CS108 Bash Grader

Aditya Neeraje ~ 23B0940

April 25, 2024

Contents

1 Objective

Designed to make life easier for generations of TAs and professors to come, this `Bash Grader` project enables easy compilation and analysis of students' grades across a course, automatic assignment of grades based on the marks distribution, and version control using a clone of git, the stupid content tracker[1].

2 Introduction

You can run `python3 initialize.py` on the terminal to create 4 csv files with random data. For replication of the examples shown below, uncomment the line `np.random.seed(0)` in `initialize.py`.

3 Combine and its Permutations

For my project, I have implemented two algorithms to combine files.
My primary algorithm makes use of associative arrays in awk, parsing the combined output of calling the cat command on every quiz to be added. For each student, awk stores his data as <QUIZ_NAME><DELIMITER><MARKS>. After the students' data from every quiz has been read, awk processes the main.csv file. For every student in main.csv, if his roll_number is present in the index of the associative array, I fill up data for all the quizzes which he attended, then go over the quizzes for which I have no data about him, filling "a"'s in those fields. I then delete his data from the associative array. At the end of processing main, I create new student entries for every roll number still in the associative array, since these students are present in the quizzes but were not initially present in main.csv.

My secondary algorithm is simpler, but however differs in one aspect. It does not care about whether a column existed in main.csv before combining, and clears main.csv at the start of every run of combine (except for however long it takes for total to note down which columns to drop.). Thus, the working of the only_flag with the second algorithm is the same as if run with the force_flag true in the first algorithm.
The secondary algorithm first cats all the quizzes' data together, using sed to append a unique number to each line to indicate which quiz it is from, and then sorts all this data according to roll_number. Then, keeping track of the previous_roll_number and current_roll_number, a while loop populates main.csv line by line, moving on to the next line and filling it if previous_roll_number is different from current_roll_number.
At the END, the last line is populated. While this is going on, empty fields are populated by default using "a".

[1] See the man page for git. Linus Torvalds calls git his stupid content tracker.

Combine has a few additional features, including the `-d/--drop` flag, which allows you to drop certain quizzes from the main.csv file.
The `-f/--force` flag allows you to forcefully combine quizzes which already have a column assigned to them in main.csv.
By passing quiz names as arguments before any other flag is passed, the only_flag is activated, which assumes the quiz names passed are the only quizzes to now be added again to main.csv.

Also, combine by default does not reevaluate columns which are already populated in main.csv, unless the force_flag is true. However, if a quiz is more recently updated than main.csv, combine recognizes this and recombines that particular quiz to ensure that updates are not missed. Combine also keeps track of which columns have been dropped from Total, and drops them again after Total is called.

4 Rescale

I have provided the user with a very easy way of rescaling the marks of any or all quizzes, with two major flags, `-w/--weights-only` and `-c/--custom`. The weights-only flag allows you to pass a sequence of n weights, which are mapped to the first n quizzes in `main.csv`. For all later columns, the user is prompted to assign weights, in order of their appearance from left to right in `main.csv`. Rescale can account for decimal values, and fractions of the form `<Numerator>/<Denominator>`, with a literal backslash.
The custom flag allows you to write out the quizzes you want to rescale in whatever order you wish, and assign weights to them in the same order. Until `EOD` is detected, the script prompts the user for the name of another quiz and its assigned weightage.
Rescale automatically updates the "Total" column, should it exist, and dynamically remembers which columns were dropped from total initially.

5 Upload

I have implemented checks to ensure the incoming file does not have the name `main.csv` and to avoid overwriting existing files without prompting the user for confirmation. Upload also checks the entire incoming file for validity of the data (all quiz scores must be positive or negative decimal numbers, or "a") By default, combine is called right after upload, as this is what I assume the general scenario of usage will be. The quiz can be dropped from main.csv at any later point in time if need be.

6 query

Uses a Levenshtein algorithm-based spell checker[1] to suggest the closest match to the student name or roll number passed as an argument. The number of

results you want returned can be specified via the `-n/number` flag.
Since this can take a few seconds to run, I have implemented animations that showcase the percentage completion to keep the user updated on the progress of the script.
If the query is a substring of any name or roll_number, all names of which it is a substring are returned. With the `-u/--uniq` flag, only the roll_number of the best match is returned.
This algorithm was inspired by the fact that for many of my Telugu batchmates, their names in main.csv are different from what we call them (formally, their last name is written first and their first name last), which would have caused an error in a naive grep-based algorithm.
Thus, my algorithm processes every word within the name queried, and finds the actual name which minimizes the sum of Levenshtein distance from each of the words within the name (i.e Aditya Neeraje and Neeraje Aditya would have a distance of 0 in my algorithm).

7 percentile

For every student whose name is given as input, I calculate his percentile in every individual quiz, as well as his total score (taken here to be the sum of the scores in all quizzes).
Automatic spell-checking has been included, as well as a progress bar.
The user can specify the precision he wants using the `-round/--round` flag, which determines the number of decimals.

8 Analyze/Analyse

I account for both British and American spellings of the word "Analyze".
(the American version is supreme and the British version will be deprecated in later releases :)).
This analyzes the average percentile of the student across quizzes and identifies the quizzes where the student has either somewhat or significantly underperformed.
If the student has performed largely consistently across all quizzes, the script will inform the user of this as well.
A flagrant assumption I am making in printint out the output of Analyze is that all datapoints are male, since it was weeks after I implemented this algorithm before a classmate proved this assumption wrong by querying her name :) Later deployed versions will hopefully have a 50% probability of printing out "her average" instead of "his average".

9 Total

This function sums up the columns of the `main.csv` file and appends the sum to the rightmost column.
Using the `-d/--drop` flag, you can exclude certain quizzes from the sum. In later iterations of total, as well as if other functions are run, this choice of dropping a quiz will be "remembered"using an algorithm which checks all possible subsets of quizzes to figure out which subset sums to total.
Keeping in mind total is likely to generally not have any quizzes dropped, the algorithm starts off with the assumption of all quizzes present, and goes down to the case of no quizzes present, so as to be efficient in real life scenarios.
If a "Total" column is already present in `main.csv`, the script will not overwrite it unless the `-f/--force` flag is passed.
Similar to combine, passing a series of quizzes as arguments before any flag is set sets the only_flag, and only those particular quizzes will be totalled in `main.csv`.

10 Update

Until Ctrl+D (EOF) is pressed, Update prompts the user for a quiz name, student roll number or name and his updated score for the quiz. The script has automatic spell checking for the second argument (student name or roll number). All arguments are first stored as two arrays ->one with scores and the other one storing data of the form <QUIZ_NAME><DELIMITER><MARKS>, where delimiter was chosen by me to be a tilde in this case.
Then, in awk, an associative array whose indices are roll numbers which have some data to be modified and whose data is of the form <QUIZ_NAME><DELIMITER><MARKS> is created. As main.csv is parsed, if the roll number is present in this array, we parse the data for that student and modify the columns corresponding to the quizzes for which he needs a marks change.
Then, I iterate over the quiz files where at least one datapoint has been changed, again creating an awk associative array of roll_number:score. If the student was absent for a particular quiz and then update is called, his name and roll_number are appended to the end of the quiz. As with combine, Total is called at the end, figuring out which quizzes were dropped earlier and dropping them once again.

11 git_init

I have implemented a clone of git, the stupid content tracker. For this purpose, I store in the `WORKING_DIRECTORY` a symlink (.my_git) that points to the remote repository.
If the remote repository is a path wherein the parent directories of the final directory themselves do not exist, the realpath command will not help get the absolute path of the final directory, so I had to implement my own logic to

recursively create all directories until the final descendant.
The script prompts the user for confirmation to change the remote repository if it already exists, unless the `-f`/`--force` flag is passed.
The script also checks to ensure that the final directory satisfies some constraints on its name, such as not allowing the current repository to also be the remote repository (that would undermine the point of having a remote backup), and also not allowing its name to terminate with ".csv". If the remote directory is changed from one directory to another, all relevant data is copied from the old directory to the new one by reading the .git_log file and transferring all the commits present in the file to the new remote.

12 git_commit

Copies the .csv files from `WORKING_DIRECTORY` which are not in the .my_gitignore file to a directory inside the remote repository.
The script prompts the user for a commit message if not specified using the `-m`/`--message` flag. This prompt is scraped from an online website using curl[2], and the user has the option of simply choosing that message by clicking w or W or the up arrow, or typing his own message. The user is prompted a maximum of five times, each time with a 10 second timeout, before git_commit fails.
With a 20% probability, a particular Easter egg appears in the commit message, whereas with a 5% probability, a different Easter egg appears. My code is designed to avoid repeating a prompt if it is present in the last 10 lines of .git_log
The `-i`/`--init` flag can be passed to initialize a remote repository at the same time as this commit.
The `-a`/`--amend` flag can be passed to amend the particular commit, which just uses the last commit's commit message for the current commit, and erases the last commit from .git_log, and does not allow git_log to be called if an amended commit has been checked out.
To greatly increase memory and time efficiency, all data after the first commit containing a particular file is stored either as a patch file w.r.t the original file or as a symlink to a patch file (if two patch files are identical).
This is also time-efficient as the patch files have the name of the source and modified files stored in them, so no time is wasted in searching for the source file.
Symlinks greatly help in memory efficiency because in an average usecase, only one or two out of the multiple csv files are liable to have changed relative to the last commit, and storing them as symlinks eliminates separate memory usage for all these files.
git_commit also informs the user which files exactly have been changed, added or removed relative to the last commit, by patching back each file in the last commit and comparing against the version of the file in `WORKING_DIRECTORY`.

13 git_status

git_status parses the .my_gitignore file to figure out which files are being tracked by git_commit, and which are not.

14 git_add

git_add allows a particular file to be tracked by git_commit by modifying the .my_gitignore file to ensure that particular file is not ignored. Running git_add . or git_add * will track all .csv files in the current directory.
With the `-r`/`--remove` flag, it does the exact opposite, adding the particular file (or all files, if . or * is specifiifed) to the .my_gitignore file.

15 git_remove

Alias of git_add -r

16 git_checkout

My version of git_checkout allows the user to specify a commit using at least the first 4 letters of its hash, a close approximation of its hash (after which a spell-checker kicks in), main (to identify the last commit), or relative to HEAD using HEAD~<num>.
If the `WORKING_DIRECTORY` has changes that are not saved in the most recent commit, then I prompt the user to save them as a new commit, which is saved as the final commit in .git_log. In an earlier version, I planned to make git_checkout main alone reorder the commit history in .git_log, so that the user would not enter an infinite loop of having to safe his data each time he checks out main, however this idea was scrapped because making the .git_log file non-chronological would cause potential problems with git_reset –hard. Instead of reordering the .git_log file, I now check whether `WORKING_DIRECTORY` is identical to any of the last two commits, to avoid looping between the last two commits.

17 grade

This function automatically assigns grades to students based on a bell curve, and displays the graph using plotly express. The relevant data is also saved as a html file.
The algorithm is simple, and proceeds first using a normal bell distribution ($\mu + 30$, $\mu + 20$ and so on until $\mu - 30$). Then, with these and len(students)+1 as boundary lines, the algorithm searches for the biggest jump between student scores in the interval of WIDTH/4 around the initial bar (where WIDTH is the

distance from the current bar to the previous bar, with the zeroth bar being len(students)+1).
This is used to assign and display grades, and pandas saves the grades to a html file as well.

18 grade_manual

This function allows the user to manually assign grades to students. The user can move his mouse around on the matplotlib figure to move the horizontal line around[3], then click to assign the grade cutoff to a particular x_coordinate (included in the grade bracket above the cutoff).
For instance, to assign all students above and including the 150th student (when sorted in increasing order of total score) AA or better, the AA cutoff would be at 150 and the AP cutoff would be at the AA cutoff or higher. Again, the created data is stored as a html file, overwriting the html file if it previously existed.

19 report_card

Creates a latex report card[4] for the students whose names are passed as a list to it. If no names are passed, it creates report cards for all students.
This command requires grade or grade_manual to have been run prior to it, because it relies on the html file with grades assigned to the students.

20 git_log

This function prints out the commit history of the repository, with the most recent commit at the top.
It also has a –oneline option which prints out only commit hash and commit names in one line each.

21 git_reset

This function allows the user to reset the repository to a particular commit. After prompting, all later commits are forgotten from the .git_log file.
If `WORKING_DIRECTORY` is modified w.r.t the commit being reset to, these modifications are stored as a separate commit which continues to exist even after reset is called.

22 git_revert

This function creates a new commit undoing the commit in question. Otherwise, similar to git_reset. No commits are forgotten from .git_log.

23 Boxplot and Stripplot

As the names suggest, they display a boxplot and a stripplot of marks for each quiz respectively. Using the `-r/--rescale` flag, you can specify a score to which to normalize all the scores (i.e all the scores are divided by the maximum score and then multiplied by the rescaling factor).

24 Describe

This function prints out the mean, standard deviation, minimum, 25th percentile[5], median, 75th percentile and maximum of the scores in each quiz.
This has been entirely implemented in awk[6][7], with no usage of numpy or other Python libraries for getting the statistics.

References

[1] URL: `https://www.geeksforgeeks.org/introduction-to-levenshtein-distance/`.

[2] URL: `https://whatthecommit.com/`.

[3] URL: `https://www.tutorialspoint.com/how-to-find-tags-near-to-mouse-click-in-tkinter-python-gui`.

[4] URL: `https://www.overleaf.com/latex/templates/the-university-of-alabama-letter/gjjyckkksphb`.

[5] URL: `https://en.wikipedia.org/wiki/Quartile`.

[6] URL: `https://stackoverflow.com/questions/22666799/sorting-numerically-with-awk-gawk`.

[7] URL: `https://www.gnu.org/software/gawk/manual/html_node/Array-Sorting-Functions.html`.