally) the number of rows to return as the second input:

```r
> tail(iris)      # Return only the last 6 rows
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
  145          6.7         3.3          5.7         2.5   virginica
  146          6.7         3.0          5.2         2.3   virginica
  147          6.3         2.5          5.0         1.9   virginica
  148          6.5         3.0          5.2         2.0   virginica
  149          6.2         3.4          5.4         2.3   virginica
  150          5.9         3.0          5.1         1.8   virginica
> tail(iris, 2)   # Return only the last 2 rows
  Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
  149          6.2         3.4          5.4         2.3   virginica
  150          5.9         3.0          5.1         1.8   virginica
```

**Viewing Your Data**

If you are using the RStudio interface, you can use the `View` function to open the data in a viewing grid. This feature in RStudio is evolving quickly, so readers of this book may find the functionality richer than that presented here (the version of RStudio being used is 0.99.441). See Figure 4.1 for an example.

If we use the `View` function, our data frame is opened in the data grid viewer in RStudio:

```r
> View(iris)     # Open the iris data in the data grid viewer
```

This window allows us to scroll around our data, and tells us the range of data we are viewing (for example, in Figure 4.1 the message at the bottom of the viewer tells us that we are looking at rows “1 to 19 of 150”).

The search bar (top right of the window) allows us to input search criteria that will be used to search the entire dataset. This is used to interactively filter the data based on a partial matching of the search term. As a quick example, look at the result of typing 4.5 in the search bar, as shown in Figure 4.2.
Exploring Your Data

FIGURE 4.1
The iris dataset viewed in the RStudio data grid viewer

```r
> ?iris
> View(iris)
```

FIGURE 4.2
Using the search bar in the data grid viewer
If we click the Filter icon from the top of the data grid viewer window, we will see a number of filtering fields appear, which we can use to interactively subset the data in a more data-driven manner. This example uses the filter feature to look only at rows for the “setosa” species with Sepal.Length greater than 5.5 (see Figure 4.3).

![Filtering data in the data grid viewer](image)

**FIGURE 4.3**
Filtering data in the data grid viewer

### Summarizing Your Data

We can use the `summary` function to produce a range of statistical summary outputs to summarize our data. The `summary` function accepts a data frame and produces a textual summary of each column of the data:

```r
> summary(iris)  # Produce a textual summary
Sepal.Length  Sepal.Width  Petal.Length  Petal.Width  Species
Min.   :4.300   Min.   :2.000   Min.   :1.000   Min.   :0.100   setosa :50
1st Qu.:5.100   1st Qu.:2.800   1st Qu.:1.600   1st Qu.:0.300   versicolor:50
Median :5.800    Median :3.000    Median :4.350    Median :1.300    virginica:50
Mean   :5.843    Mean   :3.057    Mean   :3.758    Mean   :1.199
3rd Qu.:6.400   3rd Qu.:3.300   3rd Qu.:5.100   3rd Qu.:1.800
```

Note that the summaries produced are suitable for each column type (statistical summary for numeric columns, frequency count for factor columns).
Visualizing Your Data

In this book, you will see a number of functions for creating sophisticated graphical outputs. However, let’s look at one simple function that creates an immediate visualization of the structure of our data.

We can create a scatter-plot matrix plot of our data frame using the `pairs` function as follows:

```r
> pairs(iris)  # Scatter-plot matrix of iris
```

In the graphic shown in Figure 4.4, each variable in the data is plotted against each other. For example, the plot in the top-right corner is a plot of `Sepal.Length` (y axis) against `Species` (x axis).

![Figure 4.4](attachment:image.png)

**FIGURE 4.4**
Scatter-plot Matrix of the iris data frame
From this plot we can quickly identify a number of characteristics of our data:

▶ We see that the data has five columns, whose names are printed on the diagonal of the plot.
▶ We can again see that Species is a factor column, whereas the rest are numeric.
▶ If we look at the plots on the right side of the chart, we can see each numeric variable plotted against Species and note that the numeric data would seem to vary across each level of Species.
▶ Columns Petal.Length and Petal.Width would seem to be highly correlated.

**Summary**

In this hour, we focused on two structures that store “multi-mode” data (that is, data containing more than one data type). First, we looked at lists, which allow us to store any number of objects of varying modes. Then, we looked at data frames as a special “type” of list that stores rectangular datasets in an effective manner.

Although lists are very powerful structures, when we import data into R (which you’ll see in Hour 10, “Importing and Exporting”), it will be stored as a data frame. Therefore, you need to be very comfortable manipulating this structure in particular. You should practice the syntax relating the subscripting of data frames using square brackets and the $ symbol, because this is a fundamental skill useful across all R tasks.

**Q&A**

**Q.** Can we create nested lists?

**A.** Yes. Because lists can store any type of object, they can themselves store other lists. Here’s an example:

```r
> nestedList <- list(A = 1, B = list(C = 3, D = 4))  # Create a nested list
> nestedList                                          # Print the nested list
$A
[1] 1

$B
$B$C
[1] 3

$B$D
[1] 4
```
Q. What other inputs can we use within the double square brackets?

A. In the last hour, you saw that you can use integers to directly reference elements of a list. Refer to the help file (opened using `?[[`) for a complete list of possible inputs. However, it is worth noting that you can use single-character strings to reference columns. Here's an example:

```r
> nestedList$B$C                 # Extract the C element within the B element
[1] 3
```

Q. What is the difference between `DF[ ]` and `DF[ , ]`?

A. As shown previously, you subscript data from a data frame using square brackets. Here's an example:

```r
> weather 
# The full dataset
Day Date TempF TempC
1 Saturday Jul 4 75 24
2 Sunday Jul 5 86 30
3 Monday Jul 6 83 28
4 Tuesday Jul 7 83 28
5 Wednesday Jul 8 87 31
> col <- "TempC"  # The column we want to select
> weather[[col]]  # Return the TempC column
[1] 24 30 28 28 31
```

In this example, we provide two subscripts for the data frame: blank for the rows (so all rows are returned) and a character vector to select two columns. The subscripts are separated by a comma. If we omit the comma, we appear to get the same result:

```r
> weather[[c("Day", "TempC")]]  # 2 vector elements
Day TempC
1 Saturday 24
2 Sunday 30
3 Monday 28
4 Tuesday 28
5 Wednesday 31
```

In this example, we provide two subscripts for the data frame: blank for the rows (so all rows are returned) and a character vector to select two columns. The subscripts are separated by a comma. If we omit the comma, we appear to get the same result:

```r
> weather[, c("Day", "TempC") ]   # All rows, 2 columns
Day TempC
1 Saturday 24
2 Sunday 30
3 Monday 28
4 Tuesday 28
5 Wednesday 31
```

In this example, we provide two subscripts for the data frame: blank for the rows (so all rows are returned) and a character vector to select two columns. The subscripts are separated by a comma. If we omit the comma, we appear to get the same result:

```r
> weather[, c("Day", "TempC") ]   # All rows, 2 columns
Day TempC
1 Saturday 24
2 Sunday 30
3 Monday 28
4 Tuesday 28
5 Wednesday 31
```

In this example, we provide two subscripts for the data frame: blank for the rows (so all rows are returned) and a character vector to select two columns. The subscripts are separated by a comma. If we omit the comma, we appear to get the same result:

```r
> weather[, c("Day", "TempC") ]   # All rows, 2 columns
Day TempC
1 Saturday 24
2 Sunday 30
3 Monday 28
4 Tuesday 28
5 Wednesday 31
```

Here, we are using the fact that a data frame is actually a named list of vectors. In this case, we are creating a "sub-list" containing only the two columns specified.
Q. Why, when I select a single column, is it returned as a vector?

A. When you select a single column via the square brackets approach, it is indeed returned as a vector:

```r
> weather [ , c("Day", "TempC") ]   # 2 columns - returns a data frame
  Day TempC
 1 Saturday 24
 2 Sunday 30
 3 Monday 28
 4 Tuesday 28
 5 Wednesday 31
> weather [ , "TempC" ]             # 1 column - returns a vector
[1] 24 30 28 28 31
```

In this case, the last line is equivalent to `weather$TempC`. When you select a single column of data, R simplifies the output in a way that’s similar to how you saw matrix dimensions dropped in Hour 3. If you specifically want to retain the dimensional structure, you can use the argument `drop` within the square brackets, as follows:

```r
> weather [ , "TempC", drop = F ]   # 1 column - retain dimensions
    TempC
 1 24
 2 30
 3 28
 4 28
 5 31
```

As you can see from the output, the use of `drop = F` retains the structure, returning a 5x1 data frame.

**Workshop**

The workshop contains quiz questions and exercises to help you solidify your understanding of the material covered. Try to answer all questions before looking at the “Answers” section that follows.

**Quiz**

1. What is a “list” object?
2. How do we reference elements from a list?
3. What is the “mode” of a list?
4. What’s the difference between a list and a data frame?
5. Name two ways we can return the number of columns of a data frame.
6. If we run the following code, what would the contents and structure of `result1` and `result2` contain?

   ```r
   > myDf <- data.frame(X = -2:2, Y = 1:5)
   > result1 <- myDf$Y [ myDf$X > 0 ]
   > result2 <- myDf [ myDf$X > 0, ]
   ```

7. What is the difference between the `head` and `tail` functions?

**Answers**

1. A “list” is a simple R object that can contain any number of objects of any “class.”

2. We can reference elements of a list using the “double square brackets” notation. Most commonly, we provide the index of the element we want to return from the list (for example, `myList[[2]]` for the second element). If a list has element names, we can alternatively use the dollar notation, specifying the name of the list element (for example, `myList$X` to return the `X` element of `myList`).

3. Because a list is a “multi-mode” object, it has no explicit “mode.” If you ask for a list’s mode, it simply returns “list.”

4. A list can contain any number of objects of any class—its elements may be named or unnamed. A data frame is a “named” list that is restricted to contain only same-length vectors—when printing a data frame, it uses a specific method so the data is presented in a more formatted manner.

5. We can use the `length` function to return the number of columns in a data frame, because this returns the number of vector elements in the underlying “list” structure. Alternatively, because we can treat a data frame as a matrix, we can use the `ncol` function to achieve the same result.

6. The `result1` object will contain a vector of those values from the `Y` column where the corresponding `X` column is greater than 0—specifically, this will be a vector containing values 4 and 5. The `result2` object will contain a data frame with two rows, corresponding to the rows where `X` is greater than 0 (so rows 4 and 5 of the original data frame).

7. The `head` function returns the first six rows (by default) of a data frame. The `tail` function returns the last six rows (by default) of a data frame.

**Activities**

1. Create a “named” list containing a numeric vector with 10 values (called `X`) and a character vector with 10 values (called `Y`) and a sequence of values from 1 to 10 (called `Z`). Use this list:

   - Print the number of elements and the element names.
   - Select the `X` element.
   - Select the `Y` element.
Select values of the X element that are greater than the median of X.
Select values of the Y element where the corresponding X element is greater than the median of X.

2. Adapt your code to instead create a data frame containing two columns (X = a numeric vector with 10 elements, Y = a character column containing 10 elements, Z = integers 1 to 10). Use this structure:
- Print the number of columns and the column names.
- Select the X column.
- Select the Y column.
- Select values of the X column that are greater than the median of X.
- Select values of the Y column where the corresponding X value is greater than the median of X.

3. Further subset the data in the data frame created in the last exercise as follows:
- Select all rows of the data where Z is greater than 5.
- Select all rows of the data where Z is greater than 3 and X is greater than the median of X.
- Select just the X and Z columns from the data where Z is greater than 5.

4. Print the built-in mtcars data frame. Look at the help file for mtcars to understand the origin of the data. Use this data frame:
- Print only the first five rows.
- Print the last five rows.
- How many rows and columns does the data have?
- Look at the data in the RStudio data viewer (if you are using RStudio).
- Print the mpg column of the data.
- Print the mpg column of the data where the corresponding cyl column is 6.
- Print all rows of the data where cyl is 6.
- Print all rows of the data where mpg is greater than 25, but only for the mpg and cyl columns.
- Create a scatter-plot matrix of your data.
- Create a scatter-plot matrix of your data, but only using the first six columns of the data.
Symbols & Numerics

%>% (pipe operator), 271-273
: (colon), interaction terms, 396
... (ellipsis), 157-159
= (equal sign), 19
$ (dollar sign)
  referencing list elements, 77-79
  shortened $ referencing, 78-79
\\ (double backslash), 21
\ (forward slash), 21
& operator, 144-145
[] (square brackets), 43
  double square bracket referencing, 76-77
~ (tilde), formula relationships, 381
3D lattice graphics, 352-354

A

abline function, 389-390
acf function, 448
active bindings, 544
adding
  columns, 266, 277-278
  list elements, 79-80
  rows, 278-279
aggregate function, 252
  specifying variables, 254-256
  using with a formula, 252-254
  multiple return values, 253-254
  summarizing by multiple variables, 252-253
  summarizing multiple columns, 253
aggregating data, 246
  data.table package, 280-282
dplyr package, 268-271
  grouped data, 269-270
analysis of variance, comparing nested models, 395
anova function, 395
appending, 237-238. See also combining
applications, Shiny
  server component, 564-566
  sharing, 570
  structure, 561-562
  ui component, 562-564
apply functions, 181-195, 250-251
  applying to data frames, 193-195
  example, 184-186
  lapply, 195-204
    order of “apply” inputs, 201-203
    using with vectors, 199-201
  margin values, 183-184
  multiple margins, 186-187
  passing extra arguments to “applied” function, 188-191
sapply function, 204-208
  returns, 205-207
tapply
  multiple grouping variables, 209-210
  multiple returns, 210-212
  return values, 212
  using with higher dimension structures, 187-188
xapply, 182
arguments, 116
  apply function, 183
  breaks, 111
  defining, 132-133
  ellipsis, passing graphical parameters, 159-161
  for merge function, 238
  named arguments, 131
arima function, 449
ARIMA models, in time series analysis, 448-451
arrange function, 263
arrays, 34, 58-60
  creating, 58-60
  subscripting, 60
as.numeric function, 121
assessing models, 382
  abline function, 389-390
  extractor functions, 385-386
  interaction terms, 396-398
  as list objects, 386-388
  plot function, 383-385
  predict function, 390-391
  summary function, 382-383
assignment arrows, 19-20
attributes
  of data frames, querying, 87
  of lists, 72-73
  of matrices, 52-54
  of single mode data structures, comparing, 60-62
  of vectors, 41-43
autocorrelations, in time series analysis, 448
axis limits, setting for plots, 294-295

B
bar charts, 291
Becker, Rick, 345
benchmarking, 457-458
bigmemory package, 282
binaries, installing packages from, 26
bivariate lattice graphics, 350-351
blank inputs, 44-45, 74
boxplots, 290
breaks argument, 111
bugs, reporting, 8
building packages, 471-472, 482-485

C
built-in functions, 115
character vectors, 31-32
character strings, 31-32
c function, creating vectors, 35-36
C++
  incorporating code in R, 501-502
  integrating with, 464-468
  using R functions in, 467-468
capturing input definitions, 164-167
case sensitivity for file paths, 219
cast function, 245-246
categorical data, 108-112
cbind function, 49
censoring in survival analysis, 431-432
constructor functions
581

Chambers, John, 2, 535
character data
manipulating, 123-124
searching and replacing, 124-125
character value inputs, 48, 57-58, 76
checking
function inputs, 136, 155-157
multivalue inputs, 162-164
packages, 482-484
classes, 505-509
creating, constructor functions, 510-511
example of, 507-508
extending, 518
generics, 511-516
creating, 515-516
naming conventions, 512
methods
defining for arithmetic operators, 513-514
updating, 513
object orientation, 506-508
inheritance, 508
R6, 542-544
active bindings, 544
example of, 543-544
private members, 542
public members, 542
Reference Classes, 535-542
creating, 535-537
documenting, 542
methods, defining, 537-540
objects, copying, 540-542
removing, 510
S3, 509
creating, 509-511
documenting, 518
inheritance, 516-518
limitations of, 518-519
lists versus attributes, 514-515
naming conventions, 512
S4, 523-535
defining, 525-529
documenting, 534-535
inheritance, 532-534
methods, 529-530
multiple dispatch, 531-532
summary function and, 405
writing, 505
Cleveland, William, 345
clipboard, 219
closing graphics devices, 288
code
C++, incorporating in R, 501-502
improving efficiency
benchmarking, 457-458
initialization, 458-459
integrating with C++, 464-468
with memory
management, 463-464
using alternative functions, 462-463
vectorization, 459-462
including in documents
LaTex documents, 556
RMarkdown documents, 550-552
profiling, 456
quality of, 476-477
coef function, 385-386
coefficients from logistic regression, 419-420
colon (:), interaction terms, 396
color function, 288
colors, specifying, 288
column index, 55
columns
adding, 266, 277-278
referencing, 179-180
selecting, 264-266
selecting from data frames, 88
subscripting, 88-90
combining
data.tables, 279-280
lists, 80
plot types, 318-321
vectors, 49-51
comment blocks, 15
comparing
attributes and lists, 514-515
nested models, 393-395
R and C++, 465-466
reshape and reshape2 packages, 245
single mode data structures, 60-62
conferences, 6
confint function, 420
connecting
to Excel from R, 228
to R from Excel, 226
constructor functions, 510-511
continuation prompts, 15
continuous variables, creating factors, 111-112
contrast methods, 400
controlling
  aesthetics in ggplot2 package, 322-324
  layout, 305-308
    grid layouts, 306-307
    layout function, 307-308
  strip headers, 363-364
  styles for lattice graphics, 372-376
converting objects, 156-157
coordinate systems, 338-339
copying Reference Class objects, 540-542
core packages, 23
counting records, 281
covariates, in survival analysis, 436
coxph function, 438-439
CRAN, 7
  METACRAN website, 24
  navigating to, 573
  packages
    finding, 23-24
    installing, 25-26
create function, 472-474
creating
  arrays, 58-60
  classes
    constructor functions, 510-511
    S3, 509-511
  data frames, 86-87
data.tables, 273-274
date objects, 103-104
  factors, 108-110
    from continuous data, 111-112
  functions, 130-136, 151-155
    error messages, 152-153
    warnings, 153-155
  generics, 515-516
  lattice graphs, 346-355
  lists, 71-72
    with element names, 71
    empty lists, 69
    non-empty lists, 70
  matrices, 49-52
    with a single vector, 51-52
  package structure, 472-474
  reactive functions, 567-568
  Reference Classes, 535-537
  sequence of integers, 37-38
  sequence of numeric values, 38-39
  sequence of repeated values, 39-41
  tbl_df objects, 262-263
  themes for lattice graphics, 374-376
  time objects, 104-105
  vectors, 35-41
    with c function, 35-36
CSV files, reading, 220
custom functions
  applying over dimensions, 191-192
  passing extra arguments, 192-193
custom plots, 333-339
  aes function, 333-336
  coordinate systems, 338-339
  ggplot function, 333
  multiple data frames, 336-338
cut function, 111
D
data, including in packages, 494-496
data aggregation
  aggregate function, 252
  apply functions, 250-251
  calculating differences from baseline, 257-258
  “for” loops, 250
data argument (lm function), 381
data frames, 86-93
  apply functions, 193-195
  attributes, querying, 87
  columns, selecting, 88
  creating, 86-87
  factors, creating, 108-110
  graphing, 97-98
  lapply function, 203-204
  referencing as a matrix, 90-92
  returning top and bottom of data, 93-94
  sorting, 236-237
  splitting, 197-199
  subscripting, 92-93
summarizing, 96
viewing, 94-96
working with multiple, 336-338
“data” lattice graphics, 354-355
data munging, 235
data types, 33-34
factors, 108-112
manipulating levels, 110-111
numeric factors, 109
reordering, 110
DataCamp, 5
data.table package, 273-282
aggregation, 280-282
columns
adding, 277-278
renaming, 277-278
rows, adding, 278-279
setting a key, 274-275
subscripting, 275-276
data.tables
counting records, 281
creating, 273-274
merging, 279-280
date objects, creating, 103-104
dates
lubridate package, 107-108
manipulating, 105-106
DBI (database interface), 225-226
decomposition, in time series analysis, 443-445
defining
function arguments, 132-133
keys, 274-275
methods
for arithmetic operators, 513-514
for Reference Classes, 537-540
S4 classes, 525-529
S4 generics, 530-531
time zones, 105
deleting packages, 24
deparse function, 166
dependencies, 27
descending sorts, 237
DESCRIPTION file, 474-475
developing a test framework, 490-494
incorporating tests into packages, 493-494
test_that function, 490-493
devtools, building packages, 482-485
diagnostic plots, 383-385
comparing, 387-394
in GLM framework, 416
for time series analysis, 449-450
diff function, 106
diffftime function, 106
dimensions
dropping, 56
functions, applying, 191-192
dimnames function, 53-54
distribution types, GLM framework, 412
distributions
hist function, 160-162
statistical distributions, 119-120
documentation. See also
dynamic reporting; reporting
interactive documents, 569-570
package documentation, generating, 477-482
function headers, 478-480
help pages, 480-482
R Documentation, 5
R manuals, 4-5
Reference Classes, 542
S3 class system, 518
S4 class system, 534-535
vignettes
including in packages, 496-498
markdown notation, 499
writing, 498-501
double square bracket
referencing, 76-77
dplyr package, 261-273
aggregation, 268-271
grouped data, 269-270
merge function, 267-268
mutate function, 266
pipe operator, 271-273
sorting, 263
subscripting, 264-266
with filter function, 264
with select function, 264-265
tbl_df objects, creating, 262-263
dropping dimensions, 56

duplicate function, 241-242
dynamic reporting, 547-548

LaTeX, 553-556
RMarkdown, 548-552
code chunks, including, 550-552

HTML files, building, 550

dynamic typing, 19

E

EARL (Effective Applications of the R Language) conference, 6

Eclipse, 13

efficiency of code, improving
benchmarking, 457-458
initialization, 458-459
integrating with C++, 464-468
with memory management, 463-464
profiling, 456
using alternative functions, 462-463
vectorization, 459-462

elements
extracting from named lists, 84
list elements
adding, 79-80
referencing, 76-79

ellipsis, 157-159
passing graphical parameters, 159-161
empty lists, creating, 69

errors
bugs, reporting, 8
returning, 152-153

escape sequences, 21
estimating survival function in survival analysis, 432-436
example
of apply function, 184-186
of classes, 507-508
of merge function, 239
of R6 class system, 543-544

Excel
connecting to R, 226
reading structured data, 226-227
XLConnect package, 228-231

exporting text files, 220

extending
classes, 518
packages, 489-490

extensions
to GLM framework, 422-423
to nonlinear models, 430
to survival analysis, 441
to time series analysis, 452

extracting elements from named lists, 84
extractor functions, 385-386

factor variables
in linear models, 398-401
in logistic regression, 419

factors, 108-112
creating, 108-110
from continuous data, 111-112
manipulating levels, 110-111
numeric factors, 109
reordering, 110

ff package, 282
file.choose function, 217

filter function, 264
finding
duplicate values, 241-242
packages, 23-24

fitted function, 385-386
flow control, if/else statements, 136-146

& and | operators, 144-145
example, 145-146
mixing conditions, 143
multiple test values, 139-140
nested statements, 138-139
returning early, 145
reversing logical values, 142-143
summarizing to a single logical, 140-141
switching with logical input, 141-142
using one condition, 139

for function, 174-176

loop variable, 175-176

facet_grid function, 329-331
facet_wrap function, 331-332
functions

- "for" loops, 174, 250
- foreign package, 222
- formulas, using with aggregate function, 252-254
  - multiple return values, 253-254
  - summarizing by multiple variables, 252-253
  - summarizing multiple columns, 253
- fread function, 221
- function keyword, 130-131
- functions
  - abline, 389-390
  - acf, 448
  - aes, 333-336
  - aggregate, 252-254
  - aggregate function, specifying variables, 254-256
  - anova, 395
  - apply, 181-195, 250-251
    - applying to data frames, 193-195
    - example, 184-186
    - margin values, 183-184
    - multiple margins, 186-187
    - passing extra arguments to "applied" function, 188-191
    - using with higher dimension structures, 187-188
  - arguments, 116
    - defining, 132-133
    - named arguments, 131
- arima, 449
- arrange, 263
- as.numeric, 121
- c, creating vectors, 35-36
- calling, 116
  - shortened argument calling, 162-161
- cast, 245-246
- cbind, 49
- coef, 385-386
- color, 288
- confint, 420
- constructor functions, 510-511
- coxph, 438-439
- create, 472-474
- creating, 130-136, 151-155
- cut, 111
- deparse, 166
- diff, 106
- difftime, 106
- dimnames, 53-54
- distribution functions, 119-120
- duplicated, 241-242
- error handling, 462
  - error messages, creating, 152-153
- extractor functions, 385-386
- facet_grid, 329-331
- facet_wrap, 331-332
- file.choose, 217
- filter, 264
- fitted, 385-386
- for, 174-176
  - loop variable, 175-176
- glm, 413
- logistic regression, 418-419
- methods for, 415-416
- Poisson regression, 420-422
- grep, 124-125
- group_by, 269-271
- gsub, 124-125
- head, 93-94
- help, 28-29
- hist, 160-162
- HoltWinters, 446-447
- l, 404
- ifelse, 461
- if/else structure, 136-146
  - example, 145-146
  - mixing conditions, 143
  - multiple test values, 139-140
  - nested statements, 138-139
  - returning early, 145
  - reversing logical values, 142-143
  - summarizing to a single logical, 140-141
  - switching with logical input, 141-142
  - using one condition, 139
- get, 164
- gc, 464
- grep, 124-125
- group_by, 269-271
- gsub, 124-125
- head, 93-94
- help, 28-29
- hist, 160-162
- HoltWinters, 446-447
- l, 404
- ifelse, 461
- if/else structure, 136-146
  - example, 145-146
  - mixing conditions, 143
  - multiple test values, 139-140
  - nested statements, 138-139
  - returning early, 145
  - reversing logical values, 142-143
  - summarizing to a single logical, 140-141
  - switching with logical input, 141-142
  - using one condition, 139
functions

inputs
  capturing, 164-167
  checking, 136, 155-157, 162-164
  ellipsis, 157-159
is.x, 122
lapply, 195-204
  order of “apply” inputs, 201-203
  using with data frames, 203-204
  using with vectors, 199-201
layout, 307-308
legend, 302-304
length, 41-42, 53
library, 27
lines, 299-300
  in nonlinear models, 428
lm, 380-381
  methods for, 406-407
logRange, 155
ls.str, 18-19
mathematical functions, 117-118
matrix, 51-52
melt, 243-245
merge, 238-241, 267-268
  inner joins, 240
  outer joins, 240-241
missing data functions, 122-123
mode, 34
mutate, 266
names, 42-43, 386-388
naming, 132
nchar, 123
ncol, 53
nested calls, 41
nls, 423-425
nrow, 53
objects, 18
odbcConnectAccess, 224
order, 236-237
output, saving, 131
pacf, 448
panel functions, 365-371
par, 304-305
paste, 124, 157-158
plot, 291-299, 383-385
  in GLM framework, 416
  parameters, setting, 304-305
  in proportional hazards regression, 439-441
  in survival analysis, 434
  in time series analysis, 442-443
plyr, 213
points, 299-300
predict, 390-391
  in ARIMA models, 450-451
  in logistic regression, 419
  in nonlinear models, 428
  in survival analysis, 435
  in time series analysis, 447
qplot, 314-315
  layers, 316
rbind, 50, 237-238
reactive, 566-568
read.table, 218
remove.packages, 24
rep, 39-41
replace, 122
resid, 385-386
return objects, 134-136
Rprof, 456
runif, 157
sapply, 204-208
  returns, 205-207
save, 22
scoping rules, 133-134
searchpaths, 17-18
select, 264-265
self-starting, 427
separate, 249
seq, 38-39
split, 195-197
spread, 248
sqlcolumns, 224
statistical summary functions, 118-119
stl, 443-445
stop, 152
structure, 129-130
substitute, 166
substring, 123
summary, 96, 382-383, 405
  classes and methods, 405
  in GLM framework, 415-416
  with names function, 388
  in survival analysis, 433-434
survfit, 433-434
  in proportional hazards regression, 439-441
switch, 159

Table function, 121
tail, 94
tapply, 208-213
  multiple grouping variables, 209-210
  multiple returns, 210-212
  return values, 212

test_that, 490-493
text, 300-302
ts, 441-443
tsdia, 449-450
update, 392-393
UseMethod, 512
warning, 153
warnings, 153-155
while, 180-181
window, 443
xapply, 182

G

garbage collection, 464
gather function, 247-248
Gaussian model fitting, 414
gc function, 464
generating
  classes with constructor function, 510-511
documentation
    with LaTeX, 553-556
    with RMarkdown, 548-552

package documentation with roxygen headers, 477-482
  function headers, 478-480
  help pages, 480-482
  reports, 547-548
generics, 511-516
  creating, 515-516
  multiple dispatch, 531-532
  naming conventions, 512
  S4, defining, 530-531
  Gentleman, Robert, 3
get function, 164
ggplot function, 333
ggplot2 package, 313
  aes function, 333-336
  aesthetics, 321-329
  controlling, 322-324
  grouped data, 327-329
  legend, 324-327
  combining plot types, 318-321
  custom plots, 333-339
  coordinate systems, 338-339
  working with multiple data frames, 336-338
  ggpplot function, 333
  global themes, 340-341
  legend layout, 341
  paneling, 329-333
    facet_grid function, 329-331
    facet_wrap function, 331-332
  philosophy of, 313-314
plots
  changing, 317-320
  as objects, 316-317
  qplot function, 314-315
  layers, 316
  theme layers, 339-340
  ggvis package, 342
GitHub, installing packages from, 26-27
GLM (Generalized Linear Model) framework
  defined, 412-413
  distribution types, 412
  extensions, 422-423
  Gaussian model fitting, 414
  glm function, 413
  logistic regression, 417-420
  methods for, 415-416
  Poisson regression, 420-422
  glm function, 413
  logistic regression, 418-419
  methods for, 415-416
  Poisson regression, 420-422
Global Environment. See workspaces
global themes, 340-341
graphical parameters, passing, 159-161
graphics
  colors, 288
  devices
    closing, 288
    creating, 287-288
ggplot2 package, 313
aes function, 333-336
aesthetics, 321-329
combining plot types, 318-321
custom plots, 333-339
ggplot function, 333
global themes, 340-341
legend layout, 341
philosophy of, 313-314
plots as objects, 316-317
qplot function, 314-315
theme layers, 339-340
ggvis package, 342
high-level graphics functions, plot, 291-299
lattice graphics, 345
3D, 352-354
bivariate, 350-351
“data” graphics, 354-355
graph options, 356-358
graph types, 347
graphs, creating, 346-355
groups of data, representing, 360-362
panels, 362-371
plotting multiple variables, 358-360
plotting subsets of data, 355
styles, controlling, 372-376
themes, creating, 374-376
transposing the axes, 351-352
univariate, 348-350
layout, controlling, 305-308
grid layouts, 306-307
layout function, 307-308
low-level graphics functions, 299-304
legend, 302-304
lines, 299-300
points, 299-300
text, 300-302
parameters, 304-305
trellis graphics, 345
univariate graphics, 289-291
graphing
bar charts, 291
data frames, 97-98
hist function, 160-162
Greek letters, adding to plots, 294
grep function, 124-125
grid layouts, 306-307
group_by function, 269-271
grouped data, 327-329
gsub function, 124-125
high-level graphics functions, plot, 291-299
hist function, 160-162
histograms, 289
HoltWinters function, 446-447
Holt-Winters method, 446-447
HTML files, building, 550

I

if function, 404
IDEs (integrated development environments), 13
Eclipse, 13
Notepad++, 13
R GUI, 11-12
RStudio, 12-13
if/else function, 461
if/else statements
& and | operators, 144-145
example, 145-146
mixing conditions, 143
multiple test values, 139-140
nested statements, 138-139
returning early, 145
reversing logical values, 142-143
structure, 136-146
summarizing to a single logical, 140-141
switching with logical input, 141-142
using one condition, 139
lattice graphics

Ihaka, Ross, 3
Import Wizard, 218
importing text files, 218
improving code efficiency
benchmarking, 457-458
initialization, 458-459
integrating with C++, 464-468
with memory management, 463-464
profiling, 456
using alternative functions, 462-463
vectorization, 459-462
incorporating tests into packages, 493-494
independent variables
factor variables as, 398-401
indexed printing, 36
inheritance, 508
in S3, 516-518
in S4, 532-534
inhibiting formula interpretation, 404
initialization, 458-459
inner joins, 240
inputs
ellipsis, 157-159
function inputs
capturing, 164-167
checking, 136, 155-157, 162-164
order of “apply” inputs, 201-203
list subcripting inputs
blank inputs, 74
negative integer inputs, 75
positive integer inputs, 74-75
vector subcripting inputs, 44
blank inputs, 44-45
character values, 48
logical values, 46-47
negative integer, 45-46
positive integers, 45
installing
packages, 24-27
from binaries, 26
from CRAN, 25-26
from source, 26-27
R, 573
on Linux, 574-575
on Mac OS X, 574
on Windows, 573-574
RStudio, 577-578
Rtools on Windows, 575-577
integers, creating sequence of, 37-38
interaction terms, 396-398
interactive documents, 569-570
intercepts, removing, 381
is.x functions, 122
iteration, loops
“for” loops, 250
nested loops, 177-179
performance, 180
referencing data with, 176-177
“while” loops, 176-177
J
J function, 275-276
joins
inner joins, 240
merging data in dplyr package, 267-268
outer joins, 240-241
K
Kaplan-Meier estimates, 433-434
keys
defining, 274-275
numeric keys, 276-277
keywords, function, 130-131
knitr package, 548
L
lapply function, 195-204
order of “apply” inputs, 201-203
using with data frames, 203-204
using with vectors, 199-201
LaTeX, 548
dynamic reporting, 553-556
lattice graphics, 345
3D, 352-354
bivariate, 350-351
“data” graphics, 354-355
lattice graphics

graph options
  plot types and formatting, 357-358
  title and axes, 356-357
graphs
  creating, 346-355
  types, 347
groups of data,
  representing, 360-362
panels, 362-371
  controlling strip headers,
    363-364
  functions, 365-371
  multiple “by” variables,
    364-365
plotting multiple variables,
  358-360
plotting subsets of data, 355
styles
  controlling, 372-376
  previewing, 373
themes, creating, 374-376
transposing the axes,
  351-352
univariate, 348-350
layers in quick plots, 316
layout
  controlling, 305-308
  layout function, 307-308
  grid layouts, 306-307
legend function, 302-304
length function, 41-42, 53
library function, 27
licenses for R packages, 475
limitations of S3, 518-519
linear models, 380-381
  assumptions, 411-412
  factor variables, 398-401
  interaction terms, 396-398
  methods for, 406-407
  multiple linear regression
    comparing nested models, 393-395
    creating new models, 391-392
    updating existing models, 392-393
  variable transformations, 402-404
lines function, 299-300
  in nonlinear models, 428
lines on plots, adding, 389-390
Linux
  installing R, 574-575
  installing Rtools, 575
list objects, models as, 386-388
listing
  empty lists, creating, 69
  non-empty lists, creating, 70
  objects, 18-19
lists, 68-86
  attributes, 72-73
  combining, 80
  creating, 71-72
  with element names,
    creating, 71
  elements
    adding, 79-80
    referencing, 76-79
  motivation for using, flexible simulation, 83-84
named lists, 81-82
  extracting elements from, 84
  printing, 72, 85-86
  subscripting, 73
  blank inputs, 74
  character value inputs, 76
  logical value inputs, 75
  negative integer inputs, 75
  positive integer inputs, 74-75
  subsetting, 73
  unnamed lists, 81
lm function, 380-381
  methods for, 406-407
loading packages, 27-28
logical values
  as list subscripting input, 75
  as matrix subscripting input, 56-57
  reversing, 142-143
  specifying, 36
  as vector subscripting input, 46-47
logistic regression, 417-420
logRange function, 155
loop variable, 175-176
loops
  in C++, 467
  “for” loops, 174, 250
  initialization, 458-459
  nested loops, 177-179
  performance, 180
models 

reiterating data with, 176-177
while” loops, 174
low-level graphics functions, 299-304
legend, 302-304
lines, 299-300
points, 299-300
text, 300-302
ls.str function, 18-19
lubridate package, 107-108

M

Mac OS
installing R, 574
installing RStudio, 577-578
installing Rtools, 575
mailing lists, 4
manipulating. See also sorting
character data, 123-124
dates, 105-106
factor levels, 110-111
times, 105-106
manuals, 4-5
margin values (apply function), 183-184
Markdown, 548. See also RMarkdown
masking, 27-28
mathematical functions, 117-118
matrices, 34, 49-58
attributes, 52-54
column index, 55
creating, 49-52
with a single vector, 51-52
dropping dimensions, 56
reiterating data frames as, 90-92
subscripting, 55
character values, 57-58
logical values, 56-57
transposing, 50-51
matrix function, 51-52
melt function, 243-245
memory management, 463-464
merge function, 238-241, 267-268
inner joins, 240
outer joins, 240-241
merging data.tables, 279-280
METACRAN website, 24
methods, 512
defining for arithmetic operators, 513-514
for GLM framework, 415-416
for linear models, 406-407
parametric methods in survival analysis, 434-435
for Reference Classes, 537-540
S4, 529-530
summary function and, 405
updating, 513
microbenchmark package, 457-458
Microsoft Excel. See Excel
missing data functions, 122-123
mode function, 34
models, 379
assessing, 382
abline function, 389-390
extractor functions, 385-386
interaction terms, 396-398
as list objects, 386-388
plot function, 383-385
predict function, 390-391
summary function, 382-383
GLM framework
defined, 412-413
distribution types, 412
extensions, 422-423
Gaussian model fitting, 414
glm function, 413
logistic regression, 417-420
methods for, 415-416
Poisson regression, 420-422
linear models, 380-381
assumptions, 411-412
factor variables, 398-401
interaction terms, 396-398
methods for, 406-407
variable transformations, 402-404
multiple linear regression
comparing nested models, 393-395
creating new models, 391-392
updating existing models, 392-393
nonlinear regression
assumptions, 423
extensions, 430
nls function, 423-425
Puromycin data example, 425-429
survival analysis, 430
censoring in, 431-432
estimating survival function, 432-436
extensions, 441
ovarian data frame example, 431
proportional hazards regression, 437-441
time series analysis
ARIMA models, 448-451
autocorrelations, 448
decomposition, 443-445
extensions, 452
smoothing, 446-447
ts function, 441-443
modes. See data types
motivation for using lists, flexible simulation, 83-84
multimode data structures, 36, 67-68
data frames, 86-93
apply functions, 193-195
attributes, querying, 87
columns, selecting, 88
columns, subscripting, 88-90
creating, 86-87
graphing, 97-98
lapply function, 203-204
referencing as a matrix, 90-92
returning top and bottom of data, 93-94
sorting, 236-237
splitting, 197-199
subscripting, 92-93
viewing, 94-96
working with multiple, 336-338
lists, 68-86
attributes, 72-73
creating, 71-72
with element names, creating, 71
empty lists, creating, 69
motivation for using, 83-84
named lists, 81-82
non-empty lists, creating, 70
printing, 72, 85-86
subscripting, 73
unnamed lists, 81
multiple dispatch, 531-532
multiple linear regression
comparing nested models, 393-395
creating new models, 391-392
updating existing models, 392-393
Murrell, Paul, 313
mutable objects, 538-539
mutate function, 266
N
named arguments, 131
named lists, 81-82
extracting elements from, 84
names function, 42-43, 386-388
NAMESPACE file, 475-476
naming
functions, 132
generics, 512
objects, 20
S3 classes, 512
variables, 241
navigating to CRAN, 573
nchar function, 123
ncol function, 53
negative integer inputs, 45-46, 75
nested calls, 41
nested loops, 177-179
nested models, comparing, 393-395
nicknames, 7
nls function, 423-425
non-empty lists, creating, 70
nonlinear regression
assumptions, 423
extensions, 430
nls function, 423-425
Puromycin data example, 425-429
Notepad++, 13
nrow function, 53
numeric factors, 109
numeric keys, 276-277
numeric values
creating sequence of, 38-39
simulating, 83-84

object orientation, 505-508
inherent, 508
R and, 405-406
objects, 16-22. See also packages
converting, 156-157
date objects, creating, 103-104
listing, 18-19
mutable objects, 538-539
naming, 20
packages, 17
search path, 17-18
plots as, 316-317
Reference Class objects, copying, 540-542
removing from workspace, 20
return objects, 134-136
saving, 22
tbl_df objects, creating, 262-263
time objects, creating, 104-105
workspaces, 19-22
objects function, 18
odbcConnectAccess function, 224
online resources, 4-5
operating systems
Mac OS
installing R, 574
installing RStudio, 577-578
installing Rtools, 575
Windows
building packages, 482
clipboard, 219
operators, 117-118
&., 144-145
arithmetic operators, defining methods for, 513-514
pipe, 248, 271-273
order function, 236-237
outer joins, 240-241
output of functions, saving, 131
ovarian data example (survival analysis), 431
packages, 7, 17, 23-28
bigmemory, 282
building, 471-472
with devtools, 482-485
checking, 482-484
code quality, 476-477
data, including, 494-496
data.table, 273-282
aggregation, 280-282
columns, adding, 277-278
columns, renaming, 277-278
merging data tables, 279-280
rows, adding, 278-279
setting a key, 274-275
subscripting, 275-276
deleting, 24
dependencies, 27
documentation, generating with roxygen headers, 477-482
dplyr, 261-273
aggregation, 268-271
merge function, 267-268
mutate function, 266
pipe operator, 271-273
sorting, 263
subscripting, 264-266
dplyr package, creating
tbl_df objects, 262-263
extending, 489-490
ff, 282
finding, 23-24
foreign, 222
ggplot2, 313
aes function, 333-336
aesthetics, 321-329
combining plot types, 318-321
paneling, 329-333
pacf function, 448
packages, 7, 17, 23-28
bigmemory, 282
building, 471-472
with devtools, 482-485
checking, 482-484
Pacf function, 448
packages, 7, 17, 23-28
bigmemory, 282
building, 471-472
with devtools, 482-485
checking, 482-484
packages, 593
phi

ology of, 313-314
plots as objects, 316-317
qplot function, 314-315
ggplot2 package

ggplot function, 333
global themes, 340-341
legend layout, 341
theme layers, 339-340
ggvis package, 342
installing, 24-27, 485
from binaries, 26
from CRAN, 25-26
from source, 26-27
knitr, 548
lattice, 346
licenses, 475
loading, 27-28
lubridate, 107-108
masking, 28
METACRAN website, 24
microbenchmark, 457-458
proto, 544
Rcpp, 501-502
repositories, 23
reshape, 243
cast function, 245-246
melt function, 243-245
RODBC, 223-225
sas7bdat, 223
search path, 17-18
Shiny, 561-566
applications, 561-566
interactive documents, 569-570
reactive functions, 566-568
sharing applications, 570
structure, 472-476
creating, 472-474
DESCRIPTION file, 474-475
NAMESPACE file, 475-476
tests, incorporating, 493-494
tidy, 246-249
gather function, 247-248
separate function, 249
spread function, 248
vignettes, 496-498
markdown notation, 499
writing, 498-501
XLConnect, 228-231
zoo, 123

Packages pane (RStudio), 24
paneling, 329-333
facet_grid function, 329-331
facet_wrap function, 331-332
with lattice graphics, 362-371
controlling strip headers, 363-364
functions, 365-371
multiple “by” variables, 364-365
par function, 304-305
parameters, setting for plotting functions, 304-305

parametric methods in survival analysis, 434-435
passing graphical parameters, 159-161
paste function, 124, 157-158
performance, loop performance, 180
pipe operator, 248, 271-273
plot function, 291-299, 383-385
in GLM framework, 416
paneling, facet_grid function, 329-331
parameters, setting, 304-305
in proportional hazards regression, 439-441
qplots, layers, 316
in survival analysis, 434
in time series analysis, 442-443
plots
custom plots, 333-339
aes function, 333-336
coordinate systems, 338-339
ggplot function, 333
multiple data frames, 336-338
diagnostic plots, 383-385
comparing, 387-394
in GLM framework, 416
for time series analysis, 449-450
lines on, adding, 389-390
in nonlinear models, 428-429
relational databases, 223-226
  DBI, 225-226
  RODBC package, 223-225
remove.packages function, 24
removing
classes, 510
intercepts, 381
objects from workspace, 20
renaming columns, 277-278
reordering factors, 110
rep function, 39-41
repeated values, creating
sequence of, 39-41
replace function, 122
reporting
bugs, 8
dynamic reporting, 547-548
LaTeX, 553-556
RMarkdown, 548-552
repositories
CRAN
  METACRAN website, 24
  packages, finding, 23-24
  for packages, 23
representing groups of data, 360-362
reshape package, 243
cast function, 245-246
melt function, 243-245
resid function, 385-386
restoring R sessions, 221
restructuring, 242-249
  with reshape package, 243
cast function, 245-246
melt function, 243-245
  with tidyrr package, 246-249
gather function, 247-248
spread function, 248
return objects, 134-136
returning error messages,
  152-153
reversing logical values,
  142-143
RExcel, 13
RMarkdown, dynamic reporting,
  548-552
code chunks, including,
  550-552
HTML files, building, 550
RODBC package, 223-225
rows, adding, 278-279
roxygen headers, generating
documentation with, 477-482
function headers, 478-480
help pages, 480-482
Rprof function, 456
RStudio, 12-13
data frames, viewing, 94-96
Help pane, 28-29
Import Wizard, 218
Installing, 577-578
packages, loading, 27-28
Packages pane, 24
script window, 132
sessions, restoring, 221
Source pane, 16
text files
  importing, 218
  reading, 218-220
Rtools, installing on Windows,
  575-577
runif function, 157
R Console, 14-15
R Consortium, 3, 5-6
R Development Core Team, 3
R Documentation, 5
R GUI, 11-12
R models. See models
R6 class system, 542-544
  active bindings, 544
  example of, 543-544
  private members, 542
  public members, 542
rbind function, 50, 237-238
Rcpp package, 464-468,
  501-502
.RData format, 221
reading
CSV files, 220
structured data from Excel,
  226-227
text files, 218-220
read.table function, 218
recommended packages, 23
records, counting, 281
re-creating simulated values,
  120
Reference Classes, 535-542
  creating, 535-537
  documenting, 542
  methods, defining, 537-540
  objects, copying, 540-542
referencing
columns, 179-180
data frames as a matrix,
  90-92
data with loops, 176-177
list elements, 76-79
with $, 77-79
double square bracket referencing, 76-77
regular expressions, 124, 182
relational databases, 223-226
DBI, 225-226
RODBC package, 223-225
remove.packages function, 24
removing
classes, 510
intercepts, 381
objects from workspace, 20
renaming columns, 277-278
reordering factors, 110
rep function, 39-41
repeated values, creating sequence of, 39-41
replace function, 122
reporting
bugs, 8
dynamic reporting, 547-548
LaTex, 553-556
RMarkdown, 548-552
repositories
CRAN
METACRAN website, 24
packages, finding, 23-24
for packages, 23
representing groups of data, 360-362
reshape package, 243
cast function, 245-246
melt function, 243-245
resid function, 385-386
restoring R sessions, 221
restructuring, 242-249
with reshape package, 243
cast function, 245-246
melt function, 243-245
with tidyr package, 246-249
gather function, 247-248
spread function, 248
return objects, 134-136
returning error messages, 152-153
reversing logical values, 142-143
RExcel, 13
RMarkdown, dynamic reporting, 548-552
code chunks, including, 550-552
HTML files, building, 550
RODBC package, 223-225
rows, adding, 278-279
roxygen headers, generating documentation with, 477-482
function headers, 478-480
help pages, 480-482
Rprof function, 456
RStudio, 12-13
data frames, viewing, 94-96
Help pane, 28-29
Import Wizard, 218
Installing, 577-578
packages, loading, 27-28
Packages pane, 24
script window, 132
sessions, restoring, 221
Source pane, 16
text files
importing, 218
reading, 218-220
Rtools, installing on Windows, 575-577
runif function, 157
S
S, development of, 1-3
S3 class system, 406, 509
classes, creating, 509-511
documenting, 518
inheritance, 516-518
limitations of, 518-519
lists versus attributes, 514-515
naming conventions, 512
S4 class system, 523-535
defining classes, 525-529
documenting, 534-535
generics, defining, 530-531
inheritance, 532-534
methods, 529-530
multiple dispatch, 531-532
sapply function, 204-208
returns, 205-207
Sarkar, Deepayan, 346
sas7bdat package, 223
save function, 22
saving
function output, 131
workspace objects, 22
workspaces, 221-222
scoping rules for functions, 133-134
script window (RStudio), 132
scripting, 16
search path, 17-18
masking, 28
searching and replacing character data, 124-125
searchpaths function, 17-18
select function, 264-265
selecting columns from data frames, 88
self-starting functions, 427
separate function, 249
seq function, 38-39
sequence of repeated values, creating, 39-41
server component of Shiny applications, 564-566
sharing Shiny applications, 570
Shiny package, 561-566
applications
server component, 564-566
sharing, 570
structure, 561-562
ui component, 562-564
interactive documents, 569-570
reactive functions, 566-568
shortened $ referencing, 78-79
simulated values, re-creating, 120
simulating numeric values, 83-84
single mode data structures, 34-35. See also multimode data structures
arrays, 58-60
creating, 58-60
comparing, 60-62
matrices, 49-58
attributes, 52-54
column index, 55
creating, 49-52
dropping dimensions, 56
subscripting, 55
transposing, 50-51
vectors, 35-49
attributes, 41-43
combining, 49-51
creating, 35-41
lapply function, 199-201
subscripting, 43-49
smoothing in time series analysis, 446-447
sorting
with arrange function, 263
data frames, 236-237
descending sorts, 237
Source pane (RStudio), 16
special characters, adding to plots, 294
specifying
colors, 288
logical values, 36
variables for aggregate function, 254-256
split function, 195-197
splitting data frames, 197-199
S-PLUS, 3
spread function, 248
sqlcolumns function, 224
statistical distributions, 119-120
statistical models. See models
Statistical Sciences, Inc., 3
statistical summary functions, 118-119
missing data, 122-123
stl function, 443-445
stop function, 152
structure
of functions, 129-130
of if/else statements, 136-146
of R packages, 472-476
creating, 472-474
DESCRIPTION file, 474-475
NAMESPACE file, 475-476
of Shiny applications, 561-562
tidy structure, 243
structured data, reading from Excel, 226-227
styles for lattice graphics
controlling, 372-376
previewing, 373
subscripting, 60-62
arrays, 60
columns, 88-90
data frames, 92-93
data.tables, 275-276
with filter function, 264
lists, 73
  blank inputs, 74
  character value inputs, 76
  logical values, 75
  negative integer inputs, 75
  positive integer inputs, 74-75
matrices, 55
  character values, 57-58
  logical values, 56-57
with select function, 264-265
vectors, 43-49
  blank inputs, 44-45
  character values, 48
  logical values, 46-47
  negative integers, 45-46
  positive integers, 45
subsets of time series, 443
subsetting lists, 73
substitute function, 166
substring function, 123
summarizing data frames, 96
summary function, 96, 382-383, 405
  classes and methods, 405
  in GLM framework, 415-416
  with names function, 388
  in survival analysis, 433-434
survfit function, 433-434
  in proportional hazards regression, 439-441
survival analysis, 430
  censoring in, 431-432
  estimating survival function, 432-436
  extensions, 441
  ovarian data frame example, 431
  proportional hazards regression, 437-441
switch function, 159
symbols, plotting symbols, 296-297
syntax
  comment blocks, 15
  continuation prompts, 15
  lists
    named lists, 81-82
    unnamed lists, 81
  R Console, 14-15
table function, 121
tail function, 94
tapply function, 208-213
  multiple grouping variables, 209-210
  multiple returns, 210-212
  return values, 212
Task Views, 23-24
tbl_df objects, creating, 262-263
test framework, developing, 490-494
  incorporating tests into packages, 493-494
  test_that function, 490-493
  test_that function, 490-493
test-driven development, 494
text files, 217-223
  exporting, 220
  importing, 218
  reading, 218-220
text function, 300-302
theme layers, 339-340
themes, creating for lattice graphics, 374-376
tidy data, 243
tidy package, 246-249
  gather function, 247-248
  separate function, 249
  spread function, 248
tilde (~), formula relationships, 381
time
  lubridate package, 107-108
  manipulating, 105-106
time objects, creating, 104-105
time series analysis
  ARIMA models, 448-451
  autocorrelations, 448
  decomposition, 443-445
  extensions, 452
  smoothing, 446-447
  ts function, 441-443
time zones, defining, 105
titles, labeling on plots, 293-294
transforming variables, 402-404
transposing matrices, 50-51
trellis graphics, 345
ts function, 441-443
tsdag function, 449-450

U
ui component of Shiny applications, 562-564
univariate graphics, 289-291
lattice, 348-350
unnamed lists, 81
update function, 392-393
updating methods, 513
UseMethod function, 512
user events, 6

V
variables
continuous variables, 111-112
factor variables
in linear models, 398-401
in logistic regression, 419
loop, 175-176
naming, 241
plotting, 358-360
specifying for aggregate function, 254-256
transforming, 402-404
univariate graphics, 289-291
lattice, 348-350
vectorization, 459-462
vectors, 15, 34-49
attributes, 41-43
combining, 49-51
creating, 35-41
with c function, 35-36
lapply function, 199-201
subscripting, 43-49
blank inputs, 44-45
character values, 48
logical values, 46-47
negative integers, 45-46
positive integers, 45
versions of R, 7-8
nicknames, 7
viewing data frames, 94-96
vignettes, 477
including in packages, 496-498
markdown notation, 499
writing, 498-501
Visualizing Data, 345
visualizing data frames, 97-98

W
warnings for functions, returning, 153-155
websites
METACRAN, 24
R Documentation, 5
R Project website, 3
which argument (plot function), 385
while function, 180-181
“while” loops, 174
white space, 45
Wickham, Hadley, 213, 242, 261, 313
window function, 443
Windows operating system
building packages, 482
clipboard, 219
installing R, 573-574
installing RStudio, 577-578
installing Rtools, 575
working directory, 21
workspaces, 19-22
objects
removing, 20
saving, 22
saving, 221-222
working directory, 21
writing
classes, 505
generics, 511-516
object orientation, 506-508
S3, 509
vignettes, 498-501
X

xapply function, 182
X-axis, labeling on plots, 293-295
XCode, installing Rtools, 575
XLConnect package, 228-231

Y-Z

Y-axis, labeling on plots, 293-295
zoo package, 123