GSoC ‘19 Project Proposal

Extract Scripts into a separate system

About Me

- **Name**: Harshit Bansal
- **Email**: hbansal@ec.iitr.ac.in
- **Github**: harshitbansal05
- **Skype**: live:harshit.bansalec
- **Time zone**: Indian Standard Time (GMT + 5:30)
- **Contact**: +91 81717 84091
- **University**: Indian Institute of Technology, Roorkee
- **Major**: Electronics and Communication Engineering

Abstract

Problem Statement:

The Data Retriever is a package manager for data. The Data Retriever automatically finds, downloads and pre-processes publicly available datasets and stores these datasets in a ready-to-analyze state. The Retriever project, however, suffers from some key drawbacks which require attention:

1. Currently the core software ships with json script metadata. However, this metadata must be shifted in a separate project location to help with organization, maintenance, and testing. This would also scale up the number of usable datasets for Retriever, as the source of creation may be any other project.

2. Retriever downloads all the json scripts at once during installation, or whenever the scripts folder in the home directory (~/.retriever directory) is empty. However, this would become increasingly inefficient as the number of the scripts in the upstream repository increase. We must remove this step and instead download the scripts from the upstream repository only when specifically needed.

3. Presently, Retriever does not check for newer versions of scripts upstream and continues to use the scripts present in the home directory. These scripts may become quite outdated as compared to the newer scripts. The method `check_for_updates()` synchronizes all the scripts simultaneously with the upstream scripts. However, it is only called during installation and whenever the scripts folder in the home directory is empty. Also, it is quite inefficient as it would download all the scripts at once. Thus, we must check whether a newer version of a script is available upstream, when needed and accordingly download it.

Deliverables

By the end of the GSoC period, I aim to achieve the following major goals:

1. Create a new repository named **Retriever Scripts**, containing all the json and python scripts, present in the Retriever repository. The subsequent versions of Retriever would download, compare and update the scripts from this new repository. However, the Retriever repository would still have the scripts folder present in it to provide examples for `retriever ls` and enabling users to view the format of the scripts. Secondly, it along with the file ***version.txt*** should be present to ensure backwards compatibility (i.e. proper functioning of the older versions).

2. Ability to download the scripts from the new repository, only when specifically needed, and not download all the scripts at once during installation.

3. Prompt the user if a newer version of the script (that he wishes to download) is available upstream by comparing its version with the version of the local script. If the user agrees to update, accordingly download the updated script.

4. Make corresponding changes in the Julia and R projects to ensure compatibility with the new design changes. Lastly, the functionality for installing scripts not created by Retriever would be added.

Implementation

1. Retriever Scripts repository:

It would contain the following files/directories:

1. A scripts folder containing all the scripts present in Retriever at the current time.
2. A ***version.txt*** file containing the script file names along with their versions, similar to the version.txt file present in Retriever.
3. A ***version.py*** file for updating the version.txt, whenever a script is updated, created or deleted and subsequently committed.
4. It would contain functions for creating, editing or deleting a script file similar to the Retriever repository. However, unlike Retriever, all the scripts would be only added to the scripts folder in the local git repository. It would not have any interaction with the scripts folder in the home (~/.retriever) directory.
5. It would contain proper tests for verifying the modification or updating of scripts. It would do so by trying to install the dataset in some or all of the engines. This would, however, require Retriever to be installed in the user's computer.
6. Travis CI/Appveyor integration to enable running tests outside the user's computer on the modified scripts.
7. It would also contain a **pre-commit hook** for verifying that the version number of the dataset's script is updated prior to any modification.

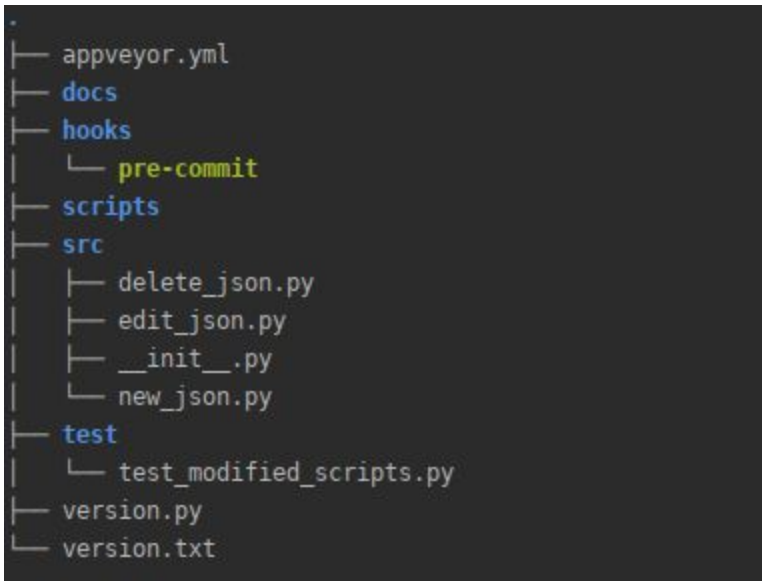
8. Finally, proper documentation and guidelines would be added for the Retriever Scripts repository.

1.1 Addition of tests:

Currently, when a script is updated/added, a manual test has to be done to verify the changes. Thus, the new repository for scripts would also add proper tests for all the functions it aims to provide, namely addition, and modification of scripts. This can simply be done by attempting to install the changed dataset script in some or all of the Retriever engines.

Similar to the original repository, **pytest** would be used for writing tests. **pytest** is a framework that makes building simple and scalable tests easy. Tests are expressive and readable, and no boilerplate code is required. A new directory named **test** would be created, which would contain all the tests for the scripts. The tests added would also be automated by running them whenever a script is added/updated prior to the git commit. This would be done by modifying the pre-commit file to initially check if Retriever is installed in the user's computer and subsequently running the tests if present.

The structure of the Retriever Scripts repository can be shown as below:



The files ***new_json.py***, ***delete_json.py*** and ***edit_json.py*** can be used for creating, deleting and editing a script respectively. They would use the same approach as followed in Retriever but would add/delete scripts only from the scripts folder present in the local git repo. Their use would be as simple as:

```
1. python new_json.py
2. python edit_json.py dataset
3. python delete_json.py dataset
```

2. Retriever repository:

2.1 Task 1 (Refer scripts from the new scripts repository):

The Retriever must now refer the new Retriever Scripts repository for downloading or updating the scripts. Also, as adding scripts to the upstream repo would now be done through Retriever Scripts, this support would be removed from this Retriever repository. To ship the scripts to a separate location, would involve some key changes throughout the codebase. Some of them are listed below as follows:

1. **setup.py:**
    - Omit the code that copies the file ***hooks/pre-commit*** to ***.git/hooks/pre-commit***, as the **hooks** folder would no longer exist. The pre-commit file looks for the modified scripts in the scripts folder of the repo and makes sure their versions are updated prior to the `git commit`. Since the Retriever would no longer be adding scripts to the upstream repo, this holds no use.
2. **version.py:**
    - Remove this file as it would be no longer needed. This file is used for updating the version.txt file when a script is added/updated to the upstream repo.
3. **test/test_modified_scripts.py:**
    - Remove this file, as adding scripts from the original Retriever repository would no longer be supported. It runs tests on modified scripts prior to the git commit.
4. **retriever/lib/defaults.py:**
    - Change **REPO_URL** to the new scripts repository.

The completion of the above tasks would result in the first version of Retriever, using scripts shipped to a separate project location.

2.2 Task 2 (Download scripts only when specifically needed):

Retriever uses the function `check_for_updates()` to download all the scripts at once during installation or whenever the scripts folder in the home (~/.retriever) directory is empty. The scripts must, however, be downloaded only when they are specifically needed. The scripts would thus be classified into the two abstract categories:

1. Online scripts (Available upstream)
2. Offline scripts (Available in ~/.retriever/scripts)

Two functions would be added, `get_dataset_names_upstream(keywords, licenses)` and `get_script_upstream(dataset)` for returning scripts or their names from the upstream repository. The working of the two methods can be described as follows:

1. **get_script_upstream(dataset):**
   It would return the script from the upstream repository (if present upstream) after downloading it, else return **None**. The following approaches are possible for downloading the script:
    a. Read the version.txt file from the new Retriever Scripts repository to check if the script exists in the list. If it exists, download it from the upstream repo.

b. Use Github's search API to check if the script exists in the upstream repository. The command "`curl https://api.github.com/search/code?q=iris+in:path+path:scripts+repo:weecology/retriever`" would check if there exists the script with the name **iris** in the upstream Retriever repository. If a result is returned, download the script file from the upstream repo. Also, Github's API does not introduce any extra dependencies.
c. Simply attempt to download the script from the upstream repository using the command: `requests.get("https://raw.githubusercontent.com/weecology/retriever/master/scripts/iris.json", allow_redirects=True, stream=True)`. If present, the script would be downloaded, else a response with status 404 would be received meaning that the script file does not exists.

After downloading the script, its path would be passed to the method `read_json()`, which would return an instance of **BasicTextTemplate**. Based on the performance, the time requirements of the above approaches, the most suitable one would be implemented after the discussion with the mentors. Thus, a function named `get_script_upstream(dataset)` would be made, which would return the script from the upstream repo after downloading (if present upstream), else return None.

2. **get_dataset_names_upstream(keywords, licenses):**
It would return the **names of the scripts** present upstream, matching the query parameters. If no argument is provided, read the version.txt file from the upstream repository and return a subset of the list. When a keyword or license is passed as an argument, the naive reading of all the upstream scripts looking for the keyword/license would be highly inefficient. Thus, to effectively search the online scripts for keywords and licenses, the following approaches could be possible:
    a. Add a separate file similar to version.txt with two more columns, features and licenses to the Retriever Scripts repository. When a query is made, this file would be read and the scripts containing at least a feature or license similar to the query would be returned.
    b. Github's search API can be used to search for the specific keywords or licenses in the scripts folder of the upstream repository. For example, the query "`curl https://api.github.com/search/code?q=CC0-1.0+in:file+path:scripts+repo:weecology/retriever`" would return all the scripts with the license **CC0-1.0** in the Retriever repository.
    c. For licenses, since the number of licenses would be limited, a folder named licenses can be created containing the files like license1, license2 and so on. These text files would then contain the script names with the respective licenses.

Thus, the method would return a simple list with names of all the scripts matching the query parameters. The best performant of the three would be implemented after the discussion with the mentors.

The code changes in the files containing the **core methods** would be as follows:

1. **retriever/lib/scripts.py:**
    - In the function `get_script(dataset)`, the function `SCRIPT_LIST()` is called, which would return the list of all the offline scripts. If the dataset exists in the offline list, it would be returned. Else, `get_script_upstream(dataset)` would be called. If a script is returned, the method would successfully terminate, else an error would be raised regarding the absence of the dataset. Figure 1 demonstrates the working of the proposed execution of the method.
2. **retriever/lib/engine_tools.py:**
    - In the function `name_matches(scripts, arg)`, scripts would be the same as SCRIPT_LIST. If **arg** = '**all**', all the offline scripts would be returned. If arg exists in the offline scripts, return that script. Else, `get_script_upstream(dataset)` would be called. If a script is returned, the method would successfully terminate. If absent in both the online and offline scripts, the default implementation of the method would follow.
    - The method `get_script_version()` would be removed, as it's only called from version.py and test_modified_scripts.py, both of which would be removed.
3. **retriever/lib/datasets.py:**
    - In the function `datasets(keywords=None, licenses=None)`, the method `get_dataset_names_upstream(keywords, licenses)` would be called to get all the upstream scripts matching the query parameters. The default implementation would follow for offline scripts. Thus, a dictionary containing all the online scripts as a list of names, and offline scripts as instances of **BasicTextTemplate** would be returned, i.e. {"online": [name], "offline": [BasicTextTemplate]}.
    - In the function `dataset_names()`, return a dictionary with the list of the names of both online and offline scripts separately.
4. **retriever/lib/download.py:**
    - In function `download()`, remove the if statement that calls `check_for_updates()`.
5. **retriever/lib/get_opts.py:**
    - The **json_list** would only be populated from the offline scripts as it would be used for editing or deleting a json script. For the **script_list**, a subset of the online scripts would be fetched by calling `get_dataset_names_upstream()` and would be combined with all the offline scripts.
    - **keywords_list** and **licenses_list** would be initially populated from the offline scripts. If a separate file containing the keywords and licenses along with the script names is made (as mentioned earlier), a subset of the keywords and licenses can also be added by reading this upstream file. This part would require further research and discussion with the mentors.
6. **retriever/lib/install.py:**
    - In function `_install()`, remove the if statement that calls `check_for_updates()`.
7. **retriever/__main__.py:**

- Remove the code that checks for scripts in the home directory and downloads all the scripts if empty.
- In the command `retriever ls`, the online and offline scripts would be printed separately.

Apart from these main changes, some changes would also be necessary for the tests folder, which would be done accordingly.

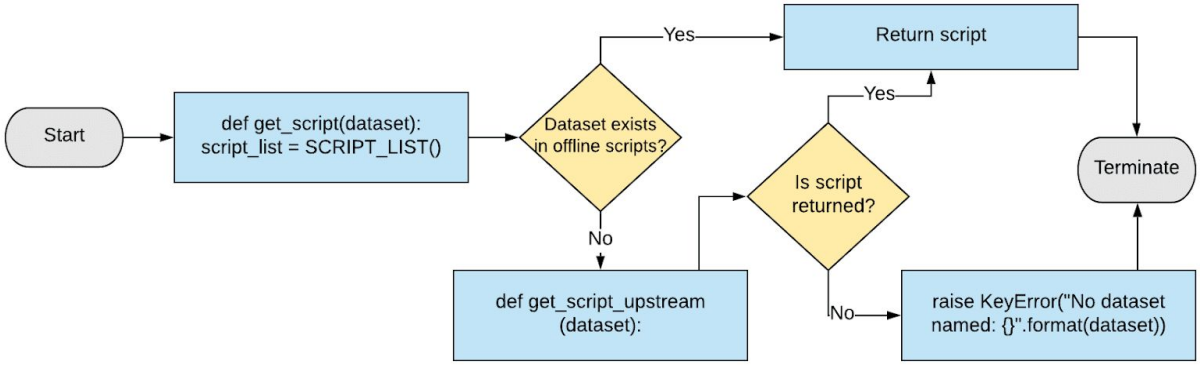


Figure 1: Proposed execution of the function get_script(dataset) to download a script only when specifically needed.

The function `check_for_updates()` currently updates all the scripts. It would be extended by adding the ability to update a specific dataset. Thus, the completion of the above tasks would result in an enhanced version of Data Retriever with scripts shipped to a separate repository and scripts only being downloaded when specifically needed.

2.3 Task 3 (Install the latest version of scripts):

Retriever currently looks for scripts in three locations, namely two local folders (~/.retriever and the local retriever git repository) and the upstream repository's scripts folder. The Retriever's Github scripts folder contains the most updated list of scripts. Retriever has a function named `check_for_updates()`, which syncs the local ~/.retriever folder with the upstream scripts folder. Currently, this method is invoked when:

1. Retriever is installed either via pip or in development mode from Github through the command `python setup.py install`.
2. In the function `_install()` only when the directory ~/.retriever/scripts is empty. This function is called by the install counterparts of the various engines, which are themselves called when running tests.
3. In the `download()` function in the file ***download.py*** when the directory ~/.retriever/scripts is empty.
4. Inside the `main` function when the command is not one of `reset` or `update` and the directory ~/.retriever/scripts is empty.
5. The command `retriever update` is explicitly called.

Thus, as can be seen from the usages, the scripts in the ~/.retriever are updated only when explicitly called. The other calls would only be invoked when the ~/.retriever/scripts is empty, which would not occur unless scripts are explicitly deleted or `retriever reset scripts` is called. Thus, Retriever currently has no way of updating the local scripts directory, unless called explicitly and thus a normal user would never be aware of the new scripts being added to the repository on a daily basis, causing the scripts in the home folder to become quite outdated.

The following function would be added as a part of the implementation:

1. **get_script_version_upstream(dataset):**
   It would return the script's version from the upstream repository (if present), else return **None**. The following approaches are possible for determining the script's version:
   a. Read the version.txt file from the new Retriever Scripts repository to read the script's version.
   b. Simply attempt to download the script from the upstream repository as was described earlier. If present, the script version can be read, else a response with status 404 would be received meaning that the script file does not exists.

The code refactoring would be as follows:

1. **scripts.py:**
   - In the function, `get_script(dataset)`, it would build on the previous changes and attempt to download the latest version of a script from the upstream repository whenever possible. Its implementation can be described as follows:
     - ❖ If dataset exists in offline scripts, `get_script_version_upstream(dataset)` would be called to get the upstream script version. If the version returned is None (the only reason is no internet connection), the offline script would be returned. Else, it would be compared with the version of the offline script. If it is equal to the offline version, the offline script would be returned. However, if greater, ask the user if he wishes to download the updated script. If the user responds in positive, call `get_script_upstream(dataset)` and return the script, else return the offline script.
     - ❖ If dataset does not exist in the offline list, `get_script_upstream(dataset)` would be called to get the script from the upstream repository. If None, an error would be raised regarding the absence of the dataset.

     Figure 2 demonstrates the working of the proposed execution of the method.
2. **engine_tools.py:**
   - In the function `name_matches(scripts, arg)`, scripts would be the same as SCRIPT_LIST. If **arg** = '**all**', return all the offline scripts. The remaining implementation would be similar to the above function. If the arg exists in the offline list, compare its version with the online version and return the script based on the user input. Else, the method `get_script_upstream(dataset)` would be called. If a script is returned, the method would successfully terminate. If absent in both the online and offline scripts, the default implementation would follow.

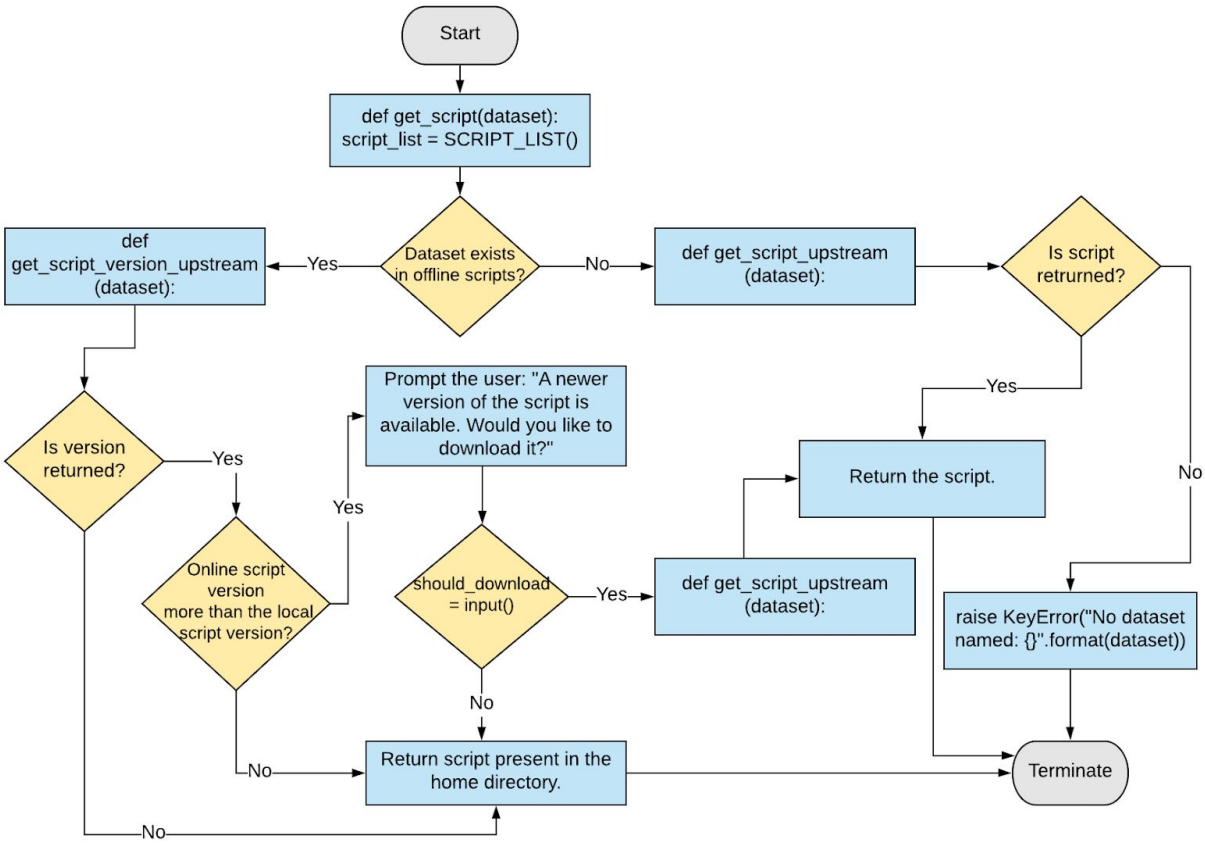


Figure 2: Proposed execution of the function get_script(dataset) to download a script only when specifically needed, installing the latest version.

3. Julia and R repositories:

Retriever.jl is a Julia wrapper for the Data Retriever software. It makes use of the Data Retriever's python package, requiring it to be installed. It has been created using PyCall hence all the functions are analogous to the functions of the Retriever python module. Since it simply wraps Retriever python functions, no change in the codebase are required to make it compatible with the design changes in Retriever. There would only be some changes in the output similar to that of Python modules. For instance, the function `dataset_names()` would now return a dictionary with two keys online and offline, containing the subset of the scripts present on the upstream Retriever repository and the home folder (~/.retriever) respectively.

RDataRetriever is an R interface to the Data Retriever so that the Retriever's data handling can easily be integrated into R workflows. Similar to the Julia counterpart, it makes use of the Data Retriever's python package, requiring it to be installed. The changes to be made in the code to make it compatible with Retriever are:

1. In the function `datasets()`, `r_data_retriever$datasets()` would return a dictionary with two keys online and offline. Thus, it would be printed accordingly.
2. Similarly, in the function `fetch()`, datasets would be handled as a dictionary.

4. Installing scripts not created by Retriever:

Retriever scripts currently use the specifications written out by frictionless data. However, around the internet, there are several data packages that follow these specifications and were not created by Retriever. Thus, the aim is to enable Retriever to also manage data delineated in the data packages not created by it. It would have a simple command line interface `retriever install csv -u [url]`. This would download the script in the home directory and subsequently install it via the specified engine. However, these scripts would be modified to have the argument **retriever** equal to **False**, to make the users aware that they are downloading a foreign file not maintained by Retriever. This can also be shown as a warning when the function is called.

Timeline

I have divided the entire timeline on a weekly basis. I would try my best to stick to the timeline and would make necessary changes to it as suggested by the mentor. I would also be writing blogs for my GSoC journey with all the timely updates. It would also be helpful for the mentors to track my progress.

Community Bonding Period:

- Learn more about testing in Python, and the use of the **pytest** package.
- Get more familiar with the codebase, coding style of Retriever and work on issues mentioned in the issue tracker.
- Communicate with the mentors about the intricate details of the idea implementation and give the planned approach an edge with the feedbacks.
- Hiatus of few days due to exams and moving back home from college for vacations.

Coding Period:

Week 1 [May 27 - June 2]

- Create the new **Retriever Scripts** repository and create the version.py file for writing to the version.txt file.
- Add the version.txt file containing the names of all the scripts with their corresponding versions.
- Add the pre-commit hook to ensure script versions are increased after updating them, prior to making the git commit.

Week 2 [June 3 - June 9]

- Create utility functions for adding, editing and deleting a json script.
- Implementation of the functions `new_json`, `edit_json`, and `delete_json` for doing the above tasks. All these functions would be operating on the scripts in the local git repo only.

Week 3 [June 10 - June 16]

- Add tests to verify any modification of scripts and add simple tests for new scripts.
- Integrate Travis CI/Appveyor tests for the Github repository.
- Create README, Contributing.md for the Retriever Scripts repository. Also, create appropriate documentation for the project.

Week 4 [June 17 - June 23]

- Make changes in Retriever to refer the scripts from the new scripts repository.
- Perform integration testing of the Retriever and the Retriever Scripts repository.
- Perform wrap-up, and any necessary code cleanup and changes.

Phase I Evaluation [June 24 - June 28]

- Check all the tasks assigned in previous weeks are brought to completion.
- Review the code properly and prepare for evaluation.
- If time permits, begin with the code changes for the task of downloading scripts only when specifically needed.

Week 5, 6 [June 29 - July 7]

- Continue the part of restructuring code from the previous week and finally finish it.
- Make changes in the tests, due to the new design changes. Also, add some more tests for the new functionalities.

Week 7 [July 8 - July 14]

- Finally, run the tests for the code changes. Thus, Retriever would now be working with scripts shipped to a separate repository and would download them only when needed.
- Update the Retriever README and docs with the new changes introduced.

Week 8 [July 15 - July 21]

- Begin with the code changes for the third task (installing the latest version of scripts), in a separate branch.
- Add more tests for the functionalities introduced in this phase.

Phase II Evaluation [July 22 - July 26]

- Complete the ongoing task of installing the latest version of scripts. Thus, Retriever would now install the latest version of scripts whenever possible.
- Perform wrap-up, and any necessary code cleanup and changes, for the 2nd evaluation.

Week 9, 10 [July 27 - August 4]

- Update the Retriever README and docs with the above new change as well. The primary tasks for the Retriever repository would have been completed till now.

- Analyze the codebase of RDataRetriever and Retriever.jl, and see the effect of the newly introduced design changes.

Week 11 [August 5 - August 11]

- Restructure the above two wrapper repositories appropriately. Mention the changes in their respective documentation and READMEs.
- Add the feature of installing a dataset from a foreign script (Installing scripts not created by Retriever).

Week 12, 13 [August 12 - August 25]

- Exhaustive checking for errors in documentation and code.
- Resolving issues that come up after the finalizing code.
- Work on the already opened pull requests and solve open issues in the Issue Tracker.
- Make the code cleaner, remove all inspection warnings and add comments (documentation or single-line) to all the methods and blocks in the code. This would be a great help to anyone willing to contribute to the project.

Stretch Goals and Post GSoC

- Continue contributing to the Data Retriever repository.
- Make efforts towards a clean issue tracker: Currently, there are many pending issues in issue tracker which are either resolved and not closed or are duplicates, and there are others which need to be worked upon. I will try to assign them the labels “bug”, “low-priority”, “feature-request”, “won’t fix”, and thus make a cleaner and better issue tracker. This would facilitate greater open source contribution.

Time Commitments

I have my semester examinations between 22 April - 5 May so I would be less active during this period, which is way off the timeline. During the vacations (6 May - 30 July), I have no other commitments and I can easily devote at least 50 hours per week till 30 July. After that, however, I would be having classes so I may be able to devote around 5 hours on weekdays and 7-8 hours on weekends. But the major part of the project would have been completed till now. I believe that the allotted work per week is completely doable by me and neither overloaded nor slacked. The GSoC timeline exactly matches with my summer break and thus it would give me enough time to work on this project.

Pre-GSoC involvements

I have been contributing to Retriever since February. Here is a list of my pull requests made so far:

- #1226 (Merged): Change read mode to remove deprecation warning
- #1233 (Merged): Add LTREB Desert Laboratory data
- #1234 (Merged): Solve minor delete key bug Fix: 1232
- #1242 (Merged): Add reset specific dataset or script functionality Fixes: #1241

- #1245 (Merged): Add path argument for altering install directories
- #1257 (Merged): Change testdb name to testdb_retriever
- #1260 (Merged): Update passwordless setup of Postgres for Windows
- #1273 (Merged): Solve minor error in fetch preserve method
- #1296 (Open): Add HDF5 engine using Pandas and SQLite
- #1300 (Open): Corrects very minor typo error
- #183 (Open): Add functionality to reset a specific dataset

Apart from these PRs and issues I have been following almost all the pull requests after my first PR. I also created the following issues in Retriever, RDataRetriever, and Retriever.jl:

- #1232: UnboundLocalError when deleting an item from keywords key in json scripts
- #1241: Add reset dataset option
- #1299: Redundant functions in tools.py and scripts.py
- #180: Add functionality to reset a specific dataset
- #181: Possible error in download function regarding argument sub_dir
- #31: No argument for data_dir in csv, xml, json and sqlite engines

Why do I fit in?

I am an active member of the Data Science Group (DSG), IIT Roorkee, a bunch of passionate enthusiasts trying to foster the culture of Data Analytics and Machine Learning on the campus. I am quite proficient in Python and also have a previous experience to work as per a proposed timeline with a mentor on a summer project.

I believe I have enough fuel to get started on my goals - as a result of my involvement with the organization in the past few weeks. I have already made some contributions, adding some features and resolving some bugs by which I have gained a lot of familiarity with the code base of Retriever. Thus, I would be setting grounds early to avoid any stopgaps during the actual GSoC period. I would fully try to learn what the mentor aims from the project and his vision of the final product. Also, I intend to update the mentors on a daily basis so that they know what I am up to. I would try to strictly follow the coding style of Data Retriever, making clean commits and well-structured pull requests.

Why this project?

Being a member of the Data Science Group, IIT Roorkee, I realize the huge time involved in acquiring, cleaning, standardizing and importing datasets, thereby hindering the main goal of data analysis. The Data Retriever automates these first steps in the data analysis pipeline, thus reducing the time for a user to get the large datasets up and running. Getting a cleaned and preprocessed dataset, ready for analysis is a luxury and therefore Data Retriever is a boon for data analysts. Hence, this project is a natural choice for me. The mentors of Retriever are always available and extremely supportive, which has encouraged me even more. I am pretty sure that the

future versions of Retriever to be rolled out would be very much better than now and I would love to be a part of the process.

References

- Retriever Github
- Documentation for Data Retriever
- Retriever.jl Github
- Documentation for Retriever.jl
- RDataRetriever Github
- Pytest Documentation
- Travis CI Docs
- Appveyor Docs