Are Simulated Coding Interviews a Fair and Practical Examination Format for Non-professional Programmers Enrolled in a Master's Degree Program in Biostatistics?

Jesse D. Troy[1,*], Gina-Maria Pomann[1], Megan L. Neely[1], Steven C. Grambow[1] & Gregory P. Samsa[1]

[1]Department of Biostatistics and Bioinformatics, School of Medicine, Duke University, Durham, NC, USA

*Correspondence: Department of Biostatistics and Bioinformatics, Duke University School of Medicine, Durham, NC, USA. Tel: 1-919-668-2932. E-mail: jesse.troy@duke.edu

Received: July 18, 2023 Accepted: October 25, 2023 Online Published: November 15, 2023

doi:10.5430/jct.v12n6p253 URL: https://doi.org/10.5430/jct.v12n6p253

Abstract

This report describes an innovative and evidence-based approach to implementing coding interviews as an examination format for non-professional programmers: namely, students in a Master of Biostatistics program taking a course in the language of SAS. In addition to its academic purpose, the coding interview examination also serves as practice for what our students will likely encounter when interviewing for jobs after graduation. We discuss our experience with coding interviews as an examination format in light of two questions: "Is it fair?" and "Is it practical?". We propose that the answer to both questions is "yes". A detailed description of the exam goals and structure is provided, along with sample questions, model answers, and a brief discussion of the rationale for each question. We also review student feedback from the course evaluation and summarize our conclusions related to fairness and practicality.

Keywords: coding interview, evaluation, programming, SAS

1. Introduction

Our teaching context is a two-year Master of Biostatistics program focusing on the constructs of Analytical skills, Biological knowledge and Communication -- the "ABCs of biostatistics" (Neely et al., 2022; G. Samsa, 2021; G. P. Samsa, 2018; G. P. Samsa et al., 2018; Troy, Granek, et al., 2022). The first-year curriculum includes a two-course sequence on computer programming, whereby all students take an R course in the fall and either a Python or a SAS course in the spring (*R: The R Project for Statistical Computing*, n.d.; *SAS: Analytics, Artificial Intelligence and Data Management*, n.d.; *Welcome to Python.Org*, n.d.). These courses are organized around algorithms and data structures, the premise being that these constructs are fundamental to effective programming and will support life-long learning (P.J. Denning et al., 1989). For example, students who have mastered algorithms and data structures should be able to learn new languages and new applications of their current languages as needed. Moreover, mastery of algorithms and data structures is necessary to effectively utilize the vast amount of external information available to students *via* web searching, ChatGPT, etc. (*Introducing ChatGPT*, n.d.). For example, while ChatGPT can write program code based on a user's request, it may not always be efficient or error-free, requiring programming knowledge to optimize or correct.

Coding interviews are a standard part of the hiring process in software engineering and other related fields where computer programming is a core task, including biostatistics (McDowell, 2020; Nolan & Temple Lang, 2010; Pomann et al., 2020). Interviewers typically use coding interviews to assess candidate's problem-solving ability and communication skills in addition to their technical knowledge (Ford et al., 2017). The typical format is that of a discussion in which the interviewer may ask the job candidate to use a whiteboard, online coding platform, or write something on paper in response to the interviewer's questions (McDowell, 2020). Interviewers may or may not require job candidates to use specific programming languages and in some cases, candidates are invited to use the programming language of their choice or to simply describe algorithms in plain language (Ford et al., 2017). There is no standard set of questions that are asked during coding interviews but there are prototypical questions covering core topics that are commonly encountered in coding interviews, and there are well-known resources available to

help job seekers prepare for such interviews (McDowell, 2020). Thus, building practice with coding interviews into an instructional program can be beneficial for students as they graduate and begin a job search.

However, coding interviews must be carefully conducted if they are to be effective for the interviewer and the interviewee. Behroozi, et al. (2019) proposed the following guidelines for conducting a coding interview: "1) Use rudimentary questions for screening [for example, prior to inviting candidates for an in-person interview]; 2) Share the interview description in advance; 3) Offer alternative interview formats; 4) Use a real problem [as the basis for the questions]; and 5) Solve problems as colleagues, not as examiners."

While the literature provides guidance on developing coding interviews for purposes of hiring employees, we are using the coding interview as an examination format in an academic setting and only secondarily as an aid for our students' future job search. We have previously described our approach to using the coding interview as a final exam for the SAS programming course offered as part of our master's program (G. P. Samsa, 2020). Since then we have accumulated additional experience using coding interviews as a final examination and discuss that experience here in light of recurring (and quite reasonable) questions from our faculty colleagues, especially: "Is it fair?" and "Is it practical?".

To address concerns around fairness and practicality, we propose in this article methods for implementing coding interview exams that follow the guidelines recommend for hiring-focused coding interviews (Behroozi et al., 2019). Specifically, we discuss our experience with coding interviews as a final examination for our Master of Biostatistics students enrolled in the SAS programming course. This paper is organized to first outline the key elements of implementing an exam that utilizes a coding interview. We provide recommendations about how others can develop and implement similar evaluations and provide examples. Then we discuss guidelines for the student assessment and discuss the questions "Is it fair?" and "Is it practical?".

2. Implementing Coding Interviews for Student Assessment

A recurrent theme throughout this report is the importance of transparency. One application of this principle is sharing with students our goals for the final examination, and also the rationale for those goals. These goals are informed with our experience in performing actual coding interviews as part of the hiring process, and address one of the key guidelines identified by Behroozi and colleagues (2019) by providing a description of the interview in advance. Additionally, to implement coding interviews we need to have a clear plan to help students prepare and a consistent structure for the exam we can follow. The following sections provide guidelines for defining goals, exam preparation, and exam structure.

2.1 Defining Examination Goals

We explain to students that an actual coding interview would try to discover (1) general facility with programming; and (2) SAS-specific skills. General facility with programming is a combination of effective computational thinking, skill in algorithms and data structures, and the use of literate and reproducible programming practices. Computational thinking refers to "the mental skills and practices for designing computations that get computers to do jobs for us, and for explaining and interpreting the world in terms of information processes" (Peter J. Denning & Tedre, 2022). As applied to programming, computational thinking involves abstracting a problem, representing that problem using data elements (i.e., data structures), organizing and processing those data elements in a systematic fashion (algorithms), and doing so using coding practices that are efficient, human-friendly and generate identical results over time (reproducibility) (P.J. Denning et al., 1989; Knuth, 1984).

Although both SAS-specific skills and general programming knowledge are crucial, the latter is arguably more important in an academic setting as it enables lifelong learning and future success, e.g., as students are required to learn new languages in the future. In other words, general facility with programming provides the tools for rapidly learning new skills, thus providing value to an employer. Indeed, in many cases, if an applicant can demonstrate to us that they are a good programmer in general, we believe that many companies would be likely to hire them, with their SAS-specific knowledge being used to identify the type of projects for which they would be competent to contribute to on day one.

We also share with students that, for many interviewers, knowledge of specific elements of SAS syntax isn't critically important. Instead, it serves as a surrogate for familiarity with the language and a check on resume assertions that the applicant knows SAS. An applicant who is a regular SAS user will have repetitively applied various elements of SAS syntax; these elements comprise building blocks of SAS programming that can be immediately recalled, and being able to do so is a reasonable focus of an interview. Students are encouraged to reply to questions about more arcane

elements of SAS syntax with an offer to demonstrate how they would search for an answer in real time (and thus verifying for the interviewer their ability to learn on the job). Indeed, after the applicant gets a job, their current SAS-specific knowledge can be supplemented by web searching, ChatGPT, etc. In other words, if you know enough about how SAS is organized, you truly can teach yourself.

2.2 Exam Preparation

A crucial element of fairness is for students to have clear expectations of how the exam will proceed, how the exam will be graded, and how they can effectively prepare. Some elements which contributed toward student preparation included:

- Exam goals were described ahead of time (as discussed above)
- A grading rubric was circulated
- Classroom exercises provided practice in exam content (e.g., by explicit discussion of algorithms and data structures, by having students explain their code)
- Students were encouraged to add copious comments to their assignment programs, and were allowed to bring these commented programs to the interview
- Typical interview questions were circulated ahead of time
- Students observed a practice interview, with another instructor playing the role of student, after which the two parties described their thought process in framing and answering the exam questions

When queried after the conclusion of their interview, students consistently reported that the interview was essentially as they expected, and also that the sample interview questions were helpful. Asking students to add comments to their programs not only provided practice in an element of literate programming, but was intended to help them engage with the material in sufficient depth to attain mastery. Interestingly, some students reported using ChatGPT as part of their preparation. This typically worked well for taking model code provided by the instructor and explaining how the relevant SAS syntax operated (and, indeed, was far less cryptic than a typical computer manual), but didn't always work well when ChatGPT was asked to write a SAS program.

2.3 Exam Structure

All students were asked the same set of general questions. In our context, which is an educational setting rather than an actual job interview, we see this as addressing the first guideline offered by Behroozi, et al. (2019) to "use rudimentary questions for screening." Whereas in a job interview these questions could be used, for example, in a phone screening to decide who meets minimum requirements for an in-person interview, in our case we use these questions to gauge students' base level of knowledge and to select the next questions to ask.

After asking all students the same set of initial questions, a sample of questions was chosen which required using SAS to complete a programming task that students are likely to encounter in a real-world setting (Behroozi et al., 2019). Again, follow-up and clarifying questions depended on the students' initial responses. In this way, the interview becomes a simulated discussion among colleagues rather than a question-and-answer session (Behroozi et al., 2019).

3. Example Questions, Answers and Commentary

In this section we provide illustrative questions and answers, along with commentary on the rationale for each question. The back-and-forth (between interviewer and interviewee) around clarifying questions isn't illustrated. There isn't a single "right answer" to any of the questions but we include model answers that would illustrate competency. The model answers are accompanied by comments that explain the rationale for the question and why we think the model answer is appropriate. The questions are divided into two groups: 1) general questions about programming with SAS that do not require interpretation or writing of program code; and 2) detailed questions about SAS programming that required reading and interpreting or writing new program code.

3.1 General Questions About SAS

3.1.1 Question: What are some similarities and differences between SAS and R?

Answer: R is an object-oriented functional language organized around lists whereas SAS is a procedurally-based language organized around data frames. R is open-source whereas SAS isn't, an implication of which is that it requires downloading packages that might or might not work as desired. R is more flexible than SAS. Work sessions

are different: R code is immediately executed whereas for SAS you write a block of code, mark it, and then execute it. Either language can be used to perform typical data management and statistical analysis tasks.

Comment: Apart from its information value, this question also serves to differentiate between the perspectives of statistical users (e.g., "the two languages can perform similar functions such as data management and analysis") and those with formal computer science training (e.g., "R is an object-oriented functional language whereas SAS is a procedurally-based language").

3.1.2 Question: What are some techniques for performing literate and reproducible programming in SAS?

Answer: Modularize the code, include human-friendly practices such as comments, white space and indentation, write code to be generalizable rather than task-specific, have a development and testing plan, use version control, when performing simulations, set a seed for the pseudorandom number generator.

Comment: This question addresses general programming knowledge, and is absolutely critical to an actual coding interview. As a rule, we only hire applicants who can describe literate and reproducible programming practices.

3.1.3 Question: What are some techniques for testing a SAS program?

Answer: Perform unit testing by checking components of the program one at a time, trace the logic by printing intermediate files and by using PUT statements to print intermediate results of iterative calculations.

Comment: This question also addresses an important element of general programming knowledge, that being the use of an explicit strategy for program development.

3.2 Detailed Questions About the SAS Language

3.2.1 Question 1: The following code illustrates the core logic of performing a patient-level simulation. What does it do?

```
data test1;
   call streaminit(1);
   do i=1 to 10;
      rand_unif=rand("uniform",0,1);
      output;
   end;
run;

proc print data=test1;
run;
```

Answer: The DATA statement creates a SAS dataset called TEST1. The CALL statement sets the seed for the pseudorandom number generator, thus ensuring that the results will be reproducible. The DO loop creates 10 records. The RAND function creates uniform random variables on the interval from 0 to 1. The OUTPUT statement explicitly writes the record. TEST1 will have 10 observations and 2 variables. The PRINT statement prints TEST1.

Comment: As discussed subsequently, rather than asking the student to write code to perform a simulation, which is typically done in interviews, this question structure allows them to get off to a good start. The data structure is a 10x2 array. The algorithm uses a DO loop to create simulated patients, and the call to the pseudorandom number generator produces the simulated data according to the desired specifications. The indentation is a literate programming practice. Because this is part of an examination, the literate programming practices of commenting program code and titling output are not illustrated. Setting a seed for the pseudorandom number supports reproducibility.

3.2.2 Question: What happens if the OUTPUT statement is dropped from the code segment shown above?

Answer: The DO loop will still be executed 10 times, but only the final record will be printed.

Comment: This question evaluates general programming knowledge about how DO/FOR loops operate. If a student is stumped, they are encouraged to run the code without the OUTPUT statement and explain the results.

3.2.3 Question: How would I simulate a standard normal random variable instead of a uniform random variable?

Answer: Even though I haven't done so before, the answer ought to be to change "uniform" to "normal" in the RAND

function -- the precise syntax might not be precisely as assumed.

Comment: This question evaluates understanding of the basic algorithm that underpins simulations. A correct answer requires understanding that the pseudorandom number is generated by the RAND function, and that the RAND function has options which can be used to specify the desired distribution.

3.2.4 Question: Suppose that you wanted to prepare a randomization list for a randomized trial, with 40 patients, where patients are randomized to either Intervention or Control with a 1:1 ratio. For any particular list, you won't necessarily have exactly 20 patients in each group. How would you modify the basic simulation code?

Answer: Change the index on the DO loop from 10 to 40. Use IF / THEN / ELSE logic to assign random numbers which fall between 0 and 0.50 to I and the rest to C. In other words, change the code to this:

```
data test1;
   call streaminit(1);
   do i=1 to 10;
      rand_unif=rand("uniform",0,1);
      if (0 le rand_unif le 0.50) then group='I';
         else group='C';
      output;
   end;
run;
proc print data=test1;
run;
```

Comment: This illustrates a generalizable programming technique, which is essentially independent of language. In algorithmic form: simulate a uniform random variable and then use IF-THEN logic to assign patients to study group based on the value of that uniform random variable. A more efficient solution, specific to this problem, is:

```
data test1;
   call streaminit(1);
   do i=1 to 10;
      group=rand("bernoulli",0.50);
      output;
   end;
run;


proc print data=test1;
run;
```

3.2.5 Question: An investigator is planning a trial comparing a new smoking cessation intervention with usual care. There will be 40 patients per group. The quit rate in the intervention group is 20% whereas the quit rate in the usual care group will be 10%. What is the estimated statistical power, based on 1,000 simulated replications of the study?

Answer: From one of the assignments, I know that the general algorithm for using simulation to perform a power calculation is (1) form an outer DO loop covering the 1,000 replications of the study; (2) within an inner DO loop, simulate the data according to the specifications provided by the investigator (here, the result should be a dataset with 80,000 rows (i.e., 1000 iterations time 80 rows per iteration); (3) for each replication, perform a statistical test and output the p-value to a new dataset; (4) map that p-value to a new variable denoting the presence or absence of statistical significance; and (5) the estimated power is the proportion of replications with a statistically significant result. The important specifications are the 20% and 10% quit rates. I'll implement this by cutting and pasting the general DO loop into one DO loop per study group. The statistical test is a chi-square test, which should be performed by iteration. I need to use some form of OUTPUT statement to write p-values to a new SAS dataset. Instead of looking up the syntax, I just cut and pasted from one of the assignments -- it uses the Mantel-Haentzel

version of the chi-square test, which I assume is OK to do here. The resulting code turns out to be:

```
data test1;
   call streaminit(1);
   do iteration=1 to 1000;
      do i=1 to 40;
         group='I';
         quit=rand("bernoulli",0.20);
         output;
      end;

      do i=1 to 40;
         group='C';
         quit=rand("bernoulli",0.10);
         output;
      end;
   end;
run;

proc freq noprint data=test1;
   by iteration;
   tables group*quit / chisq;
   output out=chisq mhchi;
run;

data chisq2;
   set chisq;
   if (p_mhchi<.05) then sig='yes';
   else sig='no ';
run;

proc freq data=chisq2;
   tables sig;
   title 'task 2';
   title2 'estimated power';
run;
```

Comment: This question requires understanding a general simulation algorithm, which is essentially independent of programming language, then translating it into a data structure that works well in SAS, and then discovering the SAS-specific syntax (e.g., how to OUTPUT p-values from the FREQ procedure). Describing the algorithm demonstrates basic programming skills. If the student is unfamiliar with specific elements of syntax, they are asked to perform a web search and then demonstrate that they can effectively utilize its results.

3.2.6 Question: How could you make the DO statement more generalizable in the code segment above?

Answer: Replace the hard-coded value of 40 with a variable such as GROUP_SIZE.

Comment: This illustrates a general programming technique that contributes to literate programming, regardless of language. It also provides a hint for the next question.

3.2.7 Question: With 40 per group the statistical power is unacceptably low. Suppose that the investigator asks you a slightly different question: namely, how many patients would be needed to achieve 80% power? How would you modify the previous code? Describing an algorithm is sufficient to answer the question.

Answer: Replace 40 with GROUP_SIZE, add an outer loop that changes the value of GROUP_SIZE (e.g., DO GROUP_SIZE=40 to 200 by 10;), stop when the desired power is achieved. Indeed, a DO WHILE structure would be more efficient than a standard DO loop.

Comment: This also translates general statistical knowledge into an algorithm, appropriate for SAS.

3.2.8 Question: What does the following code do? Your answer should mention three of the four common ways to create a SAS dataset.

```
data set1;
   input id rec_no y;
datalines;
1 1 5
1 2 6
1 3 4
1 4 8
2 1 0
2 2 2
2 3 5
2 4 6
;
run;

proc print data=set1;
run;

data set2;
   set set1;
    if (y>5) then ind=1;
      else ind=0;
run;

proc means noprint nway data=set2;
   class id;
   var ind;
   output out=out1
            mean=new_var;
run;
```

```
proc print data=out1;
run;
```

Answer: The first DATA statement creates the SAS dataset called SET1 using raw data as input (i.e., method 1). The INPUT statement provides the input directions. The DATALINES statement tells SAS that the raw data follows. The RUN statement defines the end of this logical entity. SET1 should have 8 observations and 3 variables.

The second DATA statement creates the SAS dataset called SET2 (the child) from the SAS dataset SET1 (the parent). It illustrates creating one or more child SAS datasets from one or parent SAS datasets (i.e., method 2). The IF / THEN statements create a new "indicator" variable denoting whether or not Y>5. SET2 should have 8 observations and 4 variables.

The MEANS statement creates a new SAS dataset called OUT1 as the output from a SAS procedure (i.e., method 3). In this case, we define subgroup means, with the ID variable defining the subgroups. The calculations are applied to the variable IND, and the subgroup means are saved to the new dataset in a variable named NEW_VAR. The PRINT statements print the contents of the various SAS datasets. Although I'm not completely certain about this, OUT1 should have one row per subgroup and include variables containing the subgroup name and the subgroup mean.

Comment: This assesses the general programming construct of indicator variables. These are, among others, an essential part of various counting algorithms. It also assesses familiarity with SAS, as inexperienced users are unlikely to have encountered creating SAS datasets as output from a SAS procedure.

3.2.9 Question: How would you apply the same logic to creating a new SAS dataset containing the predicted values from a linear regression model? You should have encountered this in a first-year data analysis course. Describing an algorithm is sufficient for answering this question.

Answer: Use the REG procedure to perform the regression. As part of that procedure, use the MODEL statement to define the predictor and outcome variables. This procedure ought to have an OUTPUT statement like that for PROC MEANS. Within that OUTPUT statement, there should be similar syntax that changes "MEAN = new variable name" into "PREDICTED VALUES = new variable name". The output dataset should have one row per observation in the original dataset and the variables listed here (among others).

Comment: In practice, this is followed by questions about how this task is accomplished in R, with similarities and differences.

4. Course Evaluation

The student course evaluation included the following questions pertaining to their ability to learn SAS on their own (i.e., one of the core pedagogic constructs): "One of the premises behind the course design was that understanding the structure of SAS places you in position to effectively use help (e.g., online, via SAS's help function). (1) I can find and use help to learn additional SAS content; (2) I can design a SAS program; (3) If a program design has been prepared, I can write SAS code". The median response to each of these questions was "confident".

Less formally, after each coding interview the student was asked a series of questions. Their primary method of preparation was reviewing programming assignments. Those who wrote their own code and added comments performed especially well, whereas those who took a more passive approach of starting with the instructor's code and asking ChatGPT how it works performed adequately but not as well. This is consistent with a basic principle of constructivism that deep learning requires active engagement with content (Biggs, 1996).

Students consistently found the exam questions to be as expected, and cited the sample questions and illustrative interview as helpful. A few students had previous experience with coding interviews. Some of these reported that the final examination was quite similar to those, with the exception that starting with a "show and tell" about the basic algorithm was an improvement over what they had experienced elsewhere.

5. Discussion of Fairness and Practicality

When using coding interviews as an examination for non-professional programmers, we propose that an operational definition of a fair examination includes (1) it covers important content; (2) students can adequately prepare; (3) even if students aren't asked identical questions, the questions address similar content at a similar level of difficulty; and (4) adequate accommodations are made for diverse learning styles. Here, the exam content is derived from an explicit statement of its goals. Moreover, the format of a conversation, with the opportunity to both clarify and probe deeply, helps ensure that the content is "important". Our approach to helping students prepare is described above, and its

success was supported by student performance. By asking students questions of similar difficulty and nature, we can achieve the advantages of personalization while also addressing a common concern.

Finally, we recognize that some students, whether with formal academic accommodations or not, find oral examinations to be difficult. Indeed, research has suggested that alternate formats for the coding interview should be available (Behroozi et al., 2019). One way that we addressed this issue was by allowing students to bring annotated SAS programs to the exam, so that immediate recall of facts became less critical. Students were encouraged to ask clarifying questions if needed. We were prepared to have students submit follow-up responses in writing, although it turned out that this wasn't needed. Coding interviews are often part of the process of job searching, and our hope was that good performance on a nonconfrontational interview would encourage students going forward.

Another element of fairness is our desire to give those students who choose not to cheat the same opportunity to achieve an excellent grade as those who do (Troy, Neely, et al., 2022). The course grade is made up of two major components, the assignments and the exam. Students can achieve a B by successfully completing all of their programming assignments, recognizing that they might collaborate in doing so. Indeed, such collaboration is encouraged so long as they "own" the content by writing their own comments within the program code that explain, in their own words, what the code does. Receiving an A requires an excellent performance on the coding interview, for which we see no obvious way to cheat. Students are on a level playing field in that most of the questions can be anticipated. Follow-up questions easily distinguish deep from superficial knowledge. The only guarantee for receiving an excellent grade on the coding interview is to learn the material.

In terms of practicality, with sufficient discipline a simulated coding interview can be completed in 30 minutes. In practice, ours ranged from 30-60 minutes, in part because of the desire to put students at ease and in part because this also became an opportunity to do a bit of teaching. This effort is comparable to the combination of (1) grading a traditional exam, so long as that examination is in a format other than true/false, multiple choice, or something else which sacrifices depth of assessment for ease of grading; and (2) the time and energy involved with addressing cheating. It is also consistent with what students will face as they sit for coding interviews as part of the job search process (McDowell, 2020). We also reduced interviewer burden by limiting the number of interviews per day. Students were encouraged to select an examination time that would not interfere with their other exams.

The literature on coding interviews primarily focuses on professional programmers and software engineers, although its general principles and recommendations ought to apply to circumstances such as ours, where the typical job requirement is "excellent at statistics, programs well for an amateur, and can learn more about SAS programming as needed" (Behroozi et al., 2019; Ford et al., 2017). The increasing pedagogical focus on computing within our discipline is likely to make practice with coding interviews even more relevant in the future (Reinhart & Genovese, 2021). For our audience, starting the task-oriented questions by asking the student to explain a code fragment illustrating a simple version of an algorithm appears to be a technical innovation, and one that was well received. Furthermore, we found this to be consistent with recommendations to start by assessing basic knowledge (Behroozi et al., 2019).

6. Future Directions

The most immediate practical impact of this research within our program is to support an ongoing effort to more closely align our evaluation methods with how students will use biostatistical skills during their careers. As another example of doing so, a course in survival analysis has as its final examination: 1) a comprehensive analysis of an actual dataset; 2) writing a statistical report in standard format; 3) creating a video explaining the results of the analysis to an investigator who is scientifically sophisticated but not statistically so; and 4) creating a video explaining key concepts from the class in plain English. This examination format not only covers both statistical content and the student's ability to communicate their understanding of that content, it creates a rich body of information which provides confidence that the resulting grade adequately approximates the student's skill set. As illustrated in Section 3 above, coding interviews are intended to generate a similarly rich student evaluation.

Some future research questions include exploring the trade-offs among flexibility of content, accuracy of recording, and consistency of evaluation. For example, a traditional examination generates an archival record (e.g., which could accommodate assignment of grades by multiple instructors) and has students perform identical tasks but at the cost of the inability to query students in real time to further assess depth of understanding. A coding interview, on the other hand, has the benefits of real-time queries, albeit without an archival record (unless the interview is recorded) and with tasks that aren't identical (by design). Depending on the educational context, the fact that coding interviews have no obvious way to cheat can be a non-trivial consideration as well (Noorbehbahani et al., 2022). Indeed,

recommendations for limiting cheating include upgrading evaluation methods from multiple-choice questions to include open-ended questions focusing on problem-solving skills and critical thinking, and questions where students create their own examples (Skshidlevsky, n.d.). We recommend that any evaluation method that allows the student to work outside of class include an in-person follow-up where students explain their work and, thus, reduce the likelihood that their answers relied on input from someone else or from another source such as a generative artificial intelligence (Ellis & Slade, 2023; Troy, Neely, et al., 2022).

7. Conclusion

In summary, we believe that the examination format of a simulated coding interview is both fair and practical, allows for personalization, and provides a deeper assessment than more traditional formats. We postulate that potential concerns about practicality are mostly driven by difficulty in visualizing precisely how a coding interview can be operationalized. Our report aims to offer practical guidance for those interested in using this format.

References

Behroozi, M., Parnin, C., & Barik, T. (2019). Hiring is Broken: What Do Developers Say about Technical Interviews? *Proceedings of IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC*, *2019-October*, 15-23. https://doi.org/10.1109/VLHCC.2019.8818836

Biggs, J. (1996). Enhancing teaching through constructive alignment. *Higher Education*, *32*(3), 347-364. https://doi.org/10.1007/BF00138871

Denning, P. J., Comer, D. E., Gries, D., Mulder, M. C., Tucker, A., Turner, A. J., & Young, P. R. (1989). Computing as a discipline. *Computer*, *22*(2), 63-70. https://doi.org/10.1109/2.19833

Denning, Peter J., & Tedre, M. (2022). Computational Thinking: A Disciplinary Perspective. *Informatics in Education*, *20*(3), 361-390. https://doi.org/10.15388/INFEDU.2021.21

Ellis, A. R., & Slade, E. (2023). A New Era of Learning: Considerations for ChatGPT as a Tool to Enhance Statistics and Data Science Education. *Journal of Statistics and Data Science Education*, *31*(2). https://doi.org/10.1080/26939169.2023.2223609

Ford, D., Barik, T., Rand-Pickett, L., & Parnin, C. (2017). The tech-talk balance: what technical interviewers expect from technical candidates. *Proceedings - 2017 IEEE/ACM 10th International Workshop on Cooperative and Human Aspects of Software Engineering, CHASE 2017*, 43-48. https://doi.org/10.1109/CHASE.2017.8

Introducing ChatGPT. (n.d.). Retrieved June 7, 2023, from https://openai.com/blog/chatgpt

Knuth, D. E. (1984). Literate programming. *Computer Journal*, *27*(2). https://doi.org/10.1093/comjnl/27.2.97

McDowell, G. (2020). *Cracking the Coding Interview: 189 Programming Questions and Solutions* (6th ed.). CareerCup, LLC.

Neely, M. L., Troy, J. D., Gschwind, G., Pomann, G.-M., Grambow, S. C., & Samsa, G. P. (2022). Preorientation Curriculum: An Approach for Preparing Students with Heterogenous Backgrounds for Training in a Master of Biostatistics Program. *Journal of Curriculum and Teaching*, *In Press*.

Nolan, D., & Temple Lang, D. (2010). Computing in the Statistics Curricula. *The American Statistician*, *64*(2), 97-107. https://doi.org/10.1198/tast.2010.09132

Noorbehbahani, F., Mohammadi, A., & Aminazadeh, M. (2022). A systematic review of research on cheating in online exams from 2010 to 2021. *Education and Information Technologies*, *27*(6). https://doi.org/10.1007/s10639-022-10927-7

Pomann, G.-M., Boulware, L. E., Cayetano, S. M., Desai, M., Enders, F. T., Gallis, J. A., Gelfond, J., Grambow, S. C., Hanlon, A. L., Hendrix, A., Kulkarni, P., Lapidus, J., Lee, H.-J., Mahnken, J. D., McKeel, J. P., Moen, R., Oster, R. A., Peskoe, S., Samsa, G., … Thomas, S. M. (2020). Methods for training collaborative biostatisticians. *Journal of Clinical and Translational Science*. https://doi.org/10.1017/cts.2020.518

R: The R Project for Statistical Computing. (n.d.). Retrieved June 7, 2023, from https://www.r-project.org/

Reinhart, A., & Genovese, C. R. (2021). Expanding the Scope of Statistical Computing: Training Statisticians to Be Software Engineers. *Journal of Statistics and Data Science Education, 29*(S1), S7-S15. https://doi.org/10.1080/10691898.2020.1845109

Samsa, G. (2021). Evolution of a Qualifying Examination from a Timed Closed-Book Format to an Open-Book Collaborative Take-Home Format: A Case Study and Commentary. *Journal of Curriculum and Teaching*, *10*(1), 47. https://doi.org/10.5430/jct.v10n1p47

Samsa, G. P. (2018). A Day in the Professional Life of a Collaborative Biostatistician Deconstructed: Implications for Curriculum Design. *Journal of Curriculum and Teaching*, *7*(1), 20-31 https://doi.org/10.5430/jct.v7n1p20

Samsa, G. P. (2020). Using Coding Interviews as an Organizational and Evaluative Framework for a Graduate Course in Programming. *Journal of Curriculum and Teaching*, *9*(3), 107-140.

Samsa, G. P., LeBlanc, T. W., Locke, S. C., Troy, J. D., & Pomann, G.-M. (2018). A Model of Cross-Disciplinary Communication for Collaborative Statisticians: Implications for Curriculum Design. *Journal of Curriculum and Teaching*, *7*(2), 1-11. https://doi.org/10.5430/jct.v7n2p1

SAS: Analytics, Artificial Intelligence and Data Management. (n.d.). Retrieved June 7, 2023, from https://www.sas.com/en_us/home.html

Skshidlevsky, A. (n.d.). *Prevent Cheating in College*. ProctorEdu. Retrieved October 13, 2023, from https://proctoredu.com/blog/tpost/3tih8y7or1-prevent-cheating-in-college

Troy, J. D., Granek, J., Samsa, G. P., Pomann, G.-M., Updike, S., Grambow, S. C., & Neely, M. L. (2022). A Course in Biology and Communication Skills for Master of Biostatistics Students. *Journal of Curriculum and Teaching*, *11*(4), 120. https://doi.org/10.5430/jct.v11n4p120

Troy, J. D., Neely, M. L., Pomann, G. M., Grambow, S. C., & Samsa, G. P. (2022). Administrative Considerations Pertaining to the Use of Creative Methods of Student Assessment: A Theoretically Grounded Reflection from a Master of Biostatistics Program. *Journal of Curriculum and Teaching*, *11*(5), 105. https://doi.org/10.5430/JCT.V11N5P105

Welcome to Python.org. (n.d.). Retrieved June 7, 2023, from https://www.python.org/

Acknowledgments

Not applicable.

Authors contributions

Not applicable.

Funding

Gina-Maria Pomann's support of this project was made possible (in part) by Grant Number UL1TR002553 from the National Center for Advancing Translational Sciences (NCATS) of the National Institutes of Health (NIH), and NIH Roadmap for Medical Research. Its contents are solely the responsibility of the authors and do not necessarily represent the official views of NCATS or NIH.

Competing interests

Not applicable.

Informed consent

Obtained.

Ethics approval

The Publication Ethics Committee of the Sciedu Press.

The journal's policies adhere to the Core Practices established by the Committee on Publication Ethics (COPE).

Provenance and peer review

Not commissioned; externally double-blind peer reviewed.

Data availability statement

The data that support the findings of this study are available on request from the corresponding author. The data are not publicly available due to privacy or ethical restrictions.

Data sharing statement

No additional data are available.