1. Open up the submitted\_metadata Google Sheet that the app writes submitted form data to upon submission ([submitted\_metadata - Google Sheets](https://docs.google.com/spreadsheets/d/1iqJThuAjoMhRs1njvWPi_7pbszM-Dlvl933L6wsPR6k/edit?gid=0" \l "gid=0)).
2. Also open up the submitted\_rawdata ([submitted\_rawdata - Google Drive](https://drive.google.com/drive/u/1/folders/1IRSvqDu5idxQBfJqYXlxDCl9Yo8WlN2l)) and submitted\_cleandata ([submitted\_cleandata - Google Drive](https://drive.google.com/drive/u/1/folders/1LwxRFKd9m8fNPFB0Fgl4n9v9oBRP4_FF)) folders, which are where the app caches the original files submitted by users and the processed and cleaned versions, respectively.
3. In PI Charter’s file structure, enter the “upstream” folder, then open “NewApprovalsPreLoad.R” and source it to load all contents into R’s working memory.
4. Then, in the same subfolder, open “NewSubmissionsApproval.R” and source it as well. This will begin the QA/QC process. This file contains code that will load from the Google Drive files and folders above each submission in turn, run a number of diagnostic checks on it, and ask a series of questions to confirm that the submission looks in order before adding it to the current database. It will also create a new version of the database file while caching a previous one as backup. Print messages will let you know which submission you are on and whatever else you might need to do to proceed.
5. During QA/QC, it may be beneficial to open both the raw data file and the clean file as reference, especially if you have any suspicions the app did not process the raw file correctly or if it follows an unusual structure the app may not have seen before. However, I no longer generally do this unless and until I see something that makes me feel that I need to do so.
6. For a given submission, find the corresponding row in the metadata file. We’ll be filling out the APPROVED\_BY and COMMENTS columns at the end (at time of writing, these are AB and AC).
7. One check reports all columns deleted from the raw data file, along with any comments specific to column naming the user may have provided. If any columns were deleted in error, such as taxonomic columns, you need to say Y to this prompt. The app will download the raw data file. You will specify the exact column names of the columns in the raw data file to port over to the clean data file, then the column names you will want those new columns to have. However, at present, this process only works if the two are the same numbers of rows—if they have used the row thinning options in the app, you will need to manually thin and move the columns between the two files, terminate the QA/QC process, then begin it again.
8. One check will flag any general feedback/comments the user left. Saying Y to this prompt will terminate QA/QC so specific actions can be taken in response, if any are needed.
9. One check asks you to look at the DOW, survey date, and survey time provided during submission. These should just look to be plausible values in the proper formats and should generally be fine, though they can be overwritten as needed.
10. One check asks you to look at the provided submitter and surveyors. These should match the corresponding strings found on the app’s leaderboard tab \*exactly\*, if applicable. This includes spacing, punctuations, capitals, etc. If you are unsure if they match, you should confirm that they do.
11. At any point, if you make a change that will write to the cleaned file, record it somehow in the COMMENTS column of the metadata file for posterity.
12. One check looks at any depth column and asks if there are some depths of 0 or depths > 25ish feet. In R Studio, in the view of the current.import (cleaned file), you can sort the depth\_ft column to look for these. They \*could\* be non-throws, so see if it looks like there is evidence that rakes were in fact tossed at all such points. If it seems really unlikely that they were, note which rows these are and you can remove those rows as non-throws later. However, be conservative. It isn’t a big deal to retain non-throws by accident. Also, treat any evidence that a throw was made at any low/high depth that all such depths are valid. Also, look at the bottom of the sorted depth\_ft column for NA values, as these are often indicative of non-throws.
13. One check will summarize all unique values in all taxonomic columns (that aren’t NAs). These should be numeric integers between 0 and 5, generally, but may also contain text strings like “V” or “x” or w/e. Look out for values that are more likely than not to be typos, such as “11”.
14. One of the most important checks, near the end, prints a summary() of all the columns in the submission. Look for unlikely columns or values or outliers that might be indicative of entry errors. Often, it’s this check that reveals issues not caught by other checks.
15. Some questions towards the end will ask if you want to delete any rows or columns by row number or column name. The former is mostly used to eliminate non-throws that should have been eliminated by the user but weren’t (full of NAs, e.g.), or to eliminate summary columns that should similarly have been eliminated. It’s worth giving the data file a quick once-over from top to bottom to make sure nothing jumps out to you. In particular, it might be worth checking out the depth column, as this is the most likely one to raise some flags.
16. A late question notes the user’s response to the “location data” question. If the user successfully included lat-long data (or UTM data that was successfully transformed into lat-long data), there should be latitude and longitude columns in the data set with realistic values. If this is the case, the file should be compiled (C). If location data are missing and the user said they don’t have it, the file should also be compiled (location data are ultimately “nice-to-haves”). If there are UTM data, or invalid location data, these columns should just be deleted and the submitter contacted, though the file can be conditionally compiled in the meantime (location data could be joined in later). If the user specified an alternative location data file, this can be cached in Google Drive for now in the submitted\_locdata folder ([submitted\_locdata - Google Drive](https://drive.google.com/drive/u/1/folders/14ASYFti6Yfjy3Bv2K7lpwQn5t9J46Y7P)). At present, there are no plans to join these data into the main database file unless it’s painfully obvious how to do so, which is why this route is now discouraged. However, the rest of the data can be compiled but with a note that location data will need to be joined in later (J). If the location data seem errant in any way, the file can be marked as an A instead.
17. At this point, the QA/QC script will join the data to the database file. At this point, check to see that db.new (the database file before the latest submission) and grow.dat (the database file after the latest submission) are different lengths in numbers of rows, by ~ the same number of rows as the latest submission—that would indicate the merging has been successful.
18. Assuming a successful merge, finish filling out the COMMENTS row in the metadata file, and mark the APPROVED\_BY column to be C for compiled files, A for “ambiguous/needs adjustment” if the submitter needs to clarify something (send them an email), L if the file has been compiled but location data need joining in, or N for “rejected” for any reason (usually, the wrong format, the wrong data structure, or a duplicate). Then, go to the submitted\_cleandata folder and sort it by “Date modified”. There will likely be multiple versions of the most recently approved submission—delete all but the most recent version. Then, move that most recent version into the “Approved” subfolder or the “Awaiting fixes” subfolder, as appropriate. There are a number of different subfolders in the “Approved” subfolder based on what’s going on with the location data, so make sure to move those along as appropriate too. Similarly, move the submitted\_rawdata file from the most recently approved submission into the Archived sub-folder. At this point, you’ll be ready to proceed to the next submission.
19. One check is designed to check the logic of a whole\_rake\_density column. If this column says NA or 0, but then there are taxonomic data, we should overwrite the value with the highest taxonomic abundance score seen. The most likely reason for this is as a result of some surveyors not counting algae as counting for whole\_rake\_density even though they will still give them abundance scores. Similarly, if no taxonomic data are observed, but the whole\_rake\_density column does not say 0, we should reduce it to 0 to be consistent, as that is likely a typo.
20. One check looks to see if any values for taxonomic abundance exceed the provided max rake score value. Use your discretion to overwrite these with the most likely value (e.g., “11” should probably be “1”) or else with the max rake score value.
21. Not every correction can be done in an automated fashion (at present). For example, if you need to replace an errant rake score value with something that isn’t the max rake score value (e.g., making “74” into “4” when the max rake score is “5”), you will need to terminate the QA/QC process, download the cleaned file as a CSV, fix the value manually, then reupload the file and restart QA/QC. This will mean going through some of the questions again for the current submission. You will also have to delete any older versions of the cleaned file in the cleaned file folder or the QA/QC script will gag. Downloading and opening the CSV file in Excel will mess up the survey start date and submit time columns, but the script will automatically catch this and help you correct it using the data in the submitted\_metadata file.
22. One check will look for column names in the cleaned file that aren’t in our lookup table. This is most often because of columns with duplicate names, such as multiple sta.nbr columns. Extraneous columns can be deleted. However, non-extraneous columns could be added to the lookup table (Inputs/Dynamic/column\_name\_lookup.csv) and renamed to match naming conventions.
23. One check will look to see if the station numbers fail to start at 0 or have non-consecutive values. This might be indicative of a survey whose absences (as opposed to non-throws) have been omitted. The check will then see if all remaining entries have plants at them, which would be further suspicious (the doc would lack any absences then). This combination of events isn’t super unlikely, but it does warrant a spare look—one can always double-check with the submitter as needed.
24. One check will look for the “ambiguous” substring that indicates an original column name that was ambiguous as to which exact taxon it referred. Renaming the column is easy in the QA/QC process but may require you to contact the submitter and set the file aside until the submitter responds.
25. It’s courtesy to write to each submitter post-approval to let them know that their submissions were approved and will be visible on the app when the next update is pushed through.
26. When all submissions have been approved and compiled, in the upstream folder, open and source the file named CreatingLakeSummaries.R. This will create all the new “digests” of the database that are used in selectors and figures, etc. in the app.