Designing and implementing an automated grading workflow 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

August 17, 2023

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 qualitative 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 define as an 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. 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.

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

[*]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, review and consolidation. This is crucial for both students and instructors to make informed decisions on how to improve, respectively, their learning and teaching strategies.

In order to be fully effective, assessments must receive feedback (Nicol & Macfarlane-Dick 2006, Perera et al. 2008, Irons & Elkington 2021). Feedback can broadly be categorized into qualitative (e.g. "The source of the data utilized in your analysis was not specified.") and quantitative (e.g. "9 out of 10"). To distinguish the former from the latter, in this paper we will reserve the term *feedback* for qualitative comments - given by the grader to students in response to assessment activities - , 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). Feedback, when done well, can support learning (Hattie & Timperley 2007) especially for students with little prior knowledge (Krause et al. 2009). Impactful feedback should be provided in a timely manner, should be constructive and specific to the student's work (Juwah et al. 2004, Race 2001).

Providing accurate feedback and grades to students' work can 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). Some examples of currently available tools include the Python package `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, and Otter-Grader (Pyles & UC Berkeley Data Science Education Program 2022) for grading Python and R assignments. The use of automated tools can be highly beneficial for grading (possibly closed-form) assessments involving definitions, calculations and coding where there is a correct and an incorrect answer, and it can allow for giving real-time responses to formative assessments as recommended by the GAISE college report (Carver et al. 2016).

However, providing feedback to other important types of assignments requires human judgement and still remains challenging. Indeed, many recommended assessment types, including lab reports (Carver et al. 2016), semester long projects (Carver et al. 2016, Cetinkaya-Rundel et al. 2022), and writing assignments (Woodard et al. 2020, Johnson 2016) fall outside of the framework of correct/incorrect solutions and are not amenable to automatic feedback. 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). They also present two major challenges: they tend to require more time for grading and providing feedback, and tend to be more difficult to be graded 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 paper, we consider assignments (e.g. final projects) in which human judgment is required in grading and providing feedback. We outline the steps of a typical grading workflow and highlight which of them can be automated in an approach that we define as *automated grading workflow*. We illustrate how the new R package **gradetools** implements this approach within the RStudio Graphical User Interface to enable efficient and consistent grading while providing individualized feedback.

2 Automating the grading workflow

In this section, we outline the steps of a typical grading workflow and we classify them into administrative or pedagogical tasks. We define what *automated grading workflow* means in the context of this article, and then we focus on pedagogical tasks, to identify a key source of repetitivity that technology can address to help make grading faster and smoother.

2.1 The three steps of a typical grading workflow

It is useful to start by outlining and classifying the tasks involved in a typical grading process. We can break down a grading process into the following three phases, each with their respective tasks:

1. Preparation: collecting students' assignments, drafting a rubric, setting up a grade sheet where grades can be recorded for each student;

2. Grading and providing feedback: for each student, retrieving and opening their submission, browsing and evaluating their submission, considering what grade and feedback to assign them given the current rubric, possibly updating the rubric, referring to

the grade sheet to find the student and annotate their grade, closing the submission;

3. Finalization: uploading the grade sheet on the class' learning management system, returning grades and feedback to students.

The tasks listed above can be grouped broadly into two distinct types: *pedagogical* and *administrative* grading tasks. Pedagogical tasks are those that have a direct impact on what students learn from their grade and their feedback. Administrative tasks are those that are directly related to the logistics involved with the grading process and do not have a direct impact on what is learned from the outputs of grading.

In fact, most of the tasks listed before are administrative in essence. For example, during Step 1, the only task involving pedagogical decisions is drafting a rubric that establishes what is considered as correct or incorrect and what is the importance of each component of an answer. Conversely, collecting submissions and setting up the grade sheet are administrative tasks. All other pedagogical decisions are performed in Step 2 of the grading process. Providing feedback and assigning a score is a pedagogical task for open-ended assignments. While providing feedback and scores, it is possible that a rubric may need to be updated and this would also be a pedagogical task.

Table 1 summarizes the three steps of a grading process, the tasks they involve and their type. We would like to make a remark: what we have identified as administrative choices can have (indirect) pedagogical impacts, that should be considered when designing and evaluating a grading workflow. For example, the way feedback is distributed can make it easy or difficult for a student to access their feedback, which in turn affects the chances that students will learn from it.

Administrative tasks tend to be repetitive and mostly do not require human judgement

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

1. Preparation	**2. Grading and Feedback**	**3. Finalization**
Collecting students' assignments	Retrieving and opening a submissions	Uploading grade sheets on class' learning management system
Drafting a rubric	Evaluating the submission	Returning grades and feedback to students
Setting up a grade sheet	Selecting grade and feedback from rubric Updating rubric as needed Finding student corresponding to this submission on the grade sheet	
	Closing the submission	

during their execution; in fact, automating administrative tasks 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, consider how the rubric should be applied to a given submission - require human judgement for the types of assignment that we consider in this work (see Introduction). Even when they do require human judgement, there are sources of repetitivity involved in executing pedagogical tasks that a system can be designed to automate. In this sense, we consider an automated grading workflow as a system that automates all or most repetitive grading tasks so as to reduce the time and effort required for a grader to provide high-quality feedback.

One way of designing an automated grading workflow leverages the fact that a feedback that is individualized to some feature of a student's submission is in fact applicable to all students whose submissions present the same feature. For each feature that a grader wishes to provide a feedback about, a rubric can contain an item that indicates both the feedback

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.

Student applicability	Question applicability	Example
multiple	single	When interpreting the slope coefficient make sure to use units of measurement.
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.

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. In the next two subsections we provide more considerations regarding feedback types and rubric.

2.2 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 personalized, in the sense that they are specific to a student's submission - rather than summarizing how the whole class did on the assignment. However, some of them 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. Moreover, whether they are repeatable or unique to a submission, feedback can be applicable to a single component of the assignment (or question), applicable to multiple components (or questions), 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 very rarely needed but we commonly encounter the other feedback scenarios. Whenever possible, while also allowing to provide unique feedback on the fly, an automated grading workflow should facilitate the provision of repeatable feedback across different questions and students. A rubric can be structured to serve this purpose by including information on whether each rubric item is applicable to a single, multiple questions or to the entire assignment.

2.3 Rubric

An automated grading workflow starts with the creation of a (minimal) rubric. By using a rubric, we try to (i) achieve consistency in grading and (ii) reduce grading time, by associating both a feedback and its corresponding score to a quick-to-select prompt. We present an example of such a rubric in Table 3. 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, an automated grading workflow 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. Prompt codes tend to be 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. While grading, the grader is able to see the prompt message and the prompt code, while the feedback is shown to the student when feedback is returned.

Table 3: The components of each entry (item) in the rubric, assuming that each entry corresponds to a feature of an assignment that a grader may want to remark.

Component name	Description	Example
Applicability	What question the item is available for	All questions
Feedback	What the student sees when they receive feedback	Please adhere to the Tidyverse style guide, as discussed in Lecture 1.
Prompt message	What the grader sees while grading	tidyverse code style
Prompt code	What the grader indicates	1
Grade	The penalty applied when this item is selected	0.5 points

An automated grading workflow then requires a system that allows a grader to quickly indicate (possibly multiple) items to apply among available ones in the rubric, and automatically annotates their corresponding score on the grade sheet and their corresponding feedback on a feedback file. Ideally, such system would also facilitate editing the rubric, for example to add new rubric items as new submissions get evaluated.

3 The gradetools package: considerations and capabilities

Before discussing the details of the package, it is helpful to have an understanding of what the grading process with gradetools looks like. Figure 1 shows an example of gradetools being used to grade a homework assignment. To begin, the gradetools package is loaded and the grading function is called. This triggers the grading of the first student on the roster and opens their submission automatically. Grading is an interactive process where the grader is prompted in the console to grade the submission according to the provided rubric. To specify which rubric item(s) to apply, the user provides the associated prompt

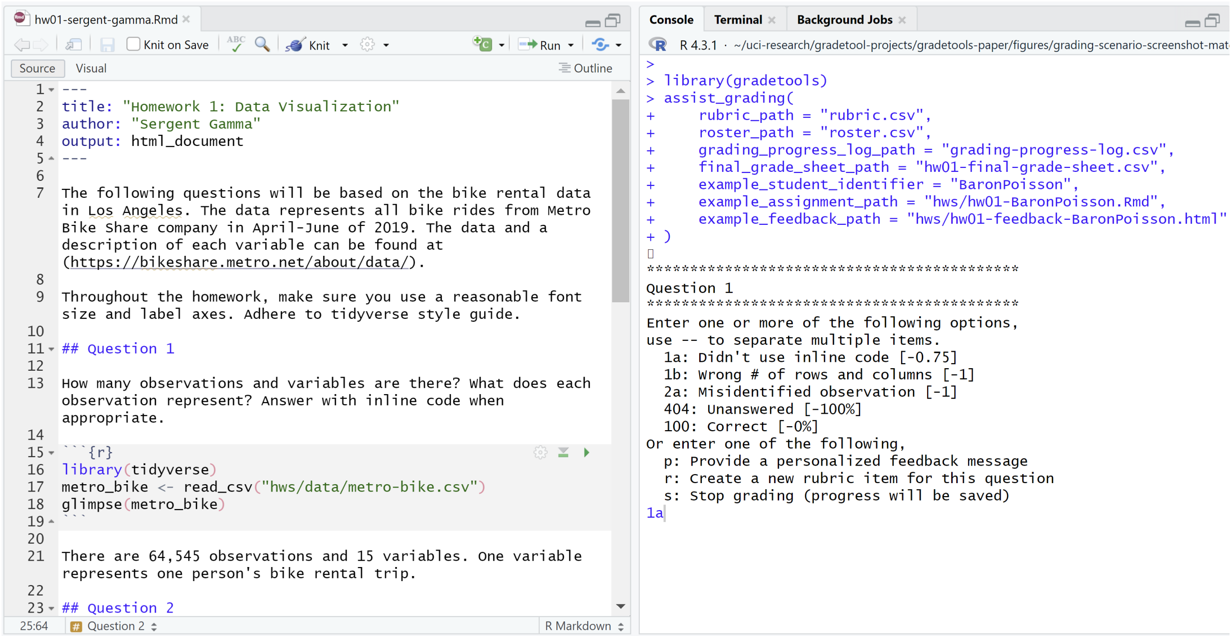

Figure 1: Screen shot of gradetools being used to grade homework in RStudio. On the left is the student's assignment which was automatically opened. On the right in the console is the grading function call and gradetools prompting the user to grade according to the provided rubric.

code. In the example displayed in Figure 1 the supplied prompt code is "1a", to specify that the students should be reminded to use in-line code (and perhaps how to use it) with some feedback and 0.75 points should be deducted for not using inline R code as instructed for Question 1. After prompt codes are entered into the console, the grader is prompted to grade the next question.

Once all questions have been graded the submission closes and the process begins again with the next student in the roster. Throughout the grading process, gradetools utilizes a single internal file containing all associated data enabling dynamic rubric creation and alteration, recording of assigned rubric items, ability to pause and resume grading, and many more features. Upon completion of grading, or termination of the grading process, a final grade sheet is created and feedback files are written from the feedback associated with the applied rubric items for each student. Figure 2 visualizes this guided grading process where the emphasis is on pedagogical tasks, and the record keeping is performed silently by gradetools.

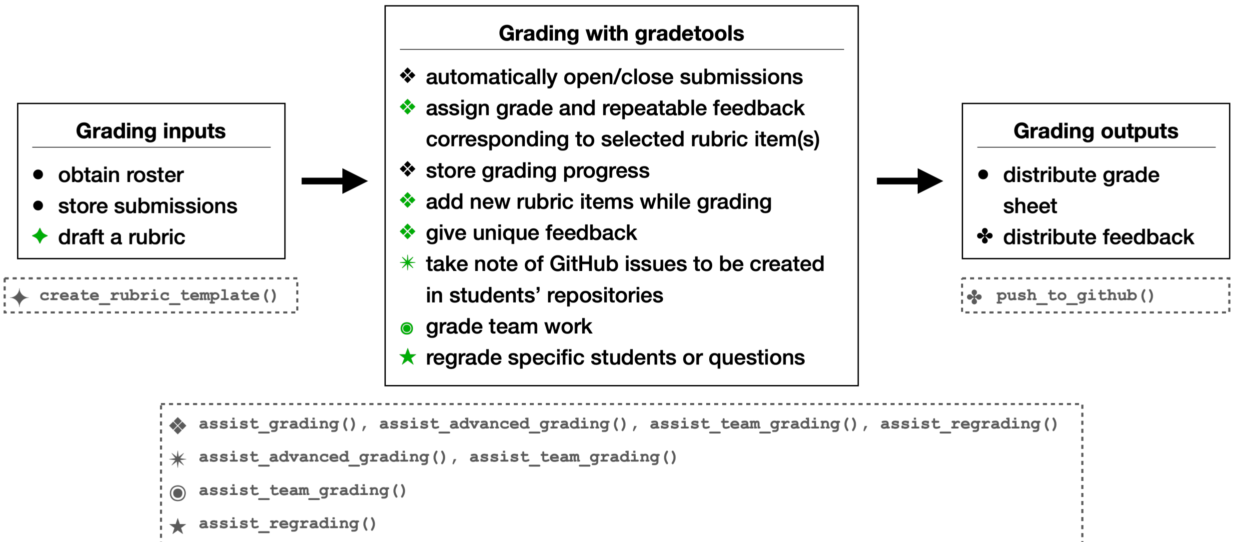

Figure 2: Diagram of automated grading workflow using gradetools in the grading step. Administrative tasks in black and pedagogical tasks in green. Special bullet shapes map tasks that gradetools assists with to the name of the functions that supports them. *Repeatable* and *unique* feedback refer to the concepts defined in Section 2.2.

To avoid repetition, the reminder of this section will discuss the conceptualization and abilities of gradetools, whereas the vignettes available on the package website (https://federicazoe.github.io/gradetools/) provide step-by-step tutorials on how to use the package.

3.1 Preparation

When developing the package, there were considerations of what is typically required to grade, and how user burden could be lessened while maintaining flexibility. To make the package accessible to a wide range of graders, a function, `assist_grading()`, was created that requires little `R` knowledge and minimal input to begin grading while allowing to grade all submissions for an assignment. Instead of providing data frames and long vectors, this grading function only requires the locations - i.e., the computer file paths - for the inputs and outputs of the grading process, as summarized in Figure 2.

The example from Figure 1 shows that the grading function needs to know where the rubric and roster are located, and where to create the grading progress log and the final grade sheet. Additionally, an identifier, assignment submission location, and feedback file path for a single student has to be provided.

3.1.1 The roster

The class roster is used by gradetools to determine how many student submissions should be expected, which ones are missing, and what to include in the final grade sheet. Any column present in the class roster will be copied over to the final grade sheet produced. There is only one requirement for the provided roster. The roster must contain a column labeled "student_identifier" which contains some identifier unique to the student such as a full name, student ID number, or GitHub username.

3.1.2 The assignment submission and feedback locations

There are a myriad of ways that assignment submissions can be collected and stored, and this can be dictated by (one or a mix of) institutional policy, software, and preference. We wanted to allow for different directory structures while limiting the information the user had to provide about where assignments are located. As discussed, our solution only requires the student identifier, assignment file path, and feedback file path for a single student to be specified.

The assignment file path and the student identifier are used to create a template to determine all other assignment file paths. Every instance of the provided student identifier is located in the provided assignment file path, and the student identifiers from the class roster are substituted into those instances to obtain a file path for each student. This requires that the student identifier is present in the student's assignment file path and that is the only part unique to that student's assignment file path.

One example of a compliant directory structure is the case where all submissions are stored in the same folder with the student identifiers in the name of the files. This is the case in the example from Figure 1, where the homework submissions are stored in a folder called "hws" and are named "hw01-BaronPoisson.Rmd", "hw01-sergent-gamma.Rmd", and "hw01-student_T.Rmd". The corresponding student identifiers are "BaronPoisson", "sergent-gamma", and "student_T" respectively. The user provided "BaronPoisson" and "hws/hw01-BaronPoisson.Rmd" as the example student identifier and assignment file path. The assignment file paths for the remaining student are inferred by gradetools to be "hws/hw01-sergent-gamma.Rmd" and "hws/hw01-student_T.Rmd".

In this example, we have shown a case where each student submits a single R Markdown file but this system also allows for submissions where each student has a folder and multiple files within their folder. This feature is especially helpful when students are graded on more extensive projects with multiple files and with GitHub submissions where each student's repository would be a folder.

At the end of grading, a feedback file is created for each student. The feedback file path operates similarly and with the same restrictions as the assignment submission file path. This restriction on the assignment submission and feedback file paths does limit the types of acceptable directory structures, but it also greatly reduces the user burden by only requiring a single submission and feedback path to be provided to gradetools.

3.1.3 The gradetools rubric

The user provided rubric is central to the gradetools workflow. The rubric is used to prompt the grader, calculate grades and write feedback for students based on the rubric items selected when grading. Due to its importance, the rubric provided to gradetools has rigid specifications.

There are five required columns in the rubric. The user has to specify what the rubric item applies to, either: a specific question, all questions, or the assignment overall. Total number of points for the question, prompt code, prompt message, and feedback for the rubric item also need to be specified. Lastly, the grader needs to decide whether they would utilize positive or negative grading and then provide points to add or subtract as a column. Question names, total points, feedback, and points to remove or add are components typical to any rubric. Whereas, prompt codes and prompt messages are additional rubric components specific to gradetools that are used when prompting the user to grade, as defined at the end of Section 2.3. The function `create_rubric_tempalte()` creates an empty csv file with the necessary column names to aid in formatting the rubric.

Formatting the rubric for gradetools can take some time, but all of these specifications allow the grading package to manage the administrative task so the user can focus on the pedagogical decisions of grading.

3.2 Grading and Feedback

There are three grading functions available in the package. The `assist_grading()` function requires minimal user input while the `assist_advanced_grading()` has more arguments allowing advanced users more flexibility. The `assist_team_grading()` has all of the functionality as the advanced grading function, but is for the special case when multiple students share a submission and the corresponding grade, such as a group project.

We will first focus on the functionalities integral to the grading process, and then we will discuss the additional capabilities of gradetools.

3.2.1 Core grading functionalities

For each question in the assignment the grader is prompted with the options to grade the question, write a personalized message, add a new rubric item, or to end the grading process.

The prompting contains all rubric items specific to a question, and rubric items specified as applicable to all questions. For each rubric item the user is prompted with its prompt code, prompt message, and the corresponding points to remove. To grade, a user selects one or more rubric items by providing the corresponding prompt codes to the console.

Before grading a question, a grader can choose to write a personalized feedback message, unique to the student and pertaining to that question. An example that motivated the creation of this option is when a student leaves a message or question to the grader within the assignment, as in the example of Table 2.

Since grading is not always a linear process, the rubric provided at the beginning can be changed. While grading, the user always has the option to create new rubric items. If selected, the user is prompted to provide all necessary information for a rubric item. Afterwards, the user will again be prompted to grade the question with the newly updated rubric items also available.

If a grader wished to edit the existing rubric items, they would have to stop grading and directly edit the rubric file. The grading progress log does not store grades, only the prompt codes associated with the selected rubric items for each question. This allows for the total points, point deductions, prompt and feedback messages to be changed in the rubric. For instance, if a grader decided that a rubric item should be worth fewer points then this can be easily modified. These updates would be reflected when the grading process is resumed with the new rubric provided. It is important to note that the question names and prompt codes should not be changed once grading has begun because this is how grading progress is recorded.

The assignment file paths are used to automatically open a student's submission when the user is prompted to grade a student, and to automatically close the file after the student has been completely graded. Not only does this save time on a tedious task, but it also removes the chance of the possible error of opening the wrong submission. While the console is busy with the interactive grading process, an open student submission can be rendered (e.g., Quarto files) or knitted (e.g., R Markdown files). This can be helpful, for example to view plots and to check that included code works.

Grading mistakes and reassessments are common to the grading process, and can be addressed using the `assist_regrading()` function. This function allows to the user to specify which student(s) and questions(s) to regrade. As the new grading input is provided, the old input is overwritten in the grading progress log. The changes are then reflected when the final grade sheet and feedback files are regenerated upon termination of the grading process.

All of these functionalities are core to the gradetools grading workflow and are therefore available with all three grading functions.

3.2.2 Additional functionalities

The `assist_advanced_grading()` function allows for increased control over the grading process and GitHub compatibility.

By default the user will be prompted to grade all questions for all students. In some scenarios a grader may want to only grade some students or only specific questions. The corresponding student identifiers and question names can be supplied as arguments to this function to overwrite the default.

This grading function is also useful for instructors managing their class on GitHub. There is an additional argument that allows the user the option of recording GitHub issues while grading. This adds an extra option to the prompting when grading a question. Before grading a question a grader can note a GitHub issue by selecting the rubric option and following the prompting to provide the issue title and body. The details of the issue will be noted in the grading progress log, but the issue will not yet be created on GitHub. This is because the grader may change their mind or note a mistake after providing the issue details.

When grading assignments submitted on GitHub, each student will have their own repository, so a logical way to distribute the feedback files is to upload (i.e., push) it to the students' repositories. The grader has the flexibility to distribute feedback files and create annotated issues whenever they prefer, by using the function `push_to_github()`.

All of these extra functionalities are also available with the team grading function. The only functional differences when using `assist_team_grading()` are an additional column in the class roster and the provision of a feedback file and grade that is *shared* across team members. When grading assignments where the students are in teams, the provided roster must also have a column containing the team identifiers. The team identifier then operates in place of the student identifier and it must be present and the only part unique to a student's submission file path. Nothing about the grade prompting differs for team grading. Upon conclusion of the grading process, each student on the same team will have the same grade and feedback.

While this package was conceptualized for a class where assignments were managed through GitHub, the package does not require any interaction with GitHub. The decisions we made for the automation process were our solutions to minimize user burden while allowing for applicability to various grading schemes and practices.

3.3 Finalization

Grading progress is maintained by the grading progress log which records applied rubric prompt codes. At the end of the grading process these prompt codes are used to produce the grades for the final grade sheet and the feedback for each student's feedback file.

The final grade sheet will contain all of the columns present in the provided class roster, with two additional columns. Columns containing the total grade scored by each student and their grade decomposed by question will be generated for the final grade sheet. It is worth noting that throughout this paper we discuss grading assignments with questions, but gradetools is also useful for grading projects. For projects, the grader could specify rubric items for different components of the project, and then the grade decomposition would reflect a student's score for each component of the project (e.g., data description, analysis, presentation).

Content for a student's feedback file is determined by the applied rubric items and any unique feedback messages supplied by the user. This means that each student, or team, receives individualized feedback specific to their work.

Similar to assignment retrieval and storage, there are a plethora of ways instructors distribute grades and feedback. Therefore, automated delivery of grades and feedback files to the students was determined to be outside of gradetools's scope, with the exception of pushing feedback files to GitHub.

4 Discussion

In Section 2.1 we have discussed grading workflow as a three-step process: preparation, grading and providing feedback, and finalization. There are many R packages that can facilitate the administrative tasks involved in the first and third steps of the grading workflow, 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 in the first and third steps.

On the contrary, 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 package also automates some administrative tasks in the third step of the workflow for those who utilize GitHub in their teaching.

The package was developed as a product of the courses we teach and grade in our department thus supports features that happen in a real grading setting. For instance, a new rubric can be added while grading or a score of a rubric item can be changed at any point during grading. These are some common mistakes graders may make while creating rubrics and gradetools allows for corrections of these mistakes.

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. To give an example, in Spring 2023, one of the authors had to grade around 200 pdf submissions. At the time, the package did not support pdf files and thus the pdf files had to be opened manually one by one, even then they chose to use gradetools simply because they believed they would still save time in other administrative tasks and also would be able to grade fairly. 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).

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 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).

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, provided different feedback types such as unique and repeated ones. 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

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).

Carver, R., Everson, M., Gabrosek, J., Horton, N., Lock, R., Mocko, M., Rossman, A., Roswell, G. H., Velleman, P., Witmer, J. et al. (2016), 'Guidelines for assessment and instruction in statistics education (gaise) college report 2016'.

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.

URL: *https://cran.r-project.org/web/packages/moodleR/index.html*

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

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.

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.

URL: *https://cran.r-project.org/web/packages/gmailr/index.html*

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.

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.

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.

Pyles, C. & UC Berkeley Data Science Education Program (2022), *Otter-Grader: A Python and R autograding solution*. Version 3.2.1.
URL: *https://github.com/ucbds-infra/otter-grader*

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.
URL: *https://github.com/daranzolin/rcanvas/tree/master*

Rundel, C. & Cetinkaya-Rundel, M. (2022), *ghclass: Tools for Managing Classes on GitHub*. R package version 0.2.1.
URL: *https://cran.r-project.org/web/packages/ghclass/index.html*

Schloerke, B., Allaire, J. & Borges, B. (2020), *learnr: Interactive Tutorials for R*. R package version 0.10.1.
URL: *https://cran.r-project.org/web/packages/learnr/index.html*

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).

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.