Book Review: *Learn R As a Language*

Author: Pedro Aphalo

General Comments

1. The concept of this book is very good. I like the tie-in between learning a programming language to learning a spoken/written language. In particular, chapters 2, 3, and 5 do this well and offer a cohesive sequence: words, sentences, paragraphs, essays. This layout is both cute and smart.
2. The amount of code examples in this book is also excellent. It is very helpful as a reader to have example code to use as a jumping off point when learning a new part of the language.
3. As I will point out in my specific comments chapter by chapter, several concepts discussed in the book need one extra sentence to *introduce* them before you start going into detail about how they work or comparing them to other things. For example, page 3, Sec 1.2.2, para 2 mentions type checks, operators, and operands but none of these terms have been defined for the reader. I think you should add a sentence that says "a type check is…". Then you're free to start discussing whether R requires type checks or not. As a second example, section 2.13 on lists starts with "Lists' main difference…". It should start with "In R, a list is…" and then you can go on to describe how it differs from other objects.
4. Far too many times, a command is used in an example before it is explained in the text. This is done several times where you explicitly acknowledge it (eg, on page 82 in Sec 2.17 you use `apply()` in the example and state "apply functions are described in section 3.4 on page 108"). Several other times, as I have tried to capture in my specific comments chapter by chapter, it is not acknowledged (e.g., `unlist()` shows up on page 55 in the playground box, but is not described until page 65).
5. This book seems to cater to experienced programmers who want to learn R, rather than someone new to programming. For example, Chapter 7 on ggplot introduces ggplot in section 7.3, which begins with 4 dense paragraphs that will make no sense to the uninitiated reader. However, the following subsections (i.e., 7.3.1 on data and 7.3.2 on mapping, etc, as well as the simple build-up of the most basic plot in section 7.3.8) are fantastic for learning how to create plots with ggplot. The author should lead with the simple parts that can be clearly understood by the novice programmer, and then launch into the precise (but not helpful when learning) "dictionary definitions". As one example, the first paragraph on ggplot2 (page 204) ends with the sentence "Furthermore, the construction is mostly semantics-based and to a large extent, how plots look when printed, displayed, or exported to a bitmap or vector-graphics file is controlled by themes." That simply isn't a "first paragraph worthy" sentence on ggplot. As the author admits when demonstrating the simple plot construction in (the fantastically written) section 7.3.8 on page 207, "the working and use of the grammar are easier to show by example than to explain in words…"

Specific Overall Comments

1. There is an inconsistent use of "I" and "we" throughout the book. For example, page 1 says "I will describe…" and "you will learn…" while page 2 says "by this we mean…" and page 3 says "We will describe…". Page 10 says "I have tried…"
2. The way plural R objects are written is inconsistent. Page 22 line 2 shows "`vector` s" with a space between the R object and the plural "s". This reads funny with the floating "s". Page 24 line 4 does the same with "NA s". However, Page 24 line 10 uses an apostrophe: "NA's" which I think reads much better. The playground box on page 46 provides a third version with no space and no apostrophe: "NAs".
3. Base R is sometimes written with a space and sometimes as base-R. Name space is sometimes namespace.

Detailed Specific Comments Chapter by Chapter

Preface

1. Page XI, para 2: The first sentence says "I do not discuss statistics or data analysis methods in this book". But Chapter 5 is literally titled "Statistics". Maybe you can make it clear that you discuss *how* to invoke some statistical methods while describing the R language, but that you do not discuss statistical *theory* underlying those methods.

Chapter 1: The Language and the Program

1. Page 1-2, Sec 1.2.1, Para 1: The distinction between the R Language and the R Program is not that clearly presented, and this is a terrifically detailed point – much too detailed to be the very first thing the reader encounters! Moreover this section (1.2.1) draws the distinction between R as a language vs R as a program, yet the next two sections introduce R as a language (1.2.2) and R as a program (1.2.3). I would reverse this order: first introduce and define a "language" and a "program" and then draw the distinction between the two. Additionally, on the bottom of page 1, you ask and answer: "Does this make a difference? Yes!" but I fail to see how this makes any difference (and thus I suspect your reader – someone brand new to R – this will make absolutely no sense or difference).
2. Page 2, Sec 1.2.1, Para 4: in addition to mentioning that R is a successor to S, it might be helpful to mention that both followed in the naming convention of the time that used a single letter for a computer program (e.g., C). Also, a cute fun fact: the letter R was chosen to be the open-source implementation of S because two original authors' first names (Robert Gentleman and Ross Ihaka) started with the letter R.
3. Page 3, Sec 1.2.2, para 2: this paragraph mentions type checks, operators, and operands. I don't think the book says who its target audience is, but assuming the reader is new to programming, the reader will not understand what these thing are. These three terms need to be defined.
4. Page 5, Sec 1.2.3.1: The example `print("hello") [1] hello` is displayed. The [1] will look very strange to someone new to R. This is not discussed until page 22. You should mention what's going on with the [1] here on page 5 or at least indicate to the reader that it will be explained later.

5. Page 4, Sec 1.2.3, para 2 (and page 5, sec 1.2.3, para 1): move discussion of IDEs to section 1.2.3.3 which is titled "Editors and IDEs". Currently these two paragraphs are just out of place and they belong in the section on IDEs.
6. Page 5, Sec 1.2.3.1, para 3: Move the first sentence about "examples in this book" to section 1.5 titled "What is needed to run examples in this book". Moreover, para 3 on page 5 is about menu-driven vs text-based programs and so discussion of the book's examples doesn't belong in this paragraph.
7. Sec 1.2.3.1: This section discusses scripts and batch jobs. I think your point is that scripts are better than working at the console. This point, however, is made on the bottom of page 3, again on the bottom on page 4, and throughout this section (1.2.3.1) on pages 5-6. You should consolidate these: Give the title to section 1.2.3.1 "Scripts and Batch Jobs" and then introduce what a script is and why it's better than working at the console.
8. Page 9, Sec 1.3: Add discussion of Quarto, which is replacing rmarkdown: https://quarto.org/
9. Page 10, Sec 1.3, para 2: This is the last paragraph of the section on "Reproducible Data Analysis" yet it talks about how the author tried to use a consistent style throughout this book. Just delete this paragraph. It's off topic for this section (1.3) and doesn't add anything.
10. Page 10, Sec 1.4, para 1: This section and paragraph discusses finding additional information and how there's often multiple ways to accomplish something in R. But you include a lot of discussion about testing code. Delete or move (to a separate section) the discussion about code testing. It's much too specific to come this early in the book. For example, the next section (1.5) tells the reader what is needed to run examples in this book. You definitely should not be emphasizing the need to test one's code before you've even told them how to run their first bit of code!
11. Page 11, Sec 1.4: I would expand this discussion to mention how R's help files have consistent structure. For an excellent overview of R's help files, see https://socviz.co/appendix.html#a-little-more-about-r
12. Page 11, Sec 1.4.1, para 3: R's manuals have ben re-written in Quarto and are much better styled, have better navigation, and generally are more enjoyable to read. You might want to point your reader to that version of online help. See https://rstudio.github.io/r-manuals/
13. Page 14, Sec 1.5, para 2: The introduction to RStudio here is repetitive of Sec 1.2.3.3 on pages 7-8. Delete this.
14. Pages 14-15, Sec 1.5: You should provide the command with the appropriate url to install the package that goes with the book. This section needs the line: `install.packages("url/to/learnrbook")`. Much later (on page 181) it is assumed that the learnrbook package is installed, but you never show the reader how to install it.
15. Page 15, Sec 1.6: This section on "Further Reading" is tremendously sparse – it's not clear to the reader if you mean further reading about programming in general (which is what you currently list in this section) or further reading about R (which is strangely absent from this section). To make sense of this for the reader, I would mention in Sec 1.6 that every chapter of this book has a "further reading" section and the later chapters will point the reader to more R resources.
16. Overall comment regarding Chapter 1: The rest of the book prints results from code using a leading double hash (##) whereas the code in Chapter 1 does not. It might be worth mentioning somewhere that this inconsistency is the result of using screenshots for Chapter 1 whereas the rest of the book's examples result from rendering the code and text using rmarkdown.

Chapter 2: Words and Sentences

6. Page 18, Sec 2.2, para 1: You discuss “interpreters” and “compilers” but the new-to-R user will not know what these things are. You should define them.
7. Page 21 R Code: the code returns TRUE and FALSE but you have not introduced these Boolean values yet. You should say that you will introduce TRUE/FALSE in Sec 2.4 on page 29. Strangely, you use code in para 2 on page 27 that also returns TRUE/FALSE values, but here you do say that you’ll explain what TRUE/FALSE means in the next section.
8. Page 21, 22, 49, and a few other places throughout the book: the R code examples on these pages create a vector object called “c”. It is likely confusing to a new-to-R programmer that you can have the c() function and the c object. It is worth discussing that this is possible, and probably also that you do this for simple examples, but that in general it’s bad practice in R to create variables called c, or T, or F, etc. I will note that a related point is made much later in the book on page 177 at the end of Sec 5.5 about how R handles functions of the same name in different loaded packages.
9. Page 21, Sec 2.3, para 1: The code shows `a <- b <- c <- 0.0` -- it is equivalent to use 0 or 0.0 here. Not sure if there’s a point to the 0.0 instead of simply 0. Similarly, the code at top of page 23 use 2L where a simple 2 would suffice. Also, I wouldn’t create an object ‘c’ which might be confused with the function c() for the new programmer, as noted in my last point.
10. Page 22, Sec 2.3, para 3: You call `c()` a “method” rather than simply a “function.” Since you have not yet discussed the object-oriented part of R yet, the word “method” does not make sense to the reader. Use the simpler “function” or something else.
11. Page 23, Sec 2.3, para 1: You say `c()` and `append()` can be used with lists, but you do not introduce lists until Sec 2.13 on page 62. I would not mention lists here on page 23.
12. Page 23, Sec 2.3, R code: You create objects `a`, `b`, `c`, and `d` and print them in the code, but you do not show the output. Why not? Show the output!
13. Page 23, Sec 2.3, para 3: You mention vector recycling. Expand the discussion to say that R *always* does vector recycling and when (or not) R will issue a warning related to vector recycling (no warning when the long vector length is an exact multiple of the short vector length; otherwise a warning; never an error!).
14. Page 24, Sec 2.3, R code: The code shows that `length(c(a, numeric(0), b))` is 9. However, the most recent definitions of the objects `a` and `b` in the book are `a <- rep(1:6)` and `b <- 5:-1`. These objects have length 6 and 7 so I don’t see how `length(c(a, numeric(0), b))` is 9. It should be 13. My suspicion is that the code in the boxes labeled with coffee mugs or lighthouses is not related to (or computed in the order presented with) the code in the main text, and this becomes confusing when an object like “`a`” are defined in both the coffee mug box and in the main text.
15. Page 25, Sec 2.3, para 1: You discuss regular expressions. Don’t do that yet! You introduce character values 14 pages later in section 2.7 on page 39. You should not discuss RegEx before a simple introduction to character values.
16. Page 21, Sec 2.3, para 3: Here you discuss floating point precision. This makes sense here in section 2.3 titled “Numeric Values and Arithmetic.” However, floating point precision is again discussed in the middle of page 26 and in the box spanning pages 33-35. These latter occurrences should be consolidated to one section and discussed once, not thrice.

17. Page 26, Sec 2.3, R code: The code at the bottom of page 26 that introduces the modulo operators (%% and %/%) are not discussed until the next page. I propose separating the one box of R code on the bottom of page 26 into two blocks of code: first block keeps the + and * operators as well as the / operator, and the second new R code block has the %% and %/% operators and comes after their discussion on page 27.
18. Page 28, Sec 2.3, R Code box: this offers a brief discussion about whether to name arguments in a function when calling the function. I think the book would benefit from a chapter subsection titled "Functions" explaining that ALL operations in R are function calls. You can then go into the details about whether you need to name the arguments of a function call or not. And you can discuss that some functions are defined with default values for arguments. Some of this is covered much later in Sec 5.3.1 on page 169.
19. Page 30, Sec 2.4, R code: Most people use the single operator (& or |) and almost never need the double operator (&& or ||) but you predominantly use the double operators in your examples. I would switch them for the single operators.
20. Page 5, Sec 2.5: You should start this section with an explanation that two equal signs (==) is used to test equality. Right now, there is simply a comment in the code box to explain this. It's important enough that it deserves its own sentence or paragraph.
21. Page 32, Sec 2.5, R code: You should explain operator precedence. Right now the lighthouse box on page 32 effectively says "hey reader: you go figure this out" which comes across as a lazy author.
22. Page 34, Sec 2.5, para 2: This paragraph uses terms like compiler, executable, and run time. These should be defined for the reader.
23. Page 34-35, R Code: I think the take-away is that type double can handle values larger than type integer. If so, I would make this more clear by saying so and showing the code. Currently, just the code is provided and the reader is left to guess about what point you are trying to make.
24. Page 36, R Code at top: What is '`a`'? Is it the same object defined 4 pages earlier on the top of page 32? If so, the two tests `a==0.0` and `abs(a) < 1e-15` which both return a vector of 10 values of FALSE are not helpful examples to demonstrate when to use ==0 and when to use <1e-15.
    Additionally, you test `sin(pi) == 0` which returns `FALSE`. You should tell the reader that `pi` is a built-in vector of length one with value 3.1415… and perhaps also remind the reader that mathematically sin(pi)=0.
25. Page 36, Sec 2.6, R Code at bottom: You create character vectors for this section, but you don't introduce character vectors until the next section (2.7 on page 39). Either put section 2.7 before 2.6 or say at the beginning of 2.6 that you'll introduce character vectors in the next section.
26. Page 37, Sec 2.6, R Code at top: The example uses `setdiff(union(bakery, dairy,), shopping)`. Show `setdiff()` and `union()` separately before combining them to make it clear to the reader what each one does. Relatedly, you define and describe `union()` on page 38 so maybe don't use it in the code example on page 37; wait until after it's introduced on page 38. Similarly, the example code uses the `%in%` operator, which isn't introduced until further down the page on page 37.
27. Page 39, Sec 2.6, R Code at top: The example is `9L %in% 2L:4L`. None of the integer notation is necessary. If you run `class(2:4)`, R will return `integer`. Is there a reason to be this specific? Are you encouraging the reader to always specify integers with L? If so, say so. If not, simplify

to just `9 %in% 2:4` (and similarly for the other examples at the top of page 39). Relatedly, in the middle of page 39 you drop the L and show the example `9 %in% (2:4)^2`. At a minimum, be consistent, preferably with the "no-L" style.

28. Page 39, Sec 2.7: The first sentence of this section says that "Character variables can be used to store any character." I find this a little misleading. R makes no real distinction between "a" and "aa" – both are a character vector of length one. So I would make that opening sentence plural or more general, as in: '…store any character string."
29. Page 39, Sec 2.7: You explain that there are three types of quotes in the ASCII character set. Say why this matters? Does R only "understand" ASCII characters? What about those curly quotes that Microsoft Word uses – the reader will likely wonder whether those can be used?
30. Page 40, Sec 2.7, para 4: You introduce things like \n and \t as "escape sequences" and then call them "escape codes" 4 times later in the paragraph. I would introduce them as "escape <u>codes</u>" to keep the terminology consistent.
31. Page 41, Sec 2.8: the functions `mode()`, `class()`, and `typeof()` are confusing. You present `mode()` as the go-to function out of the three. However, the book by Chambers "Software for Data Analysis" pages 141-142 says that `class(x)` is the way to determine what kind of think `x` really is, and that "classes are intended to be the official, public, view, with as clear and consistent a conceptual base as possible." My recommendation would be to expand your comparison of `mode()`, `class()`, and `typeof()` possibly also providing a table that compares what they return for different objects, e.g. what is the mode, class, and type of `x=1:10` or `x=c("a", "b")`.
32. Page 43, Sec 2.9, para 1: starts with a statement about the "least intuitive type conversions". You should start this section by saying what a type conversion is before launching into which ones are intuitive or not.
33. Page 43, Sec 2.9, last sentence: you say `print()` and `sprintf()` are used to "make numbers pretty". This is a strangely colloquial way to refer to output of those functions and starkly contrasts your careful, thorough description of the R language up until this point.
34. Page 45, R code: This code uses brackets to subset vectors, but this comes just <u>before</u> section 2.10 where you introduce the bracket-subsetting notation.
35. Page 46, coffee mug box: this talks about a "locale" but you haven't explained what that is yet.
36. Page 47, last sentence: this refers to vector recycling "as discussed above" but "above" actually refers to 24 pages earlier (page 23 in Section 2.3). I would reference page 23 explicitly.
37. Page 53, Section 2.11, R Code at top: You use the code `A[1,1]` to show how matrix subsetting works. This examples comes right after you describe which index selects rows and which selects columns. As a result, don't pick the `[1,1]` index, because it doesn't make the row/column choice clear. Use `A[1,2]` or `A[2,1]` instead!
38. Page 55, Sec 2.11, para 1: You say many matrix operators are available in R but that you're not going to describe them. But you then describe the transpose function `t()` and the matrix multiplication operator `%*%`. I think you should also show how `diag(6)` makes an identity matrix of size 6 and how `solve()` finds the inverse of a square, non-singular matrix. I agree that you don't need to exhaust the matrix computations available in R (e.g., leave out `chol()` for finding a Cholesky root), but I think a couple of extremely common operators (like identity matrices and inverses) should be at least briefly mentioned.

39. Page 55, playground box: You instruct the user to test what happens when they use `unlist()`. This function is not described until page 65 and again in the playground box on page 66. It feels out of place here on page 55.
40. Page 59, Sec 2.12, para 1: You mention how factors are used by ANOVA and linear models. I would also add and make clear that factors save the user from "one hot encoding" a bunch of binary variables when fitting a statistical model. You could use `model.matrix()` here to show a simple example. This would also motivate why someone might need to re-order the levels of a factor variable (as you discuss on the following page, page 60)
41. Pages 56-61, Sec 2.12: You should mention the (now mostly-obsolete) `stringsAsFactors` argument to many "data import" functions like `read.csv()`. This best fits with the discussion about reading in data from files in the coffee cup box at the bottom of page 57. Note that as of R 4.0 released in April 2020, `stringsAsFactors=FALSE` is the default.
42. Page 62, Sec 2.13, first sentence: This section on lists starts with the words "Lists's main difference…" It should start with something like "In R, a list is…" and then you can go on to describe how lists differ from other R objects.
43. Page 62, Sec 2.13.1, bottom line: You use the command `try()` but don't explain what it does. The result will be quite confusing to a new R user, since the particular example you give looks a bit like something worked (when in fact, it didn't).
44. Pages 66-67, Sec 2.14: you repeat on both pages 66 and 67 that data.frames are derived from class list. You also repeat on those two pages that vectors and factors in a data frame are required to be of the same length. Consolidate or remove the redundancy.
45. Page 69, Sec 2.14, R code at top of page: When you subset a data frame using matrix-style indexing, you get back a "smaller" data frame and, importantly, it prints the row names. In the case of this example, the row names are simply the integer number of the row. The reader has not seen row names before and may be puzzled by what is being printed (especially since the row numbers are equivalent to the values stored in the "x" column, so it looks like x is printing twice). I would explain what row names are and that they are present in this example. Row names/numbers similarly print in many subsequent examples on pages 69-70.
46. Page 75, Sec 2.14, last para of the coffee cup box: what is "cleanup"?
47. Page 80, Sec 2.16.2: this page discusses the function `ls()`. `ls()` is commonly used with `save(list=ls())`. You should make clear that the function argument "list" does not accept a list object as input, but rather a vector of character strings, where each character string is an object in the global environment. In particular, the playground box in the middle of the page tells the reader to look at the "list of object names returned" by `ls()`. Here, you're using "list" to mean "collection", but the reader might be assuming that you're referring to the R object of type list.
48. Page 80, Sec 2.16.2 and page 81, Sec 2.16.3: you use `unlink()` to delete files that you create as part of the examples. You should critically warn the user that this command doesn't just delete objects in R's global environment but rather irrevocably deletes files from their computer and therefore should be used very carefully! Also, why use the strangely-named `unlink()` function rather than the better-named function `file.remove()`?
49. Page 82, Sec 2.17, last line: The line says "we can also use sapply(), lapply(), or vapply()…" but the apply family or functions is not described until page 108. I would omit this sentence.

50. Page 84, Sec 2.18, para 1: When discussing formula syntax with plotting, you should reference that R's formula syntax operates a bit like the `with()`, `within()`, and `attach()` functions mentioned earlier in Section 2.14 that the reader is now familiar with. This will help clarify that the user can write simply `dist~speed` instead of `cars$dist ~ cars$speed`.
51. Chapter 2 has a "structural break" at section 2.10. Prior to Section 2.10, you focus on verbs (operations) that typically rely on vectors (or occasionally length-one vectors that look like constants). With section 2.10, you switch to talking about nouns, like vectors, matrices, lists, and data.frames. I wonder if you want to add this structure to the table of contents by breaking chapter 2 into two separate chapters. One other idea would be to move section 2.10 on vectors to be one of the first things introduced in chapter 2 (eg, move section 2.10 to become, say, section 2.3). This might make it more natural to talk about the verbs/operations in sections 2.3-2.9 since the reader has now been formally introduced to vectors as the most-primitive object in R and these objects are used for all of the examples in sections 2.3-2.9.

Chapter 3: Paragraphs and Essays

1. Page 88, Sec 3.2.2, second to last paragraph on page: you refer to `ggplot()` but this hasn't been introduced yet. Just refer to `plot()` here.
2. Page 89, Sec 3.2.2, para 1: you use the default R prompt (>) when showing commands executed in the system's shell. A different prompt could be used to emphasize that the command is not an R command. Many new-to-R users are not familiar with command-line programming and the distinction between an R command and a Shell command should be made as clear as possible.
3. Page 88-89, Sec 3.2.1 and 3.2.2: When reading these sections, the reader gets the sense that most of the time they should be "sourcing" whole R scripts or clicking the run button in RStudio. In my experience, they should mostly be hitting ctrl+enter to run a script line-by-line. While the ctrl+enter approach is mentioned, it certainly is not highlighted. I think much more emphasis should be given to this approach. Relatedly, in section 3.2.3, you mention testing things at the command line until they work, and then copying/pasting the working commands from R's history. A better approach is to write the command in the R script (rather than at the command line), test the command by hitting ctrl+enter, and editing until the line works. It requires the exact same amount of typing without any cumbersome copty/paste, and it keeps the focus on the script instead of the console.
4. Page 91, Sec 3.2.4: mention quarto as the predecessor to rmarkdown: https://quarto.org/
5. Page 100, Sec 3.3.3.1, R code: The first three R code examples each use only a single statement in the for-loop and are written on the same line as the `for()` command. However, the first example does not use curly braces while the second and third do. Why do that? Curly braces are only needed for compound statements, so they're not necessary for any of these three examples, and there's no explanation for why you include them sometimes but not others. The second example on page 101 is the first instance of multiple statements in a for-loop and necessarily uses the curly braces. Stick to that.
6. Page 100, Sec 3.3.3.1, para 5: just before the R code examples, you say "Some examples of use of for loops – and of how to avoid their use." But you don't motivate why anyone would want to avoid their use. You need to tell the reader that vectorized operations offer simpler code and faster execution. The "faster execution" bit gets mentioned a few pages later in section 3.3.4, but would be helpful information to have before seeing examples of "how to avoid their use".

7. Page 101, Sec 3.3.3.1, bottom of page: You discuss `seq(along.with=a)` but the more common and simpler use is `seq_along(a)`.
8. Pages 100-102, Sec 3.3.3.1: You should mention that the iterator (for example, the `i` in `for(i in 1:10)`) persists at the termination of the loop. So after the `for()` command, it is possible to call `print(i)` for example. This is an unusual feature of R and likely worth mentioning.
9. Pages 102-103, both coffee cup boxes: You describe `break()` and `next()` but never provide an example. Examples of these commands would be helpful to clarify how they operate.
10. Page 104-105, Sec 3.3.4, coffee cup box: The point of this box is to demonstrate how vectorized code can run faster, and it comes just after a paragraph introducing the `system.time()` command (presumably for measuring execution time). But then you don't actually time your examples. You show the examples, untimed. Their execution speed is then listed in the paragraph text that follows. I think you should provide the evidence in the code! Show the `system.time()` *results in the code* for each of the approaches.
11. Pages 104-105, Sec 3.3.4: This section on "loops can be slow" feels out of place. It breaks the flow between (1) introducing commands that handle iteration and (2) nesting iteration. I think a more natural place to move Sec 3.3.4 would be just before Section 3.4 on page 108 where you introduce the `apply()` family of functions.
12. Page 108, Sec 3.4, first sentence: The first sentence of this section is, verbatim, "Apply functions apply a function passed as an argument to parameter FUN or equivalent, to elements in a collection of R objects passed as an argument to parameter x or equivalent." This is painful to read and makes the help-page look easy by comparison. I contrast this with Hadley's section on "motivation for functional programming" (http://adv-r.had.co.nz/Functional-programming.html#fp-motivation) in his Advanced R book where he introduces `lapply()` as a way to write code so that you don't have to repeat yourself. Even though his introduction is technical, I find it to be a much more intuitive introduction compared to the "dictionary definition" used to start Section 3.4 here.
    Additionally, this section first mentions `apply()` in the sentence: "In summary, apply() is used to… while lapply() and sapply() are used to…" Regarding apply(): you shouldn't refer to a function following the words "in summary" if you haven't introduce that function yet!
13. Page 109, Sec 3.4.1, R code at top: You create a custom function my.fun() but you don't tell the user about creating their own functions until Sec 5.3 on page 166.
14. Page 109, Sec 3.4.1, para 2: You introduce anonymous functions here. You could mention the new shorthand `\(x)` for `function(x)` when introducing an anonymous function (but do this later, in Sec 5.3, rather than here!).
15. Page 113, Sec 3.4.2, first para: You describe how infix functions like `a+b` could be written as `` `+`(a,b) ``. by providing the example `` sapply(…, FUN=`x`) ``. There is no need to do this in the context of `apply()` functions. This just makes the concept of infix functions too difficult because they're "nested" in the complexity of the apply-family of functions. I recommend moving discussion of infix functions out of the section on apply-functions, and show a simple example like `` `+`(a,b) ``.

Chapter 4: Statistics

1. Page 119, Sec 4.2, para 1: Way too many parentheses in this paragraph.

2. Page 120, Sec 4.2, middle of page: the argument is not `na.omit` but rather `na.rm`. As a result, the R code example shows two results that are both NA, rather than one result that is NA and one result that is 10.5. Change the code and discussion to refer to `na.rm`.
3. Page 123, Sec 4.3.3, warning box: You say "probability functions like `pnorm()` always do computations based on a single tail of the distribution." It might be helpful to elaborate that p-values are cumulative probabilities that "add up" starting from the left tail of a distribution. Thus `qnorm(0.025)` equals -1.96. And that this behavior can usually be switched to the right tail with the argument `lower.tail=FALSE`.
4. Page 123, Sec 4.3.4, bottom of page: The R code to show random draws just prints values. This doesn't "show" the reader that the code is "working." I recommend running `hist(rnorm(1e3))` or something similar, where the reader can see that the histogram looks similar to the desired distribution, but with sampling error.
5. Page 126, Sec 4.5.1, first sentence: You say "When the data frame (or matrix) contains only two columns, the returned value is equivalent to that of passing the two columns individually as vectors". This isn't true. Yes, the correlation is the same. But when you pass vectors into `cor()`, you get a length-one vector out. When you pass the two-column data frame into `cor()`, you get a 2x2 matrix out.
6. Page 126, Sec 4.5.1, middle of page: You say "… cor.test() also computes the t-value, p-value, and confidence interval for the estimate." This is a bit colloquial. It is correct to say that the estimate has a confidence interval. But it's not correct to say the estimate has a t-value or p-value without specifying the hypothesis being tested.
7. Page 127, Sec 4.5.2, first sentence of this section: typo. "passing <u>and</u> argument" should be "passing <u>an</u> argument".
8. Page 130, Sec 4.6.1, top half of page: You omit the intercept from the linear regression by using the formula `dist ~ speed - 1`. Equivalently, you can use `dist ~ 0 + speed`. I find the latter version more intuitive as it "multiplies" the intercept by zero (therefore omitting it from the model) rather than by "subtracting 1".
9. Page 134, p-value calculation in code: You incorrectly calculate the p-value. You use `dt()` where you should instead use `pt()`. In order to do a two-tailed hypothesis test (as indicated in the text) you will also need to do `2*(1-pt(…))`.
10. Page 138, Sec 4.6.3. This section on ANCOVA should be dropped. There is nothing new (from a coding perspective) that the reader learns and from my experience, this perspective on linear modeling has been almost entirely replaced with linear regression.
11. Page 141, Sec 4.8: I am not familiar with `nls()` or the self-starting functions. From reviewing the code in the book, I gather that the self-starting functions are not just for specifying the starting values for the iterative algorithm, but also basically define the model? Is that true? Can you use `nls()` without a self-starting function? As a seasoned R user, I am not familiar with `nls()` and I have a lot of questions and so I suspect that other readers will feel the same. Consider expanding the exposition on NLS to clarify these questions.
12. Page 146, Sec 4.9, bottom para: This paragraph is trying to say that packages for more complex models introduce extensions to Base R's model formula syntax. This is a great point! However, this lengthy paragraph starts with the sentence: "Nesting of factors in experiments using hierarchical designs such as split-plots or repeated measures, results in the need to compute additional error terms, differing in their degrees of freedom". What does any of this have to do

with formula syntax?!? The perplexed reader might eventually deduce the answer, but it would be substantially clearer to say: "Packages exist that fit more complex models and some of these packages introduce extensions to the model formula syntax." Then you can talk about which packages, which complex models, etc.

13. Page 147, Sec 4.9, last R example: You use `inherits()` because `is.formula()` does not exist. You should provide a description or introduction of `inherits()`. What is this function? How does it differ from the `is.x()` family of functions? When do I use it vs an `is.x()` function?
14. Page 148, Sec 4.9, first R example: You use a `for()` loop with the iterator formula, as in `for(formula in formulas)`… This code `calls lm(formula, …)`. This might be confusing to the reader since the first argument to `lm()` is `formula`, such that a verbose call would look like `lm(formula=formula, …)`. Since you have not yet introduced that function inputs can match the names of function arguments without ambiguity, it would be best to change the code here to use, say, `form` as in `for(form in formulas)`.
15. Page 148, Sec 4.9, second paragraph: This is super minor, but you say "as could be expected, a conversion constructor is available with name `as.formula()`…" but on the preceding page you say that `is.formula()` doesn't exist. So, at this point, the reader probably assumes `as.formula()` also does not exist, so I wouldn't start the sentence with "as could be expected".
16. Page 151, Sec 4.10: It is strange to me that you introduce time series before any discussion of dates. Dates are quite tricky, as they are stored numerically, but need to print such that a human can read them, and they must properly account for time zones, daylight saving time, etc. And while I appreciate the desire to keep the time series introduction quite brief, I'm surprised that the autocorrelation function `acf()` makes no appearance.
17. Page 153, Sec 4.11.1, last line on the page: There are 3, not 2, species of flower in the Iris dataset.
18. Pages 153-155, Sec 4.11.1: This section contains 1.5 pages of code and output with no explanation as to what is happening or why the reader should care. At a minimum, the model that is being fit should be described before the code is run to fit the model. Perhaps compare the results of this MANOVA to the separate results of two ANOVAs or something.
19. Page 157-158, Sec 4.11.3: This section is on multi-dimensional scaling. This feels like a weird choice. I would have expected to see factor analysis or ARIMA or nonparametric density estimation with `density()`, each of which -- I suspect -- is more commonly used than MDS.

Chapter 5: Adding New Words

1. Page 163, Sec 5.2: This section is about packages. Curiously, you define what a package is in the second sentence of the second paragraph on page 165 (the third page of this section). Don't do that to your reader! Put the definition of a package as the first sentence of the first paragraph of the section back on page 163.
2. Page 163, Sec 5.2.1: You use "source" and "binary" to describe file types. It may help the reader to define these terms.
3. Page 164, Sec 5.2.1, para 1: You note that as of January 2017 there are more than 10,000 packages on CRAN. This figure needs to be updated. As of January 2023, there are almost 20,000 packages on CRAN.
4. Pages 164 and 166: It is strange that you introduce CRAN and Bioconductor on page 164, but then re-introduce them with hyperlinks on page 166. Consolidate these two introductions.

5. Page 164, Sec 5.2.1, para 2: You refer the reader to the devtools package for installing packages from github. The remotes package is also an excellent choice (newer than devtools and a bit "lighter weight" with the focus just on installing packages from non-CRAN repositories).
6. Pages 164-165, Sec 5.2.2: This section discusses "name mangling" and "compilers" and "namespaces". None of these terms has been defined. It may help the reader to define these terms.
7. Page 168, Sec 5.3, para 1: You use the colloquial term "disappear" which feels out of place with the exactness of the rest of the text.
8. Page 169, Sec 5.3.1, top half of page: base R uses the function argument `na.rm` not `na.omit`.
9. Chapter 5 overall thought: most R users will employ a simple two-step process to obtain and use functions not in base R: `install.packages(newpkg); library(newpkg)`. Somewhere between the start of the chapter on page 163 and the end of section 5.2 on page 166 this needs to be made explicit. You have all of the gory detail "behind" these two functions, but you don't tell the reader: "hey, use install.packages() to "install" a new package into R, and use the library() function to make objects (like functions and datasets) from that package 'available' in the current R session." And unless you want to talk about namespace/environment precedence (and masking) for function name lookups, I would leave the description of "make available" simple.

Chapter 6: New Grammars of Data

1. Page 180, Sec 6.2, para 2: Typo. "difficult too read" should be "difficult to read"
2. Page 180, Sec 6.2: It is strange to me that you talk about packages data.table, tibble, and dplyr in this introduction as though the reader is familiar with them. Because you then go on to *introduce* what these packages are on the following pages (181-188). For example, I find it confusing that you talk about how "data.table is best example" of something (page 180) when the reader does not yet know what the data.table package is.
3. Page 181, Sec 6.4.1: The subsection introduces the data.table package. Yet you don't actually say the word "data.table" until the second paragraph. Not a great introduction! The first sentence of this section should start with something like, "the data.table package offers a different way to interact with data frames, and it is optimized for speed." You can then go into the details about how data.table uses arguments by reference rather than by copy.
4. Page 182, Sec 6.4.2: This section introduces tibbles. You sort of do this, but you mostly focus on describing what is not backwards compatible with data frames. Sure, explain the imperfect backwards compatibility, but lead with a proper introduction: "The tibble package offers a different set of functionality from data.table (which we just discussed) but is similarly designed to provide enhanced data frames. These enhancements include…". And then you can end with: "One caution with tibbles is that they are not always backwards compatible with data frames, although the authors of the tibble package have worked hard to ensure that they often are. For example, one common instance of broken backwards compatibility is…"
5. Page 185, Sec 6.4.2, lighthouse box: You say that the column A is a character in the data frame and it is a factor in the tibble. This is incorrect; it is a character in both places.
6. Overall comment on section 6.4: Both data.table and tibbles via the tidyverse are popular. However, you devote two paragraphs toward the discussion of data.table and 6 pages to the discussion of tibbles. Why the disparity? Are you advocating for tibbles over data.tables?

Aren't some of the topics you discuss regarding tibbles (printing, conversion) also applicable to data.tables?

7. Page 187-188, Sec 6.5: You focus on pipes as a means of "relieving the user from the responsibility of creating and deleting temporary objects" and "of enforcing the sequential execution of the different steps". ALL code (written with pipes or otherwise) enforces the sequential execution of the different steps, so I would omit that point. Also, code can usually be written (by nesting function calls) to avoid the creation of temporary objects. So your two main benefits aren't really benefits of pipes. In my opinion, the REAL benefit of pipes is to convert code from looking like math g(f(x)) to looking like English x |> f() |> g(). The benefits of doing so include clarity of code and fewer errors or bugs.
8. Page 188, Sec 6.5, first line: This is a shell command. You provide no prompt, so (1) it looks the R commands on the rest of the page and could easily be confused for an R command, and (2) it is inconsistent with the shell prompt (">") you use on page 89.
9. Page 188, Sec 6.5, coffee cup box: You mention a function's "suitable signature." The reader likely does not know what this. Provide a definition.
10. Overall comment on section 6.5: As of R 4.0, there is now a "native" pipe operator in R (|>) which seems to work very similarly to %>%. Discussion of |> should be added to section 6.5
11. Page 190, Sec 6.6: the discussion in the first paragraph is about long and wide datasets. The reader would benefit from an example of each at this point in the text. A very simple (3 row, 4 column) example would suffice, but the description alone is not sufficient to communicate the differences to the uninitiated reader.
12. Pages 190-192, Sec 6.6: tidyr functions `gather()` and `spread()` have been superseded by `pivot_longer()` and `pivot_wider()`. This section should be updated to refer to the newer functions.
13. Page 192, Sec 6.7: It's not clear to the reader which packages are part of the tidyverse and which are not. This is particularly clear in the warning box that starts section 6.7 where you discuss dplyr, data.table, dtplyr, seplyr, and tibble. Only dplyr and tibble are part of the tidyverse. All others except data.table were written by the authors of the Tidyverse, but aren't strictly speaking part of the tidyverse.
14. Page 193, Sec 6.7.1, last paragraph: you use a function from package 'strigr'. This package has never been introduced to the reader. You should tell the reader it's part of the tidyverse.
15. Page 194, Sec 6.7.1, last paragraph: Using `select()` to subset the columns of a tibble has an importance difference from base R's bracket [] column subsetting in that `select()` always returns a tibble, whereas bracket subsetting returns a data frame unless only one column is chosen to be returned, in which case the chosen column is returned as a vector. This importance difference is worth mentioning.
16. Page 195, Sec 6.7.1, last code example: The code highlighting is strange because you have chosen the new column name "dim" which registers to the rmarkdown syntax highlighting as a keyword and is therefore colored green when printing. This makes your use of the `rename()` function see like one of the arguments is "dim" taking value "dimension" when in fact your assigning the new name "dim" to the column currently called "dimension". Pick a different new name so that the new name isn't colored green when rendered.
17. Page 196, Sec 6.7.2, warning box: You say "grouped_df" is the most derived class? What does this mean? It was not discussed in section 5.4 where you introduced classes.

18. Page 201, Sec 6.8: this section on "further reading" is specific to additional reading on the tidyverse. However, chapter 6 is about much more than just the tidyverse. Packages data.table and magrittr are also mentioned, as well as the general topic of "new grammars" in R. I might also refer the reader to the excellent data.table vignettes, the book "extending R" by John Chambers, or any of the 4 books listed on the "learn" webpage of the tidyverse website: https://www.tidyverse.org/learn/.

Chapter 7: Grammar of Graphics

1. Page 204, Sec 7.3: Here you introduce + as the operator for ggplot commands. Since ggplot2 is part of the tidyverse set of packages and it "plays nicely" with them, it may be worth emphasizing that the operator of choice for ggplot is the + and not the pipe (|> or %>%). The reason for this is that Hadley Wickham developed ggplot2 before discovering the pipe. He maintains + as the operator for backwards compatibility, but has said publicly that he would change it to the pipe if he could.
2. Page 205, Sec 7.3.1: The last sentence of this paragraph lists the variable types that can be plotted. Why do this? Are there any types that cannot be plotted? Moreover, you only list numeric, factor, character, and posixct, but then in Section 7.4.6 on page 228 you indicate that spatial coordinates are also acceptable.
3. Page 214, Sec 7.3.9, warning box: Here you introduce `str()` from the gginnards package. This is redundant as you have already introduced `str()` in the playground box on the top of page 209.
4. Page 221, Sec 7.4.1, last para of section: You discuss plotting error bars using geom_errorbar but provide no example. Please add an example.
5. Chapter 7 overall comment: you introduce many geometries, scales, and other ggplot elements. It would help a reader to have a one-page summary of all geometries, perhaps shown as a two-panel table where the top panel lists the most commonly used geoms and the bottom panel lists the rest. I recommend a similar one-page summary for scales or whatever other major categorization of plotting elements you feel makes sense.

Chapter 8: Data Import and Export

1. Page 299, Sec 8.6, para 1: You introduce the two types of data storage formats in plain text files here, and then again two pages later on page 301, Sec 8.6.1, para 1. This is redundant.
2. Page 302, Sec 8.6.1, para 3: You say that "by default columns containing character strings are converted into factors" and that you can change this behavior by providing the argument `stringsAsFactors=FALSE`. Since R 4.0 released in April 2020, `stringsAsFactors=FALSE` has been the default. This paragraph is therefore out of date with the current base R behavior. This is also mentioned and needs to be updated on the top of page 306.