SOFTWARE TOOL ARTICLE

REVISED DataPackageR: Reproducible data preprocessing, standardization and sharing using R/Bioconductor for collaborative data analysis [version 2; referees: 2 approved, 1 approved with reservations]

A practical approach to wrangling multi-assay data sets for standardized and reproducible data analysis

Greg Finak[1-3], Bryan Mayer[1-3], William Fulp[1-3], Paul Obrecht[1-3], Alicia Sato[1-3], Eva Chung[1-3], Drienna Holman[1-3], Raphael Gottardo[1-3]

[1]Vaccine and Infectious Disease Division, Fred Hutchinson Cancer Research Center, Seattle, WA, 98109, USA
[2]Vaccine Immunology Statistical Center, Fred Hutchinson Cancer Research Center, Seattle, WA, 98109, USA
[3]Statistical Center For HIV AIDS Research and Prevention, Fred Hutchinson Cancer Research Center, Seattle, WA, 98109, USA

v2 **First published:** 22 Jun 2018, **2**:31 (doi: 10.12688/gatesopenres.12832.1)
Latest published: 10 Jul 2018, **2**:31 (doi: 10.12688/gatesopenres.12832.2)

Abstract

A central tenet of reproducible research is that scientific results are published along with the underlying data and software code necessary to reproduce and verify the findings. A host of tools and software have been released that facilitate such work-flows and scientific journals have increasingly demanded that code and primary data be made available with publications. There has been little practical advice on implementing reproducible research work-flows for large 'omics' or systems biology data sets used by teams of analysts working in collaboration. In such instances it is important to ensure all analysts use the same version of a data set for their analyses. Yet, instantiating relational databases and standard operating procedures can be unwieldy, with high "startup" costs and poor adherence to procedures when they deviate substantially from an analyst's usual work-flow. Ideally a reproducible research work-flow should fit naturally into an individual's existing work-flow, with minimal disruption. Here, we provide an overview of how we have leveraged popular open source tools, including Bioconductor, Rmarkdown, git version control, R, and specifically R's package system combined with a new tool *DataPackageR*, to implement a lightweight reproducible research work-flow for preprocessing large data sets, suitable for sharing among small-to-medium sized teams of computational scientists. Our primary contribution is the *DataPackageR* tool, which decouples time-consuming data processing from data analysis while leaving a traceable record of how raw data is processed into analysis-ready data sets. The software ensures packaged data objects are properly documented and performs checksum verification of these along with basic package version management, and importantly, leaves a record of data processing code in the form of package vignettes. Our group has implemented this work-flow to manage, analyze and report on pre-clinical immunological trial data from multi-center, multi-assay studies for the past three years.

Open Peer Review

Referee Status: ✓ ✓ ?

	Invited Referees 1	2	3
REVISED version 2 published 10 Jul 2018	✓ report	✓ report	? report
version 1 published 22 Jun 2018	? report		

1 **Aaron T.L. Lun**, University of Cambridge, UK

2 **Ted Laderas**, Oregon Health & Science University (OHSU), USA

3 **Ben Marwick**, University of Washington, USA

Discuss this article

Comments (0)

Keywords
Assay data, Bioconductor, Collaboration, Data science, Package, Reproducibility, Rmarkdown, Rstats, Version control

Corresponding authors: Greg Finak (gfinak@fredhutch.org), Raphael Gottardo (rgottard@fredhutch.org)

Author roles: Finak G: Conceptualization, Funding Acquisition, Investigation, Methodology, Software, Supervision, Writing – Original Draft Preparation, Writing – Review & Editing; **Mayer B**: Data Curation, Methodology, Supervision, Validation, Writing – Review & Editing; **Fulp W**: Data Curation, Formal Analysis, Writing – Review & Editing; **Obrecht P**: Data Curation, Software, Validation; **Sato A**: Project Administration, Resources, Supervision; **Chung E**: Project Administration; **Holman D**: Data Curation, Supervision, Validation; **Gottardo R**: Conceptualization, Funding Acquisition, Investigation, Project Administration, Supervision, Writing – Review & Editing

Competing interests: No competing interests were disclosed.

Grant information: This work was supported by the Bill and Melinda Gates Foundation [OPP1032317; to RG]; and the National Institute of General Medical Sciences [R01 GM118417-01A1; to GF].
The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

How to cite this article: Finak G, Mayer B, Fulp W *et al.* **DataPackageR: Reproducible data preprocessing, standardization and sharing using R/Bioconductor for collaborative data analysis [version 2; referees: 2 approved, 1 approved with reservations]** Gates Open Research 2018, **2**:31 (doi: 10.12688/gatesopenres.12832.2)

First published: 22 Jun 2018, **2**:31 (doi: 10.12688/gatesopenres.12832.1)

REVISED **Amendments from Version 1**

This version of the manuscript has been updated to address the comments of Dr. Aaron Lun.

We have added text noting that scripts in languages other than R (e.g., bash, python) are supported through rmarkdown/knitr's multi-language code chunk support.

We have described our support of multi-script workflows by extending the description of the YAML configuration to include the render_root property and describing the datapacakge_object_read() API.

We have added a paragraph describing how DataPackageR has benefitted the processing and analysis of the MX1 study described in the manuscript and described this in the context of the overall workload of VISC.

We have added a Caveats section describing some of the shortfalls of the framework, specifically the need to support remotely cached data, and the risk of git repository bloat, and how to mitigate this issue.

This version also includes updates that reflect v 0.14.0 of DataPackageR. We have updated function names that now use snake_case and made several small typo corrections.

See referee reports

Introduction

A central idea of reproducible research is that results are published along with underlying data and software code necessary to reproduce and verify the findings. Termed a *research compendium*, this idea has received significant attention in the literature[1–5].

Many software tools have since been developed to facilitate reproducible data analytics, and scientific journals have increasingly demanded that code and primary data be made publicly available with scientific publications[2,6–27]. Tools like git and Github, figshare, and Rmarkdown are increasingly used by researchers to make code, figures, and data open, accessible and reproducible. Nonetheless, in the life sciences, practicing reproducible research with large data sets and complex processing pipelines continues to be challenging.

Data preprocessing, quality control (QC), data standardization, analysis, and reporting are tightly coupled in most discussions of reproducible research, and indeed, literate programming frameworks such as Sweave and Rmarkdown are designed around the idea that code, data, and research results are tightly integrated[2,25]. Tools like Docker, a software container that virtualizes an operating system environment for distribution, have been used to ensure consistent versions of software and other dependencies are used for reproducible data analysis[16]. The use of R in combination with other publicly available tools has been proposed in the past to build reproducible research compendia[3,28,29]. Many existing tools already implement such ideas. The *workflowr* package provides mechanisms to turn a data analysis project into a version-controlled, documented, website presenting the results. The *drake* package[30] is a general purpose work-flow manager that implements analytic "plans", caching of intermediate data objects, and provides scalability, and provides tangible evidence of reproducibility by detecting when code, data and results are in sync.

However, tight coupling of preprocessing and analysis can be challenging for teams analyzing and integrating large volumes of diverse data, where different individuals in the team have different areas of expertise and may be responsible for processing different data sets from a larger study. These challenges are compounded when a processing pipeline is split across multiple teams. A primary problem in data science is the programmatic integration of software tools with dynamic data sources.

Here, we argue that data processing, QC, and analysis can be treated as modular components in a reproducible research pipeline. For some data types, it is already common practice to factor out the processing and QC from the data analysis. For rnaseq data, for example, it is clearly impractical and time consuming to re-run monolithic code that performs alignment, QC, gene expression quantification, and analysis each time the downstream analysis is changed. Our goal is to ensure that downstream data analysis maintains dependencies on upstream raw data and processing but that the processed data can be efficiently distributed to users in an independent manner and updated when there are changes.

Here, we present how the Vaccine Immunology Statistical Center (VISC) at the Fred Hutchinson Cancer Research Center has addressed this problem and implemented a reproducible research work-flow that scales to medium-sized collaborative teams by leveraging free and open source tools, including R, Bioconductor and git[22,31].

Methods

Operation

In order to use *DataPackageR* an R installation (≥3.5.0) is required. Associated dependencies are listed in the package's DESCRIPTION file and are automatically installed when using the *install_github()* API from the *devtools* package. There are no minimum memory, CPU, or storage requirements apart from what is necessary to perform data processing, which varies on a case–by-case basis.

Implementation

Our work-flow is built around the *DataPackageR* R package, which provides a framework for decoupling data preprocessing from data analysis, while maintaining traceability and data provenance[18].

DataPackageR builds upon the features already provided by the R package system. R packages provide a convenient mechanism for including documentation as part of the built-in help system, as well as long-form `vignettes`, and version information and distribution of the entire package. Importantly, R packages often include data stored as R objects, and some packages, particularly under BioConductor, are devoted solely to the distribution of data sets[22]. The accepted mechanism for such distribution is to store R objects as `rda` files in the `data` directory of the package source tree and to store the source code used to produce those data sets in `data-raw`. The *devtools* package provides

some mechanisms to process the source code into stored data objects[32].

Data processing code provided by the user (in the form of Rmd files preferably, and R files optionally) is run and the results are automatically included as package vignettes, with output data sets (specified by the user) included as data objects in the package. Additional languages such as *bash* and *python* scripts are supported via rmarkdown's multi-language code chunk support. Notably, this process, while apparently mirroring much of the existing R package build process, is disjointed from it, thereby allowing the decoupling of computationally long or expensive data processing from routine package building and installation. This allows *DataPackageR* to decouple data munging and tidying from data analysis while maintaining data provenance in the form of a vignette in the final package where the end-user can view and track how individual data sets have been processed. This is particularly useful for large or complex data that involve extensive preprocessing of primary or raw data (e.g. alignment of fastq files for rnaseq or gating of fcm data), and where computation may be prohibitively long or involve software dependencies not immediately available to the end-user.

DataPackageR implements these features on top of a variety of *tidyverse* tools including *devtools, roxygen2, rmarkdown, utils, yaml, purrr*. The complete list of package dependencies is in the package `DESCRIPTION` file.

Package structure

To construct a data package using *DataPackageR*, the user invokes the `datapackage.skeleton()` API, which behaves like R's *`package.skeleton()`*, creating the necessary directory structure with some modifications. A listing of the structure of *DataPackageR* skeleton package directory, with other associated files is shown below:

```
package root
|--- datapackager.yml # Configuration file controlling
|                     # the package build process.
|--- DESCRIPTION # Adds a DataVersion
|                # string to version the
|                # data set.
|--- NAMESPACE
|--- DATADIGEST # Stores an MD5 hash of each
|               # data object in the package.
|--- R
|--- Read-and-delete-me.txt    # Further instructions
|                              # on building the package.
|--- data             # Holds processed, analysis-ready data objects.
|--- data-raw         # User code for data
|                     # processing is placed here by
|                     # datapackage.skeleton().
|--- documentation.R # Auto generated roxygen documentation
|                   # for data set objects.
|--- inst
| |___ extdata # (small) raw data files.
| |___ doc   # Processed vignettes are moved here.
|            # Data processing code is accessible in the
|            # final package via the vignette() API.
|--- vignettes # Scripts in data-raw
|              # are processed into vignettes.
|___ man # Autogenerated documentation is processed
         # into rd files.
```

The `datapackage_skeleton` API takes several new arguments apart from the package *name*. First, *`code_files`* takes a vector of paths to *Rmd* or *R* scripts that perform the data processing. These are moved into the `data-raw` directory. The argument *`r_object_names`* takes a vector of quoted R object names. These are objects that are to be stored in the final package and it is expected that they are created by the code in the *R* or *Rmd* files. These can be *tidy* data tables, or arbitrary R objects and data structures (e.g. *S4* objects) that will be consumed by the package end-user. Information about the processing scripts and data objects is stored in a configuration file named `datapackager.yml` in the package root directory and only used by the package build process. The scripts may read raw data from any location, but generally the package maintainer should place it in `inst/extdata` if file size is not prohibitive for distribution.

The build_package API

Once code and data are in place, the `build_package()` API invokes the build process. This API is the only way to invoke the execution of code in `data-raw` to produce data sets stored in `data`. It is not invoked through R's standard `R CMD build` or `R CMD INSTALL` APIs, thereby decoupling long and computationally intensive processing from the standard build process invoked by end-users. Upon invocation of `build_package()` the *R* and *Rmd* files specified in `datapackager.yml` will be compiled into *package vignettes* and moved into the `inst/doc` directory, data objects will be created and moved into `data`, data objects will be version tagged with their *checksum* and recorded in the *DATADIGEST* file in the package root, and a *roxygen* markup skeleton will be created for each data object in the package.

YAML configuration

The `datapackager.yml` configuration file in the package root controls the build process by specifying which `R` and `Rmd` files should be processed and which `named R objects` are expected to be included as data sets in the package. The `render_root` property points to a directory where R and Rmd files will be processed, allowing multi-script pipelines to share file system artifacts. The `datapackage_object_read()` API also allows downstream scripts to read data objects creted by earlier scripts. The listing below shows the structure of the YAML configuration file used by *DataPackageR* to control compilation and inclusion of data objects in the package:

```
configuration:
  files:                              # files property lists
    process_dataset_one.Rmd:          # R or Rmd code files
      name: process_dataset_one.Rmd   # Each file has a name
      enabled: yes                    # The enabled property specifies
                                      # if the file should be processed
    process_dataset_two.Rmd:
      name: process_dataset_two.Rmd
      enabled: yes
  objects:                            # A list of the data objects created
  - dataset_one                       # by processing the files.
  - dataset_two
  - dataset_three
render_root:                          # root directory where scripts are
  tmp: '298918'                       # render()ed for multi-script
                                      # pipelines
```

The API for interacting with the YAML configuration file is outlined in Table 1.

Table 1. The API for interacting with a YAML config file used by DataPackageR allows the user to add and remove data objects and code files, toggle compilation of files, and read and write the configuration to the data package.

API call	Property	return value
`yml_add_files()`	file	config object
`yml_remove_files()`	file	config object
`yml_add_objects()`	objects	config object
`yml_remove_objects()`	objects	config object
`yml_find()`	config file	config object
`yml_write()`	config file	null
`yml_enable_compile()`	enable	config object
`yml_disable_compile()`	enable	config object

Accessing raw data from scripts

Users can access the package root and `inst/extdata` directories from within R and Rmd scripts in a portable manner using the `project_path()` and `project_extdata_path()` APIs. Raw data stored outside the package root should be accessed relative to these locations.

Dataset versioning

During the build, the `DATADIGEST` file is auto-generated. This file contains an `md5` hash of each data object stored in the package as well as an overall data set version string. These hashes are checked when the package is rebuilt; if they do not match, it indicates the format of the processed data has changed (either because the primary data has changed, or because the processing code has changed to update the data set). In these cases, the *DATADIGEST* for the changed object is updated and the minor version of the `DataVersion` string in the `DESCRIPTION` file is automatically incremented. The DataVersion for a package can be checked by the `data_version()` and `assert_data_version()` APIs, allowing end-users to produce reports based on the expected version of a data set (Figure 1).

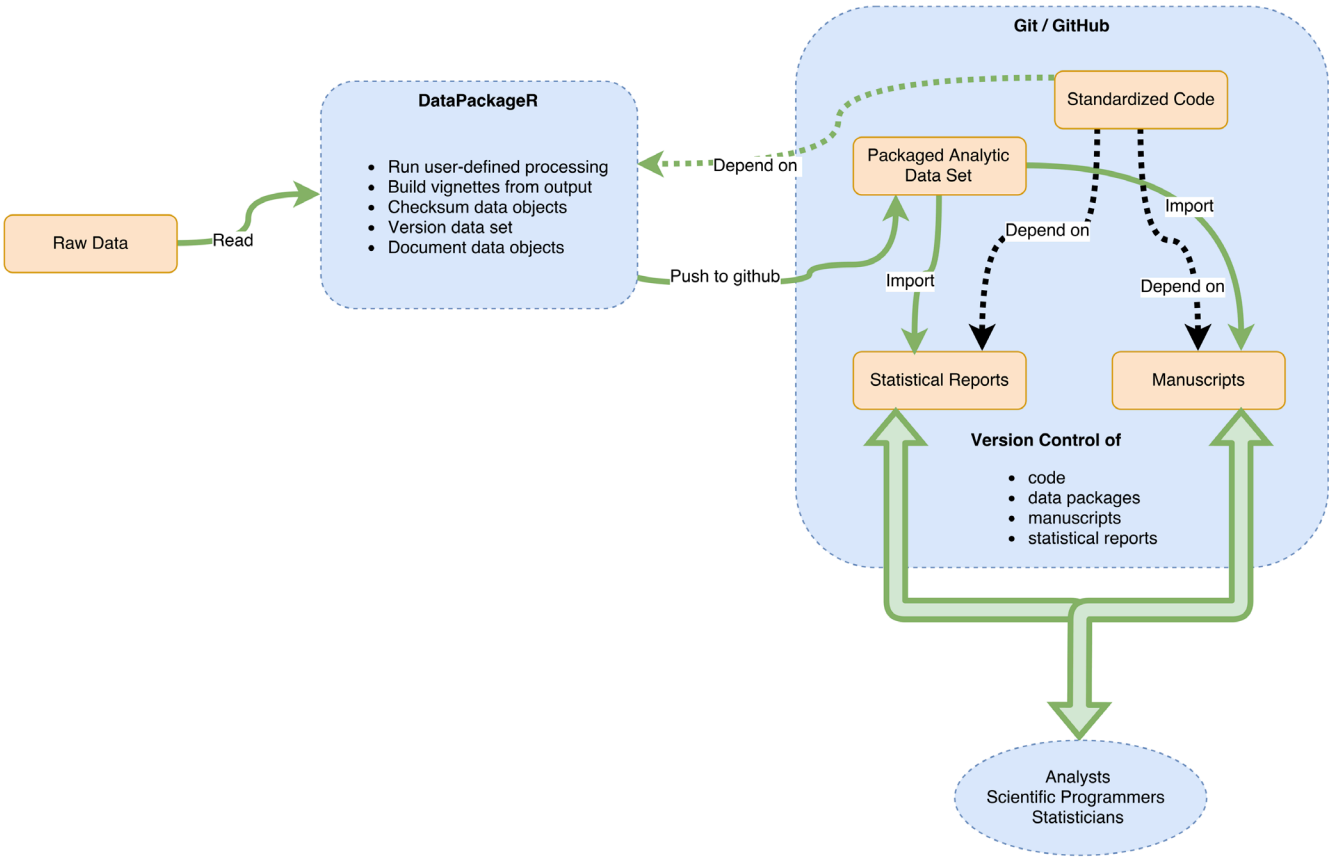

Figure 1. A schematic overview of the components in our reproducible data packaging work-flow that decouples data processing from data analysis.

Data documentation

DataPackageR ensures that documentation is available for each data object included in a package by automatically creating a *roxygen* markup stub for each object that can then be filled in by the user. Undocumented objects are explicitly excluded from the final package.

Packages can be readily distributed in source or tarball form (together with the processed data sets under `data/` and raw data sets under `inst/extdata`). Within VISC we leverage *git* and *github* to provide version control of data package source code. By leveraging *DataPackageR*, the data processing is decoupled from the usual build process and does not need to be run by the end-user each time a package is downloaded and installed. Documentation in the form of *Rd* files, one for each data object in the package, as well as *html vignettes* describing the data processing, are included in the final package. These describe the data sets as well as how data was transformed, filtered, and otherwise processed from its raw state.

Use cases

DataPackageR was developed as a lightweight alternative to existing reproducible work-flow tools (e.g. *Galaxy*[33]), or to fully fledged database solutions that are often beyond the scope of most short-term projects. *DataPackageR* plugs easily into any existing R-based data analysis work-flow, since existing data processing code needs only to be formatted into Rmarkdown (ideally). It is particularly suited for long-running or complex data processing tasks, or tasks that depend on large data sets that may not be available to the end user (e.g. FASTQ alignment or raw flow cytometry data processing). Such tasks do not fit well into the standard `R CMD build` paradigm, for example either as vignettes or *.R* files under `/data` since these would be invoked each time an end user builds a package from source. We desire, however, to maintain a link between the processed data sets and the processing code that generates them. We note that *DataPackageR* is distinct from other reproducible research frameworks such as *workflowr* or *drake*[30], in that it is designed to *reproducibly prepare data for analysis*, using an *existing code base*, with little additional effort. The product of *DataPackageR* is nothing more than an R package that can be used by anyone. The resulting data packages are meant to be shared, to serve as the basis for further analysis (Figure 1) and distributed as part of publications. These downstream analyses may leverage any of the existing work-flow management tools. Our goal is that data sets forming the basis of scientific findings can be confidently shared in their processed form which is often much smaller and easier to distribute.

Within the VISC, a team of analysts, statistical programmers, and data managers work collaboratively to analyze pre-clinical data arising from multiple trials. There are multiple assays per trial. The challenges associated with ensuring the entire team works from the same version of a frequently changing and dynamic data set, motivated the development of *DataPackageR*. The tool is routinely used to process and standardize trial data for submission to The Collaboration for AIDS Vaccine Discovery (CAVD) Data Space.

We demonstrate how *DataPackageR* is used to process and package multiple types of assay data from an animal trial of an experimental HIV vaccine.

Data

We demonstrate the use of *DataPackageR* for processing data from a vaccine study, named *MX1*, designed to examine the antibody responses to heterologous N7 Env prime-boost immunization in macaques. The study had four treatment groups plus a control arm, with six animals per group. Samples were collected at three time points: t1: baseline, post-prime 2, post-boost 1, t2: post-boost 2, t3: post-boost 3. Six assays were run at each time point, using either serum samples or peripheral blood mononuclear cells (PBMCs). The assays were: 1) enzyme-linked immunosorbent assay (ELISA), an immunological assay that enables detection of antibodies, antigens, proteins and/or glycoproteins (serum); 2) a neutralizing antibody (Ab) assay (serum); 3) a binding antibody multiplex assay (BAMA) to assess antibody response breadth (serum); 4) a BAMA assay to permit epitope mapping (serum); 5) an antibody dependent cellular cytotoxicity (ADCC) assay (serum); and 6) an intracellular cytokine staining assay to assess cellular responses (PBMCs).

The raw data and environment to reproduce the processing with *DataPackageR* are distributed as a Docker image on `hub.docker.com` as `gfinak/datapackager:latest`. We have restricted the number of FCS files distributed in the container to limit the size of the image and speed up processing of FCM data for demonstration purposes.

Flow cytometry and other assay data

Flow cytometry (FCM) is a high content, high throughput assay for which VISC leverages specialized data processing and analytics tools. Raw FCS files and manual gate information in the form of FlowJo (FlowJo LLC, Ashland, OR) workspace files are uploaded directly to VISC by the labs. The raw data are processed with open source BioConductor software (*flowWorkspace*) to import and reproduce the manual gating, extract cell subpopulation statistics, and access the single-cell event-level data required for downstream modeling of T-cell polyfunctionality and immunogenicity[34–36]. Tables of extracted cell populations, cell counts, proportions, and fluorescence intensities are included in study packages, together with an Rmarkdown vignette describing the data processing. Due to the size of the raw FCS files, they are imported for processing from a location external to the data package source tree, so that the raw files are not part of the final package, but vignettes outlining the data processing are automatically included.

Remaining assay data are of reasonable size and are provided raw data in tabular (csv) form, imported into the package, processed and standardized from the `inst/extdata` package directory. Users can run and connect to an Rstudio instance in the container, where code and data to build the *MX1* data package reside.

MX1 is just one study from over 33 prepared by VISC over the past 3 year using the *DataPackageR* infrastructure. Each

study consists of three to seven data types, each with different QC and standardization criteria, each produced by different labs, all on a different time line. For each data type, VISC produces QC reports for the labs, statistical analysis plans for each data type, statistical reports and analyses for each data type, and an integrated analysis for manuscripts and publications. The data generally arrive over a time period of several months, and reports are also generated over several months. Different individuals are responsible for the generation of different reports. It is critical that all reports are based off the same data set, and that they are updated when individual data sets are updated. For the MX1 study above, the data package was updated 29 times between 2015 and 2017. The data set version was incremented from 0.1.0 to 0.2.21 over that time frame. Six data sets were added to the package over two years, 17 commits were associated with corrections or updates to data sets (e.g. additional or corrected annotations). Six follow up reports based on the data package were updated each time there was an addition, or correction to the data. DataPackageR helped ensure consistency between the different reports and the data, something that would have been incredibly difficult without this framework.

Caveats

We note a number of limitations with the existing framework that we hope to address in the future. First, our dependence on git and gtihub to version control data sets means that in the long-term the size of the local copy of a repository will grow with every changed data set. We are exploring approaches to cache older versions of data sets remotely, analogous to approaches taken by BioConductor's ExperimentHub (https://doi.org/doi:10.18129/B9.bioc.ExperimentHub), and packages like datastorr (https://github.com/ropenscilabs/datastorr).

Summary

Reproducibility is increasingly emphasized for scientific publications. We describe a new utility R package, *DataPackageR* that serves to help automate and track the processing and standardization of diverse data sets into *analysis-ready data packages* that can be easily distributed for analysis and publication. *DataPackageR*, when paired with a version control system such as *git*, decouples data processing from data analysis while tracking changes to data sets, ensuring data objects are documented, and keeping a record of data processing pipelines as vignettes within the data package. The principle behind the tool is that it remains a lightweight and non-intrusive framework that easily plugs into most R-based data analytic work-flows. It places few restrictions on the user code therefore most existing scripts can be ported to use the package. The VISC has been using *DataPackageR* for a number of years to perform reproducible end-to-end analysis of animal trial data, and the package has been used to publicly share sets for a number of published manuscripts[35,37,38].

Software and data availability

Source code available from: http://github.com/RGLab/DataPackageR

Archived source code as at time of publication: https://doi.org/10.5281/zenodo.1292095[39]

Reproducible examples: http://hub.docker.com/r/gfinak/datapackager/

License: MIT license

Partial data for the MX1 study to demonstrate processing using *DataPackageR* are available as a docker container from `gfinak/DataPackageR:latest`. The processed MX1 data are available on the CAVD DataSpace data sharing and discovery tool at http://dataspace.cavd.org under study identifier, CAVD 451. At the time of publication the complete data set is available to CAVD members only.

Competing interests

No competing interests were disclosed.

Grant information

This work was supported by the Bill and Melinda Gates Foundation [OPP1032317; to RG]; and the National Institute of General Medical Sciences [R01 GM118417-01A1; to GF].

The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript.

Acknowledgements

The authors wish to acknowledge the contributions of the members of the VISC and the CAVD Data Space (CDS) for contributions to testing and feedback on the software. We also acknowledge the Collaboration for AIDS Vaccine Discovery (CAVD), the Comprehensive Antibody Vaccine Immune Monitoring Consortium (CAVIMC) and the Comprehensive Cellular Vaccine Immune Monitoring Consortium (CCVIMC), as well as Dr. Shiu-Lok Hu.

References

1. Baggerly KA, Coombes KR: **What information should be required to support clinical "omics" publications?** *Clin Chem.* 2011; **57**(5): 688–690.
PubMed Abstract | Publisher Full Text

2. Gentleman R, Lang DT: **Statistical analyses and reproducible research.** *Bioconductor Project Working Papers.* 2004.
Reference Source

3. Marwick B, Boettiger C, Mullen L: **Packaging data analytical work reproducibly using R (and friends).** *PeerJ Preprints*; PeerJ Inc., 2018.
Publisher Full Text

4. Stodden V: **Enabling reproducible research: Open licensing for scientific innovation.** *International Journal of Communications Law and Policy.* 2009.
Reference Source

5. Stodden V, Borwein J, Bailey DH: **Publishing standards for computational science: "Setting the default to reproducible"** 2013.
Reference Source

6. Lortie CJ: **A review of R for data science: Key elements and a critical analysis.** *PeerJ Preprints*; PeerJ Inc., 2017.
Publisher Full Text

7. Wickham H, Grolemund G: **R for data science: Import, tidy, transform, visualize, and model data.** 'O'Reilly Media, Inc.', 2016.
Reference Source

8. Huang Y, Gottardo R: **Comparability and reproducibility of biomedical data.** *Brief Bioinform.* 2013; **14**(4): 391–401.
PubMed Abstract | Publisher Full Text | Free Full Text

9. Buck S: **Solving reproducibility.** *Science.* 2015; **348**(6242): 1403.
PubMed Abstract | Publisher Full Text

10. Peng R: **The reproducibility crisis in science: A statistical counterattack.** *Significance.* 2015; **12**(3): 30–32.
Publisher Full Text

11. Morrison SJ: **Time to do something about reproducibility.** *eLife.* 2014; **3**: e03981.
PubMed Abstract | Publisher Full Text | Free Full Text

12. Yaffe MB: **Reproducibility in science.** *Sci Signal.* 2015; **8**(371): eg5.
PubMed Abstract | Publisher Full Text

13. Begley CG, Ioannidis JP: **Reproducibility in science: Improving the standard for basic and preclinical research.** *Circ Res.* 2015; **116**(1): 116–126.
PubMed Abstract | Publisher Full Text

14. Stodden V, Leisch F, Peng RD: **Implementing reproducible research.** CRC Press, 2014.
Reference Source

15. Freedman LP, Inglese J: **The increasing urgency for standards in basic biologic research.** *Cancer Res.* 2014; **74**(15): 4024–4029.
PubMed Abstract | Publisher Full Text | Free Full Text

16. Boettiger C: **An introduction to docker for reproducible research.** *Oper Syst Rev.* 2015; **49**(1): 71–79.
Publisher Full Text

17. McNutt M: **Journals unite for reproducibility.** *Science.* 2014; **346**(6210): 679.
PubMed Abstract | Publisher Full Text

18. Peng RD: **Reproducible research in computational science.** *Science.* 2011; **334**(6060): 1226–1227.
PubMed Abstract | Publisher Full Text | Free Full Text

19. Gentleman R: **Reproducible research: A bioinformatics case study.** *Stat Appl Genet Mol Biol.* 2005; **4**(1): Article2.
PubMed Abstract | Publisher Full Text

20. Peng RD: **Reproducible research and Biostatistics.** *Biostatistics.* 2009; **10**(3): 405–408.
PubMed Abstract | Publisher Full Text

21. Mesirov JP: **Computer science. Accessible reproducible research.** *Science.* 2010; **327**(5964): 415–6.
PubMed Abstract | Publisher Full Text | Free Full Text

22. Gentleman RC, Carey VJ, Bates DM, *et al.*: **Bioconductor: open software development for computational biology and bioinformatics.** *Genome Biol.* 2004; **5**(10): R80.
PubMed Abstract | Publisher Full Text | Free Full Text

23. Finak G, Gottardo R: **Promises and Pitfalls of High-Throughput Biological Assays.** *Methods Mol Biol.* 2016; **1415**: 225–243.
PubMed Abstract | Publisher Full Text

24. Allaire J, *et al.*: **Rmarkdown: Dynamic documents for R**. 2015.

25. Xie Y: **Knitr: A comprehensive tool for reproducible research in R.** *Implement Reprod Res.* 2014; **1**: 20.
Reference Source

26. Baumer B, Udwin D: **R markdown.** *WIREs Comput Stat.* 2015; **7**(3): 167–177.
Publisher Full Text

27. Ram K: **Git can facilitate greater reproducibility and increased transparency in science.** *Source Code Biol Med.* 2013; **8**(1): 7.
PubMed Abstract | Publisher Full Text | Free Full Text

28. Project, T: **rOpenSci. Use of an r package to facilitate reproducible research**. 2015.

29. Project, T: **rOpenSci. A guide to reproducible research**. 2015.

30. Michael Landau W: **The drake R package: A pipeline toolkit for reproducibility and high-performance computing.** *JOSS.* 2018; **3**(21): 550.
Publisher Full Text

31. Ihaka R, Gentleman R: **R: A language for data analysis and graphics.** *J Comput Graph Stat.* 1996; **5**(3): 299–314.
Publisher Full Text

32. Wickham H, Chang W: **Devtools: Tools to make developing r packages easier**. 2018.
Reference Source

33. Goecks J, Nekrutenko A, Taylor J, *et al.*: **Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences.** *Genome Biol.* 2010; **11**(8): R86.
PubMed Abstract | Publisher Full Text | Free Full Text

34. Finak G, Jiang M, Andre M, *et al.*: **FlowWorkspace: A new R package for importing flow cytometry data into bioconductor from flowJo.** Fred Hutchinson Cancer Research Center; Poster B232, CYTO 2011 XXVI Congress of the International Society for Advancement of Cytometry Baltimore Convention Center, Baltimore, Maryland, USA May 21 Ð 25, 2010.

35. Lin L, Finak G, Ushey K, *et al.*: **COMPASS identifies t-cell subsets correlated with clinical outcomes.** *Nat Biotechnol.* 2015; **33**(6): 610–616.
PubMed Abstract | Publisher Full Text | Free Full Text

36. Finak G, McDavid A, Chattopadhyay P, *et al.*: **Mixture models for single-cell assays with applications to vaccine studies.** *Biostatistics.* 2014; **15**(1): 87–101.
PubMed Abstract | Publisher Full Text | Free Full Text

37. Finak G, McDavid A, Yajima M, *et al.*: **MAST: A flexible statistical framework for assessing transcriptional changes and characterizing heterogeneity in single-cell RNA sequencing data.** *Genome Biol.* 2015; **16**: 278.
PubMed Abstract | Publisher Full Text | Free Full Text

38. Bolton DL, McGinnis K, Finak G, *et al.*: **Combined single-cell quantitation of host and SIV genes and proteins *ex vivo* reveals host-pathogen interactions in individual cells.** *PLoS Pathog.* 2017; **13**(6): e1006445.
PubMed Abstract | Publisher Full Text | Free Full Text

39. Finak G, Obrecht P: **RGLab/DataPackageR v0.13.2 (Version v0.13.2).** *Zenodo.* 2018.
Data Source

Open Peer Review

Current Referee Status: ✔ ✔ ?

Version 2

Referee Report 25 July 2018

doi:10.21956/gatesopenres.13917.r26542

? **Ben Marwick**
Department of Anthropology, University of Washington, Seattle, WA, USA

This is a well-written and clear paper about a very interesting package that helps to raise awareness about the challenges of collaborating on data sets that change over time. The package describes is a useful addition to the toolkit of R-based infrastructure to promote transparent and open research.

Previous work includes workflowr and drake. This is a very incomplete list. There are many other R packages that deal specifically with data provenance that would be relevant to note here. And it's not clear specifically what the previous work lacks? The authors say that rerunning rnaseq code is impractical and time consuming, but there are many existing approaches to manage that problem, e.g. the venerable make. It's not clear how this package solves a problem that can't be solved with existing packages. How is their package different to any previous effort? What specific problem does it solve? What makes it notable and a unique contribution? I'm confident the authors can answer these questions, and I think it would make the ms. more useful and impactful if they could do so more directly and specifically. The reader will want to know that this is not just a re-invention of the wheel, or a thin wrapper around something they already use heavily.

The detail that it is a lightweight alternative to Galaxy is an important motivation that should be noted to the reader earlier in the paper. This sentence seems to hint at the unique value of this package: "it is designed to reproducibly prepare data for analysis, using an existing code base" but it needs more description and elaboration to more clearly show the reader why this package is necessary. It would also be helpful to hear more about the research context where this package is optimal, you say something about medium-sized organisations. But for researchers who don't do DNA analysis and flow cytometry, a little more detail on the size and structure of the research team and typical research process will help them recognise if this solves a problem they also have.

It is a nice touch to prompt documentation of data sets. I wonder about having so many top-level non-standard package files, that will be off-putting to some people who are not used to overloading the package structure that.

How do packages made by this package communicate to the user the initial source of the data, and the final location? For example, how does it capture the date, time, instrument, etc where the data came from, and the DOI and repository of where the data has been finally deposited? Is this added in free text in the roxygen documentation, or are there fields especially for this? The system provided by this package seems a bit disconnected from the full pipeline, which might limit its archival value. Seems that it's optimised for work-in-progress data management, with less concern for the long term. Where do the

authors recommend the resulting packages go when the work is complete? I don't think GitHub is a good solution for long term archiving.

There is a mention of Docker in the 'Use Case' section, but no explanation of whether or not the package no handles this, or not. Also in this section it would be good to have URLs to the packages being discussed so the reader can inspect them in more detail, and see proof of the concept.

In the 'caveat' section there is a mention of the size of the data, but no specific details. It would be more helpful to users to know the ballpark sizes that this package is good for. It is up to 5 MB of processed data files? Do the authors have any guidance or rules of thumb such as 10 MB of FCM raw data can typically be stored as 5MB of package data? These would be very helpful for potential users to gauge the usefulness of this tool their own situation.

In the console output for DataPackageR::datapackage_skeleton it would be nice to use usethis:::done() for more consistent formatting of the messages. Also, in the example code on the GitHub readme, creating the pkg in a tmp directory makes it quite hard to find to edit the yml file and inspect the contents. I understand that using a tempfile makes it easy to test and knit, but it's not very user-friendly. Perhaps worth noting somewhere that the packages this package creates are not CRAN-ready (I got 1 warning and 1 note with the mtcars20 package). It would be good to have some guidance about how the authors recommend packages created by this method be shared (presumably github?).

Is the rationale for developing the new software tool clearly explained?
Partly

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: No competing interests were disclosed.

Referee Expertise: I have co-authored a similar R package to help with organising research materials into a compendium to enable reproducibility and versioning. I am not familiar with omics research.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Referee Report 24 July 2018

doi:10.21956/gatesopenres.13917.r26579

Ted Laderas
Department of Medical Informatics and Clinical Epidemiology, Knight Cancer Institute, Oregon Health & Science University (OHSU), Portland, OR, USA

The authors have designed a package that allows for bundling of multiple data types that can be versioned, along with preprocessing scripts and raw data into an R data package. The main use case for this package seems to be systems biology projects where multiple datatypes (such as RNAseq, DNAseq, flow cytometry, etc.) need to be aggregated together for analysis by multiple groups. As a systems biologist, I believe that `DataPackageR` does fill a useful niche between complete research compendia and storage of results in a database, which may require dedicated data modeling.

I have run the docker container example and built the MR1 data package in the vignette. The ability to version aggregated datasets and tie an analysis to a version is a very important one. As the other reviewer did suggest, putting large datasets into GitHub would cause the repo to become very large and I'm glad that the authors have tried to address this.

I see `DataPackageR` working well with both workflow tools (such as `drake`) but also potentially data integrity testing tools (such as `assertr`) to guarantee a common version of the data. I believe it serves a useful purpose within data analysis.

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Yes

Competing Interests: I utilize some of the data processing tools that have been developed in the Gottardo lab for analysis of flow cytometry data.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Referee Report 11 July 2018

doi:10.21956/gatesopenres.13917.r26556

Aaron T.L. Lun
Cancer Research UK Cambridge Institute, University of Cambridge, Cambridge, UK

The authors have addressed my concerns in their responses and revisions.

Competing Interests: No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard.

Version 1

Referee Report 25 June 2018

doi:10.21956/gatesopenres.13908.r26541

Aaron T.L. Lun
Cancer Research UK Cambridge Institute, University of Cambridge, Cambridge, UK

In this article, Finak et al. describe a new package for pre-processing and sharing of data within the R programming language. Their **DataPackageR** package provides a standardized framework for traceable pre-processing and version control of R data objects, facilitating reproducible research across a diverse team of collaborators. The manuscript is clear and concise, and the function and benefits of the software are clear. Nonetheless, I have some comments (listed below) that the authors might consider to improve both the software and the manuscript.

1. The processed data are stored as *RData* files in the *data/* directory and distributed as part of the constructed data package. These files can become somewhat large (> 100 MB), which in and of itself is not a problem; indeed, having to deal with large data files may not be avoidable in some contexts. However, the text on page 5 and Figure 1 suggest that the user should place the generated data files under version control via Git prior to distribution. With Git, every clone of the package will contain every version of all data files, inflating the size of the repository for download and on disk. This may become prohibitive for practical use (e.g., GitHub forbids uploads of files above a certain size), and feels unnecessary given that only one version of the data will be active at any one time. Perhaps the authors would consider supporting the versioning and acquisition of *RData* files from a separate location, mimicking the behaviour of (or directly using) Bioconductor's **ExperimentHub** where large data files are downloaded and cached locally as needed?

2. The YAML header seems to be limited to R scripts or *Rmd* files for data pre-processing. However, as the authors would appreciate, a lot of pre-processing is performed on the command line, e.g., with aligners and related software. It would be awkward (and involve extra work) to have to wrap these scripts in R/*Rmd* files in order for them to be executed by **DataPackageR**. To provide a concrete example: I would like **DataPackageR** to directly call my existing Bash scripts for

alignment of RNA-seq data, followed by execution of *Rmd* reports for assigning reads to genes to get a count table that is saved as an *RData* object.

3. Does the YAML header permit dependencies between scripts? Say I have a long pre-processing pipeline that is split across multiple scripts for convenience. Does the order of files in the header reflected in the order of execution? And are they executed in the same location, so that intermediate files produced by one script (that are not intended to be included in the final package) can be picked up as inputs to the following script?

4. The MX1 use case on pages 6-7 seems under-described - I don't get an intuition as to why **DataPackageR** was beneficial for this project. It would be demonstrative to have some specific examples of how the version control and change tracking were advantageous in a collaborative environment. For example, how many *DataVersion* bumps were performed throughout the course of the project, and why? Were there any issues arising from changes to the processed data, and how were these resolved?

Is the rationale for developing the new software tool clearly explained?
Yes

Is the description of the software tool technically sound?
Yes

Are sufficient details of the code, methods and analysis (if applicable) provided to allow replication of the software development and its use by others?
Yes

Is sufficient information provided to allow interpretation of the expected output datasets and any results generated using the tool?
Yes

Are the conclusions about the tool and its performance adequately supported by the findings presented in the article?
Partly

Competing Interests: No competing interests were disclosed.

I have read this submission. I believe that I have an appropriate level of expertise to confirm that it is of an acceptable scientific standard, however I have significant reservations, as outlined above.

Reader Comment 29 Jun 2018

Greg Finak,

Thank you for taking the time to review our work and providing feedback that will improve the manuscript and the software. You will find our responses the points you raised below. I look forward to continue the conversation. We will submit a revised version of the manuscript in due course.

1. In response to the first point, I agree that this can become a problem, particularly with large data sets. ExperimentHub is an interesting option to circumvent the file size limits of github (and indeed of data packages in general), although I feel that it is not a sufficiently general solution since ExperimentHub is “moderated” and requires a submission and approval process in order to have data appear there. One option to get around files bloat in local git repositories is simply to checkout the latest revision via `git clone –depth 1` . Git LFS could also be used to track large files, although this is a `pay to play` option. None of these options are particularly satisfactory for me.

A general solution may be support a remotely stored data types via some new class or object type that stores a pointer to the data and makes it the users’ responsibility to provide a method for pushing and pulling data from the remote resource. Such an object could point to an ExperimentHub resource or any other remote resource the user wishes to support, but I don’t believe it’s in the scope of DataPackageR to provide such support for specific resources. We will discuss this limitation in the revised manuscript and aim to provide support for remote data files in a future release, as it is going to require some more thought on our part. Again we thank the reviewer for this suggestion.

2. Regarding the second point, the infrastructure does not directly supporting running bash or other script via the YAML configuration at this time, but it is none the less possible to do so. Of course we are well aware of the need for support of other languages and at the moment this is handled through the multiple language support of rmarkdown and knitr. Code chunks in knitr can be used to process different languages, not just R, including bash, python, SQL, javascript, STAN, Rcpp, and javascript by replacing the language specifier in the code chunk header (e.g., the “```{r}” language specifier at the beginning of a code chunk) with another supported language. For the bash example specifically, a code chunk in an Rmd file would begin with “```{bash}”, and the code would be no different than a command-line invocation of a bash script.
We do not see this as particularly awkward or extra work, but if the reviewer has something else in mind, we are happy to follow up. We will highlight this manner of support for bash and other languages in the revised manuscript.

3. Regarding point 3, we feel this is an excellent suggestion. At the time of submission, dependencies between scripts were not supported, althought the order of execution is preserved and the scripts are all executed in the same location. We have made changes to the latest version of the software so that scripts run in a user-defined or temporary (default) directory, configurable in the YAML file as the “render_root” property and passed to *render*’s *knit_root_dir* argument. This will allow multi-script jobs to execute correctly, so the outputs of one script can be used as the inputs for another (without polluting the *data-raw* directory). Additionally, we have made the objects created by previous processing scripts in the pipeline accessible to subsequent processing files via the `datpackager_object_read()` API. In this way, if *file1.Rmd* creates table1, then file2.Rmd can access table1 using the API without polluting the environment where file2.Rmd is executed. This should make file-system artifacts as well as objects available to a multi-script pipeline. We will revise the manuscript to mention these features. We thank the reviewer for this suggestion.

4. Regarding point 4, we include more detail here, and will update the revised manuscript accordingly.
MX1 is just one study from over 33 prepared by VISC over the past 3 year using the *DataPackageR* infrastructure. Each study consists of three to seven data types, each with different QC and standardization criteria, each produced by different labs, all on a different time line. For each data type, VISC produces QC reports for the labs, statistical analysis plans for each data

type, statistical reports and analyses for each data type, and an integrated analysis for manuscripts and publications. The data generally arrive over a time period of several months, and reports are also generated over several months. Different individuals are responsible for the generation of different reports. Clearly, for a single project, perhaps this can be handled in an ad-hoc manner, but for VISC, with many studies on the go, it is critical that all reports are based off the same data set, and that those are updated appropriately when a data set is updated. For the MX1 study, the data package was updated 29 times between 2015 and 2017. The data set version was bumped from 0.1.0 to 0.2.21 over that time frame. Six data sets were added to the package over two years, 17 commits were associated with corrections or updates to data sets (e.g. additional or corrected annotations). Six follow up reports based on the data package were updated each time there was an addition, or correction to the data. Ensuring consistency between the different reports and the data would have been incredibly tedious without this framework (and has been in the past, prior to our adoption of this framework). We will update the revised manuscript with discussion of how DataPackageR has been beneficial for this study and in general.

Competing Interests: No competing interests were disclosed.