deposits: Deposit Research Data Anywhere

Scott Chamberlain

2021-04-19

Signatories

Project team

Scott Chamberlain, rOpenSci Co-founder and Technical Lead, University of California, Berkeley

Consulted

- Karthik Ram, Maëlle Salmon, Noam Ross - rOpenSci
- Aaron Wolen - TileDB

The Problem

Publicly depositing datasets associated with published research is becoming more common. This is in part due to an increase in journals requiring data sharing, though the proportion of journals requiring sharing is still quite small (Resnik et al. 2019). In addition, there is ongoing cultural change happening in academia in which data sharing is becoming more of an accepted gold standard. However, some journal policies do not enforce their own data sharing policies, and authors often do not comply (Vines et al. 2014, Stodden et al. 2018, Christensen et al. 2019).

One reason authors do not share data is because of the time involved. Fecher et al. (2015) found in a survey that respondents often said data sharing was too much effort and took too much time. One of the major pieces of sharing data is the preparation of data and metadata into a format that a data repository expects. Better programmatic tools can transform data sharing from a mountainous climb into a pit of success. Documentation and training can help get users familiar with the process, but a browser based data and metadata submission workflow can only be so fast, is not easily reproduced, and does not facilitate updates as data and metadata are updated due to revisions/corrections.

Why make a programmatic tool in R? R is widespread in academia in many disciplines. Thus, the potential user base of tools to make data deposition easier is likely quite large.

In addition to the large number of R users in research, the number of datasets uploaded to data repositories is enormous. The Registry of Research Data Repositories provides the number of datasets in each repository. For example, Dryad has 30,000 datasets; Knowledge Network for Biocomplexity has 27,000 datasets; and Pangaea has 400,000 datasets. Only a fraction of the people uploading datasets would consider using R to upload datasets, but together with a large number of R users, and increased prevalence of open science practices, there is likely significant demand for a high quality R software tool for data deposition.

There are some existing solutions for data deposition in R:

- dataone: DataOne (Data Observation Network for Earth) is a general data repository. This package does support uploading datasets to the DataOne cloud
- rdryad: Like DataOne, Dryad is a general data repository. Karthik Ram and I maintain this package. Data submission to Dryad is in development in this package.
- osfr: Open Science Framework (OSF) is a project management platform that includes hosting research datasets. osfr is an rOpenSci package maintained by a community member.

The above tools interact with single data repositories. I know of no tools that work across data repositories.

The proposal

I propose a framework that is a single user interface to many different research data repositories. My approach is like that of dplyr; whereas dplyr "verbs" work identically across many backends (e.g., spreadsheet, SQL database), this project will simplify data deposition under a common set of commands. This model of decoupling the interface from the backend has proven successful, but has not yet been applied to the important task of researchers sharing data.

There are thousands of different data repositories, with many aspects that differ among them (e.g., disciplinary scope, openness, data submission format, authentication methods). While only a handful of repositories may be relevant to any individual researcher, the fact that each one likely implements totally different interfaces and protocols for depositing/updating data creates significant barriers for end users. We aim to simplify all of that complexity to make depositing data much easier.

As there are so many data research repositories, it would not be feasible for a few developers to maintain extensions for so many different sources and verify they work. Instead, I will design a package that implements a modular/plugin system so that users can contribute their own plugins to extend functionality to other repositories. Detailed vignettes will be written to document this process.

One of the first steps will be to complete research on data repositories using the Registry of Research Data Repositories. This will allow us to quickly get familiar with the diversity of technical details behind data repositories. I will narrow the set down to two repositories that are widely used so we can provide the largest immediate payoff upon project completion. In addition, the focus on two widely used repositories will make the complexity manageable while still providing enough variety so that we'll cover a large number of users.

My significant experience in working with web services will help make interactions with data repositories as smooth as possible. I created and maintain packages for major aspects of working with web services in R: crul, an HTTP client (similar to httr); webmockr, for mocking HTTP requests; and vcr, for caching HTTP requests in unit tests.

Many of the pieces for interacting with repositories are already implemented by rOpenSci staff or community members. For example, Karthik Ram and I maintain the R package rdryad for interacting with Dryad, one of the larger general purpose data repositories, to which we've recently added the ability to upload datasets. Another major data repository is the Open Science Framework (OSF) - an rOpenSci community member Aaron Wolen maintains the package osfr for OSF (which also has data upload functionality). Using these existing packages as dependencies, I can more quickly build the plumbing for data repositories.

The package structure will have the following components (see figure below): authenticate, prepare data and metadata, submit data, fetch data, and browse data. Authentication will entail abstracting a lot of complexity, making the process simple for the user. Next we'll help users prepare their data and metadata. This step will vary based on the data repository; in some cases this will be the last step before the user has to submit data manually on a website; in other cases we can proceed to submitting data. Data submission will entail typically a POST HTTP request with metadata and data, returning the status of the submission attempt. After submitting data, I'll encourage users to fetch their data to make sure that the data submission worked. Last, if the data repository has a landing page for the uploaded dataset within a short period, we can bring the user to that page to make sure everything is correct.

The following is an example of what a workflow will look like using pseudo-code.

Authenticate:

```
library(deposits)
Sys.setenv(DRYAD_TOKEN = "some-token")
deposits::auth(repo = "dryad")
```

Prepare metadata and data:

```
x <- readr::read_csv("users-data.csv")
my_metadata <- deposits::prep_metadata(x)
my_data <- deposits::prep_data(x)
```

Submit metadata and data:

```
out <- deposits::submit(data = my_data, metadata = my_metadata)
```

Fetch and browse data:

```
out$id
#> "dataset-identifier"
deposits::fetch(x$id)
deposits::browse(x$id)
```

I anticipate using the following packages to support the proposed components:

- Authentication: httr, crul
- Prepare data: vroom, data.table
- Prepare metadata: various, including: xml2, jsonlite, rdflib
- Submit in R / Fetch data / Browse data: httr, crul

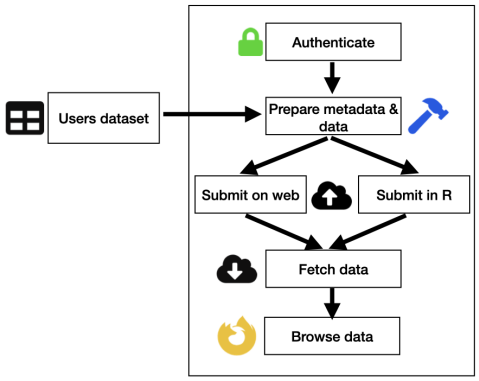

Figure 1: Icons from FontAwesome

It may make sense after some work is done to break off parts of the package into additional package(s). The R package devtools has shown us that placing too many disparate tasks in a single package can be better accomplished by separating them.

Project plan

Start-up phase

We'll start with research on data repositories. This will entail using the Registry of Research Data Repositories, including communication with repository managers to clarify any details which are not clear. In addition, we'll identify users of each repository to help us test the package. Details on data repository research:

- Select two data repositories to focus on. Build out documentation on how to add additional data repositories so that most of the work of extending to new repositories is done by community members
- Find the most common set of actions we can perform with data repositories
- Find data repositories that do not support programmatic data upload - for these we can help users prepare data for browser based submission
- Does each data repository use any standard or vocabulary we can use to help prepare/validate data for users?

Technical delivery

- Design plugin system to allow for anyone to extend the package for any additional data repositories
- Design public API (user facing functions) with function names that convey meaning following the usethis package model.
- Implement authentication following best practices for security
- Implement data fetch methods first - as this is easier than creating data and users will want it anyway - this will help us get familiar with each data repository
- Implement metadata and data preparation methods
- Implement data upload methods
- Documentation: write detailed documentation, including many vignettes, on a pkgdown site
- User testing: we'll test the package ourselves throughout the development process; this step is to get users to test the package to smooth out any user interface problems

Dissemination

- License will be MIT
- Blog posts, ropensci community call, conf. talk, R Consortium blog post
- The code will be developed in the open in the rOpenSci GitHub organization. Discussion of the deposit project will happen in public issues of the repository. The license will be MIT.

Project plan

Technical delivery

Eight weeks in total. 4 hours per day, 4 days per week (16 hrs/wk * 8 weeks = 128 hrs)

- Phase 1 (Weeks 1-2 (32 hrs)): Data repositories research and implement authentication
- Phase 2 (Weeks 3-6 (64 hrs)): Implement methods for fetch, metadata/data preparation, and data upload
- Phase 3 (Week 7-8 (32 hrs)): Documentation and user testing

Requirements

Programming time is the only requirement. All of the requested funds are for paying salary of the one programmer.

People

Scott Chamberlain - rOpenSci Scott is an rOpenSci Co-founder and technical lead. He has 14 years of experience with R, and maintains dozens of R packages on CRAN, most of which have significant components concerned with remote resources. In addition, Scott maintains one of the three major HTTP clients for R.

Processes

This project will use a Code of Conduct following the rOpenSci Code of Conduct https://ropensci.org/code-of-conduct/

Funding

I request a total of $16,000 to support 2 months work for myself, all of which will be salary.

Success

Definition of done

This project will be complete when a stable version of deposit is on CRAN and users are starting to give feedback - at that point it will be on its way to wider adoption.

Measuring success

Success will be measured by interest gaged via GitHub (stars, forks, opened issues), and downloads from CRAN. Most importantly for the long-term success of the software, I'd like to attract one or more additional maintainers to make sure the project is more sustainable moving forward - and because complex projects are easier with more maintainers. There are many technical university librarians that have been involved in bug reports and writing tutorials for software I maintain for academic metadata sources. Thus, it's likely these same people will be interested in the proposed software herein and perhaps aid in maintenance.

We can also measure success by what users do with the software. It is hard to measure how R users use R packages because there's no way to track usage. However, we will include user agent strings uniquely identifying HTTP requests from our software; then we can ask data repositories to tell us how many requests they get from our user agent string. Citations will not likely work since it's not likely that users would cite this kind of software in a paper. It is likely that this software could show up in training materials created by librarians at universities.

Key risks

A software client for any remote API runs the risk of breaking because remote APIs can often change with little notice. We have a lot of experience with this issue. Two ways to minimize effects of changing APIs are to a) subscribe to any mailing lists, fora, etc. where API changes are announced, and b) make sure the package fails well, making it easy for maintainers to find problems as well for users to understand errors.

References

Christensen G, Dafoe A, Miguel E, Moore DA, Rose AK (2019) A study of the impact of data sharing on article citations using journal policies as a natural experiment. PLoS ONE 14(12): e0225883. https://doi.org/10.1371/journal.pone.0225883

Fecher, B., Friesike, S., & Hebing, M. (2015). What Drives Academic Data Sharing? PLOS ONE, 10(2), e0118053. https://doi.org/10.1371/journal.pone.0118053

Resnik, D. B., Morales, M., Landrum, R., Shi, M., Minnier, J., Vasilevsky, N. A., & Champieux, R. E. (2019). Effect of impact factor and discipline on journal data sharing policies. Accountability in Research, 26(3), 139–156. https://doi.org/10.1080/08989621.2019.1591277

Stodden, V., Seiler, J., & Ma, Z. (2018). An empirical analysis of journal policy effectiveness for computational reproducibility. Proceedings of the National Academy of Sciences, 115(11), 2584–2589. https://doi.org/10.1073/pnas.1708290115

Vines, T. H., Andrew, R. L., Bock, D. G., Franklin, M. T., Gilbert, K. J., Kane, N. C., ... Yeaman, S. (2013). Mandated data archiving greatly improves access to research data. The FASEB Journal, 27(4), 1304–1308. https://doi.org/10.1096/fj.12-218164