A Randomized Controlled Trial on the Wild Wild West of Scientific Computing with Student Learners

Timothy Rafalski
rafalski@unlv.nevada.edu
University of Nevada, Las Vegas

P. Merlin Uesbeck
uesbeck@unlv.nevada.edu
University of Nevada, Las Vegas

Cristina Panks-Meloney
panksc1@unlv.nevada.edu
University of Nevada, Las Vegas

Patrick Daleiden
patrick.daleiden@unlv.edu
University of Nevada, Las Vegas

William Allee
william.allee@unlv.edu
University of Nevada, Las Vegas

Amelia Mcnamara
amelia.mcnamara@stthomas.edu
University of St Thomas

Andreas Stefik
stefika@gmail.com
University of Nevada, Las Vegas

ABSTRACT

Scientific computing has become an area of growing importance. Across fields such as biology, education, physics, or others, people are increasingly using scientific computing to model and understand the world around them. Despite the clear need, almost no systematic analysis has been conducted on how students in fields outside of computer science learn to program in the context of scientific computing. Given that many fields do not explicitly teach much programming to their students, they may have to learn this important skill on their own. To help, using rigorous quantitative and qualitative methods, we looked at the process 154 students followed in the context of a randomized controlled trial on alternative styles of programming that can be used in R. Our results suggest that the barriers students face in scientific computing are non-trivial and this work has two core implications: 1) students learning scientific computing on their own struggle significantly in many different ways, even if they have had prior programming training, and 2) the design of the current generation of scientific computing feels like the wild-wild west and the designs can be improved in ways we will enumerate.

CCS CONCEPTS

• **Mathematics of computing** → **Statistical software**; • **Applied computing** → **Education**; • **Human-centered computing** → *Empirical studies in HCI*; • **Software and its engineering** → *Domain specific languages.*

KEYWORDS

statistics education, programming languages, scientific computing

ICER '19, August 12–14, 2019, Toronto, ON, Canada

ACM ISBN 978-1-4503-6185-9/19/08…$15.00
https://doi.org/10.1145/3291279.3339421

ACM Reference Format:
Timothy Rafalski, P. Merlin Uesbeck, Cristina Panks-Meloney, Patrick Daleiden, William Allee, Amelia Mcnamara, and Andreas Stefik. 2019. A Randomized Controlled Trial on the Wild Wild West of Scientific Computing with Student Learners. In *International Computing Education Research Conference (ICER '19), August 12–14, 2019, Toronto, ON, Canada.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3291279.3339421

1 INTRODUCTION

Academic conferences such as ICER, like academic venues in many fields, often use empirical data, statistics, and evidence-based reasoning as part of the typical publication process. Outside of academia, politicians have encouraged the use of evidence as part of the grant process. For example, the Every Student Succeeds Act (ESSA [26]) in the United States requires that the Department of Education uses tiers of evidence in its decision-making process. A "Tier 1" study under ESSA has a variety of requirements (e.g., must be a true randomized controlled trial, have over 350 participants, be multi-site), as interpreted by the the What Works Clearinghouse [46]. Other fields, like medicine, have different standards of evidence, like the CONsolidated Standards Of Reporting Trials (CONSORT [16]). For these and many other important reasons, educational programs are investing into scientific computing [11].

One interesting characteristic of scientific computing is that the programming languages often used are made by computer scientists, but the people using them may not be. As an example, consider a medical doctor publishing in any of the more than 600 biomedical journals following the CONSORT evidence standard. In these journals, they must follow specific methodologies and report the results following very specific standards [16]. This means they, or a lab associate, must learn scientific computing tools like SPSS, R, or Python, or hire this portion of their research out. While learning these languages and their tools is clearly evident, we know of no systematic empirical investigation into how much training people receive, the problems they face, or strategies our community can take to help scientists in other fields gain and improve these skills.

To help begin carving out this space, we conducted an empirical investigation into how students program in the context of scientific computing. To inform our study, we first looked at relevant data from the field, like the survey of Stack Overflow developers from

2017. In it, it was reported that people working in the context of statistics or mathematics make up about 11.3% of users. The same survey reports that, broadly, 45% of the people they surveyed used Python, 11.2% used R, and 8.2% used Matlab [39], which suggests that studying Python or R users would be a good first step. While either would be interesting to investigate, given that only 21% of Python users work in scientific computing and engineering [32], we chose to investigate how people use alternative packages in R [33]. We investigated four key research questions:

- **RQ1:** Do student programmers or non-programmers have differences in mean completion times, measured in seconds, when solving tasks in either the tidyverse, base R, or using the tilde style?
- **RQ2:** What errors do student programmers or non-programmers commonly make when solving tasks in R using the three alternative styles mentioned in RQ1?
- **RQ3:** Does previous experience learning programming in an undergraduate computer science sequence impact a student's ability to successfully solve tasks in R?
- **RQ4:** Given that students in education, biology, or other fields may have no training in computer science, what changes could be made to R to make the language less error prone or easier to use?

We then designed a 10-task experiment and recruited from in and out of the computer science department (recruitment N = 206). Our analytics platform tracked users every 5 seconds, providing 217,260 snapshots of student code. We then used automated tools like the R interpreter to look at errors in these snapshots and then separately hand-coded snapshots according to a set of qualitative categories of misconceptions, which we derived and validated.

R as a language is supported by many additional libraries made available on the Comprehensive R Archive Network [33]. The archive does not restrict style and three primary ones have emerged in the community [24]. While there is consensus one style should be taught, there is debate about which [34, 35]. The evidence we gathered here does not support that style had an effect on students when first learning on their own (RQ1). Second, in terms of errors, the barriers students face when learning on their own are not trivial (RQ2). In looking carefully at student snapshots, we believe the most likely explanation is not an individual issue, but a cornucopia that we will outline. Third, we found that experience in formal programming classes does help reduce errors, but accounts for only about 3.4% of the variance in regard to time or number of errors received (RQ3). This means experience matters less than we expected going into the study. Finally, we suspect that many of the ways students are struggling are byproducts of a wild wild west approach to design (RQ4) that is used by many languages, including R. We suspect that working toward greater overarching organization, and re-thinking naming conventions of all of the libraries and functions, would help.

The rest of this paper is as follows. First, our research team is interested in the concept of evidence standards, as used in other fields. Thus, we report this randomized controlled trial in an adapted form of CONSORT format [16], which was designed as a standard for reporting randomized controlled trials. This includes first discussing the background for this paper, then following an explicit and externally vetted approach to reporting our methodology. Finally, we present our results, discuss them, and conclude.

2 BACKGROUND AND RELATED WORK

Although there is still relatively little evidence-based research in computer programming [20], there have been an increasing number of trials and studies examining programmer behavior. This includes studies from type systems [12, 14], inheritance [30], lambdas [45], concurrency [27, 36], and compiler errors with and without enhancements [3, 6]. With the exception of a few studies examining younger students learning experiences with programs on block-based languages [15, 19, 47] and some on novice programmers in their first experiences [40, 41], many of these studies have examined undergraduate computer science students as test subjects.

As evidence-based research develops in the field of computer science and becomes institutionally required by funding agencies like the Department of Education [26], it is important for the community to agree on common evidence standards, just as the CONSORT [16] community and others have done. The CONSORT guidelines provide concrete and specific methods for reporting studies and results, which we have followed in this paper as closely as possible. Ideally, we would have preferred to follow CONSORT reporting exactly, but this is not possible at ICER. To complete the requirement requires two-phase peer review, which ICER, and to our knowledge all other computer science conferences, do not follow. This is because an important component of CONSORT requires trial registration and peer review of experimental design and methods prior to running a study, then again after it is complete. Notwithstanding our inability to comply fully with CONSORT, we followed it quite closely.

A number of papers have been written about how novice programmers approach the problem of learning and executing the logic of programs, including papers on human cognitive models [5, 29] and learning barriers [5, 44]. Other papers have emphasized the importance of statistical education research taught in tandem with computational tools using programming languages like R [9, 21, 25, 31]. Other research relevant to documenting novice programmers experiences include examinations of application programming interfaces (APIs) [43], integrated development environments (IDEs) [28], programming language pedagogy [44], compiler errors and messages [4, 10], and error types and frequencies [7, 23, 37].

R and Python are the two main languages in use today for scientific computing, according to a recent Stack Overflow survey [39]. The online learning environment for Python has allowed some researchers to analyze novice programmer errors such as syntax [22] and error frequencies [38]. R is a common first language choice for programmers to perform data analysis [1, 17, 39]. The R community has come to consensus that teaching one programming style is easier for novices. However, there is debate over which style is easiest to learn [2, 8, 35]. To our knowledge, there are no rigorous investigations into these or other issues, although some qualitative research has taken place [18]. Tools such as RStudio are designed to make learning R easier, although we are unaware of formal empirical data on such claims [17, 18, 48]. To be clear, our team is independent from the R community. As such, while we are

interested in investigating the claims of others, we have no vested interest in the outcome of our study.

3 METHODS

In this section, we describe the core methods of our approach. Again, we are are following CONSORT style reporting for trials.

3.1 Trial Design

We conducted a double-blind repeated measures randomized controlled trial with ten tasks on using the programming language R. We had two fixed-factors: 1) style of using R, and 2) major of study. For style of using R, we included the three discussed previously in our research questions: 1) the tidyverse, 2) base R, and 3) tilde style. Our intended allocation ratio for our primary factor was 1:1:1, or an equal number across groups. Because there are more non-computer science than computer science students on our campus, we expected to have a larger number for our second factor in order to recruit across a variety of majors of study. Our study was conducted in the Fall semester of 2018 and was approved by the UNLV IRB Ethics Board under protocol 1326857.

3.2 Participants

We recruited from lower level classes in our computer science program that used the imperative language C++, as well as across other majors at our university. Recruitment was conducted by giving short speeches in front of classes during class time or via a university research participation system. Participants were incentivized through extra class credit or research class credit. Participants were eligible to participate only if they had completed a statistics course in high-school or later. All participants were provided with informed consent before entering the study.

3.3 Interventions

We recorded participants completing tasks using a web-based analytics system, custom designed to track data for this experiment. For each of the tasks, participants were given a sample data table and an instruction set. This included a function name to call, the column names used in a calculation, and the number of lines of code that were required for a successful solution. They were also given a code sample guide specific to their assigned style that provided an example of how to complete a similar task. The guide was available to students throughout the experiment for reference. What students were trying to solve was standardized across experimental groups, varied only in the R style in which they were solving it.

Every five seconds, a snapshot of each student's code was submitted to our database. Each task had a 20 minute limit, after which the system registered a timeout event, ended the task, and prompted the participant to the next one. These 20 minute timeouts could plausibly lead to ceiling effects, but were necessary for two reasons. First, IRBs consider it unethical for experiments to be too long, so as scholars we must balance our desire to be experimentally thorough with ethical practice. Second, if a student cannot solve the task, there is no reason for them to continue in perpetuity.

3.3.1 Tasks. Developing tasks is a difficult part of designing any experiment. For ours, we first analyzed both online resources for R and the textbook by Field et al [13] to get a sense of the types of tasks relevant in data science. Many books and resources could have been chosen, but the one by Field et al., like many other books, contains a variety of common analysis procedures that data scientists might do. Next, we drafted a set of tasks and had them reviewed by a statistician with expertise in R. We did this as a check and balance on what we chose to test and why. We made a variety of changes to our tasks during this process, including the decision to test our three alternative designs in RQ1.

Next, because we were studying students, not professionals, we did extensive pilot testing. This included three pilot tests in the Spring of 2018. In each, we revised tasks to make sure they were not too easy or too hard for our group, adjusted the instructions, and interviewed our participants to learn what we should change.

We discuss the iterative process of task design to underscore that it is always easy to dismiss tasks as not being the "right" ones, no matter what was chosen. This is also partially justified, because different tasks can lead to different conclusions. That said, no formal RCTs exist in the literature that can both provide evidence on our research questions and that use replication packets with reported effect sizes. As such, while we think the field will need to test many other things, our tasks 1) were carefully selected, 2) match the kinds of tasks we see referenced in textbooks and online resources, 3) have gone through expert feedback, 4) were vetted in three independent pilot studies. Figure 1 provides high level descriptions of the tasks and the full values are in our replication packet, listed at the end of this paper.

Num	Task
1	Calculate mean of a column
2	Calculate standard deviation of a column
3	Calculate sum of a column
4	Calculate mean with a built-in function
5	Calculate standard deviation with built-in function
6	Calculate sum with built-in function
7	Filter data from column
8	Filter using if else statement
9	Calculate new column using division
10	Calculate new column with subtraction

Figure 1: Description of the Tasks

Students with little to no programming experience can only solve simple tasks without hitting ceiling effects. Thus, our tasks had "behind the scenes" R code, which we concatenated on to the students' answers on our server. Students would then solve one small, but key, piece of a task and we could observe snapshots for what they ended up with and the process they used. An example task is in figure 2.

Once submitted, a task's solution code was concatenated to the file and the code was run with R version 3.5.0 using the Rscript command. The results from the interpreter were displayed on the screen and recorded as an event in the database. These "events" were recorded separately from our snapshots, so that we could tell what users explicitly requested.

```
The tilde symbol(~) is used to represent a relationship
between column variables. The format is
'dependent variable ~ independent variable.
It is also acceptable to use the tilde when no dependent
variable is indicated. Here are some code examples
that show the use of the tilde. The first code is an
example that uses the method 'min', the table 'soccerLeague',
and the column variable 'goals'. The second is an example
that plots the relationship of the independent variable
'player' and the dependent variable 'goals'
from the table 'soccerLeague'.

min(~goals, data=soccerLeague)
bwplot(goals~player, data=soccerLeague)
```

(a) Sample of Tilde guide page

```
# For each task there exists a table called
americanStats that looks like this:

# playerID  yearID  teamID  lgID  G  AB  R  H  HR  RBI  SB
#1 abreujo02  2017  CHA  AL  156  621  95  189  33  102  3
#2 altuvjo01  2017  HOU  AL  153  590  112  204  24  81  32
#3 anderti01  2017  CHA  AL  146  587  72  151  17  56  15
#4 andruel01  2017  TEX  AL  158  643  100  191  20  88  25
#5 aokino01  2017  HOU  AL  71  202  28  55  219  5
#6 barneda01  2017  TOR  AL  129  336  34  78  6  25  7
#7 bautijo02  2017  TOR  AL  157  587  92  119  23  65  6
#8 beninan01  2017  BOS  AL  151  573  84  155  20  90  20

# Using the sample section on the left as a guide, you are to write a line
# of R code that will use the sd() method to calculate the standard deviation
# of the column variable HR from the americanStats table.

# Type your code here
```

(b) Sample of task page

Figure 2: Guide and Task pages

3.4 Outcomes

We have a composite primary outcome. First, we investigated how long it took for our participants to solve tasks correctly. Our analytics server ran participants' code, allowing them to move on if they received a correct answer. Second, we tracked errors at every snapshot, similar to the way errors are tracked on the Blackbox project [7], in the work of Becker [4], or other more recent examples [22]. Similarly, we looked at snapshots qualitatively to determine what students actually did wrong, as compared to what the interpreter output as an error.

As a secondary outcome, we were curious how student demographics impacted the results. Namely, we hypothesized that formal training in computer science could give students an advantage. We tracked this information through a self-reported survey, which is included in our replication packet.

3.5 Sample Size

Power test results would be preferred to determine a reasonable sample size for running an experiment, but we are aware of no Randomized Controlled Trial in the literature that has explicitly evaluated research questions like ours. Thus, we can only provide the sample sizes we targeted, but cannot report an effect size until after the study is conducted. We targeted approximately 200 people, with approximately three quarters of them from outside of computer science. We achieved our goal, recruiting 206 participants. While participants could leave at any time, no outside stopping conditions were needed ahead of time.

3.6 Randomization

A covariate adaptive randomization approach from Suresh [42] was used. Participants were first binned into an experience category based on their college year. Experience measures we have examined suggest that college year and experience are correlated. Random group assignment was done after participants consented. A participant was randomly assigned to one of three language styles, the next participant was randomly assigned to one of the remaining two, and finally the last style was assigned. This assured that the groups were randomly assigned and balanced automatically by the computer.

3.7 Blinding

We argue our study was double blind, but there are limitations to our approach. It is important to realize that any study on programming requires that a participant can observe the code they are writing. This means that our participants could have plausibly known that they were solving tasks in R, if they happened to know the language. Further, it is also possible that they could have known they were writing code in a particular style (e.g., base R). However, follow-up questions with our participants suggest they did not know and, more crucially, it would be impossible to blind in totality for this kind of study. In a sense, our participants were blind by ignorance, but not by observation. As experimenters, we were blinded by the fact that a computer program controlled our group assignments. This approach is not perfect blinding, but we believe this to be about as close as is realistically feasible.

3.8 Statistical Methods

We took several approaches for analyzing our results. First, we used standard parametric statistics to analyze time, measured in seconds, on task, following the recommendations of Field et al. [13] for a repeated measures design and using partial-eta squared as a normed effect size for RQ1. Before we ran them, we checked normal assumptions, as is also recommended by Field et al. Second, for the compiler errors output by the R interpreter, we took a similar approach, although we also present descriptive statistics visually in a graph. Third, we analyzed student code snapshots, by hand, qualitatively. We did this because we wanted to see if we could find any theoretical explanation for why students performed as they did. To evaluate our qualitative coding, we conducted a standard Kappa Analysis, which we will discuss in the results.

3.9 Additional Analysis

3.9.1 Qualitative Error Category Discovery. While documenting what errors the R interpreter reports is interesting, the messages R outputs can be opaque. Thus, we investigated the mismatch between what R outputs and the mistakes students seemed to actually make. To do this, we analyzed a small selection of student snapshots to see what kinds of mistakes they were making as a first pass. We describe our coding specification here.

3.9.2 Qualitative Error Categories. Our broad categories for the errors students made are summarized in Table 1. The reader should note that in our analysis, one might assume students made a wide variety of errors. On inspection, however, the vast majority of

```
1  min(~teamID, data=playerID)
2  playerId$teamID
3  transform(data=#$G*#$AB*#$R*#$H*#$HR*#$RBI*#$SB
4  is=min(#1)171.28
5  is=min(#2)170.85
6  is=min(#3)149.14
7  is=min(#4)175
8  is=min(#5)54.57
9  is=min(#6)87.85
10 is=min(#7)1043
11 is=min(#8)156.14
```

Figure 3: This is an example of a real error from a participant's code snapshot.

problems could be categorized into a few simple problems. This might be because of the constrained tasks we had developed for R, but it might also be a function of how students naturally try to use R at first.

In either case, the first error type we called "Method or variable name from sample page (**SP**)." We observed that some students relied too heavily on the example code, copying more liberally than would have been sensible to solve the task. For example, some participants would leave one or more of the variable names used in the example code untouched. An example of this type of error is shown in figure 3 in line 1 of the participant's submission. This uses the min() method, shown on the sample tilde guide page, rather than the necessary mean() method requested in the instruction for this task. This observation suggests that there is a misunderstanding in the use of method and variable names.

A second type of error was incorrect usage of variables. This type of error was categorized as "Incorrect variable or variable in the wrong place (**IV**)." An example in this category was when participants selected an incorrect replacement variable name from the data table when constructing their submissions. Other examples included incorrect replacement of a variable from the sample code with another variable, placement of a variable in the wrong place, and a misspelling of a variable name. Figure 3 shows two examples of an (**IV**) category error on line 1. In this case, it falls in this category because it uses the incorrect independent column variable name (teamID rather than the necessary RBI) and because it uses an incorrect table name (playerID rather than the necessary americanStats). This observation suggests that there is a misunderstanding in the placement of names within a function call.

A third type of error that was observed we categorized as "extra characters (**EC**)." These errors included characters like extra commas, misused pound signs, parenthesis, brackets, or the equal sign. An example error is shown in figure 3 on line 3. It uses several unnecessary characters showing that the participant was unsure of the operation that each character is responsible for (e.g. the multiple # characters which are actually "commenting out" anything that follows the # character in that line of code). This observation suggests that participants were not familiar with what characters could be used in R, or what those characters meant.

The fourth type of error observed was the use of unnecessary lines of code within the participant's submission. This error was categorized as "extra lines of code (**EL**)." The guide's example code occasionally provided more than one example of how the method being introduced could be used in various situations as they pertained to the tasks. An example of this error was that participants attempted to construct their submission using two lines of code rather than a single one. Again, figure 3 is used as an example, which contains a participant submission made up of eleven lines of code, when only one was requested. This observation suggests that participants were not familiar with how documentation with examples should be used to implement code.

A final type of error observed was when participants used the numbers from the table rather than using the instructed function. This error was categorized as "raw numerical data (**RD**)." Figure 3 shows an example of this on lines 4 through 11, in which the participant self-calculated the mean for each row of the table and attempted to use the self-calculated values in their submission. This observation suggests that a participant had no knowledge of how method abstraction is implemented in programming languages.

Table 1: Qualitative Error Categories

Error	Description
SP	Method or variable name from sample page
IV	Incorrect variable or variable in the wrong place
EC	Extra characters
EL	Extra lines of code
RD	Raw numerical data

3.9.3 R Interpreter Messages. As part of the experiment, we recorded all attempts to run code that were made when participants were trying to solve the programming tasks. To take a closer look at all diagnostic errors made in the attempts to solve the tasks, we collected all unsuccessful attempts to run the programs. Each of these events was represented as a state of the program the participant had submitted, as well as the error message that was displayed. From these error messages we derived a short descriptive name of these errors to be able to more easily aggregate the events into error statistics. This resulted in a total of 44 different diagnostic error categories.

4 RESULTS

4.1 Participant Flow, Losses, and Exclusions

In total, there were 206 student participants recruited for our trial, of which 41 were computer scientists, 6 were undecided, and the rest came from other fields that might use scientific computing. All of the students that had taken programming were recruited intentionally from the computer science department because we knew what kind of training they received. For students outside of computer science, we recruited widely, with few exceptions, from fields that were relevant to scientific computing. The students were selected from courses in various disciplines at the University of Nevada, Las Vegas.

Of the 206 original participants, a number of the results were omitted from our analysis for the following reasons: (i) 19 for invalid group assignments due to a software bug, (ii) 5 for having over 2+ years of programming experience, which we considered too much experience to be regarded as a novice, and (iii) 28 for terminating the study prior to finishing at least task 6. The remaining participant

count was 154, with 55 in the base R group, 52 in the tilde group, and 47 in the tidyverse group.

The instruction guide for tasks 3 and 7 were found to be ambiguous to the participants so we removed these tasks from the study before any analysis of the data was performed.

4.2 Recruitment

The web based testing system was available online at the beginning of November 2018 for approximately a month.

4.3 Baseline Data

Table 2 shows the demographics of our study. It includes their distribution across major, level of education, gender, spoken language, programming experience, and academic year.

4.4 Numbers Analyzed

4.4.1 Time. Overall we analyzed 1,232 different task times, measured in seconds, each the outcome of one participant working on one task. Of these, 440 (35.7%) events were in group base R, 416 (33.8%) were in group tilde, and 376 (30.5%) were in the tidyverse group. Broken down by major, 328 (26.6%) events belong to CS, while 904 (73.4%) belong to other majors. The tidyverse group had the highest mean time (M=657, SD=534), with the tilde style having the second highest mean time (M=621, SD=519), leaving the base R group with the lowest mean time (M=537, SD=508). Comparing the majors, computer science had a lower mean task time (M=439, SD=488) than the other majors (M=661, SD=522).

4.4.2 Qualitative Validation. Once we had derived the errors we thought people made, we conducted a Kappa Analysis of the snapshots. Notably, we had two independent reviewers code a randomly selected 10% of our sample snapshots. Once complete, we then conducted a Cohen's Kappa, receiving a score of 0.7674 and a raw agreement of 92.83%. To remind the reader, Kappa ranges between 0 and 1 and is a chance-corrected measure of reliability between two reviewers. A score of 0.7674 would be generally high agreement. We determined that the Kappa score was sufficient, so reviewers then independently coded the remaining snapshots. In points of coding disagreement, the reviewers reviewed the differences, revised the coding specification, and agreed on a final coding.

4.4.3 Qualitative Analysis. We performed qualitative error analysis as described in subsection 3.9.1. From the 1,232 events we had to exclude each instance where qualitative assessment was not possible (e.g., the snapshot was blank) and after these exclusions, there were 1,112 events. Of these, 403 (36.2%) were in group base R, 365 (32.8%) in tilde, and 344 (30.9%) in tidyverse. There were 297 (26.7%) of these events associated with participants in CS majors and 815 (73.3%) with the other majors. The results are summarized in Figure 4, which shows the number of errors in relation to the total possible errors for each of the 5 error categories.

There was a possibility that each of the 5 errors could be found in each task (for an example see figure 3), so the following percentages reflect in how many of the tasks a specific error was found and they do not add up to 100%. We found that 468 of the events contained the error **IV**, which means that we found this error in 42.1% of all tasks. Further, we found that 25% of all possible instances contained the error **EC** with 278 instances and error **EL** was found 130 times which means that 11.7% of tasks contained this error. The **SP** error was found in 11.7% of the tasks (n=130), and lastly, the **RD** error was found in 112 events which means it was found in 10.1% of tasks.

4.4.4 R Interpreter Messages. We analyzed 2609 error events as reported by the R interpreter. 1119 (42.9%) of the errors were produced by group base R, 827 (31.7%) were produced by group tilde, and the tidyverse group produced 663 (25.4%). When looking from the perspective of major, 441 of the errors were produced by participants in the CS major (16.9%), while the 2168 (83.1%) remaining errors were produced by the participants in other majors. On average, participants produced 17 errors (M= 17, SD=22.7) over the course of the experiment. Group base R produced the most (M=20.8,

Table 2: Participant Demographics

Degree Sought	**n**
Computer Science	33
Computer Engineering	8
Engineering (Other than Computer)	7
Counseling	7
Psychology	9
Biology	3
Education	71
Hospitality	3
Accounting	2
Other Liberal Arts	11
Undecided	6
Total	154

Education	**n**
High School/GED	13
Some College / University	75
Associates Degree	19
Bachelors Degree	32
Advanced Degree	12
NA	3
Total	154

Gender	**n**
Female	95
Male	59
Total	154

Primary Language	**n**
English	120
Non-English	34
Total	154

Programming Experience (yrs)	**n**
0	105
1	36
2	13
Total	154

School Year	**n**
Freshman	12
Sophomore	43
Junior	45
Senior	11
Graduate or Above	40
None of the above	3
Total	154

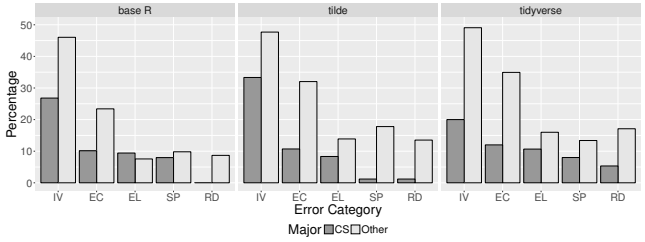

Figure 4: Percent of tasks by participant with qualitative errors by style and major. SP=Copied sample, IV=Incorrect variable, EC=Extra characters, EL=Extra lines, RD=Raw numerical data

SD=24.1) errors per person, followed by tilde (M=15.9, SD=22.2), and the tidyverse (M=13.9, SD=21.4).

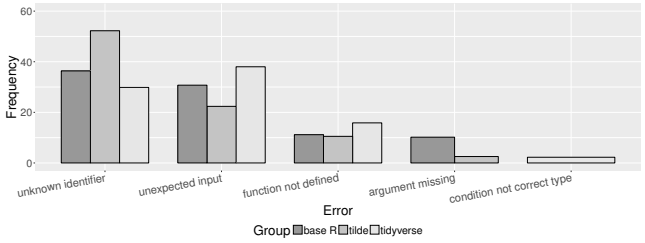

Figure 5: This figure displays the R interpreter messages output by R.

The frequency of the 4 most common errors per group can be seen in figure 5. While the 3 most common errors are shared across all 3 groups, "argument missing" was the fourth most common error for base R and tilde, while "condition not correct type" was the fourth most common error in the tidyverse group. The majority of errors that were reported seemed to be triggered by the individual R libraries used and differed depending on which libraries were involved. This made it difficult to judge the root cause of the error across groups. It also made it difficult to determine a direct mapping between the qualitative assessment of errors, made by inspecting the code by hand (as discussed in subsection 3.9.1), and the recorded error categories.

4.5 Outcomes and Estimation

In this section, we will present the inference analysis of the experiment outcomes time, qualitative errors, and quantitative errors.

4.5.1 Task Completion by Time. Figure 6 is a plot of paired boxplots, in a grid by task number and code style of the task completion. If a participant did not enter any code and still timed out on a task, an entry of the maximum time (1200 seconds) was used.

A repeated measures ANOVA of the completion time with between-subjects factors style and major, and within-subjects factor task, suggests that we observed no significant difference between styles in R $F(2, 148) = 0.74$, $p = 0.480$, $\eta_p^2 = 0.007$ at significance level $\alpha = 0.05$. The difference between majors is significant $F(1, 148) =$

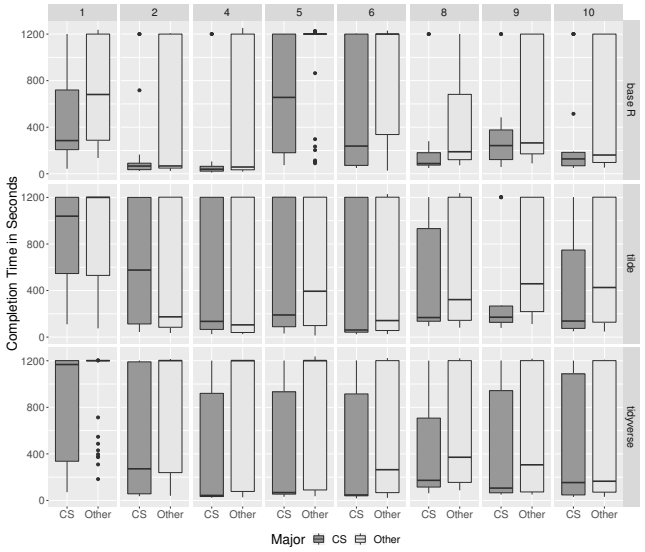

Figure 6: Comparison of Task Completion Times

7.35, $p = 0.008$, $\eta_p^2 = 0.034$, as is the difference between tasks $F(7, 1036) = 20.35$, $p < 0.001$, $\eta_p^2 = 0.038$. There is also a significant interaction effect between style and task $F(14, 1036) = 9.82$, $p < 0.001$, $\eta_p^2 = 0.037$. All numbers here and going forward are reported with Greenhouse-Geisser correction applied if they failed the sphericity test, as is standard practice.

4.5.2 Qualitative Errors. To further analyze the qualitative error assessments, we ran a repeated measures ANOVA of the number of qualitative errors, using style, type of error, and major as between-subjects factors to compare whether the error distributions are distinct. The results show that style is not significantly different $F(2, 730) = 2.642$, $p = 0.071$, $\eta_p^2 = 0.007$, while major is significant $F(2, 730) = 26.077$, $p < 0.001$, $\eta_p^2 = 0.034$. Further, the type of error is significant as well $F(4, 730) = 27.134$, $p < 0.001$, $\eta_p^2 = 0.129$. None of the interactions were significant.

4.5.3 R Interpreter Messages. A repeated measures ANOVA of the number of R interpreter errors with between-subjects factor group and major, and within subjects factor task shows that there is no significant difference between styles in R $F(2, 148) = 0.657$, $p = 0.520$, $\eta_p^2 = 0.003$. The difference between majors is significant $F(1, 148) = 4.166$, $p = 0.04$, $\eta_p^2 = 0.010$, as is the difference between tasks $F(7, 1036) = 8.424$, $p < 0.01$, $\eta_p^2 = 0.035$ and there is a significant interaction effect between task and style $F(14, 1036) = 5.033$, $p < 0.01$, $\eta_p^2 = 0.042$. The amount of errors in each task and group, split by whether the participants were CS majors can be seen in figure 7.

5 DISCUSSION

5.1 Limitations

We think there are several important limitations of our study. First, we looked at potential scientists at the beginning of their career,

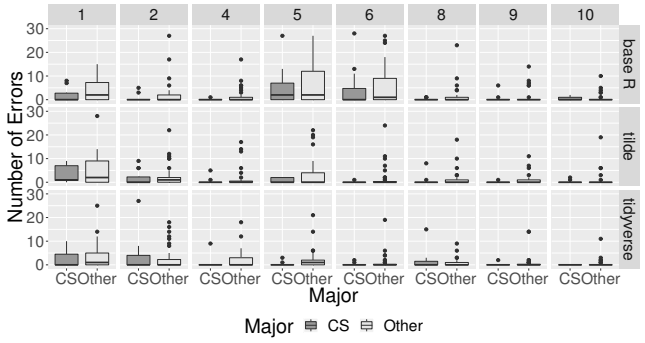

Figure 7: Number of Errors by Syntax Group and Major

learning on their own in an undergraduate setting. We suspect many learning scientific computing must learn this way, but this is not the case for everyone. Students in statistics, or other areas, may obtain training and thereby have different outcomes.

Second, we want to be careful in saying that while we found very little evidence that the alternative styles of using R measurably benefited students, this is not the same thing as saying there was no difference. While we wanted to test with novices first, we acknowledge that packages like the tidyverse have a design philosophy that differs a lot from base R and might benefit different groups in different ways. As such, experts doing different kinds of tasks may have different outcomes than our students did.

5.2 Generalizability

We have already mentioned that studying students may not be predictive of expert performance, nor would we expect it to be. However, other issues may impact how our study generalizes. Namely, the tasks that we had students complete were rather small. It is possible that larger tasks would have produced different results. However, we suspect many students in this experience range could not complete large and complex tasks at all. This is true in typical programming classes as well, which is why we often scaffold what students are learning.

5.3 Interpretation

In terms of our first research question (**RQ1**), whether styles made a difference, at N = 154, we were unable to conclude that it impacted our participants error rate or mean completion times. Qualitatively, we also found no evidence that the errors students were making were different. That said, we find it interesting that in tasks 4 and 5, we see the base R group did have significantly more errors. Given the significant interaction, more research is needed to determine how, and in what way, style in R is task dependent.

For (**RQ2**), the most common kind of error was the misnaming or misplacement of a variable name. This might indicate a lack of understanding of how different features are mapped to names in the programming language. The second most common error was extra characters. We theorize thus that syntax and naming conventions might be a significant initial barrier in R. While we did not test this formally, this might also imply that cheat sheets on special characters could be helpful as a teaching tool for the language.

For (**RQ3**), it is clear that computer science students with training in an imperative language like C++ did make fewer errors and completed tasks in less time. However, the differences were not universal and very small, which surprised us. For example, in Task 4, it took computer science students in the base R group longer to complete the tasks than our non-majors with no programming training at all. Even when computer science students outperformed the non-majors, as was normal, the effect only explained approximately 3.4% of the variance.

For (**RQ4**), we have two recommendations for the designers of R. First, error causes were not clear to our users. A good example is the use of phrases like "unexpected input," which often did not give our users enough information to solve their problems. We recommend trying to make the causes of errors more clear, following the advice of existing literature on the topic (e.g., Becker [3]).

Second, the design of some libraries in R sometimes "feels" messy and unorganized as a whole. We observed in our qualitative data that the sometimes peculiar naming conventions were a struggle for our users. We think our data provides evidence that R's users would benefit from 1) better naming conventions across the board that diminish the use of single letter variables, shortened words, and acronyms and 2) a re-thinking of the overarching organization of the statistical routines in R. For example, perhaps functions in the language could be named to imply what the function does, as opposed to honoring the mathematician that invented it. The former could be used to mentally categorize ideas in a mental model, while the latter must be raw memorized across the board. In any case, R's designers have the difficult job of trying to make the field of statistics understandable in code. Reorganizing and renaming would not fix everything, but likely would have diminished some of the challenges we observed.

6 CONCLUSION

In this paper we have presented a randomized controlled trial with 154 participants, comparing three different styles of programming in R. We found that there was no significant difference between the styles regarding completion time, qualitatively assessed errors, or R interpreter errors. We did find differences when comparing participants who had some computer science training compared to participants without. This suggests that learning programming in one language can help with learning R, but also shows the initial effect size is small. Finally, we documented that users made a variety of errors in R. We suggest that improving R error messages and re-thinking the naming conventions and organizational structure of the language could help make R easier for novices to use.

6.1 Protocol

The code and replication packet for this study can be cloned from https://bitbucket.org/stefika/replication.

6.2 Funding

This work was funded by the National Science Foundation under grants #1640131, #1644491, #1738259.

REFERENCES

[1] Ben Baumer, Mine Cetinkaya-Rundel, Andrew Bray, Linda Loi, and Nicholas J. Horton. 2014. R Markdown: Integrating A Reproducible Analysis Tool into Introductory Statistics. *arXiv:1402.1894 [stat]* (Feb. 2014). arXiv:stat/1402.1894

[2] Benjamin Baumer, Daniel Kaplan, and Nicholas J. Horton. 2017. *Modern Data Science with R.* CRC Press,Taylor & Francis Group, CRC Press is an imprint of the Taylor & Francis Group, an informa business, Boca Raton.

[3] Brett A. Becker. 2016. An Effective Approach to Enhancing Compiler Error Messages. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education (SIGCSE '16).* ACM, New York, NY, USA, 126–131. https://doi.org/10.1145/2839509.2844584

[4] Brett A. Becker and Catherine Mooney. 2016. Categorizing Compiler Error Messages with Principal Component Analysis. In *12th China-Europe International Symposium on Software Engineering Education (CEISEE 2016), Shenyang, China, 28-29 May 2016.*

[5] A. F. Blackwell. 2002. First Steps in Programming: A Rationale for Attention Investment Models. In *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments.* 2–10. https://doi.org/10.1109/HCC.2002.1046334

[6] Neil C. C. Brown and Amjad Altadmri. 2017. Novice Java Programming Mistakes: Large-Scale Data vs. Educator Beliefs. *ACM Trans. Comput. Educ.* 17, 2, Article 7 (May 2017), 21 pages. https://doi.org/10.1145/2994154

[7] Neil Christopher Charles Brown, Michael Kölling, Davin McCall, and Ian Utting. 2014. Blackbox: A Large Scale Repository of Novice Programmers' Activity. In *Proceedings of the 45th ACM Technical Symposium on Computer Science Education (SIGCSE '14).* ACM, New York, NY, USA, 223–228. https://doi.org/10.1145/2538862.2538924

[8] Jennifer Bryan and Hadley Wickham. 2017. Data Science: A Three Ring Circus or a Big Tent? *arXiv:1712.07349 [stat]* (Dec. 2017). arXiv:stat/1712.07349

[9] Mine Çetinkaya-Rundel and Colin Rundel. 2018. Infrastructure and Tools for Teaching Computing Throughout the Statistical Curriculum. *The American Statistician* 72, 1 (Jan. 2018), 58–65. https://doi.org/10.1080/00031305.2017.1397549

[10] Paul Denny, Andrew Luxton-Reilly, and Dave Carpenter. 2014. Enhancing Syntax Error Messages Appears Ineffectual. In *Proceedings of the 2014 Conference on Innovation & Technology in Computer Science Education (ITiCSE '14).* ACM, New York, NY, USA, 273–278. https://doi.org/10.1145/2591708.2591748

[11] David Donoho. 2017. 50 Years of Data Science: Journal of Computational and Graphical Statistics: Vol 26, No 4. https://www.tandfonline.com/doi/full/10.1080/10618600.2017.1384734.

[12] Stefan Endrikat, Stefan Hanenberg, Romain Robbes, and Andreas Stefik. 2014. How do api documentation and static typing affect api usability?. In *Proceedings of the 36th International Conference on Software Engineering.* ACM, 632–642.

[13] Andy P. Field, Jeremy Miles, and Zoë Field. 2012. *Discovering statistics using R.* Sage, London; Thousand Oaks, Calif.

[14] Lars Fischer and Stefan Hanenberg. 2015. An empirical investigation of the effects of type systems and code completion on api usability using typescript and javascript in ms visual studio. In *ACM SIGPLAN Notices*, Vol. 51. ACM, 154–167.

[15] Diana Franklin, Charlotte Hill, Hilary A Dwyer, Alexandria K Hansen, Ashley Iveland, and Danielle B Harlow. 2016. Initialization in Scratch: Seeking knowledge transfer. In *Proceedings of the 47th ACM Technical Symposium on Computing Science Education.* ACM, 217–222.

[16] The CONSORT Group. [n. d.]. CONSORT: Transparent Reporting of Trials. http://www.consort-statement.org/. Accessed: 2017-10-13.

[17] Nicholas J Horton and Johanna S Hardin. 2018. Challenges and Opportunities for Statistics and Data Science Undergraduate Major and Minor Degree Programs. (2018), 6.

[18] Carl Howe. 2019. The Next Million R users. https://resources.rstudio.com/rstudio-conf-2019/the-next-million-r-users. In *rstudio::conf.*

[19] Christopher D Hundhausen, Sean F Farley, and Jonathan L Brown. 2009. Can direct manipulation lower the barriers to computer programming and promote transfer of training?: An experimental study. *ACM Transactions on Computer-Human Interaction (TOCHI)* 16, 3 (2009), 13.

[20] Kaijanaho Kaijanaho. 2015. *Evidence-based programming language design : a philosophical and methodological exploration.* University of Jyväskylä, Finnland. http://urn.fi/URN:ISBN:978-951-39-6388-0

[21] Daniel Kaplan. 2017. Teaching Stats for Data Science. *The American Statistician* 72, 1 (2017), 89–96. https://doi.org/10.1080/00031305.2017.1398107

[22] Tobias Kohn. 2019. The Error Behind The Message: Finding the Cause of Error Messages in Python. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19).* ACM, New York, NY, USA, 524–530. https://doi.org/10.1145/3287324.3287381

[23] D. McCall and M. Kölling. 2014. Meaningful Categorisation of Novice Programmer Errors. In *2014 IEEE Frontiers in Education Conference (FIE) Proceedings.* 1–8. https://doi.org/10.1109/FIE.2014.7044420

[24] Amelia McNamara. 2018. R Syntax Comparison Cheatsheet. https://github.com/rstudio/cheatsheets/raw/master/syntax.pdf.

[25] Deborah Nolan and Jamis Perrett. 2016. Teaching and Learning Data Visualization: Ideas and Assignments. *The American Statistician* 70, 3 (July 2016), 260–269. https://doi.org/10.1080/00031305.2015.1123651

[26] U.S. Department of Education. [n. d.]. Every Student Succeeds Act(ESSA). https://www.ed.gov/essa?src=rn. Accessed: 2017-03-29.

[27] Victor Pankratius and Ali-Reza Adl-Tabatabai. 2014. Software engineering with transactional memory versus locks in practice. *Theory of Computing Systems* 55, 3 (2014), 555–590.

[28] Fernando Perez and Brian E. Granger. 2007. IPython: A System for Interactive Scientific Computing. *Computing in Science & Engineering* 9, 3 (2007), 21–29. https://doi.org/10.1109/MCSE.2007.53

[29] James Prather, Raymond Pettit, Kayla McMurry, Alani Peters, John Homer, and Maxine Cohen. 2018. Metacognitive Difficulties Faced by Novice Programmers in Automated Assessment Tools. In *Proceedings of the 2018 ACM Conference on International Computing Education Research - ICER '18.* ACM Press, Espoo, Finland, 41–50. https://doi.org/10.1145/3230977.3230981

[30] Lutz Prechelt, Barbara Unger, Michael Philippsen, and Walter Tichy. 2003. A controlled experiment on inheritance depth as a cost factor for code maintenance. *Journal of Systems and Software* 65, 2 (2003), 115–126.

[31] Purdue University, Pete Pascuzzi, Megan Sapp Nelson, and Purdue University. 2018. Integrating Data Science Tools into a Graduate Level Data Management Course. *Journal of eScience Librarianship* 7, 3 (Dec. 2018), e1152. https://doi.org/10.7191/jeslib.2018.1152

[32] PyCharm. 2016. Python Developers Survey 2016:Findings. https://www.jetbrains.com/pycharm/python-developers-survey-2016/

[33] R Core Team. 2019. Comprehensive R Archive Network. http://cran.r-project.org/.

[34] David Robinson. 2014. Don't teach built-in plotting to beginners (teach ggplot2). http://varianceexplained.org/r/teach_ggplot2_to_beginners/.

[35] David Robinson. 2017. Teach the tidyverse to beginners. http://varianceexplained.org/r/teach-tidyverse/.

[36] Christopher J. Rossbach, Owen S. Hofmann, and Emmett Witchel. 2010. Is Transactional Programming Actually Easier?. In *Proceedings of the 15th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP '10).* ACM, New York, NY, USA, 47–56. https://doi.org/10.1145/1693453.1693462

[37] Tom Schorsch. 1995. *Cap: An Automated Self-Assessment Tool To Check Pascal Programs For Syntax, Logic And Style Errors.*

[38] Rebecca Smith and Scott Rixner. 2019. The Error Landscape: Characterizing the Mistakes of Novice Programmers. In *Proceedings of the 50th ACM Technical Symposium on Computer Science Education - SIGCSE '19.* ACM Press, Minneapolis, MN, USA, 538–544. https://doi.org/10.1145/3287324.3287394

[39] StackOverflow. 2017. Developer Survey Results 2017. https://insights.stackoverflow.com/survey/2017

[40] Andreas Stefik and Susanna Siebert. 2013. An Empirical Investigation into Programming Language Syntax. *ACM Transactions on Computing Education* 13, 4 (Nov. 2013), 1–40. https://doi.org/10.1145/2534973

[41] Andreas Stefik, Susanna Siebert, Melissa Stefik, and Kim Slattery. 2011. An Empirical Comparison of the Accuracy Rates of Novices Using the Quorum, Perl, and Randomo Programming Languages. In *Proceedings of the 3rd ACM SIGPLAN Workshop on Evaluation and Usability of Programming Languages and Tools - PLATEAU '11.* ACM Press, Portland, Oregon, USA, 3. https://doi.org/10.1145/2089155.2089159

[42] KP Suresh. 2011. An overview of randomization techniques: an unbiased assessment of outcome in clinical research. *Journal of human reproductive sciences* 4, 1 (2011), 8.

[43] Kyle Thayer. 2018. Using Program Analysis to Improve API Learnability. (2018), 2.

[44] Kyle Thayer and Andrew J. Ko. 2017. Barriers Faced by Coding Bootcamp Students. In *Proceedings of the 2017 ACM Conference on International Computing Education Research - ICER '17.* ACM Press, Tacoma, Washington, USA, 245–253. https://doi.org/10.1145/3105726.3106176

[45] Phillip Merlin Uesbeck, Andreas Stefik, Stefan Hanenberg, Jan Pedersen, and Patrick Daleiden. 2016. An Empirical Study on the Impact of C++ Lambdas and Programmer Experience. In *Proceedings of the 38th International Conference on Software Engineering (ICSE '16).* ACM, New York, NY, USA, 760–771. https://doi.org/10.1145/2884781.2884849

[46] U.S. Department of Education Institute of Education Sciences. 2019. *What Works Clearinghouse Procedures and Standards Handbook* (3.0 ed.). U.S. Department of Education.

[47] David Weintrop and Uri Wilensky. 2015. Using Commutative Assessments to Compare Conceptual Understanding in Blocks-based and Text-based Programs.. In *ICER*, Vol. 15. 101–110.

[48] Karlijn Willems. 2013. R and Education: A Survey on the Use of R in Education. https://www.datacamp.com/community/blog/survey-on-r-and-education.