Automated grading workflows for providing personalized feedback to open-ended data science assignments

Federica Zoe Ricci*
and
Catalina Mari Medina†
and
Mine Dogucu†
Department of Statistics, University of California Irvine

Submitted: Aug 17, 2023; Accepted: Dec 11, 2023; Published: Jul 19, 2024

Abstract

Open-ended assignments - such as lab reports and semester-long projects - provide data science and statistics students with opportunities for developing communication, critical thinking, and creativity skills. However, providing grades and formative feedback to open-ended assignments can be very time consuming and difficult to do consistently across students. In this paper, we discuss the steps of a typical grading workflow and highlight which steps can be automated in an approach that we call automated grading workflow. We illustrate how gradetools, a new R package, implements this approach within RStudio to facilitate efficient and consistent grading while providing individualized feedback. By outlining the motivations behind the development of this package and the considerations underlying its design, we hope this article will provide data science and statistics educators with ideas for improving their grading workflows, possibly developing new grading tools or considering use gradetools as their grading workflow assistant.

Keywords: data science education, statistics education, R, formative assessment, fair grading, reproducible teaching

*Gratefully acknowledges funding by the Hasso-Plattner-Institute Research Center in Machine Learning and Data Science at UCI.

†Gratefully acknowledge funding by NSF IIS award 2123366.

1 Introduction

From a learner's standpoint, assessments are fundamental moments of the learning process. Assessments (especially *formative* ones (Dixson & Worrell 2016)) give students opportunities to practice, interiorize, deepen and demonstrate the concepts learned with lectures and readings. By requiring the use of class material for answering questions and solving problems, assessments reveal what parts of the syllabus are well understood and what parts need, instead, revision. This is crucial for both students and instructors to make informed decisions on how to improve, respectively, their learning and teaching strategies (Pai 2024).

Following up on assessment activities, teachers can give students qualitative (e.g., "The source of the data utilized in your analysis was not specified.") and quantitative (e.g., "9 out of 10") comments. To distinguish the former from the latter, in this paper we will reserve the term *feedback* for qualitative comments and we will call *grades* the scores that quantitatively summarize how well a student did on the assignment. Grades are a convenient summary of students' performance (Lipnevich & Smith 2009). On the other hand, as stated by the Guidelines for Assessment and Instruction in Statistics Education (GAISE 2016), assessment must receive feedback in order to lead to learning, and teachers should provide assessment *and* feedback throughout their courses.

The importance of feedback is underlined by several studies in the literature on education. Reviewing research on formative assessment, Black and William (1998) found evidence that innovations designed to improve frequent feedback can bring substantial learning gains. In addition, a number of studies observe that feedback, when done well, can support learning (Hattie & Timperley 2007, Carless 2006), especially for students with little prior knowledge (Krause et al. 2009). In concordance with these observations, several scholars have argued

for the need to raise awareness among teachers on the usefulness of formative feedback (Nicol & Macfarlane-Dick 2006, Perera et al. 2008, Irons & Elkington 2021).

Impactful feedback should clarify what good performance is and encourage positive motivational beliefs (Nicol & Macfarlane-Dick 2006), it should be provided in a timely manner, and be constructive and specific to the student's work (Juwah et al. 2004, Race 2001). Providing accurate feedback and grades to students' work can therefore be very time consuming and become a struggle for (even moderately) large classes. Data science courses are particularly affected, as they face higher course enrollment numbers as one of their major challenges (National Academies of Sciences, Engineering, and Medicine 2018).

Automated grading and feedback tools have been proposed by many as a possible solution (Galassi & Vittorini 2021). For types of assessment where all ways of stating the correct solutions can be enumerated, like multiple-choice, select-all and short-answer questions, automated grading tools are available for assignments distributed using popular online grading systems such as Gradescope (Singh et al. 2017) or learning management systems (LMS) such as Canvas. Similarly, for computing assignments where the correct solutions can be defined through a series of if-else statements, several automated grading tools are currently available, including the Python library `nbgrader` (Blank et al. 2019) for grading Jupyter notebooks, the R package `learnr` (Schloerke et al. 2020) for creating self-paced interactive tutorials in teaching R and R packages, Otter-Grader (Pyles & UC Berkeley Data Science Education Program 2022) for grading Python and R assignments, and the language-agnostic autograder in Gradescope. These tools allow to give fast or even real-time responses (as recommended by GAISE 2016) to those assignments for which, upon setup, providing grades and feedback can be done without human judgement. However, they are not amenable for many recommended assessment types, for which providing feedback

remains challenging.

Indeed, open-ended assessments including lab reports (GAISE 2016), semester long projects (GAISE 2016, Cetinkaya-Rundel et al. 2022), and writing assignments (Woodard et al. 2020, Johnson 2016) cannot be autograded, as enumeration of all possible correct or incorrect approaches is not possible. These types of open-ended assignments are known to provide opportunities for developing communication, critical thinking, and creativity skills in addition to supporting statistical knowledge (Garfield & Gal 1999). Providing good feedback to such assessments involves pedagogical choices and requires human judgement, which brings two major challenges: a higher time-cost for grading and more difficulty to grade consistently across students. These challenges can affect the number of open-ended assignments and the quality of feedback that can be provided, even for small and medium-sized classes. Using a detailed rubric for grading can greatly help with consistency (Ragupathi & Lee 2020, Timmerman et al. 2011), but it requires time and can sometimes be overlooked during the grading workflow.

In this article, we consider grading workflows that can assist with providing grades and good-quality feedback to data science and statistics assignments that require human judgement to be assessed. Gradescope (Singh et al. 2017) and the LMS Canvas, among other online tools, both provide valuable options for grading open-ended assignments. Particularly Gradescope, as reported by teachers from many STEM disciplines (Garcia 2015, Reck 2019, Yen et al. 2020), has made grading easier and faster, with features such as rubric-based grading for transparency and consistency, dynamic rubric creation and group assignments, to name a few. However, data science and statistics undergraduate classes often include assignments that involve computing and may be completed, e.g., as R scripts (see for example Hu & Dogucu 2022; and Dogucu & Çetinkaya-Rundel 2021), combination

of computing, data visualization and writing using, e.g., R Markdown and Quarto files (for instance Loy et al. 2019), multiple of these files (e.g., for a final project) and may be administered using GitHub for a top-down approach to teaching Git (Beckman et al. 2021, Fiksel et al. 2019). These cases are not easily handled with Gradescope.

The contributions of this article are threefold. In Section 2 we present the steps of a typical grading workflow for data science open-ended assignments, highlighting the steps that can be automated, and discussing pedagogical tools such as rubrics and feedback, to build the concept of an automated grading workflow. Next, in Section 3, we introduce the package **gradetools** (Ricci et al. 2022), that implements an automated-grading workflow within the RStudio Graphical User Interface (GUI). This package enables efficient and consistent grading directly within RStudio, as well as scalable yet individualized feedback provision, and integrates with the existing R package **ghclass** (Rundel & Cetinkaya-Rundel 2022) to streamline feedback distribution for assignments managed with GitHub. Following this, in Section 4 we examine the key underpinnings of the gradetools package that allow it to be an automated grading workflow. In doing so, we intend to provide data science educators with ideas for improving their grading workflows, and possibly developing new automated-grading workflow tools adjusted to their own grading needs. Lastly, in Section 5 we summarize key points and discuss the implications of this work for the statistics and data science education community.

2 What is an automated grading workflow?

The assessment at scale of assignments that cannot be auto-graded requires automated grading workflows, that is, systems that automate all or most repetitive grading tasks so

Table 1: Phases of a typical grading workflow and corresponding tasks. Tasks colored in green are pedagogical, in black are administrative. The tasks listed in the grading phase need to be repeated for each submission to grade.

1. Preparation	**2. Grading and Feedback**	**3. Finalization**
Collecting students' assignments	Retrieving and opening a submission	Uploading grade sheets on class's learning management system
Setting up a rubric	Assigning grade and writing feedback based on current rubric	Returning grades and feedback to students
Setting up a grade sheet	Updating rubric as needed	
	Updating record of student corresponding to this submission on the grade sheet	
	Closing the submission	

as to reduce the time and effort required for a grader to assess students' work and possibly provide high-quality feedback.

A natural starting point for designing an automated grading workflow is to outline the tasks that are executed in a typical grading workflow. We can break down a grading process into three phases - (1) preparation, (2) grading and providing feedback, (3) finalization - each with their respective tasks. Table 1 lists these phases and groups tasks into two types: *pedagogical* and *administrative*. The former have a direct impact on what students learn from their grade and their feedback, while the latter are related to the logistics involved with the grading process.

Automated grading workflows should automate administrative grading tasks. These tasks tend to be repetitive and mostly do not require human judgement during their execution; in fact, their automation can minimize the occurrence of errors such as miscomputing the overall grade or assigning a grade to the wrong student in the grade book.

Some pedagogical tasks - drafting and updating a rubric - always require human judgement; other pedagogical tasks - evaluating a submission, considering how the rubric should be

applied to a given submission - require human judgement for open-ended assignments. Even when they do require human judgement, there are sources of repetitivity involved in executing pedagogical tasks that an automated grading workflow can automate.

Unless a class has very few students, most of the time required for grading open-ended data science assignments is typically spent in Phase 2, that is, evaluating submissions and assigning grades and feedback. Providing individualized feedback is especially time-demanding and may often be sacrificed, even though it is extremely valuable for students as discussed in Section 1. Automated grading workflows can scale provision of individualized feedback by leveraging the fact that some feedback that is individualized to features of a student's submission is in fact applicable to all students whose submissions present the same features. To better illustrate this, the next subsection outlines different types of feedback and explains how automation may be achieved.

2.1 Feedback types and automation

Feedback can differ based on its applicability across students and across questions. Table 2 distinguishes and exemplifies six types of feedback, based on whether they are applicable to a single or to multiple students and to a single question, to multiple questions or to the entire assignment.

Note that all of these feedback are *individualized*, in the sense that they are specific to a student's submission - rather than merely stating what the correct solution is or summarizing how the whole class did on the assignment. However, some of these feedback are *repeatable* - those that can be applied to multiple students - while some of them are *unique* because they are specific to a feature that is present in a single submission.

Table 2: Examples of feedback that can be given to only a single or to multiple students, and for only one question or for multiple questions (or components).

Student applicability	Question applicability	Example
multiple	single	When interpreting the slope coefficient make sure to use units of measurement (in this case, miles).
multiple	multiple	Please adhere to the Tidyverse style guide.
multiple	entire assignment	Great job on this assignment!
single	single	Recall our conversation about the p-value during office hour...
single	multiple	The soft g letter (ğ) encoding is not displayed correctly on your output. In LaTeX try: \u{g}.
single	entire assignment	Thank you for your note, Menglin. I am glad you had fun doing the assignment.

Whether they are repeatable or unique to a submission, feedback can be applicable to a *single question* of the assignment (or component), applicable to *multiple questions* (or components), or be *general* feedback that refer to how a student did overall on the assignment. In our experience, we found that multiple-question, single-student feedback is rarely needed but we commonly encounter the other feedback scenarios.

An automated grading workflow should expedite the provision of repeatable feedback across different questions and students. It should also facilitate providing unique feedback on the fly. One way of scaling the evaluation of submissions and the assignment of grades and feedback is by setting up a rubric that has an item for each encountered feature of students' assignments and that, for each item, indicates both its associated feedback and its associated score (that is, a number of points to remove, or add, to a student's grade when their work presents this feature). This will be further discussed in Section 4.

Given the rubric and a selection of rubric items prepared by the grader, a system can then be designed to update the grade sheet and write individualized feedback to the assignment that is being graded. An automated grading workflow should also facilitate updating the rubric dynamically, when the grader encounters previously-unobserved features that may

occur in other submissions yet to be assessed.

The R package gradetools is an automated-grading-workflow system designed around these ideas, for executing Phase 2 tasks within RStudio. The next section shows what this system looks like from the user perspective.

3 Introducing gradetools

3.1 Motivating grading scenario

Earlier functions of the gradetools package were first developed to assist with grading assignments of the Stats 68 class - Statistical Computing and Exploratory Data Analysis with roughly 60 students enrolled. Since its creation as a package, we also used gradetools extensively in our Stats 6 - Introduction to Data Science course with around 120 students. The courses share a similar computational structure. Every week students receive and submit assignments through individual GitHub repositories. Formerly, the assignments generally consisted of a R Markdown file (.Rmd), with questions involving a combination of coding and text (e.g., running a model and interpret its results), or occasionally R scripts (.R). Currently, we utilize Quarto files (.qmd) in our courses.

Teaching students good coding practices and reproducible workflows via R Markdown/Quarto and GitHub have been key learning goals of computationally rigorous courses that we teach. In these courses, we assesses the raw .Rmd/.qmd files rather than their rendered pdf or HTML files. In addition, students work on projects that have multiple files with directories consisting of multiple folders. For instance, a project folder has subfolders consisting of data, project proposal, and presentation with each subfolder

having multiple files.

For such teaching settings, we had a few options to consider. The first one was the LMS Canvas. Grading on Canvas can be efficient when each student submits a single pdf file for an open-ended assignment. The interface allows for annotation and use of rubric. Canvas did not meet our grading needs since students submitted multiple files (e.g., a dataset they found and an Rmd file) and Rmd files could not be displayed. We did not consider downloading students' submissions as a zipped folder and opening one-by-one as that would defeat the goal of efficient grading.

Another option was autograders including the aforementioned packages or Gradescope's automated grading feature. These options did not allow for assessing higher-levels of thinking that we wanted to assess. Lastly, Gradescope also has an online submission feature, which in its capacity is similar to an LMS. Contrary to Canvas, Gradescope does display markdown documents. However, this also limits student's submission to a single file submission at a time and cannot help instructors assess a full project in a typical folder structure.

We believe that in similar computationally rigorous statistics and data science courses, students' files are best suited to be evaluated in a GUI such as RStudio, where a grader can look at the raw file and simultaneously, if needed, its rendered version. To avoid the need of switching between different applications used to maintain a grade sheet and manage a rubric, we created the gradetools package to carry out grading within RStudio. With this package we also automated repetitive parts of the grading workflow, and we automated writing a rich, individualized feedback file for each student that could then be shared with them.

3.2 Grading example with the gradetools package in RStudio

To better understand the process of grading with gradetools, we will walk through a simple example of grading a quiz with two questions. In this example we begin with a properly formatted class roster, quiz rubric, and student submissions. Details on gradetools's format requirements can be found in Section 4 and in the package's introductory vignette (https://federicazoe.github.io/gradetools/articles/a-grading-with-gradetools.html).

Grading in action

To begin the process, the gradetools package is loaded and the grading function, `assist_grading()` is called - this is shown in the first image of Figure 1. This function requires the locations of the roster, rubric, and submissions, as well as where to write the grading progress log, grade sheet, and feedback files. The function call triggers the grading of the first student on the roster and opens their submission automatically. Based on the provided location of the submission and on the desired location for the feedback file of a *single* student, gradetools determines the file path for submission and feedback files of all students in the roster. Grading is an interactive process where the grader is prompted in the console to grade a submission according to the provided rubric.

Figure 1 displays three screenshots, each showing the student's quiz on the left, and on the right in the console is the rubric prompting and selection for a component of the quiz. The first screenshot shows the grading function call (on the right), which begins the grading process by automatically opening the first student's assignment submission (on the left) and prompting the grader to grade the first question on the quiz according to rubric options (on the right). This student, Gia Bayes, has answered both parts of Question 1 correctly so the "Correct" rubric item is selected, removing zero points, by entering "100",

the corresponding prompt code, into the console. This example is using a negative grading scheme, where each rubric item is associated with points to remove from the total number of points the question is worth. After the code is entered into the console, the user is then prompted to grade the next question in the rubric. Note that there are additional options - providing a personalized feedback message, creating a new rubric item, and terminating the grading process - and these options will be discussed later.

Moving onto question 2, displayed in the second screenshot in Figure 1, the student was asked to produce a plot and use it to compare within-group patterns. Overall the student addressed this question, but plotted counts instead of proportions. We want to convey that plotting proportions within groups would visualize all group patterns, while plotting counts hinders comparison of patterns within the groups with relatively small counts. The corresponding rubic item is the first available, "Plotted counts" which deducts 0.25 points, and is selected by supplying "1a" to the console.

After all the questions have been graded, the user is prompted to provide feedback for how the student did on the quiz overall, called general feedback. This rubric has one general feedback rubric option, which commends the student's adherence to the tidyverse style guide. Gia Bayes did a great job using tidyverse style so this option is selected by entering "100" into the console.

Once all questions have been graded, the submission automatically closes, the student's overall quiz grade is displayed, and grading continues with the next student in the roster, Lee Kim. As shown in Figure 2, Lee's quiz is automatically opened and the grader is prompted to grade question 1. Lee correctly identified the number of observations and variables, but did not use inline code in their answer, so the rubric item "Didn't use inline code" is applied by entering the prompt code "1a", which will deduct 1 point from the

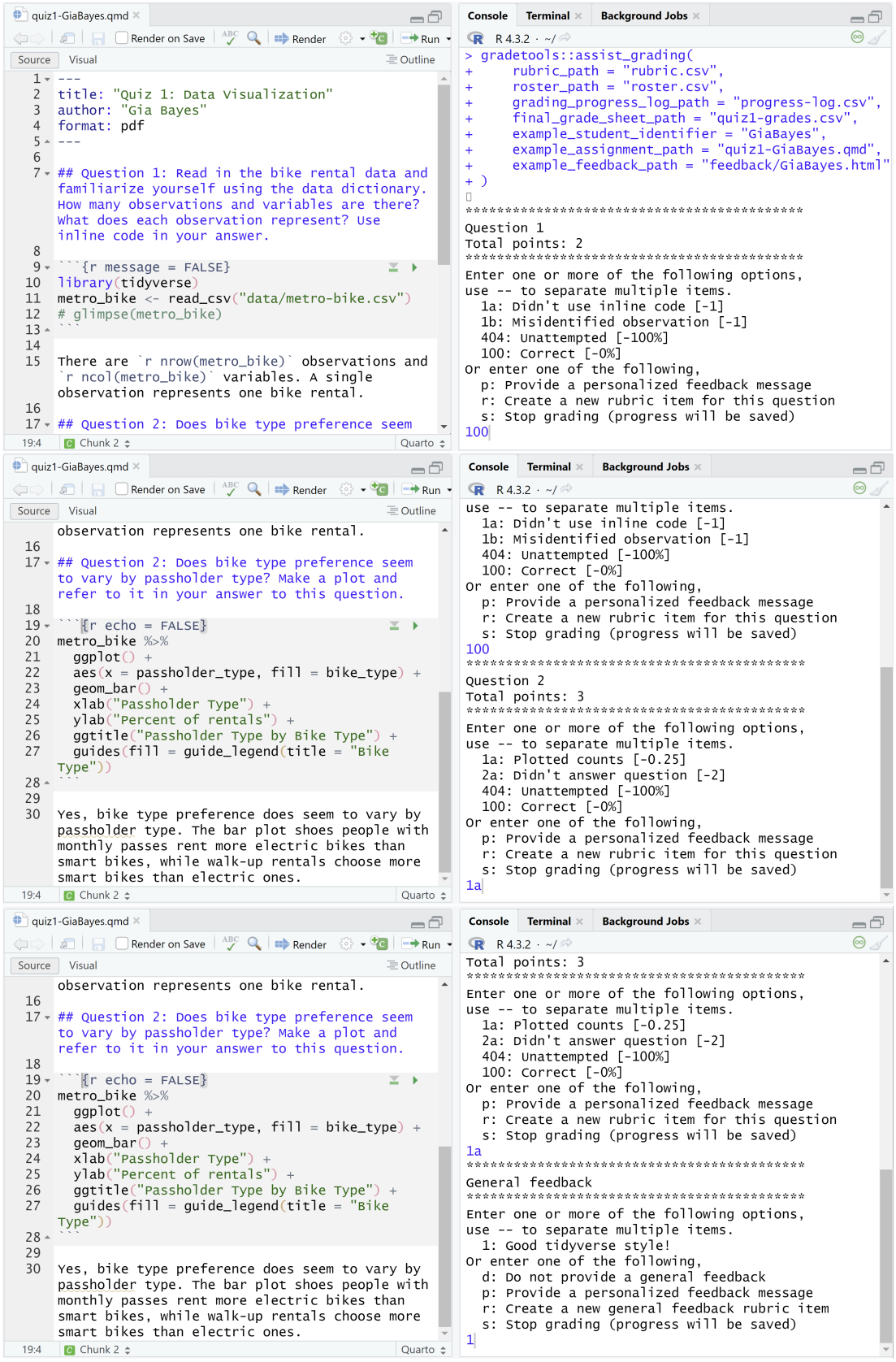


Figure 1: Example of grading a quiz using gradetools in RStudio. Each image shows the grading of a component of a student's quiz. This quiz belongs to the first student on the roster, Gia Bayes.

question's total of 2 points. For question 2 we see that the student did a bar plot of counts, like the first student we graded, and they did not provide an explicit answer to the question "Does bike type preference seem to vary by passholder type?". These mistakes each correspond to a different rubric item. Multiple rubric items are applied by using "–" to separate prompt codes. After entering "1a – 2a", the grader is asked to provide general feedback for Lee's quiz. Tidyverse style was not used for naming the bike rental data, so we may choose to avoid providing a general feedback by entering "d" into the console.

Grading outputs

To conclude this example grading session we will proceed as if all remaining quizzes have been graded. Upon completion of grading, or termination of the grading process, a final grade sheet is created and feedback files are automatically written from the feedback associated with the applied rubric items for each student, displayed in Figure 3.

Figure 3c shows the grade sheet for this quiz, with student identifiers obtained from the class roster, total grade, and points earned for each question, separated by an ampersand. The grade sheet could now be formatted and uploaded to a LMS and the feedback files could be distributed to students. While grade sheets are common in practice, feedback files may be less familiar.

The first student's feedback file displayed in Figure 3a contains feedback associated with the rubric items selected when grading this student. Notice that the prompt message for the rubric item applied for question two, "Plotted counts", is different than the feedback message written to the file. Both messages were specified for that rubric item in the rubric file, one meant as a concise summary for the grader and the other as a more thorough note for the student. The association of feedback messages with rubric items is a key aspect of gradetools which avoids redundancy of retyping feedback for the same mistake,

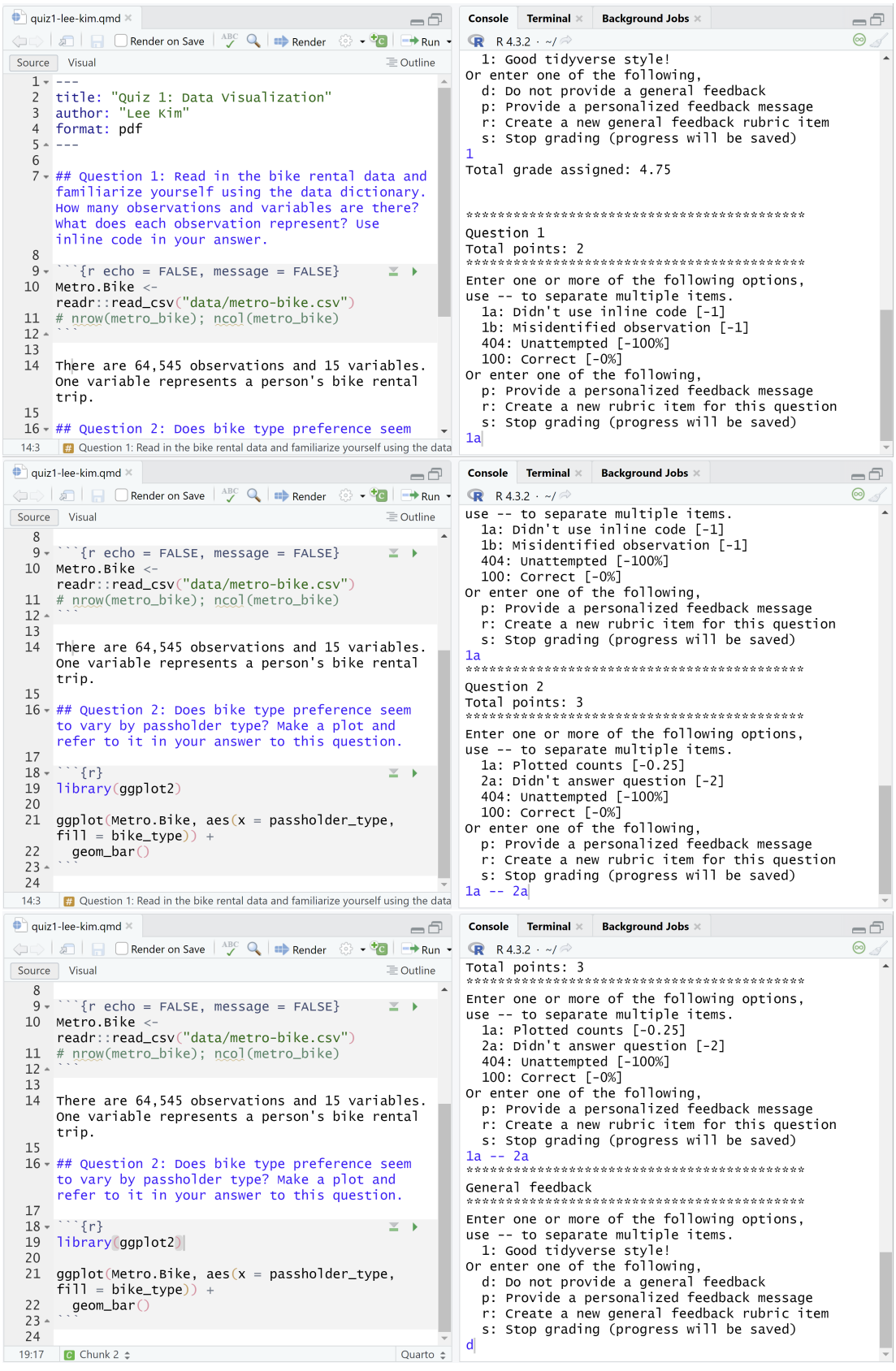
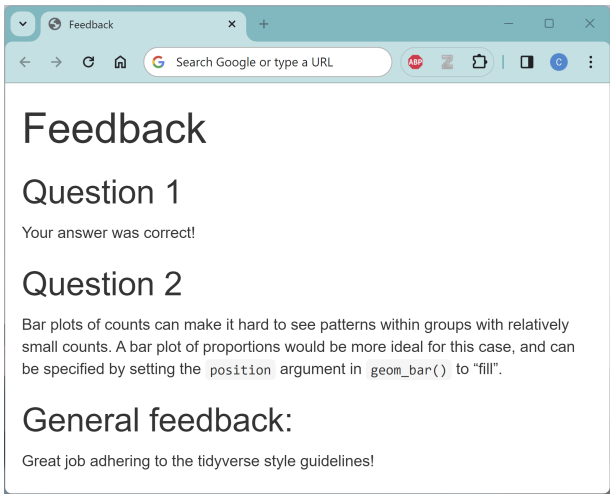
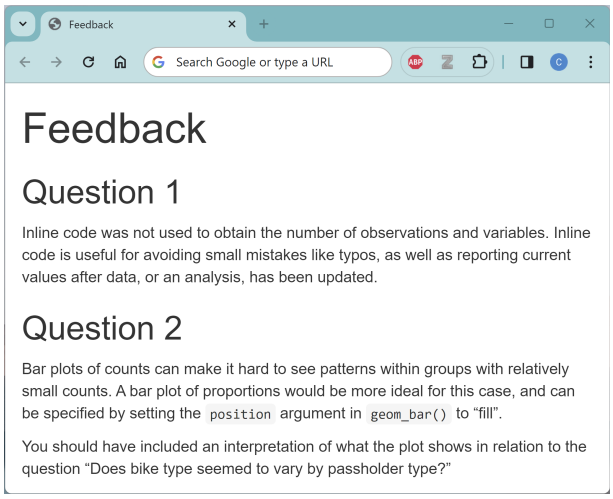


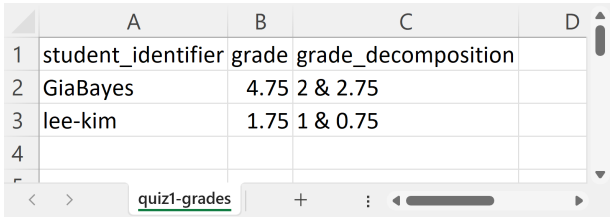
Figure 2: Continuation of example of grading a quiz using gradetools in RStudio. Each image shows the grading of a component of a student's quiz. This quiz belongs to the second student on the roster, Lee Kim.



(a) Feedback for first student, Gia Bayes



(b) Feedback for second student, Lee Kim



	A	B	C	D
1	student_identifier	grade	grade_decomposition	
2	GiaBayes	4.75	2 & 2.75	
3	lee-kim	1.75	1 & 0.75	
4				

quiz1-grades

(c) Grade sheet

Figure 3: Outputs of grading: feedback files and grade sheet corresponding to example of grading a quiz using gradetools in RStudio.

while creating feedback documents that are specific to each students' performance on the assignment. For example, both students made the same mistake of making a bar plot of counts for question two, so they both have the same corresponding message written in their feedback file. The second student made the additional mistake of not explicitly answering the question posed in question 2, so they have an additional message in their feedback.

Dynamic rubric editing

In this grading example the rubric already had rubric items for all instances we encountered. When that is not the case, the grader would likely want to add rubric items as they grade and encounter new responses. This is possible with gradetools by entering "r" into the console instead of specifying an available rubric item. Another possibility is that the grader may want to edit a preexisting rubric item, e.g., change the number of points possible for a question or the feedback for a rubric item. Whenever the grader re-runs the grading function with an updated rubric, all feedback files and the grade sheet are updated to reflect the latest rubric version.

Unique on-the-fly feedback

Another option that was not showcased in the previous grading example is the ability to write feedback message unique to a student - that is, a "single-student" type of feedback in the terminology of Table 2. By entering "p" into the console while grading, the user can enter a note to be written to the feedback file not associated with any points. For example for the second student, question 2, we could have left a note in their feedback telling the student they could receive partial credit if they resubmit their assignment with an interpretation of the plot in response to the question.

Fixing grading mistakes

Lastly, in this example we did not make any mistakes, but mistakes can happen in reality. Grading can be stopped at any time by entering "s" into the console. Doing so will end the grading process and all grading progress will be maintained through the grading progress log produced. The function `assist_regrading()` can be used to regrade specified students and questions (see the vignette for step-by-step instructions at https://federicazoe.github.io/gradetools/articles/b-regrading-with-gradetools.html).

3.3 Grading scenarios

The grading function `assist_grading()` has the core grading functionalities (later summarized in Figure 6) and is useful for users with limited R knowledge. The previous grading example demonstrated using gradetools to grade a single file per student. We will now discuss other grading scenarios and their respective gradetools functions.

Grading projects

Sometimes assignments involve multiple files to grade, for example a final project that includes a README file to describe where the data was obtained and a Quarto file that generates a presentation. Assignments that involve multiple submissions can be handled by gradetools, by providing a vector of file paths for a student's submission, instead of just a single location. Another useful ability for grading projects in RStudio with gradetools is the option to render documents while grading, so the raw and rendered files can be viewed at the same time. Examples of these features for grading projects can be found in gradetools' comprehensive vignette (https://federicazoe.github.io/gradetools/articles/e-comprehensive-example.html).

Grading team assignments

We use *team grading* to refer to the case when multiple students share a submission and grade. The `assist_team_grading()` function allows for grading team assignments and functions similarly to `assist_grading()`. The only additional requirement is a column in the roster denoting what team each student is on. The grading process for team assignments results in a single feedback file and grade for each team. A vignette for team grading can be found at https://federicazoe.github.io/gradetools/articles/c-extended-capability-teams.html.

Grading assignments managed through GitHub

Assignments can be managed through GitHub, teaching students reproducibility practices and version control, but collecting assignments from GitHub, grading, and returning feedback to student can be time consuming without the appropriate tools, such as the R package ghclass. Our package complements the streamlined collection of GitHub repositories with ghclass, by allowing the noting of issues while grading with `assist_advanced_grading()` or `assist_team_grading()`, and allowing the user to push feedback and create issues on GitHub using `push_to_github()`. A vignette on using gradetools with assignments managed through GitHub can be found at https://federicazoe.github.io/gradetools/articles/d-extended-capability-github.html.

Multiple graders

When an instructor has helpers, such as teaching assistants, the grading load can be split accross multiple people. This may mean the students submissions are partitioned between the graders (e.g. grader 1 the first 20 students and grader 2 the last 20 students), or the questions for all submission may be partitioned between the graders (e.g. grader 1 grades the odd questions and grader 2 the even). The functions `assist_advanced_grading()` and `assist_team_grading()` allow for the user to specify which students and questions

are to be graded, with the default of all students and questions. Simultaneous grading with gradetools would result in different grading log files for each grader (see Section 4.3 for details on this file). Merging grading log files would then need to be implemented by the grading team, as gradetools does not currently include a function for it.

3.4 Considerations for adoption

When considering adopting gradetools as your automated-grading-workflow assistant, it is important to take into account your grading scenarios and if gradetools is compatible, and weigh the advantages against the learning curve. This package was made with coding and report scripts for data-science assignments in mind, for classes with frequent assignments and/or 30 or more students per grading staff - where there are no grading packages or software that we are aware of. But gradetools can also be helpful for grading scenarios beyond its original purpose, especially for teachers at institutions that do not pay for grading software such as Gradescope.

A key consideration for adopting gradetools, or any other software, is the learning curve. Minimal R knowledge is required for gradetools, since the user only needs to know how to call a function from a package, but an understanding of file paths is necessary since the arguments for the grading functions are almost all file paths. The biggest challenge in adoption would be learning the details of how the rubric must be formatted and the file naming conventions in order to use gradetools. These requirements are detailed in the introductory vignette (https://federicazoe.github.io/gradetools/articles/a-grading-with-gradetools.html), and their purpose is discussed in Section 4. Once a user has successfully called the grading function, the grading process is straightforward, as exemplified in Section 3.2.

The time to grade is an important decision when creating assignments and deciding on a

grading workflow. Grading open ended questions can be time consuming, and gradetools speeds up the process by automating repetitive tasks, but still requires considerable time relative to assignments that could be autograded. For large classes and limited teaching staff, the time gain for using gradetools may still not be enough to allow grading many open-ended assignments throughout the quarter, and a combination of frequent quizzes autograded with another tool (e.g., Online assignments in Gradescope) with few open-ended assignments graded with gradetools may be considered.

We restricted the scope of gradetools to mostly the grading stage, leaving collection of student submissions, distribution of feedback files, and uploading grade sheets mostly beyond the scope of our package, with the exception of GitHub-compatible functionalities. Our package could be combined with other available software, such as a package to distribute feedback files through email or packages which can aid in the collection of student submissions, in order to speed up other components of your grading workflow.

4 Underpinnings of automation in gradetools

In this section we provide more details on how gradetools is designed. The package's vignettes that can be found at https://federicazoe.github.io/gradetools/ are the best documentation to get started to *use* gradetools functionalities. Instead, this section is for the reader who would like to understand how these functionalities are implemented in gradetools. This would be especially valuable for someone who is considering to extend gradetools or to develop new automated grading-workflow tools to better suit their grading needs.

In the next subsections we discuss the main strategies adopted by gradetools for automating

repetitive tasks encountered in Phase 2 of grading (i.e. assigning grades and feedback), that were outlined in Table 1. Specifically, we consider automation with: (1) retrieving, opening and closing submissions; (2) assigning grade and feedback based on rubric; (3) maintaining the grade sheet. For each of these tasks, we are going to see that the way in which they are automated is tied to specific choices made during Phase 1 of grading (i.e. grading preparation).

4.1 Retrieving, opening and closing submissions

With many submissions, considerable time may go into retrieving, opening and closing submission file(s) without automation of these tasks. As seen in the grading example of Section 3.2, these are tasks that are automated by gradetools. To automatically identify the student corresponding to each submission, some assumption must be made on the way in which students assignments are collected (or better, stored). The solution that we adopted with gradetools is to assume that students have a unique student identifier, specified in the roster, and that this identifier is present (at least once) in the student's assignment file path and is the only part unique to that student's assignment file path. For example, if the first quiz consists of a single Quarto file, all submissions may be stored in a folder named `quiz1` and the file names may be `quiz1-GiaBayes.qmd`, `quiz1-lee-kim.qmd`, etc. Thanks to this structure, at grading time, the grader needs to provide the student identifier and assignment file path for only one student in the roster. For example, a grader can call `assist_grading()` with inputs `example_student_identifier = 'GiaBayes'` and `example_assignment_path = 'quiz1/quiz1-GiaBayes.qmd'`, and provide the location of the roster where the other student identifiers are listed. Then, the template `quiz1/quiz1-` + student identifier + `.qmd` is assumed for all assignment file paths, and gradetools is able

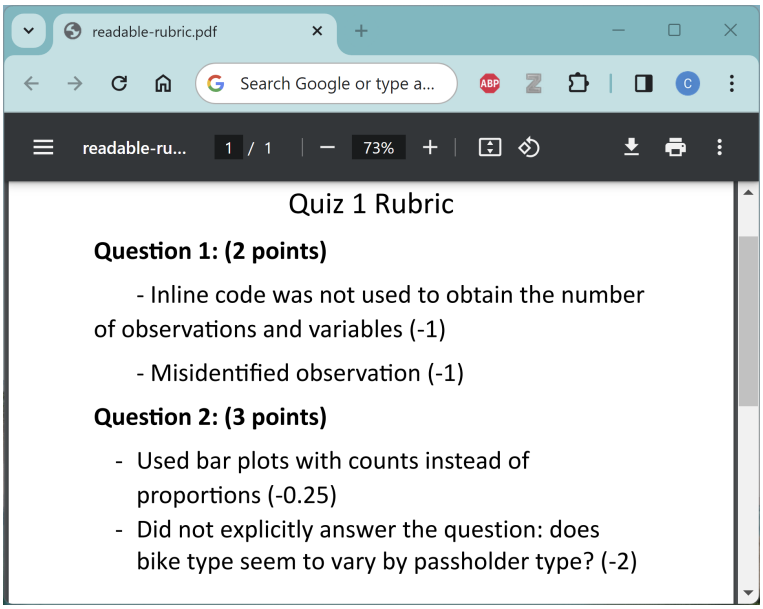
to iterate through opening and closing all submissions.

An assignment made of multiple files can be handled in a similar way. For example, if the first quiz consisted of two files (e.g., a Quarto document with a data analysis and a R script with some functions) all submissions may be stored in a folder named `quiz1` and the file names for GiaBayes may be `analysis-GiaBayes.qmd` and `functions-GiaBayes.R`, those for lee-kim may be `analysis-lee-kim.qmd` and `functions-lee-kim.R`, etc. As another example, both files could have the same name but be stored in a folder with naming specific to the student, e.g., files `analysis.qmd` and `functions.R` may be located in folders `quiz1-GiaBayes`, `quiz1-lee-kim`, etc. In the above cases, when grading we could provide `example_student_identifier = 'GiaBayes'` and the vector `example_assignment_path = c('quiz1/analysis-GiaBayes.qmd', 'quiz1/functions-GiaBayes.R')` for the former case or `example_assignment_path = c('quiz1-GiaBayes/analysis.qmd', 'quiz1-GiaBayes/functions.R')` for the latter case.

Provided that student submissions are stored with these regular file paths, besides iterating through all submissions, gradetools can also retrieve the file(s) of one or more *specific* students, for example because we wish to grade or re-grade only their submissions. The functions `assist_advanced_grading()` or `assist_regrading()` have optional arguments, respectively `students_to_grade` and `students_to_regrade`, that can be set to a single string or to a vector of strings specifying the identifiers of the students we wish to (re-)grade.

4.2 Assigning grade and feedback based on rubric

Once the submission file(s) of a students have been retrieved and opened, we need to read (i.e., evaluate) the students' work. For each component of the assignment, we must choose which items from the rubric we want to apply, possibly add new items to the rubric, and



(a) Unformatted rubric

	A	B	C	D	E	F
1	name	total_points	prompt_code	prompt_message	feedback	points_to_remove
2	Question 1	2	1a	Didn't use inline code	Inline code was not used to obtain the number of observations and variables. Inline code is useful for avoiding small mistakes like typos, as well as reporting current values after data, or an analysis, has been updated.	1
3	Question 1		1b	Misidentified observation	A single observation in this data set represents a single bike rental.	1
4	Question 2	3	1a	Plotted counts	Bar plots of counts can make it hard to see patterns within groups with relatively small counts. A bar plot of proportions would be more ideal for this case, and can be specified by setting the `position` argument in `geom_bar()` to "fill".	0.25
5	Question 2		2a	Didn't answer question	You should have included an interpretation of what the plot shows in relation to the question "Does bike type seemed to vary by passholder type?"	2
6	all_questions		404	Unattempted	This question was left unanswered.	100
7	all_questions		100	Correct	Your answer was correct!	0
8	general_feedback		1	Good tidyverse style!	Great job adhering to the tidyverse style guidelines!	

rubric

(b) Formatted rubric

Figure 4: Rubric from grading example in its readable form and formatted to be compliant with gradetools rubric requirements.

ideally give the student some qualitative feedback specific to their work. Without a system like gradetools, these tasks may be extremely time and energy consuming, as they require to simultaneously refer to and possibly edit multiple grading files (the submission files, the rubric, the grade sheet and some file where we write feedback). As illustrated with the grading example of Section 3.2 (e.g., see Figure 1), the strategy we adopted with gradetools enables to visualize the available rubric items directly in RStudio, next to the open file(s), and only requires the grader to enter one or more short prompts corresponding to the rubric items to be applied to the submission. In this process, on the back end, the grade sheet gets updated and a feedback file gets written for each student that is graded.

The key to this automation is setting up a rubric that has all the necessary information to (i) make grading prompts, such as those shown in Figure 1 and Figure 2; and, based on the choices made by the grader, (ii) compute the grade and (iii) assign personalized feedback. Without gradetools, when grading the example assignment from Section 3 a grader may use a rubric such as the one displayed in Figure 4a. This rubric breaks down the assignment into components (in this case, questions) and, for each component, specifies its assigned points (e.g., 2 points for Question 1). Each item of the rubric represents a potential error (e.g., Mispecified observation) and notes the points to remove for that error (e.g., -1). The rubric required by gradetools encodes all this information, plus additional information to make rubric prompts and write feedback. Figure 4b displays the rubric provided to gradetools in the examples of Section 3. Each entry represents a rubric item, and for each item it specifies (i) what the item applies to (an assignment component/question, all questions, or the assignment overall); (ii) the total number of points for the component/question (only for items specific to a component/question); (iii) the prompt code the grader needs to type to apply the item; (iv) the prompt message to be shown next to the prompt code; (v) the

Table 3: The specifics of each entry (item) in the rubric, in the case of an item applicable to all questions.

Specifics	Description	Example
Name	What question the item is available for	All questions
Prompt code	What the grader enters to apply this item	1
Prompt message	The description that the grader sees while grading	tidyverse code style
Feedback	What the student sees when they receive feedback	Please adhere to the Tidyverse style guide, as discussed in Lecture 1.
Points to remove	The penalty applied when this item is selected	0.5 points

feedback to apply when the item is selected; and (vi) the points to remove from (or to add to) the total grade when the item is selected. The function `create_rubric_template()` creates an empty csv file with the necessary column names to aid in formatting the rubric.

The meaning of each column of the rubric formatted for gradetools is further illustrated in Table 3, with an example rubric item applicable to all questions (note that total points are left blank for items applicable to all questions, as shown in Figure 4b). Let's assume that the grader wants to provide the feedback "Please adhere to the Tidyverse style guide" for a specific question or for the overall assignment. This is the feedback that the student will see. However, writing out this comment each time the grader encounters a submission that needs such feedback is time consuming. To save time, gradetools utilizes a short code, that we denote prompt code, in this case defined as 1. By selecting 1, the grader is able to provide the full length comment and apply the corresponding score policy. To be quick-to-enter, prompt codes can be short and uninformative and therefore difficult to remember, so a short prompt message can be shown along with the prompt code to remind the grader what feedback the prompt code corresponds to.

In gradetools, while grading, the grader is able to see the prompt message and the prompt

code, while the student will be able to see the corresponding personalized and extended feedback when feedback files are returned. As shown in Figure 2, the grader can quickly indicate (possibly multiple) items to apply among available ones in the rubric.

As anticipated in Section 2, linking each rubric item with some corresponding feedback allows gradetools to scale the provision of detailed individualized feedback. In addition to expediting provision of repeatable feedback across different questions and students, gradetools supports the provision of unique feedback on the fly and the dynamic update of the rubric, for which prompt codes "p" and "r" are reserved, respectively (as shown in Figure 1).

4.3 Maintaining the grade sheet and the feedback files

	A	B	C
1	**student_identifier**	GiaBayes	lee-kim
2	**feedback_path_Rmd**	feedback/GiaBayes.Rmd	feedback/lee-kim.Rmd
3	**feedback_path_to_be_knitted**	feedback/GiaBayes.html	feedback/lee-kim.html
4	**assignment_path**	quiz1-GiaBayes.qmd	quiz1-lee-kim.qmd
5	**assignment_missing**	FALSE	FALSE
6	**grading_status**	all questions graded	all questions graded
7	**feedback_codes**	100&&&1a&&&1	1a&&&1a---2a&&&NA
8	**graded_qs**	Question 1&&&Question 2&&&general_feedback	Question 1&&&Question 2&&&general_feedback
9	**last_time_graded**	2024-02-28T14:58:10Z	2024-02-28T14:58:29Z
10	**comments**	NA	NA
11	**comment_qs**	NA	NA
12	**grade_student**	TRUE	TRUE
13			

grading-progress-log

Figure 5: Screenshot of grading progress log file, at the end of grading two students as in the example from Section 3.2.

Finally, we consider how gradetools assists with maintaining the grade sheet and feedback files. Grading progress is recorded by gradetools in a *grading progress log file*. Figure 5 shows the content of this file for the example of Section 3.2. When beginning to grade a new assignment, this file is created with a row for each student in the provided roster (displayed as column in Figure 5, for readability), including information such as where assignments

files are located and indicating that all students are ungraded. When grading a student's submission, the grading log file is dynamically updated to keep track of any prompt code entered by the grader and eventually students grading statuses change from "ungraded" to "all questions graded". The grading progress log file is not meant to be edited directly by the grader, but is used by gradetools to keep track of the grading progress so that grading can be interrupted and resumed at any time. Specifically, as shown in Figure 5, the grading progress log saves all rubric prompt codes that have been applied by the grader. At the end of the grading process, as discussed in Section 4.2, applied prompt codes are used in combination with the rubric to produce the grades for the final grade sheet and the feedback for each student's feedback file, e.g., the ones shown in Figure 3. Only storing the selected prompt codes allows the grader to change the grade and feedback corresponding to any rubric item, anytime until grading is completed, and see the changes be reflected in the feedback files and final grade sheet (as mentioned in Section 3.2). Importantly, a grader should never change prompt codes that have already been used, as this would result in the loss of grading progress.

At the end of grading, a feedback file is created for each student. When calling gradetools' functions, the grader specifies where the feedback files should be stored by providing an example feedback path. Similarly as with the example assignment path described in Section 4.1, the provided student identifier must be present in the provided example feedback path, so that files with feedback for different submissions can always be distinguished. Information on where the grader wishes to store feedback files is also recorded in the grading progress log, as shown in Figure 5.

Since there are a plethora of ways instructors distribute grades and feedback, automated delivery of grades and feedback files to the students was determined to be outside of

gradetools' scope, with the exception of pushing feedback files to GitHub. In the next section, we provide some alternatives in the R ecosystem.

5 Discussion

In Section 2 we have discussed the grading workflow as a three-step process: preparation, grading and providing feedback, and finalization. In Section 3 we have shown what automation of this grading workflow looks like with the R package gradetools and in Section 4 we have detailed how gradetools supports this automation. Figure 6 summarizes gradetools functions, emphasizing the pedagogical tasks, and noting the tasks managed by each function. From this figure, it is evident that gradetools has some limitations and does not serve every step of the grading process.

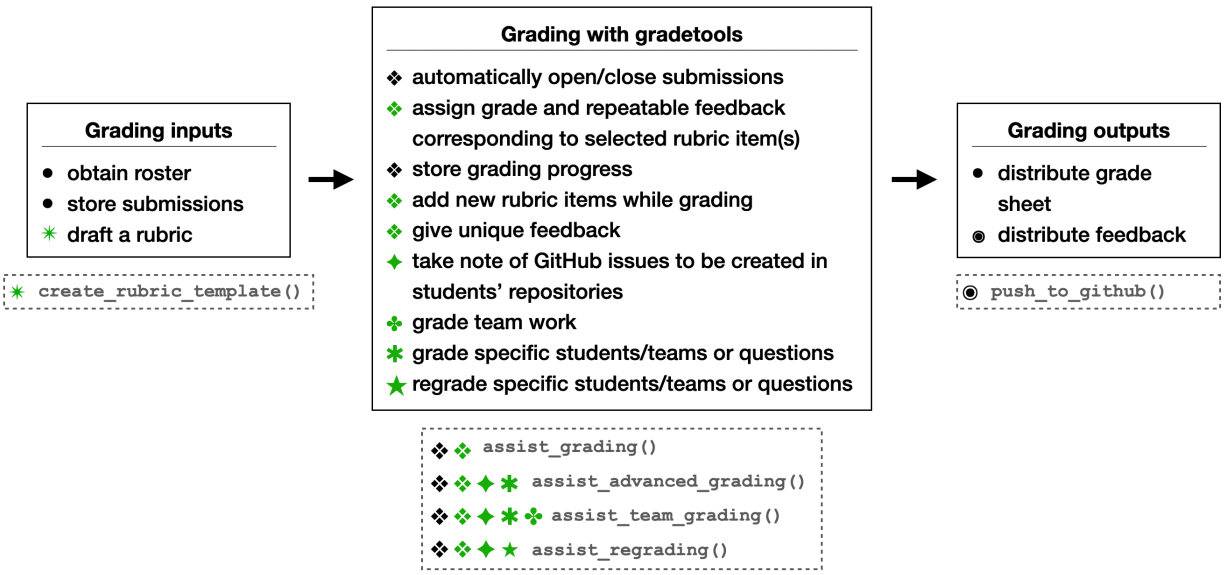


Figure 6: Diagram of automated grading workflow using gradetools in the grading step. Administrative tasks in black and pedagogical tasks in green. Special bullet shapes map tasks that gradetools assists with to the name of the functions that support them. *Repeatable* and *unique* feedback refer to the concepts defined in Section 2.1.

In terms of inputs, graders will need to provide the roster and students' submissions and

gradetools does not support retrieval of them. In terms of outputs, the package provides feedback files for each student and the overall gradesheet. Gradetools only supports returning of feedback files to students via GitHub. For the tasks that are not supported by gradetools, there are many R packages that can facilitate these tasks such as **rcanvas** (Ranzolin et al. 2017), **moodleR** (Dietrichson 2022), **ghclass** (Rundel & Cetinkaya-Rundel 2022), **gmailr** (Hester & Bryan 2023) for working with the Canvas, Moodle, GitHub, and GMail Application Programming Interfaces (APIs) respectively. These packages can support tasks like retrieving students' work and returning students' scores and feedback. Some of the Learning Management Systems (LMSs) may also provide interfaces that allow bulk downloads and uploads manually.

The gradetools package focuses mainly on the second step of the grading workflow by improving the grading and feedback process through automating the administrative tasks. The most important benefit of using gradetools is that it helps adopt an efficient and fair grading workflow. Even though we did not study it rigorously, in terms of efficiency, gradetools saves a lot of time in grading once the initial learning curve has been passed. In terms of fairness, the fact that gradetools enforces use of a rubric allows for consistent grading and feedback across different students and questions. Use of rubrics are pivotal to fairness especially in performance-based assessments (Shepherd et al. 2008).

In summary, gradetools automates many administrative tasks in the grading workflow with many pedagogical considerations but it is by no means a single solution to a fully automated grading workflow. Instructors who are interested in fully automating the grading workflow, would need to be proficient in R and rely on packages other than gradetools. For instance, if an instructor downloads files from an LMS they might need to do string manipulation to have filing name consistency across different students' file names. In our courses, we

have managed to fully automate our grading workflow by supplementing gradetools with GitHub features of ghclass (Rundel & Cetinkaya-Rundel 2022) and data wrangling features of the tidyverse packages (Wickham et al. 2019).

The gradetools package can be an important addition to an instructor's toolkit, especially to support reproducibility. With a stronger emphasis on teaching of reproducibility skills, students working directly in literate programming notebooks such as Quarto and learning to manage and name these files are important parts of their data science training (Pruim et al. 2023). In these teaching scenarios, it is important to use grading-workflow tools like gradetools, that allow to provide students with feedback on their coding practices in addition to their coding outputs. Along with teaching reproducibility, reproducible teaching is also important (Dogucu & Çetinkaya-Rundel 2022). The gradetools package allows for grading code and files (e.g., rubric) to be reused (or modified) from one academic term to the next, and to be picked up by other instructors or TAs grading the same assignment.

Besides increased emphasis on reproducibility, another important change in data science education that can motivate the use of automated-grading-workflow tools is the development of generative artifical intelligence (AI) tools. Even though there is not a consensus on how AI tools can benefit the learning process, examples of use show that AI is here to stay to be part of the learning process (Ellis & Slade 2023). With students' increased use of AI, we expect instructors to modify some of their assignments to alternative formats with more reliance on open-ended assessments, deriving a greater need for tools like gradetools.

Contrary to some other grading tools, since gradetools is an R package, it is free to use. It does not require internet connection during the extensive period of grading and providing feedback. However, users would still need internet connection in the first and third steps of the grading workflow.

When students' work is considered, needless to say privacy is important (e.g., grades) and can be protected under law depending on the country. In our grading process we have used the package on our local computers and stored the grade sheets and other private documents locally. However, it is worth noting that users who choose to use R and gradetools on different platforms such as in the Cloud will need to be mindful of what they are storing, what is legal and ethical to store in that specific platform.

In this paper, in addition to introducing gradetools and how it can be utilized in data science classes, we have also shared our vision of an automated grading workflow and defined the distinction between pedagogical tasks and administrative tasks in grading, defined different feedback types such as unique and repeated ones. We will continue to maintain gradetools for use in our own data science courses and beyond. We hope that this work will help the community of data science and statistics educators use gradetools as their grading workflow assistant or develop their own tools for assisting their grading workflow.

References

Beckman, M. D., Çetinkaya-Rundel, M., Horton, N. J., Rundel, C. W., Sullivan, A. J. & Tackett, M. (2021), 'Implementing version control with git and github as a learning objective in statistics and data science courses', *Journal of Statistics and Data Science Education* **29**, S132–S144.

Black, P. & Wiliam, D. (1998), 'Assessment and classroom learning', *Assessment in Education: Principles, Policy & Practice* **5**(1), 7–74.

Blank, D. S., Bourgin, D., Brown, A., Bussonnier, M., Frederic, J., Granger, B., Griffiths, T. L., Hamrick, J., Kelley, K., Pacer, M. et al. (2019), 'nbgrader: A tool for creating and

grading assignments in the jupyter notebook', *The Journal of Open Source Education* **2**(11).

Carless, D. (2006), 'Differing perceptions in the feedback process', *Studies in Higher Education* **31**(2), 219–233.

Cetinkaya-Rundel, M., Dogucu, M. & Rummerfield, W. (2022), 'The 5ws and 1h of term projects in the introductory data science classroom', *Statistics Education Research Journal* .

Dietrichson, A. (2022), *moodleR: Helper Functions to Work with 'Moodle' Data*. R package version 1.0.1.

Dixson, D. D. & Worrell, F. C. (2016), 'Formative and summative assessment in the classroom', *Theory into practice* **55**(2), 153–159.

Dogucu, M. & Çetinkaya-Rundel, M. (2022), 'Tools and recommendations for reproducible teaching', *Journal of Statistics and Data Science Education* **30**(3), 251–260.

Dogucu, M. & Çetinkaya-Rundel, M. (2021), 'Web scraping in the statistics and data science curriculum: Challenges and opportunities', *Journal of Statistics and Data Science Education* **29**, S112–S122.

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), 128–133.

Fiksel, J., Jager, L. R., Hardin, J. S. & Taub, M. A. (2019), 'Using github classroom to teach statistics', *Journal of Statistics Education* **27**(2), 110–119.

GAISE (2016), 'Guidelines for assessment and instruction in statistics education (GAISE): College report'.

Galassi, A. & Vittorini, P. (2021), Automated feedback to students in data science assignments: improved implementation and results, *in* 'CHItaly 2021: 14th Biannual Conference of the Italian SIGCHI Chapter', pp. 1–8.

Garcia, D. D. (2015), 'Tech launch with gradescope: exam grading will never be the same again!', *ACM Inroads* **6**(2), 82–83.

Garfield, J. B. & Gal, I. (1999), 'Assessment and statistics education: Current challenges and directions', *International statistical review* **67**(1), 1–12.

Hattie, J. & Timperley, H. (2007), 'The power of feedback', *Review of educational research* **77**(1), 81–112.

Hester, J. & Bryan, J. (2023), *gmailr: Access the 'Gmail' 'RESTful' API.* R package version 2.0.0.

Hu, J. & Dogucu, M. (2022), 'Content and computing outline of two undergraduate bayesian courses: Tools, examples, and recommendations', *Stat* **11**(1), e452.

Irons, A. & Elkington, S. (2021), *Enhancing learning through formative assessment and feedback*, Routledge.

Johnson, K. G. (2016), Incorporating writing into statistics, *in* 'Mathematics Education', Springer, pp. 319–334.

Juwah, C., Macfarlane-Dick, D., Matthew, B., Nicol, D., Ross, D. & Smith, B. (2004), 'Enhancing student learning through effective formative feedback', *The Higher Education Academy* **140**, 1–40.

Krause, U.-M., Stark, R. & Mandl, H. (2009), 'The effects of cooperative learning and feedback on e-learning in statistics', *Learning and instruction* **19**(2), 158–170.

Lipnevich, A. A. & Smith, J. K. (2009), '"i really need feedback to learn:" students' perspectives on the effectiveness of the differential feedback messages', *Educational Assessment, Evaluation and Accountability* **21**, 347–367.

Loy, A., Kuiper, S. & Chihara, L. (2019), 'Supporting data science in the statistics curriculum', *Journal of Statistics Education* **27**(1), 2–11.

National Academies of Sciences, Engineering, and Medicine (2018), *Data science for undergraduates: Opportunities and options*, National Academies Press.

Nicol, D. J. & Macfarlane-Dick, D. (2006), 'Formative assessment and self-regulated learning: A model and seven principles of good feedback practice', *Studies in higher education* **31**(2), 199–218.

Pai, G. (2024), 'Using Formative Assessment and Feedback from Student Response Systems (SRS) to Revise Statistics Instruction and Promote Student Growth for All', *Journal of Statistics and Data Science Education* **0**, 1–15.

Perera, J., Lee, N., Win, K., Perera, J. & Wijesuriya, L. (2008), 'Formative feedback to students: the mismatch between faculty perceptions and student expectations', *Medical teacher* **30**(4), 395–399.

Pruim, R., Gîrjau, M.-C. & Horton, N. J. (2023), 'Fostering better coding practices for data scientists', *Harvard Data Science Review* **5**(3).

Pyles, C. & UC Berkeley Data Science Education Program (2022), *Otter-Grader: A Python and R autograding solution*. Version 3.2.1.

Race, P. (2001), 'Using feedback to help students to learn', *The Higher Education Academy* .

Ragupathi, K. & Lee, A. (2020), Beyond fairness and consistency in grading: The role of rubrics in higher education, *in* 'Diversity and inclusion in global higher education', Palgrave Macmillan, Singapore, pp. 73–95.

Ranzolin, D., Hua, C., Solt, F., van Atteveldt, W. & Hathaway, J. (2017), *rcanvas: R Client for Canvas API*. R package version 0.0.0.9001.

Reck, R. M. (2019), 2019 asee annual conference & exposition.

Ricci, F. Z., Medina, C. M. & Dogucu, M. (2022), *gradetools: Tools to Assist with Providing Grades and Personalized Feedback to Students*. R package version 0.2.0.

Rundel, C. & Cetinkaya-Rundel, M. (2022), *ghclass: Tools for Managing Classes on GitHub*. R package version 0.2.1.

Schloerke, B., Allaire, J. & Borges, B. (2020), *learnr: Interactive Tutorials for R*. R package version 0.10.1.

Shepherd, C. M., Mullane, A. M. et al. (2008), 'Rubrics: The key to fairness in performance based assessments', *Journal of College Teaching & Learning (TLC)* **5**(9).

Singh, A., Karayev, S., Gutowski, K. & Abbeel, P. (2017), Gradescope: A fast, flexible, and fair system for scalable assessment of handwritten work, L@S '17, Association for Computing Machinery, New York, NY, USA, p. 81–88.

Timmerman, B. E. C., Strickland, D. C., Johnson, R. L. & Payne, J. R. (2011), 'Development of a 'universal'rubric for assessing undergraduates' scientific reasoning skills using scientific writing', *Assessment & Evaluation in Higher Education* **36**(5), 509–547.

Wickham, H., Averick, M., Bryan, J., Chang, W., McGowan, L. D., François, R., Grolemund, G., Hayes, A., Henry, L., Hester, J. et al. (2019), 'Welcome to the tidyverse', *Journal of open source software* **4**(43), 1686.

Woodard, V., Lee, H. & Woodard, R. (2020), 'Writing assignments to assess statistical thinking', *Journal of Statistics Education* **28**(1), 32–44.

Yen, M., Karayev, S. & Wang, E. (2020), Analysis of grading times of short answer questions, L@S '20, Association for Computing Machinery, New York, NY, USA, p. 365–368.