To Tidy or not When Teaching R Skills in Biology Classes

Kelly Carscadden[1] & Andrew Martin[1]

[1] University of Colorado, USA

Correspondence: Andrew Martin, University of Colorado, USA. E-mail: am@colorado.edu

Received: February 25, 2022 Accepted: March 27, 2022 Online Published: March 30, 2022

doi:10.5430/ijhe.v11n5p39 URL: https://doi.org/10.5430/ijhe.v11n5p39

Abstract

An essential skill for STEM undergraduates is the ability to understand the world by manipulating, visualizing, and analyzing data to make or evaluate claims. Current online debate, without peer-reviewed literature, explores which of two common R syntax environments (base R or tidyverse) is best for teaching novice R users. In an in-person undergraduate course on evolutionary biology, we implemented two coding curricula: one using base R (n = 49 students) and the other using tidyverse (n = 58 students). We compared these two curricula using several dimensions of student success: interpretation of syntax, creation of appropriate data visualizations and analyses, and an absence of sex bias in performance. A linear model revealed prior experience had the largest estimated effect, followed by syntax environment; sex had the smallest effect. Pedagogical approaches that ensure students have repeated opportunities for practice and that implement techniques to overcome student frustration and anxiety are likely more important than syntax environment when learning coding in biology classes. Furthermore, the small effect of sex combined with the high proportion of females in the biological sciences suggests introducing computer programming in biology may allow females to discover interest and ability that they may not have had if computer programming was the sole propriety of computer science departments.

Keywords: Base r, tidyverse, quantitative analysis, biology

1. Introduction

1.1 The Importance and Challenge of Quantitative Analysis in Biology Education

Computational thinking and constructing algorithms are essential practices for making sense of the world using data. It is becoming increasingly clear that learning computer programming exercises key elements of being a productive scientific thinker. Some of these elements include thinking through a problem, breaking down the problem and abstracting it to remove extraneous details, devising a solution, and explicitly communicating the solution (to a computer) using unambiguous and purposeful language (Saeli et al., 2011; Buitrago Flórez et al., 2017). There are, however, a number of challenges and barriers to successfully training students in computer programming, including helping students overcome computer anxiety, understand syntax and the flow and manipulation of data, and troubleshoot their own work using available resources.

Student anxiety towards computers, programming, and statistics is commonly reported (Onwuegbuzie & Wilson, 2003; Connolly et al., 2009). Negative attitudes like anxiety, underestimated self-efficacy, and perceptions that computers are difficult to use or unnecessary are linked to reduced computer skills in students (Mayer, 1981; Speier et al., 1996). Similarly, several studies have linked statistics anxiety with poor course performance (reviewed in Onwuegbuzie & Wilson, 2003). Hence, teaching R in ways that help reduce student frustration and anxiety may be instrumental to improving student performance and their willingness to engage in programming in the future. This is important because much of biology—and other disciplines—increasingly involves being able to effectively manipulate, analyze, visualize, and interpret data.

Gender equality and diversity have been important issues in STEM, and especially so in computer sciences, where females are dramatically underrepresented at every career stage (Sax, 2012; Whitney et al., 2013). Historically, males may have had more time with computers and developed greater computing interest and confidence (e.g., Chen, 1986; Busch, 1995; He & Freeman, 2019). A recent meta-analysis suggests gains in gender equity have been made in the last several decades. Females reported greater enjoyment and self-efficacy in their technological abilities than before; however, a gender gap in attitudes persists (Cai et al., 2017). There is an emerging consensus that the limited success and representation of females in computer sciences stems from stereotype threat, which "occurs when targets of stereotypes alleging intellectual inferiority are reminded of the possibility of confirming these stereotypes" (Inzlicht &

Ben-Zeev, 2000: 365). The possibility of stereotype threat underscores the necessity of ensuring the introduction of computer programming into a biology course does not adversely affect a gender.

Pedagogical approaches that can help students, particularly underrepresented groups, increase their enjoyment, interest, and confidence in computer programming may make students more likely to engage with technology and remove one barrier to success in STEM fields (Mayer, 1981). Thus, one motivation for our study was assessment of whether computational and algorithmic thinking goals revealed sex-based differences in learning gains, and if so, what can be done to limit, or eliminate these differences in success.

1.2 The Use of R

R is an open-source, widely-used, object-oriented programming language that is transforming the way biologists do and share science (Lowndes et al., 2017; R Core Team, 2018). Incorporating R into undergraduate biology courses lets students readily manipulate, visualize, and analyze data for constructing or evaluating evidence-based claims about the world. Furthermore, in comparison to other point and click 'black box' programs, R encourages algorithmic thinking; students must think through and specify a series of steps to accomplish their data task. Lastly, R is a useful tool for open and reproducible science because it requires users to create a shareable stepwise set of executable commands, referred to as a script.

Because R is a flexible and evolving programming language, there are numerous different ways of constructing algorithms for solving problems. For instructors, this means there are many choices when introducing students to R coding (Fox & Anderson, 2005; Ross et al., 2017; Wickham et al., 2019). Here, we focus on two different, popular syntax environments within R: base R and tidyverse. Base R is the most common syntax environment mostly because it has a longer history than tidyverse. Many online courses, online resources, and internet discussion forums focus on the base R syntax environment. Tidyverse is a derivative syntax environment that enables more efficient data manipulation, analysis and visualization (Wickham & Grolemund, 2016; Wickham et al., 2019). Our comparison of R instructional methods happened in the context of an undergraduate biology course. Students in this course typically have limited previous exposure to R and generally very little experience or comfort writing original code. Additionally, many students begin the semester with some negative attitudes towards learning and using R. The power and flexibility of R comes along with hurdles to new learners, such as a plethora of functions with different naming conventions and syntax, a large number of different ways to accomplish a given task, and the absence of graphical user interface (Muenchen, 2014). The need to overcome technical challenges and student anxiety and frustration with R raises the question of how to best teach students to code.

1.3 Choice of Syntax Enviroment: Base R or Tidyverse for An Introduction to Coding

An active point of discussion in online forums is whether the base R or tidyverse syntax environment is best for teaching new students. Several chat forums, blogs, twitter posts, and other sources contain claims supporting one or the other approach. However, all evidence for either alternative appears anecdotal, and we are not aware of any primary literature about whether one is better than the other. Broadly, those in favor of teaching base R have touted its fundamental nature. Using base R syntax, there are often many alternative ways to solve a given problem; this allows creativity and flexibility in how students think about coding problems, and students may emerge as more capable problem solvers after overcoming this initial learning curve. Online forums (e.g., Stack Overflow) are invaluable for troubleshooting code, and many solutions are still not in tidyverse syntax. The streamlined flow of data in the tidyverse syntax environment could decrease students' exploration and understanding of intermediate steps and data products. In contrast, advocates of tidyverse suggest the syntax should get students up and running with code more quickly, given the consistency among tidy functions in the order of arguments they accept and how functions work together to create and manipulate data objects (e.g., Wickham & Grolemund, 2016; Robinson, 2017; Heppler, 2018; Wickham et al., 2019). Developers of tidyverse packages argue that the names of tidy functions more clearly convey what a given function does (e.g., *select* certain variables; Wickham & Grolemund, 2016). Tidyverse code is designed to contain "English-like" functions and be read and understood in a typical linear fashion. In contrast, base R syntax can result in nested, onion-like series of functions that might be more challenging to interpret. Lastly, the multiplicity of different solutions to a coding challenge using base R is seen as a source of potential confusion for new learners. Tidyverse seems to offer fewer different ways to accomplish a task, which may generate a more consistent and shareable data analysis pipeline.

From this online discussion and our own experience with the two approaches, it is conceivable that the differences between the two syntax environments may influence students' ability to generate functional algorithms and develop computational and quantitative thinking skills. The two environments have three key differences in how students develop algorithms for solving problems: (1) how functions are named, (2) how data are used to generate new objects,

and perhaps most importantly (3) how data flow through functions (i.e., use of "pipe" syntax in tidyverse). First, students need to learn syntax and the functional meaning of words used to call functions. The naming of functions may therefore influence how students think about what the algorithm directs the computer to do. For example, the functions *subset* and *filter* in base R and tidyverse, respectively, do the same thing: extract data from an object that meet some explicit criteria. So, although using either function on a data object will generate the same general output, the words subset and filter may mean different things to students and impact the ease of use of these functions.

Second, base R and tidyverse differ in how data are used to generate new objects. In some cases, base R requires "for loops" and row-column indexing of data frames for repeated calculation of some relevant values. So, for example, if there are multiple observations within several different groups, a for loop can be used to calculate and store summary statistics for each group. In this way, base R works by decomposing the problem into smaller, replicate parts, and iterating across groups. In tidyverse, the for loop is replaced by combining the functions *group_by* and *summarise*.

Third, in base R, the script is often built either as a set of separate, sequential commands resulting in new generated data objects at each step, or as a series of nested functions that must be read from the inside out. That is, a 'core' function transforms some data, and that output is taken by the next function, and the next, like layers of an onion radiating outward. By contrast, tidyverse uses piping: a process of moving data from one function to another in a series of dependent, linear steps without generating novel intermediate objects. Wickham and Grolemund (2016, Ch. 18.2) provide a useful comparison of these syntax environments using a children's story (Figure 1).

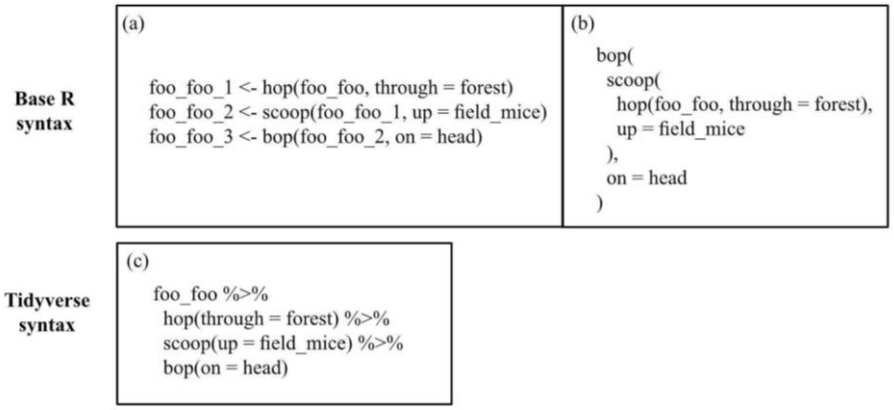

Figure 1. Comparing flow of information and organization of functions in base R (a, b) and tidyverse (c) syntax environments. Example text from Wickham and Grolemund (2016, Ch. 18.2). Re-creating little bunny foo foo's tale would either require several intermediate objects (a) or an onion-like series of functions (b), whereas the tidyverse version can be read linearly from left to right (c).

Prior to our study, we were not aware of data for making an objective, evidence-based decision about whether to teach beginners programming using base R or tidyverse. Hence, we set out to test whether it matters if students learn how to code in base R or tidyverse by evaluating student performance when confronted with the same set of challenges. We sought to determine whether there was a difference between base R and tidyverse in terms of: student gains in understanding syntax, student gains in constructing effective data visualizations and analyses, and whether one syntax environment was more sex-equitable than the other. We empirically addressed these questions by comparing the performance of students in lab sections of an undergraduate course that received instruction in either base R or tidyverse. The course focused on evolutionary biology, not computer programming, and our evaluation of how students used computers was in the context of learning key principles in evolutionary biology. Our exploratory analysis was motivated by the goal of engaging in data-driven revision of the curriculum and teaching strategies aimed at improving students' abilities as scientific thinkers.

2. Method

2.1 Characteristics of the Study

We conducted our study in an upper-division course with a strong emphasis on data manipulation, analysis, and visualization using R. The course was implemented in-person, prior to the pandemic. Many of the students had been exposed to R prior to this course, but most of their experience involved using existing algorithms and executing prepared scripts. For most students, this was the first course in which they were given examples and practice exercises

and were expected to adapt the examples to construct their own algorithms for manipulating, analyzing, visualizing, and ultimately making sense of data. There were six separate lab sections (~ 13-20 students in each). Two different graduate teaching assistants taught the labs, but the curriculum and coding challenges were identical across all sections. In addition, there were five undergraduate teaching assistants that helped students with troubleshooting. Three randomly chosen lab sections received instruction and coding resources in base R and the other three received instruction and coding resources in tidyverse syntax. Otherwise, the lab content, skills, and assessments were the same across all lab sections. We stress, however, that there were many reasons why students in a particular lab section may have experienced a different learning environment that may have influenced their gain in R coding skills such that the degrees of freedom for our statistical analyses is unknown and less than expected based on the number of students. Our results apply to the specific course investigated and are not meant to be general; however, the study could help inform the ongoing debate about the relative effectiveness of teaching base R or tidyverse programming syntax and rules for algorithms. Because the purpose of the study was to promote data-driven revision of the curriculum and teaching strategies, the characteristics of our study did not warrant IRB approval.

2.2 Measures

We implemented pre- and post-instruction assessments focused on understanding syntax and the ability to construct algorithms that effectively manipulate, analyze, and visualize data. In the pre-assessment, very few students were able to understand R syntax or construct functional algorithms, so we focused only on the post-instruction assessment data. When we did use pre-post comparisons, the analyses used composite scores and not individuals scores to maintain anonymity of students. To gauge students' understanding of syntax, we constructed three short pieces of code in each of the two syntax environments (base R and tidyverse) and asked students to explain the function and the predicted products from each short code example (Table 1). To assess students' skill at coding, we presented students with a simple coding challenge. We confronted students with a dataset that included mid-parent and average offspring beak sizes for a set of Darwin's finches. Students were asked to visualize the data and estimate heritability (e.g., the slope of an offspring ~ parent regression). Students had access to their notes, previous work, and the Internet during the assessment. The particular challenge was unique, however; students had not previously been asked to estimate the heritability of beak size.

Our coding challenge rubric evaluated whether students were successful (1) or not (0) for seven items that were common to both treatments (base R, tidyverse): construct a graph of the data, use the knit function that is part of the process of making the script available as an html file, change the axes and axes labels to better convey relevant information, construct a linear model, use the linear model to draw a line showing predicted values on the graph, insert informative annotations into the code using the # sign, and access the statistics from the linear model. Other possible rubric items could have been scored, but these represent core aspects of students practice in both syntax environments.

Table 1. Queries aimed at estimating whether students understand syntax.

tidyverse	base R
InsectSprays %>% filter(spray == "B")	my_dat <- subset(InsectSprays, spray == "B")
ggplot(data, aes(x=x, y=y)) + geom_point(shape=1) + geom_smooth(method=lm)	plot(data$x, data$y, pch=1) model <- lm(data$y ~ data$x) abline(model)
data %>% group_by(group) %>% summarise(mean_size = mean(size))	means <- rep(NA, 10 for (t in 1:10){ means[t] <- mean(data$size[group==t])}

2.3 Samples and Non-Independence

The number of individuals categorized by the three predictor variables included 49 and 58 students in the base R and tidyverse treatments, respectively; 63 and 44 females and males, respectively; and 12 and 95 students with and without known experience coding in R, respectively. The known experience coding in R was based on record of successfully completing a first-year, R-based statistics class and showing evidence of some knowledge of coding in R on a pre-assessment. Importantly, students were not fully independent observations because certain students interacted repeatedly, the course emphasized collaboration and cooperation, and we know students shared code; however, the assessments were implemented as an individual challenge and we evaluated all students as independent data points. However, because of the non-independence of individuals, all statistical estimates of uncertainty likely underestimate the true uncertainty rendering p values misleading (Hurlbert, 1984). Therefore, we do not include p values or estimates of uncertainty and do not refer to any of our results as significant. Our aim was simply to estimate the relative effects of syntax environment, prior experience using R, and sex on student performance. We conducted all analyses using R (R Core Team, 2018).

2.4 Analyses

To evaluate differences in student understanding of key syntax, we used Chi-square tests comparing base R and tidyverse student performance on each of three syntax components related to manipulating, plotting, and summarizing data.

Students' competency with constructing a script that manipulates, analyzes, and visualizes data in R was estimated by the sum of the binary rubric scores for the seven rubric items. We included three predictor variables in a linear model constructed for explaining some of the variation in the estimated R competency of students; the syntax environment (base R and tidyverse), sex (male or female), and known student experience with R. For each of the predictor variables, the difference in the summed rubric scores was visualized.

Additionally, we visualized the composite binary rubric scores for all individuals using non-metric multidimensional scaling (NMDS)(Oksanen et al., 2019). NMDS is a widely used method for visual representation of the similarity (or differences) among entities based on multiple variables. Visualization of NMDS scores was used to reveal the diversity of student answers. We further characterized the diversity of students' composite rubric score by counting the number of different types of answers.

3. Results

3.1 Analysis of Syntax

Our assessment of syntax revealed that one advantage of tidyverse lies in the interpretability of code that calculates summary statistics by groups (a common coding problem in biology and many other fields; Table 2). Tidyverse students asked to interpret tidyverse code for calculating group means far outperformed base R students asked to interpret base R code, which relies upon for loops to accomplish the same task ($p < 0.001$). While 51% of tidyverse students correctly described what the group means code would accomplish (the highest success rate, by far, for any syntax question), only 4% of base R students correctly described what the for loop code would produce (Table 2). There was no discernible difference among treatment groups when it came to understanding syntax for data manipulation or visualization (e.g. subset and filter, and plot and ggplot, for base R and tidyverse, respectively); only 16-22% of students correctly interpreted these code snippets (Table 2).

Table 2. Comparison of students' understanding of tidyverse vs base R syntax based on chi-square tests

Syntax	Proportion correct tidy	Proportion correct base	Difference
Data manipulation (filter/subset)	0.22	0.20	0.02
Data visualization (ggplot/plot)	0.16	0.17	-0.01
Group means (summarize/for loop)	0.51	0.04	0.47

3.2 Analysis of coding abilities

Students' overall scores across the coding challenges ranged from 0 to 7 (of a maximum possible score of 7)(Figure 2). The average and median scores for all 107 students were both 3. The linear model analysis revealed base R students were generally more successful than tidyverse students (Table 3 and Figure 3): the base R effect was mostly due differences in students' abilities to modify axes (Figure 3). There were was a detectable but small effect of sex (females > males [Table 3 and Figure 3]). Student experience showed the largest effect (more experience = higher score [Table 3 and Figure 3]).

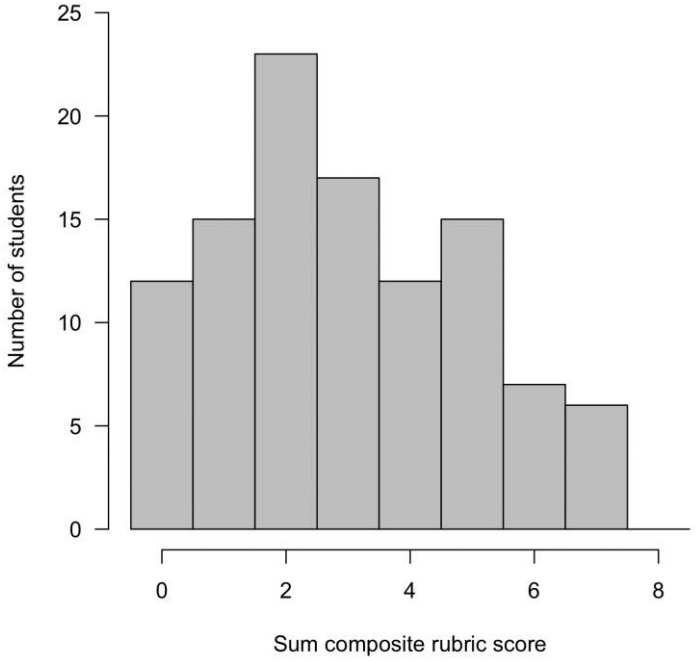

Figure 2. Histogram showing the variation in student scores for 7 rubric items used to evaluate student coding performance in the post-instruction assessment.

Table 3. Results from an additive linear model.

Predictor variable	Estimate
Intercept	2.73
Syntax environment (base R)	0.78
Prior experience	1.54
Biological sex (male)	-0.68

3.3 Diversity of Student Answers

There was a total of 31 different student answers based on the composite rubric scores. These 31 answers spanned the whole range of possible summed scores and the plot of NMDS scores revealed remarkable diversity (Figure 4). Moreover, because the proximity of points in graphical space defined by the two NMDS reflects similarity of answers, it is instructive that in some cases, some composite scores (e.g. 3) were more similar to composite scores of 2 or 4 depending on which of the seven rubric items for students demonstrated competence. The rank order distribution of the frequency of the 31 student answers followed a log-normal distribution (data not shown), with a few answers that were very common and many answers that were either unique or infrequent.

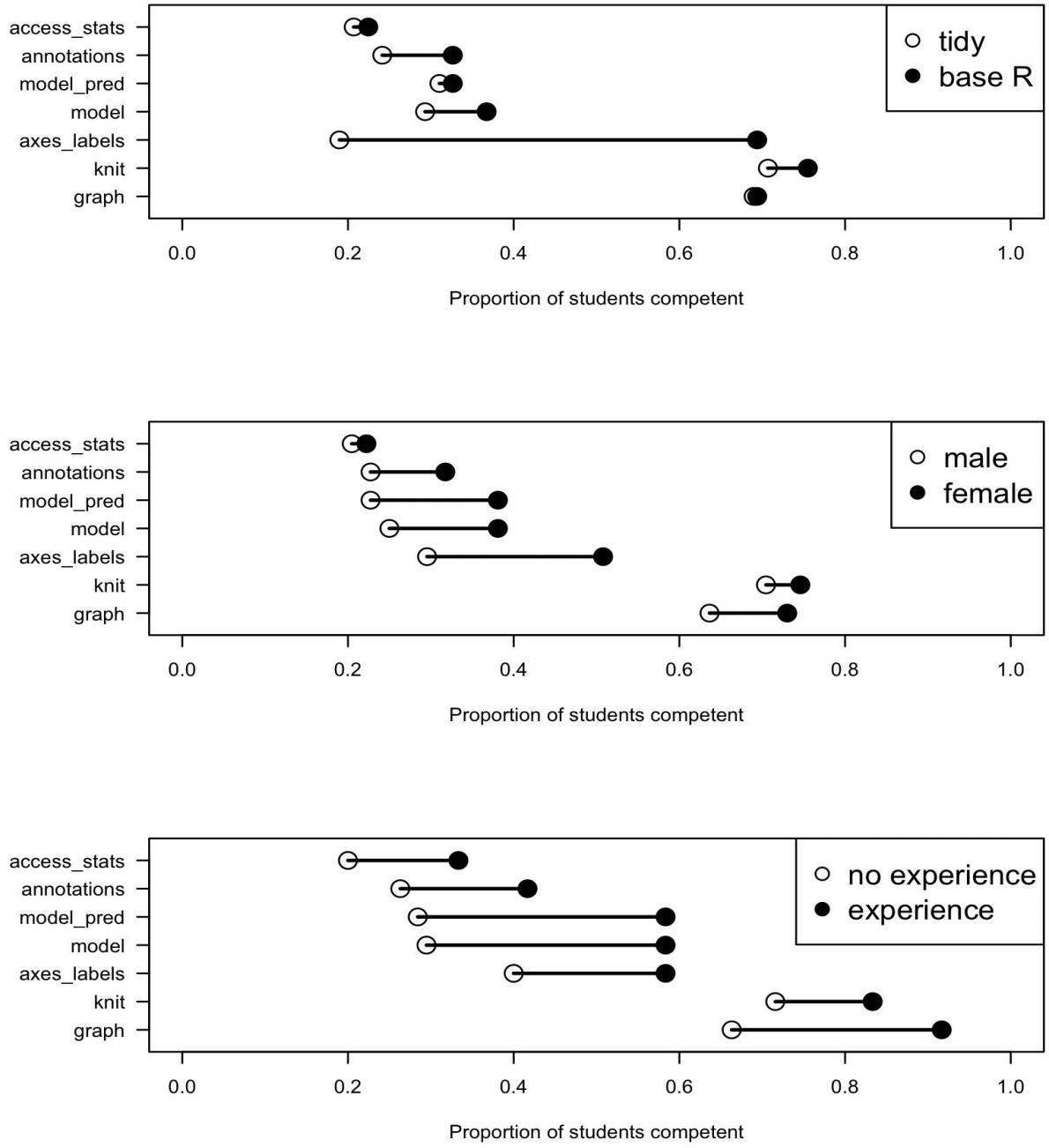

Figure 3. Comparison of the proportion of students that were assigned a value of one for each of seven rubric items between the two different syntax environments (top), depending on whether they had prior experience with R (middle), and whether students were male or female (bottom).

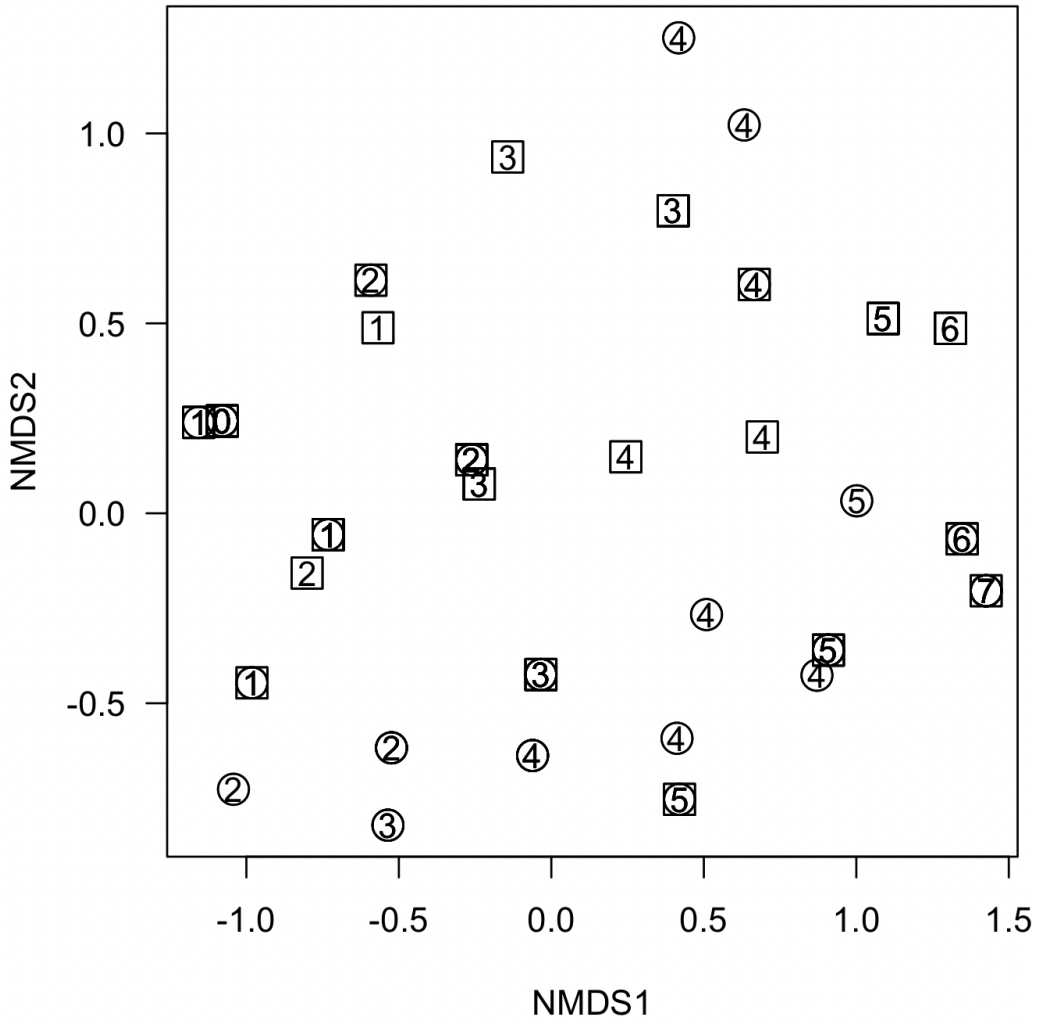

Figure 4. Visualization of NMDS data based on the presence and absence of rubric items for all students. Circles and squares identify individuals in the base R and tidyverse syntax environments, respectively. Numbers inside the symbols indicate the sum across the seven rubric items. Points that are in closer proximity indicate more similar composite rubric scores than points that are further apart.

Student responses on the coding challenges were remarkably variable given that they received identical instructional materials during class and had access to all of their scripts and notes and could search the internet during the final assessment. Moreover, it was clear some students failed to learn the best type of graph for visualizing particular data (e.g. scatterplot versus line graph). So, for example, of the 74 students who were scored as successfully producing a graph visualizing the data, eight students produced graphs other than a scatterplot.

4. Discussion

Data manipulation, analysis, and interpretation are essential skills for students across a broad array of quantitative disciplines (Hoar, 2014; Feser et al., 2017). For this reason, our department—of biology—has recently decided to focus on R because it is free, flexible, powerful, and widely used within and beyond the department. As is true for any new training program initiatives, curriculum revision may be necessary to adequately integrate these skills within and across courses. To make data-driven recommendations for improving our curriculum in ecology and evolutionary biology, we quantified and compared undergraduate student learning and attitude outcomes when they used base R or tidyverse syntax environments to make sense of data in an upper-division evolution course. We evaluated the impact of R syntax environment by characterizing student ability to interpret syntax and manipulate and visualize data to address evolutionary questions. Lastly, we tested for sex bias in student coding performance to ensure our pedagogical approach was equitable. Overall, we discovered that syntax environment and sex had little to no discernible effect. The

largest effect was whether students had prior experience coding in R. Our conclusion is that what matters most is why coding is emphasized and how it is taught and not the syntax environment.

In our study, the final assessment was challenging for students; the average score was less than 50%. Moreover, the variation among students indicated that it was not just one aspect of coding that all students found consistently challenging. Rather, each component of the coding challenge was difficult for some subset of students. Furthermore, students appeared to struggle to demonstrate their programming competency even though they had encountered similar coding challenges throughout the semester, and they had access to notes, prior scripts, and the internet during the final exam. This observation suggests that the symbolic and abstract thinking inherent in constructing algorithms for making sense of data presents a significant cognitive challenge. Put differently, providing students with the necessary experience recognizing common structures and linking the common structural elements of problems to an appropriate mode of analysis, adapting existing code to achieve a unique goal, and becoming fluent in the use of functions for solving common problems were significant challenges. This is true in part because students were simultaneously learning biology and how to manipulate, analyze, visualize, and make sense of data using R. Thus, imposing quantitative analysis and computational literacy as key learning goals in a biology class might be elevating cognitive load to a degree that limits success for some students.

4.1 Base versus Tidyverse

We discovered students in base R achieved greater gains in constructing scripts that effectively manipulate, analyze and visualize data for the purpose of making or evaluating evidence-based claims. Matloff (2019) contended teaching tidyverse to beginners may add complexity up front, for example by requiring instructors to explain the pipe (%>%) and *ggplot* syntax for students to make basic summaries and plots of their data. For individuals with familiarity with coding in R, plotting with *ggplot* is seen as an easier and more efficient alternative to base R plotting, so we were surprised that students using base R syntax scored slightly better when it came to constructing and modifying plots. Although we did detect a difference in success depending syntax environment, most of the difference was explained by students' ability to modify the axes in ways that better conveyed the story in the data. Another possible explanation for difference between the two syntax environments may reflect that there were more numerous online resources for troubleshooting base R than tidyverse.

The main message from the comparison of performance between the syntax environments was not that students performed better in one than the other, but that, overall there were clear deficits in student learning. This result suggests a continuing challenge remains developing more effective curriculum and teaching strategies. As educators interested in developing critical and quantitative thinking abilities, our goal should be supporting student gains in using R (or another appropriate language) for making sense of data regardless of syntax environment. We recommend a pragmatic approach, wherein instructors use the functions necessary for particular data manipulation, analysis, and visualization goals that may entail borrowing strengths from each syntax environment or committing to the approach with which they are most familiar. Indeed, several widely used online instruction materials for data analysis have featured a mixture of base R fundamentals and tidyverse functions (e.g., stat545.com/basic-data-care.html; datacarpentry.org/semester-biology/materials/). Others suggest an alternative approach designed to remove some of the challenging idiosyncrasies among functions and help students get up and running with R more quickly. For example, within **mosaic** and **ggformula** R packages, students use streamlined and consistent formula-based syntax (y ~ x) to summarize, plot, and analyze data (Pruim et al., 2017; Kaplan & Pruim, 2020). Lastly, data manipulation that would require several tidyverse functions (e.g., *filter*, *select*, *summarize*, etc.) could instead be accomplished with a single, flexible **data.table** function (Dowle & Srinivasan, 2019; Matloff, 2019).

4.2 The Success of Males and Females

Analyses of the coding challenge results indicate that, overall, there was little or no evidence of sex-bias in the performance of students learning R; however, females performed better, on average, than males in both syntax environments. The lack of a large sex effect, relative to prior experience, suggests the curriculum and teaching strategies created an equitable environment across this important axis of student identity. Moreover, the high proportion of females in the biological sciences suggests introducing computer programming in biology may allow females to discover interest and ability that they may not have had if computer programming was the sole propriety of computer science departments.

4.3 Limitations of the Study

This particular study was not designed in a manner that enabled robust inference of the effect of syntax environment, prior experience, or sex on student gains in coding ability; there were simply too many uncontrollable variables. For

instance, the results may reflect differences in the effect of the graduate student instructors and the five learning assistants that worked with students; differences in time of day and day of the week for the different lab sections; and the variation in student interactions among the six sections. The many uncontrolled variables that likely influenced student performance precluded robust statistical analysis; consequently, we did not attempt to establish whether the estimated effects were larger than expected from random sampling error. We emphasize the purpose of publishing the results from our quasi-experimental study was to emphasize that coding is a useful skill for biology students and teaching and learning coding in a biology class is challenging.

4.4 Going forward

Developing students' quantitative skills is essential for improving critical thinking and students' chances of gaining employment in increasingly data-dependent careers (Marbach-Ad et al., 2019; Holdren et al., 2012). Yet, the low average students scores suggest we did not achieve our learning goals. In some cases, the group work component of the lab, instead of promoting cooperative learning (Gillies, 2003), was found to encourage peers to copy each others' scripts. Copying obviates the need for problem solving and undoubtedly reduced learning outcomes. Students may have perceived they were successful because they were able to produce graphs of the data; however, in many cases, the graphs were nonsensical. Therefore, in future classes, we will focus on developing a "Does it make sense?" mindset. One further issue was evident from a student's paraphrased semester-end review of the course: if I wanted to learn computer programming, I would have enrolled in computer science. Clearly there is work to be done to make the purpose and value of coding relevant for early career biological thinkers.

The education literature indicates that instructors could improve student learning by actively considering student affect, engagement, and cognitive load (Kirschner, 2002; Trujillo & Tanner, 2017; Tanner, 2017; Guzman et al., 2019). For example, an emphasis on short worked examples that introduces only the minimum necessary R functions, and prompting students to check their understanding by explaining small sections of code, can improve student confidence and motivation while reducing stress and frustration (Guzman et al., 2019). In our study, new R functions were introduced through a combination of lecture and worked examples, where students were asked to follow along and then adapt the code for a new dataset. Additionally, R functions were introduced sequentially only as needed. However, adding questions for students to interpret code segments, rather than just the end result, in assignments or in-class quizzes (e.g., "clicker questions", real-time surveys) would provide valuable opportunities for students and instructors to confirm understanding. Perhaps the best way to help students create more functional algorithms is a stronger emphasis on the habit of adding informative annotations, comprehensible to their peers, that explicitly describe each line of code (Matloff, 2019).

Finally, as has been shown in many areas in STEM, repeated exposure over time may aid student mastery by improving their ability to learn and recall material, solve problems, and apply their knowledge to new challenges (Kang, 2016). Having multiple opportunities to practice using particular functions is likely critical for student coding success. Since the greatest gains in student learning likely require coordinated inclusion of programming and data analysis across courses (Guzman et al., 2019), we need to incorporate computational and algorithmic thinking challenges throughout the training program, from freshman through senior levels.

Our ultimate goal is to build a biology curriculum with a strong emphasis on quantitative thinking aided by developing students' abilities to manipulate, visualize, and analyze data for the purpose of making or evaluating evidence-based claims. To do so requires we change our focus from evaluating the effect of syntax on achieving coding challenges to one centered on evaluating students' development of quantitative and algorithmic thinking as they progress through courses (Guzman et al., 2019). The current research is a springboard for our next study: testing how different courses and instructors influence students' development into effective scientific thinkers confident in their ability to find, display, and tell the biological story in the data.

Acknowledgements

We thank the graduate teaching assistants, the students, and learning assistants for agreeing to participate. We also thank two anonymous reviewers for their thoughy and diligent reviews.

References

Buitrago Flórez, F., Casallas, R., Hernández, M., Reyes, A., Restrepo, S., & Danies, G. (2017). Changing a generation's way of thinking: Teaching computational thinking through programming. *Review of Educational Research, 87*, 834-860. https://doi.org/10.3102/0034654317710096

Busch, T. (1995) Gender differences in self-efficacy and attitudes towards computers. *Journal of Educational Computing Research, 12*, 147-158. https://doi.org/10.2190/H7E1-XMM7-GU9B-3HWR

Cai, Z., Fan, X., & Du, J. (2017). Gender and attitudes toward technology use: A meta-analysis. *Computers & Education, 105*, 1-13. https://doi.org/10.1016/j.compedu.2016.11.003

Chen, M. (1986). Gender and computers: The beneficial effects of experience on attitudes. *Journal of Educational Computing Research, 2*(3), 265-282. https://doi.org/10.2190/WDRY-9K0F-VCP6-JCCD

Connolly, C., Murphy, E., & Moore, S. (2009). Programming anxiety amongst computing students - A key in the retention debate?. *Institute of Electrical and Electronics Engineers - Transactions on Education, 52*, 52-56. https://doi.org/10.1109/TE.2008.917193

Dowle, M., & Srinivasan, A. (2019). data.table: Extension of `data.frame`. R package version 1.12.2. https://CRAN.R-project.org/package=data.table.

Feser, J., Vasaly, H., & Herrera, J. (2017). On the edge of mathematics and biology integration: Improving quantitative skills in undergraduate biology education. *CBE-Life Sciences Education, 12*, 124-128. https://doi.org/10.1187/cbe.13-03-0057

Fox, J., & Anderson, R. (2005). Using the R statistical computing environment to teach social statistics courses. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.308.13&rep=rep1&type=pdf.

Gillies, R. M. (2003). Structuring cooperative group work in classrooms. *International Journal of Educational Research, 39*, 35-49. https://doi.org/10.1016/S0883-0355(03)00072-7

Guzman, L. M., Pennell, M. W., Nikelski, E., & Srivastava, D. S. (2019). Successful integration of data science in undergraduate biostatistics courses using cognitive load theory. *CBE—Life Sciences Education, 18*(4), ar49. https://doi.org/10.1187/cbe.19-02-0041.

He, J., & Freeman, L. A. (2019). Are men more technology-oriented than women? The role of gender on the development of general computer self-efficacy of college students. *Journal of Information Systems Education, 21*(2), 203-212. http://jise.org/Volume21/n2/JISEv21n2p203.pdf

Heppler, J. (2018). Teaching the tidyverse to R novices. https://medium.com/@jaheppler/teaching-the-tidyverse-to-r-novices-7747e8ce14e

Hoar, R. (2014). Generally educated in the 21st century: The importance of computer literacy in an undergraduate curriculum. *Canadian Conference on Computing Education*. https://doi.org/10.1145/2597959.2597964

Holdren, J. P., Lander, E., & the president's council of advisors on science and technology. (2012). Engage to excel: Producing one million additional college graduates with degrees in science, technology, engineering, and mathematics. Report to the President. https://obamawhitehouse.archives.gov/sites/default/files/microsites/ostp/pcast-engage-to-excel-final_2-25-12.pdf

Huber, W. Carey, V.J. , Gentleman, R., et al. (2015). Orchestrating high-throughput genomic analysis with Bioconductor. *Nature Methods, 12*, 115. https://doi.org/10.1038/nmeth.3252

Hurlbert, S. H. (1984) Pseudoreplicatation and the design of ecological field experiments. *Ecological Monographs, 54*, 187-211. https://doi.org/10.2307/1942661

Inzlicht, M., & Ben-Zeev, T. (2000) A threatening intellectual environment: Why females are susceptible to experiencing problem-solving deficits in the presence of males. *Psychological Science, 11*, 365-371. https://journals.sagepub.com/doi/pdf/10.1111/1467-9280.00272?casa_token=8mFtCtDAxs4AAAAA:7p4YPmgjdwq9yrKyNI4NyxaG7p1wCylQqT4RLpXJ6_rE7vXZw7I-FGCGdKq_Ee2C_tkQ59Tt_S0n

Kang, S. H. (2016). Spaced repetition promotes efficient and effective learning: Policy implications for instruction. *Policy Insights from the Behavioral and Brain Sciences, 3*(1), 12-19. https://doi.org/10.1177/2372732215624708

Kaplan, D., & Pruim, R. (2020). ggformula: Formula interface to the grammar of graphics. R package version 0.9.4. https://CRAN.R-project.org/package=ggformula

Kirschner, P. A. (2002). Cognitive load theory: Implications of cognitive load theory on the design of learning. *Learning and Instruction, 12*, 1-10. https://doi.org/10.1016/S0959-4752(01)00014-7

Lowndes, J. S. S., Best, B. D., Scarborough, C., Afflerbach, J. C., Frazier, M. R., O'Hara, C. C., …, & Halpern, B. S. (2017). Our path to better science in less time using open data science tools. *Nature Ecology & Evolution, 1*(6), 0160. https://doi.org/10.1038/s41559-017-0160

Marbach-Ad, G., Hunt, C., & Thompson, K. V. (2019). Exploring the values undergraduates students attribute to

cross-disciplinary skills needed for the workplace: An analysis of five STEM disciplines. *Journal of Science Education and Technology, 28*, 452-469. https://doi.org/10.1007/s10956-019-09778-8

Matloff, N. (2019). An opinionated view of the tidyverse "dialect" of the R language. https://github.com/matloff/TidyverseSkeptic

Mayer, R. E. (1981). The psychology of how novices learn computer programming. *Association for Computing Machinery - Computing Surveys, 13*(1), 121-141. https://doi.org/10.1145/356835.356841

Muenchen, R. A. (2014). Why R is hard to learn. http://r4stats.com/articles/why-r-is-hard-to-learn/

Oksanen, J., Blanchet, F. G., Friendly, M., Kindt, R., Legendre, P., …, & Wagner, H. (2019). vegan: Community ecology package. R package version 2.5-5. https://CRAN.R-project.org/package=vegan

Onwuegbuzie, A. J., & Wilson, V. A. (2003). Statistics anxiety: Nature, etiology, antecedents, effects, and treatments—a comprehensive review of the literature. *Teaching in Higher Education, 8*(2), 195-209. https://doi.org/10.1080/1356251032000052447

Pruim, R., Kaplan, D. T., & Horton, N. J. (2017). The mosaic package: Helping students to 'think with data' using R. *The R Journal, 9*(1), 77-102. https://doi.org/10.32614/RJ-2017-024

R Core Team. (2018). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. https://www.R-project.org/

R Studio Community page. (2020). https://community.rstudio.com/t/base-r-and-the-tidyverse/2965

Robinson, D. (2017). Teach the tidyverse to beginners. http://varianceexplained.org/r/teach-tidyverse/

Ross, Z, Wickham, H. & Robinson, D. (2017). Declutter your R workflow with tidy tools. *PeerJ*. https://doi.org/10.7287/peerj.preprints.3180v1

Saeli, M., Perrenet, J., Jochems, W. M. G., & Zwaneveld, B. (2011). Teaching programming in secondary school: A pedagogical content knowledge perspective. *Informatics in Education, 10*, 73–88. https://doi.org/10.15388/infedu.2011.06

Sax, L. J. (2012). Examining the underrepresentation of women in STEM fields: Early findings from the field of computer science. UCLA Center for the Study of Women. https://escholarship.org/uc/item/84j9s1v1

Speier, C., Morris, M. G., & Briggs, C. M. (1995, August). Attitudes toward computers: The impact on performance. In Association for Information Systems Americas Conference held in Indianapolis, Indiana (pp. 10-15). https://aisel.aisnet.org/amcis1995/43

Tanner, K. D. (2017). Structure matters: Twenty-one teaching strategies to promote student engagement and cultivate classroom equity. *CBE-Life Sciences Education, 12*, 322-331. https://doi.org/10.1187/cbe.13-06-0115

Trujillo, G., & Tanner, K. D. (2017). Considering the role of affect in learning: Monitoring students' self-efficacy, sense of belonging, and science identity. *CBE-Life Sciences Education, 13*, 6-15. https://doi.org/10.1187/cbe.13-12-0241

Whitney, T., Gammal, D., Gee, B., Mahoney, J., & Simard, C. (2013). Priming the pipeline: Addressing gender-based barriers in computing. *Computer, 46*(3), 30-36. https://doi.org/10.1109/MC.2013.40

Wickham, H., & Grolemund, G. (2016). *R for data science: Import, tidy, transform, visualize, and model data*. O'Reilly Media, Inc. https://batrachos.com/sites/default/files/pictures/Books/Wickham_Grolemund_2017_R%20for%20Data%20Science.pdf

Wickham, H., Averick, M., …, & 21 authors. (2019). Welcome to the tidyverse. *The Journal of Open Source Software, 4*(43), 1686. https://doi.org/10.21105/joss.01686