A Realistic Guide to Making Data Available Alongside Code to Improve Reproducibility

"Data! data! data!" he cried impatiently. "I can't make bricks without clay." - Sherlock Holmes (PBS Series, 2010-2017)

1 Introduction

Data are a fundamental currency upon which scientific discoveries are made. Without access to good data, it becomes extremely difficult, if not impossible, to advance science. Yet, a large majority of data on which published research papers are based rarely see the light of day and are only visible to the original authors (Rowhani-Farid and Barnett 2016; Stodden, Seiler, and Ma 2018). Sharing data sets upon publication of a research paper has benefits to individual researchers, often through increased visibility of the work (Popkin 2019; Kirk and Norton 2019). But there is also a component of benefit to the broader scientific community. This primarily comes in the form of potential for reuse in other contexts along use for training and teaching (McKiernan et al. (2016)). Assuming that the data have no privacy concerns (e.g., human subjects, locations of critically endangered species), or that the act of sharing doesn't put the authors at a competitive disadvantage (data can be embargoed for reasonable periods of time), sharing data will always have a net positive benefit. First and foremost, sharing data along with other artifacts can make it easier for a reader to independently reproduce the results, thereby increasing transparency and trust in the work. The act of easy data sharing can also improves model training, as many different models can be tried and tested on latest data sources, closing the loop on research and application of statistical techniques. Existing data sets can be combined or linked with new or existing data, fostering the development and synthesis of new ideas and research areas. The biggest of these benefits is the overall increase in reproducibility.

For nearly two decades, researchers who work in areas related to computational science have pushed for better standards to verify scientific claims, especially in areas where a full replication of the study would be prohibitively expensive. To meet these minimal standards, there must be easy access to the data, models, and code. Among the different areas with a computational bent, the bioinformatics community in particular has a strong culture around open source (Gentleman et al. 2004), and has made releasing code and associated software a recognized mainstream activity. Many journals in these fields have also pushed authors to submit code (model implementations) alongside their papers, with a few journals going as far as providing a "reproducibility review" (Peng 2011).

In this paper we focus on the practical side of sharing data for the purpose of reproducibility. Our goal is to describe various methods in which an analyst can share their data with minimal friction. We steer clear of idealistic ideas such as the FAIR data principles (Wilkinson et al. 2016) since they still do not help a researcher share their data. We also skip the discussion around citation and credit because data citations are still poorly tracked and there is no standardized metric or a h-index equivalent for data as of this writing.

For a piece of computational research to be minimally reproducible, it requires three distinct elements: 1) Code; 2) Computing environment, and 3) Data. The first two of these challenges have largely been solved (Poisot et al. 2019).

Although code sharing in science had a rocky start (Barnes 2010), more and more code writing by scientists is being shared, partly due to the rapid increase in training opportunities made available by organizations like The Carpentries, combined with the rapid adoption of Github by scientists (Ram 2013). The bigger driver for this may also be connected to the rise in popularity of data science as a discipline distinct from statistics (Donoho 2017). This rapid growth in data science has largely been fueled by easy access to open source software tools. Programming languages such as Python, R and Julia help scientists implement and share new methods to work with data. Each of these languages is supported by thriving communities of researchers and software developers who contribute many of the building blocks that make them popular. As of this writing, Python, Julia, and R have 167k packages ("Pypi," n.d.), ~ 14k packages ("Cran," n.d.)

and ~2k packages ("Julia-Pkgman," n.d.) respectively. These packages form the building blocks of a data scientists daily workflow. In a modern computational workflow it is trivial for a user to `pip install` a package in Python, or use `install.packages` in R to install all software dependencies for a project. By relying on the idea of having research compendia (Gentleman and Temple Lang 2007) or something as simple as a requirements file, one can easily install all the necessary scaffolding. Data on the other hard are rarely accessible that easily.

A typical data analysis loads a dozen or two of these open source libraries at the top of a notebook and then relies on existing routines to rapidly read, transform, visualize, and model data. Each package depends on a complex web of other packages, building upon existing work rather than re-implementing everything from scratch. Working from script and a list of such dependencies, a data analyst can easily install all the necessary tools in any local or remote environment and reproduce the computation. When new functionality is developed, it is packaged into a separate entity and added to a language's package manager.

The computing environment is also easily captured with modern tools such as Docker (Boettiger 2015; Jupyter et al. 2018). Modern tools such as Binder (citation) can parse Docker files and dependency trees to provide on demand, live notebooks in R and Python that a reader can immediately execute in the browser without dealing with the challenges of local installation. This makes it simple to load a specific environment to run any analysis. Code is handled by version control with tools like Git and GitHub ("Git," n.d.; "Github," n.d.), paired with archiving such as Zenodo provide access to code (particularly model implementations)(Zenodo 2016). All the necessary software is available from various package managers (and their numerous geographic mirrors and archives) making it easy to install any version of a software package. However, the biggest challenge, even today, remains easy and repeatable access to data in a data analysis.

Datasets are often far more diverse than code in terms of complexity, size, and formats. This makes them particularly challenging to standardize or easily "install" where the code is running. While there are numerous public and private data repositories, none of them function as package managers, which is what provides so much robustness to code sharing. As a result, data used in an analysis is often read from various locations (local, network, or cloud), various formats, varying levels of tidiness (Wickham 2014). There is also the

overhead associated with data publishing, the act of archiving data in a repository that also mints permanent identifiers, that are not required of all projects due to the effort and time involved. It is worth drawing the distinction between data sharing (making the data available with little effort) and data publishing (archiving the data in a long-term repository, with our without curated metadata).

What aspects of data make them particularly challenging to share from a reproducibility perspective? This is the question we tackle in this paper. While there are several papers that serve as best-practice guides for formatting data and getting them ready for sharing, the aims of this paper are a bit different. Our aim to address the issue of data in the context of reproducibility in data science workflows. In particular we discuss the various tradeoffs one has to consider when documenting and sharing data, when it is worth publishing and how this would change depending on the use case, perceived impact, and potential audience.

We discuss how to share and/or publish data and cover various tradeoffs when deciding how much to do. We skip detailed discussions of the minutiae of data preparation (TODO: We need to cite these articles), licenses (there are not many options when it comes to data), or citation (as of this writing, data citation is still in its infancy). We also analyze the state of data contained inside software packages, shared and made available as part of modern data analysis. How often are data shipped inside software packages? How does data format and size impact availability?

2 Challenges in documenting your dataset

There are many features to include with data alongside publications. However, not all of these are needed for every dataset. Working out which are needed is a challenge that is not discussed in the literature. This section discusses a framework to use decide how much they should document their data. To frame discussion around the challenges of data documenting, we can think of how an individual dataset falls on two features: "Effort to prepare", and "Ease of understanding" in Figure 1. The ideal space to be in the graph would be the top left hand corner. But what we notice is that taking more effort to prepare data means that the data is easier to understand.

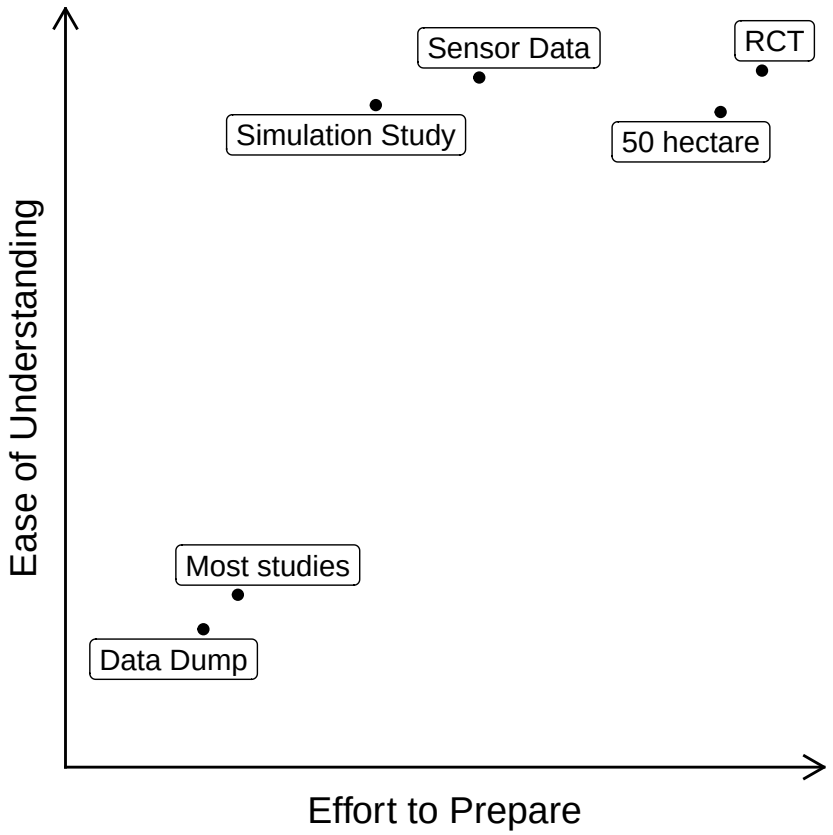


Figure 1: *There is a big difference in the effort to prepare data, and how easy it is to understand - look at the difference between most datasets, and something like a Randomized Control Trial (RCT).*

Data with higher potential impact and reuse should be made as easy to understand as possible; but it also requires more time and effort to prepare. Impact is hard to measure, and varies from field to field, but as an example, take some data from medical randomized control trials (RCTs) on cancer treatment. These can have high impact, so requires a lot of time and effort document. Comparatively, a small survey on a few demographics can have low impact. Placing these on the graph above, we see they might not have a worthwhile tradeoff for ease of understanding and ease of preparation. This means the effort put into preparing data documentation for a small survey should be kept relatively simple, not over complicated. Likewise, data that can be created via simulation from open source software could arguably not be shared since it can be generated from scratch with code; a reproducible process that requires computer time, not person time to create.

Deciding how much data documentation to include should be based on the data's impact. The more impactful the data, the more documentation features to include. Figure 2 shows the practical types of steps that can be taken against the effort required.

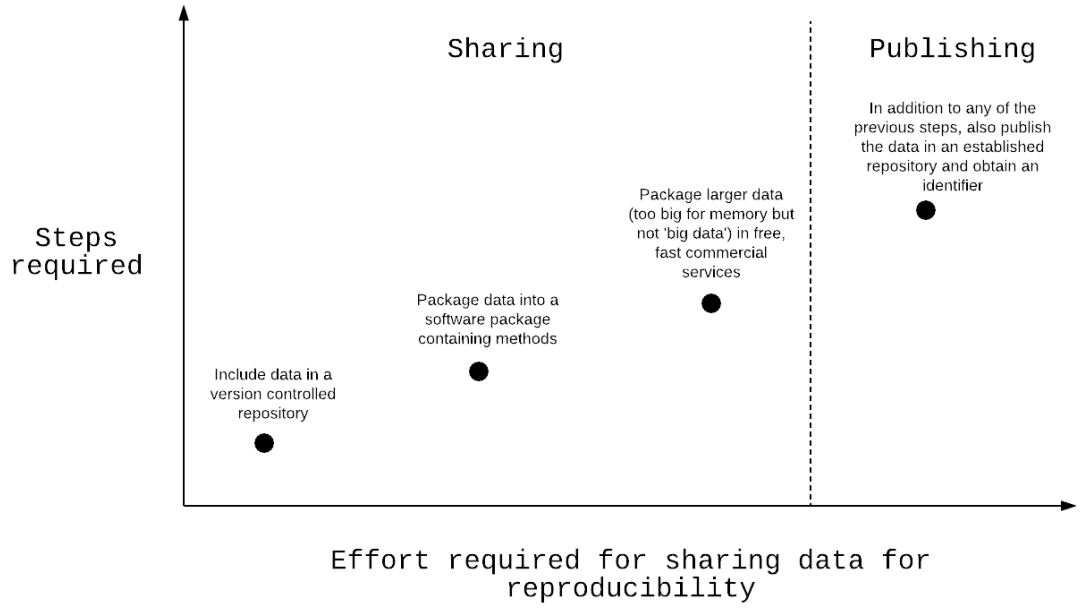


Figure 2: *The steps required compared to effort required for data sharing for reproducibility.*

Documentation challenges may evolve over time as the cost of making them easy to prepare and understand change. For example, new technologically may automate rigorous data documentation, so thorough documentation can take the same time as it takes to do poorly now. In this case, there would be no reason why more people cannot do this, even when the benefits are not immediately apparent. So if something might appear to have low impact, it should still be made easy to understand, if it doesn't take long to prepare that documentation.

Creating good documentation has similar challenges to good writing: it takes time, and it can be hard to know when you are done. Thinking about two features: 1) the impact of data, and 2) the current effort to do each step, provides guidelines for the researcher to decide how much documentation they should provide for their data.

3 Releasing data in R

One low cost and easy way to distribute data alongside compute is to package the datasets as a data only package or as part of something larger where the methods are being implemented. The R ecosystem has many noteworthy examples of data-only packages. One exemplar is the `nycflights13` package by Hadley Wickham (Wickham 2018), This package makes available airline data for all flights departing New York city in 2013 in a tidy format, with distinct tables for metadata on airlines, airports, weather, and planes. The package not only provides ready to use binary data but also ships raw data (in a folder called `data-raw`) along with scripts used to clean them. The package was originally created as a way to provide example data to teach tidy data principles and serves as a great model for how one can publish a data package in R.

A more common use case is to include data as part of a regular package where analytical methods are being implemented and shared. This serves the dual purpose of exposing methods and data together, making it extremely easy for a researcher to simply install and load the library at the top of their script. CRAN's distributed network (along with the global content distribution network maintained by RStudio) ensure that the package is quickly accessible to anyone in the R ecosystem. A second critical advantage in this approach is that one could also benefit from R's package documentation syntax to generate useful metadata for fast reference. This approach can also easily be adapted to other languages. Python for example, is far more flexible with respect to including arbitrary files as part of a package.

Other benefits of packaging data in R

1. Packaging data can be a very powerful pedagogical tool to help researchers understand how to transform data and prepare it for further analysis. To do so, one can package raw data alongside scripts. Long form documentation such as vignettes can provide further detailed discussion on the process. Users can also skip the raw data and scripts and proceed directly to the fast binary data, which can hold a lot of data when heavily compressed.

2. When the primary motivation for shipping data is to illustrate visualization techniques or to run examples, one can skip the raw data and processing scripts and only include the binary data. As long as the total package size doesn't exceed 5 megabytes, it would be acceptable as part of CRAN. For

cases when this size is hard to maintain, CRAN recommends data-only packages that will be rarely updated. For a detailed discussion on this issue and alternative approaches, see Brooke Anderson and Eddelbuettel (2017).

One major disadvantage of packaging data inside R is that it makes the data availability very language centric. Non R users are unlikely to download and export data out of a package. This is why we recommend, as a rule, that researchers also archive data in a long-term data repository. These include domain specific repositories (see Section 6) or more general purpose ones such as Zenodo or Figshare and include the persistent identifier in all locations where the data is referenced such as the manuscript, notebook and data package.

Of the 15539 packages on Central R Archive Network (CRAN), 6278 contain datasets either as binary data (5903 packages) or as external datasets (766). Binary files comprise a bulk of the data shipped in the `data` folder (68.06%) with other plain text formats such as txt, CSV, dat, `json` comprising less than one percent of data shipped in packages.

4 Data Sharing in Julia and Python

Sharing data in a programming language happens to some extent in other languages, such as Julia and Python. Julia was established in 2012, and is a high level language with great performance (Bezanson et al. 2012). Similar to R, Julia provides packages for database access, API access, and data packages ("Metadata for Registered Julia Packages," n.d.). This includes access to MySQL databases, NOAA, and actual datasets from Econometrics, faces, Market data, and even R datasets ("NOAA-Jl," n.d.; "Econ-Data-Jl," n.d.; "Face-Data," n.d.; "Market-Data-Jl," n.d.; "R-Data-Jl," n.d.).

Python is a popular general purpose language, used in engineering, statistics, and computer science. Including data in a python package is straightforward; a user can include data (or any other arbitrary files) inside a python package by declaring it in the manifest ("Include-Data-Py-Pkg," n.d.).

5 Dealing with medium to large data

Another common situation that researchers face is in dealing with data that fall somewhere between small and large. For example, small data could be tabular, as a CSV, in the order of bytes to megabytes to gigabytes that can be easily compressed, and large could be what falls under the umbrella of **big data**. The happy medium is generally data that are too big to fit on memory of most standard machines, but can successfully fit on disk (https://ropensci.github.io/arkdb/articles/articles/noapi.html). In this case, users who do not have the support to maintain resource intensive database servers can instead rely on light serverless databases such as MonetDB or SQLite. These databases provide disk based storage and using language agnostic interfaces, a analyst can easily query these data in manageable chunks that don't overrun memory limitations. Using software packages such as `arkdb` (Boettiger 2018) one could easily chunk large data from flat text files to these lite databases without running into memory limitations.

To make these files available alongside compute, one ingenious but short-term solution is to use the GitHub release feature to attach such large database dumps. GitHub releases are designed to serve software releases and provide the opportunity to attach binary files. This mechanism can also be used to attach arbitrary files such as large data files, binary data, and database dumps as long as each file does not exceed 2gb. The R package `piggyback` (`https://docs.ropensci.org/piggyback/`) allows for uploading and downloading such files to GitHub releases, making it easy for anyone to access data files wherever the script is being run. We emphasize again that this is a short-term solution that is dependent on GitHub maintaining this service.

6 Publishing and repositories

It is worth distinguishing between sharing data and publishing data. One can share data in numerous ways without going through the trouble of publishing it, which often requires metadata that a human must verify. Data can be shared in numerous ways, including placing it in a repository, packaging it with methods, or using various free tiers of commercial services. However, one must publish data when appropriate.

There are a three common options for publishing data in research:

1. **Least Moving Parts**
2. **Domain Specific Venue**
3. **Long Term Archive**

We discuss each of these options and provide recommendations on how to publish data in these areas.

In **Least Moving Parts**, the data might be published with an R package, or as part of a GitHub release using piggyback ("Piggyback," n.d.), or a in serverless databases. This approach **gets the data somewhere rather than nowhere**. Its minimal features means it is simple to maintain. A downsides is that it does not scale to larger data. Self hosting the data is an option, but we discourage this, as it may succumb to bit rot.

In **Domain Specific Venue**, data can be published in journal data papers, or venues specific to the type of data. For example, in astronomy the data is hosted at the SDSS (SDSS) or Genetic data can be hosted at GenBank (Benson et al. (2005)), and so on. Some other places to put data include the open science framework (Erin D. Foster 2017), Dataverse, ("Dataverse," n.d.), and Data World, ("Dataworld," n.d.). Other datasets might be produced

The purpose, use, and origin of the data is an important component to consider. Data for research has a different domain compared to data collected by governments, or by journalists. Many governments or civil organizations are now making their own data available through a government website interface. Media and journalism groups are also making their data available either through places like GitHub ("Pudding-Data," n.d.), or organizations such as Data World, or may self host their data, such as five thirty eight ("Data-538," n.d.).

This is a good option when the data is appropriate for the domain. This is often decided by community standards. We recommend adhering to community standards for a location to publish your data, as this will likely improve data reuse. The guidelines suggested in this paper for sharing the data should be included.

A **Long Term Archive** is the best option to share the data. Long term archives provide information such as DOI (Digital Object Identifier) that make it easier to cite. Data can be placed in a long term archive and a DOI can be minted. This DOI can then be used in other venues, such as domain specific or even self hosted, and will ensure that the projects refer back appropriately.

If the dataset you are shipping has a research application, the most relevant home for it would be Zenodo, a research data repository. Launched in 2013 in a joint collaboration between openAIRE and CERN, Zenodo provides a free, archival location for any researcher to depositor their datasets. The only limits to file sizes are 50gb for individual files, which is generous enough to accommodate a large number of use cases. Zenodo is able to accommodate larger file sizes upon request.

Principles for all three

The data should contain information on metadata, data dictionaries, the README, and data used in analysis (See Section 7). No matter where data is submitted, there should ideally be a canonical data repo in one long term archive that links to others with a DOI.

6.1 Publishing data in a data repository

6.2 Publishing data through a data journal

Data used in publications are often shared in the supplementary materials of articles, or served on repositories such as the Dryad Digital Repository ((“Dryad,” n.d.)). Dryad makes data from scientific publications discoverable, reusable, and citable. It is well funded through grants from the NSF, European Commission.

To provide better context around the data used in research and better expose data for reuse, journals are now adding “data papers”. These are specifically designed for publishing articles about the data, and sharing it. This benefits both researchers and readers. Researchers receive credit for data they have collected or produced. Readers get more context about the data.

Data papers are similar to research articles, they have titles, authors, affiliations, abstract, and references. They generally require an explanation of why the data is useful to others, a direct link to the data, description of the design, materials, and methods. Other information on the subject area, data type, format, and related articles are usually required.

Whilst useful, these requirements do not tell the author how to actually structure the data and folders. Instead providing ideas on what they should include. This is a useful step towards improving data reuse, but

it *lacks some minimal structure that allows a researcher to have a predictable way to access and interpret the data.*

Other journals operating in this space include journals like "data in brief", "Data", and "Nature Scientific Data". Guidelines for what is required in the content of the paper, and the required information along with the data (meta data, data format, etc.) vary by journal.

7 What belongs in the minimal structure for researchers

There are 8 pieces of content to consider for data sharing:

1. README: A Human readable description of the data
2. Codebook: Human readable dictionary of data contents
3. License: How to use and share the data
4. Citation: How you want your data to be cited
5. Machine readable meta data
6. Raw data: The original/first data provided
7. Scripts: To clean raw data ready for analysis
8. Analysis ready data: Final data used in analysis

One basic suggested directory layout is given below in Figure 3.

```
proj-name/
          |
          | - README.md
          | - Codebook.csv
          | - LICENSE.md
          | - metadata.json
          | - raw-data/
                      | - raw-data.csv
                      | - 01-read-tidy-raw-data.csv
          | - data/
                | - final-data.csv
```

Figure 3: *Example directory layout and structure for a data repository.*

Out of these sections, the minimal viable format are:

1. README
2. Codebook
3. License
4. Citation
5. Analysis ready data

What goes into each of these sections is now described.

7.1 README: A Human readable description of the data

The README is often the first place people will go to learn more about data. It is meant for someone to read and understand more about the data and contains the "who, what, when, where, why, & how":

- **Who** collected it
- **What** is the data
- **When** was it collected
- **Where** was it collected

- **Why** it was collected
- **How** is was collected

The README should be placed in the top level of the project. It should be brief, and provide links to the other aforementioned sections. Saving a README with the extension `.md` file gives the author the formatting benefits of `markdown`, making it simple to insert links, tables, and make lists. In systems like GitHub, a README file is detected and rendered in nice HTML on the repository by default.

7.2 Codebook: Human readable dictionary of data contents

Codebooks provide human readable description of the data, providing context on the nature and structure of the data. This helps someone not familiar with the data understand, and use the data. At a minimum they should contain:

- **variable names**
- **variable labels**
- **variable codes**, and
- special values for **missing data**.

Variable names are short, descriptive names with no spaces or special characters. For example, "job_position", "faculty_level", and "years_at_company". **Variable labels** are longer descriptions of variables. For example "University Job Position", "University Faculty Position Level", and "Number of Years Worked at University" ("Codebook Cookbook: A Guide to Writing a Good Codebook for Data Analysis Projects in Medicine" n.d.; Arslan 2019). **Variable codes** apply to categorical (factor) variables, and are the values for their contents. For example, `0 = no, 1 = yes` for one variable, and `0 = inside` and `1 = outside`. These should be consistent across similar variables to avoid problems where `0 = yes` for one variable, and `1 = yes` in another. Date variables should have consistent formatting. For example, all date information could be in format "YYYY-MM-DD", and this should not be mixed with "YYYY-DD-MM". **Missing data** are values that should have been observed, but were not. The code for missingness should be documented in the codebook, and should nominally be `NA`. If the reason for missingness is known it should be recorded. For

example, censored data, patient drop out, or measurement error can have different values, such as "unknown" or -99 (White et al. 2013; Broman and Woo 2017).

Table 1 shows an example data dictionary table taken from the Tidy Tuesday repository on incarceration trends ("Tidy-Tuesday-Incarcerate," n.d.). This includes information on the variable the class (type) of the variable, and a longer description of the variable.

Table 1: *The prisoner summary data dictionary, with columns on the variable, it's class, and a short description of the contents of the variable.*

Variable	Class	description
year	integer (date)	Year
urbanicity	character	County-type (urban, suburban, small/mid, rural)
pop_category	character	Category for population - either race, gender, or Total
rate_per_100000	double	Rate within a category for prison population per 100,000 people

Data dictionary tables should be placed in the README and presented as a table. Every data dictionary should also be provided in its raw form in the repository, for example, saved as a CSV.

7.3 License: How to use and share the data

Data with a license clearly establishes rules on how everyone can modify, use, and share data. Without a license, these rules are unclear, and can lead to problems with attribution and citation. It can be overwhelming to try and the right license for a use case. Two licenses that are well suited for data sharing are:

1. Creative Commons Attribution 4.0 International Public License (CC BY), and
2. Creative Commons CC0 1.0 Universal (CC0)

7.3.1 CC BY

The CC BY enforces attribution and due credit by default, but gives a lot of freedom for its use. Data can be shared and adapted, even for commercial use, with the following conditions:

- You must provide appropriate credit to the source. This means listing the names of the creators.
- Link back to the CC BY license, and

- Clearly show if changes were made.
- Data cannot be sub-licensed, that is - a change to the existing license
- There is also no warranty, so the person or people who obtained the data cannot be held liable.

The journal PLOS Comp Bio requires that data submitted cannot be more restrictive than CC BY ("PLOS Computational Biology," n.d.). For a brief overview of the CC BY, suitable to include in a README, see ("Ccby-Short," n.d.). For the full license, see ("Ccby-Long," n.d.).

7.3.2 CC0

The CC0 is a "public domain" license. Data with a CC0 license means the data owners waive all their rights to the work, and it now "owned" by the public. The data can be freely shared, this means it can be copied, modified, and distributed, even for commercial purposes *without asking permission.* When using data with CC0, it is good practice to cite the original paper, but it is not required. If you wish to use the CC0, see ("Choose-Cc0," n.d.). For a brief overview of the CC0, see ("Cc0-Short," n.d.), and for the full license, see ("Cc0-Long," n.d.).

Other licenses or notices to be aware of are **copyrighted data**, and **data embargoes**. If you are working with data **already copyrighted**, (for example under CC BY or CC0), you must give follow appropriate guidelines for giving credit. Data may also be under **embargo**. This means data cannot be shared more widely until a specific release time. If sharing data under an embargo, include detailed information on the embargo requirements in: the README, and in separate correspondence with those who receive the data.

7.4 Citation: How you want your data to be cited

A DOI is a prerequisite for citation. When citing data, it only makes sense to cite datasets that have been deposited into a DataCite compliant repo. If, for example, the data are deposited in dryad or Zenodo, the best practice would be to copy the citation created by these repositories. Under the hood, DataCite provides the DOI. If a DOI is unavailable, a citation will be meaningless, as it cannot be tracked by any means.

7.5 Machine readable metadata

The README, and data dictionary provides *human readable* information on the data. To ensure data types are preserved - dates are dates, names are characters, and so on - there needs to be some form of *machine readable* metadata. An excellent standard for metadata is Table Schema written by frictionless data ("Fritionless-Table-Schema," n.d.). For example, a dataset called "demographics" with three variables is shown in Table 2, which has the Java Script Object Notation (JSON) equivalent in Figure 4.

Table 2: *Example demographics table of age, height, and nationality.*

age	height	nationality
12	161.5	Australian
21	181.2	American
37	178.3	New Zealand

```
{
  "name": "demographics",
  # here we list the data files in this dataset
  "resources": [
    {
      "path": "demographics.csv",
      "schema": {
        "fields": [
          {
            "name": "age",
            "type": "integer"
          },
          {
            "name": "height",
            "type": "number"
          },
          {
            "name": "nationality",
            "type": "string"
          }
        ]
      }
    }
  ]
}
```

Figure 4: *Example snippet of some Table Schema data for a dataset with three variables. This provides a description of each field, and the type of field, and it's description.*

This contains fields such as path, and a schema with subfields name and type, for each variable. It also provides information for licensing and features such as line breaks, and delimiters. It is built on JSON (JavaScript Object Notation), a lightweight, human-friendly, machine readable data-interchange format. Table schema is baked into formats such as csvy ("Csvy," n.d.), an extended `csv` format, which has additional front matter in a YAML format using Table Schema.

7.6 Raw data: The original or first data provided

Raw data is usually the first format of the data provided before any tidying or cleaning of the data. If the raw data is a practical size to share, it should be shared in a folder called `raw-data`. The raw data should

be in the form that was first received, even if it is in binary or some proprietary format. If possible, data dictionaries of the raw data should be provided in this folder as well.

7.7 Scripts: To clean raw data ready for analysis

Any code used to clean and tidy the raw data should be provided in the `raw-data` directory. Ideally this would involve only scripted languages, but if other practical steps were taken to clean up the data, these should be recorded in a plain text or markdown file.

7.8 Analysis ready data: Final data used in analysis

The data used in the data analysis should be provided in a folder called `data`. Ideally, the data should be in "Tidy Data" format (Wickham 2014), where tidy data contains variables in columns, and observations in rows. Contrasting "raw data", "analysis ready data" should be in an easily readable plain-text format, such as CSV, tab separated, or semicolon separated. Binary or proprietary formats are discouraged in favor of interoperability, as it requires special software to read, even if it is sometimes slower to read in and out, and harder to share due to size.

8 Tooling for packaging data

Tooling for producing data documentation information speeds up the process of sharing data. Machine-readable metadata that can be indexed by google is created using the `dataspice` package (NotComplete 2018). To help create codebooks, we recommend the `codebook` package in R (Arslan 2019), which also generates machine readable metadata. Codebooks are implemented in other software such as STATA, which provides a "codebook" command. Data can be packaged up in a "data package" with DataPackageR (Greg Finak 2019), which provides tools to wrap up data into an R package, while providing helpers with MD5sum checks that track versioning. Note that is different to Frictionlessdata's tabular data package spec ("Frictionlessdata-Data-Pkg," n.d.).

9 Example datasets

We now explore the variety of documentation practices of a few selected datasets.

9.1 Dataset 1: Forest Census Plot on Barro Colorado Island

Long-term field surveys, where the same type of data is repeatedly measured and collected over a long period of time, are an example of data collection where the time and financial investment would necessitate meticulously curated and easily accessible metadata. Both from the fact that the same protocol is being followed year after year, and that field data collection efforts are quite expensive, these data need to be well documented, with documentation available alongside the data.

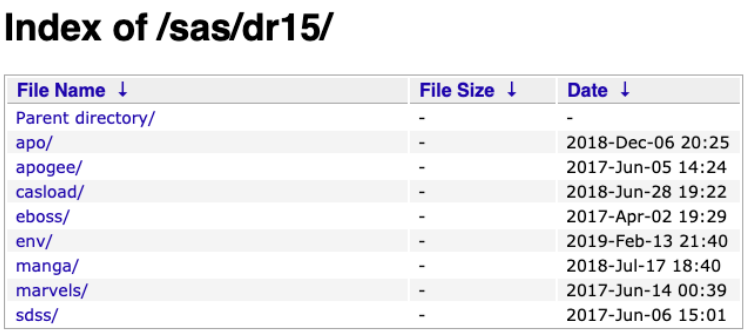
One example of a very laborious, long-term study is the 50-hectare plot on Barro Colorado Island in Panama. Researchers at the field station have censused every single tree in a pre-defined plot 7 times over the past 30 years. More than a million trees have been counted. The data and metadata however are hard to reach. To obtain the data, one must fill out a form and agree to terms and conditions which require approval and authorship prior to publication. The biggest challenge with this study is that the data are stored on a personal FTP server of one of the authors. While the data are available in CSV and binary Rdata formats, the data storage servers do not have any metadata, README files or codebooks. A separate, public archive hosts the metadata (https://repository.si.edu/handle/10088/20925) in a PDF file (https://repository.si.edu/bitstream/handle/10088/20925/RoutputFull.pdf?sequence=1&isAllowed=y) that describe the fields.

9.2 Dataset 2: Sensor data

Datasets obtained from sensors such as meteorological data are typically easy to prepare and understand. This is because sensors measure specific features, so the description of data type happens upstream at the instrument-design level, and flows down to data collection. The telescope data from the Sloan Digital Sky Survey (SDSS) is a great example of sensor data.

The SDSS data includes photometric and spectroscopic information obtained from a 2.5m telescope at Apache Point Observatory in New Mexico, operating since 2000, producing over 200Gb of data every day (York 2000; Blanton et al. 2017; "Sdss-Website," n.d.). This data has been used to produce the most detailed, three dimensional maps of the universe ever made.

The data are are free to use and publicly accessible, with the interface to the data being designed to cover a wide range of skills. For example, the marvin service to streamline access and visualisation of MaNGA data (Cherinka et al. 2018), through to raw data access (see Figure 5).



Index of /sas/dr15/

File Name ↓	File Size ↓	Date ↓
Parent directory/	-	-
apo/	-	2018-Dec-06 20:25
apogee/	-	2017-Jun-05 14:24
casload/	-	2018-Jun-28 19:22
eboss/	-	2017-Apr-02 19:29
env/	-	2019-Feb-13 21:40
manga/	-	2018-Jul-17 18:40
marvels/	-	2017-Jun-14 00:39
sdss/	-	2017-Jun-06 15:01

Figure 5: *Screenshot of Raw data avaialbe through DR15 FITS.*

The telescope data is very high quality, with the following features:

- Metadata in the machine readable standard, SCHEMA
- Data in multiple formats (e.g., .csv and .fits, database)
- Previously released data available in archives
- Entire datasets available for download
- Raw data before processed into database is also available

For example, the optical and spectral data can be accessed in FITS data format, or even as a plain CSV, the first few rows shown in Table 3. Wavelength plotted against Flux is shown in Figure 6, with 6A showing the output from the SDSS website, and Figure 6B showing the output from the CSV. The fact that figure Figure 6A is virtually replicated in Figure 6B demonstrates great reproducibility.

Table 3: *Example spectra data from SDSS, showing values for Wavelength, Flux, BestFit, and SkyFlux.*

Wavelength	Flux	BestFit	SkyFlux
3801.893	48.513	44.655	14.002
3802.770	54.516	42.890	14.113
3803.645	52.393	47.813	14.142
3804.522	45.273	42.612	14.186
3805.397	51.529	39.659	14.221
3806.273	44.530	44.183	14.349

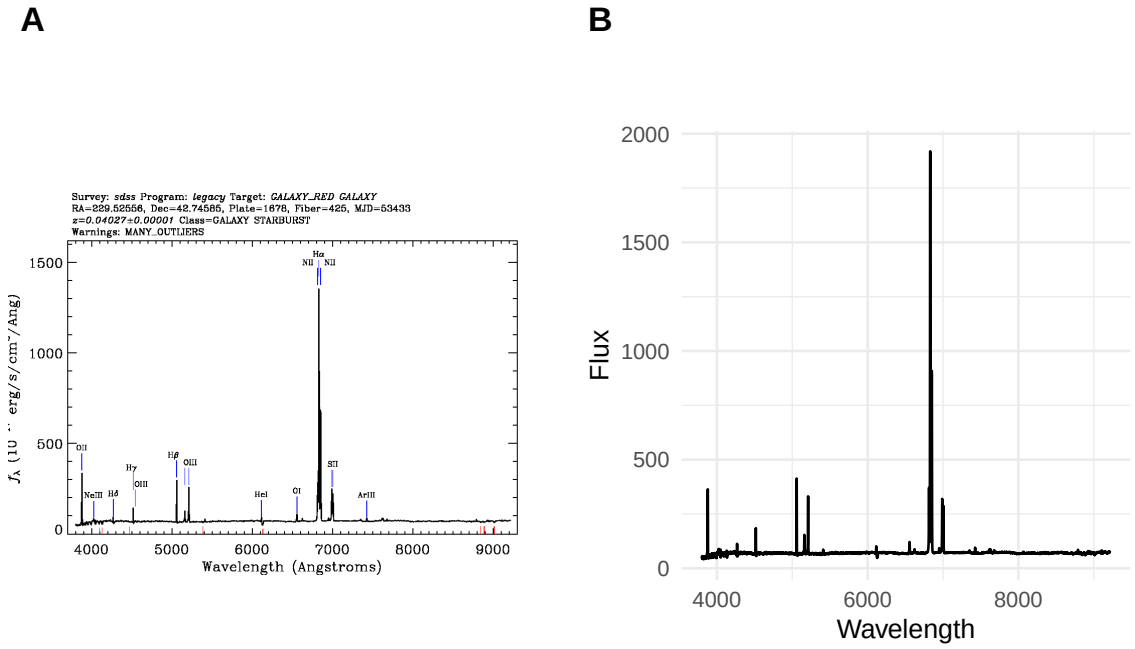


Figure 6: *Spectra image of wavelength against flux from SDSS from two sources. Part A shows the example image taken from SDSS website. Part B shows the replicated figure using data provided as CSV, demonstrating great reproducibility.*

Metadata from the data cannot be contained in the CSV file, but is available in the URL the csv was downloaded from: `http://dr15.sdss.org/optical/spectrum/view/data/format=csv?plateid=1678&mjd=53433&fiberid=425&reduction2d=v5_7_0`

Which contains details of the data, specifically:

- plateid = 1678
- mjd = 53433
- fiberid = 425
- reduction2d = v5.7.0

These are unique identifying features of the data, which tell us:

- **plateid:** the unique plate used to explore a given section of night sky.
- **mjd:** an integer that marks the "modified julian date" of the night of observation.
- **fiberid:** an integer of the fiber optic cable used to transmit light to the sensor.
- The **version** marked as `v5_7_0` is the pipeline version used for BOSS in DR12.

As a researcher who does not work in astrophysics, this information on the data was quite easy to retrieve from the SDSS website section called "Understanding SDSS spectroscopic data": https://www.sdss.org/dr12/spectro/spectro_basics/.

The SDSS provides data at different levels of complexity for different degrees of understanding. It is a huge project involving many people, and a staggering effort to prepare. Despite this, it is still very easy to understand. This is a further reflection on the idea that high effort can create highly understandable data, (see Figure 1). The impact of this data is high, having changed the way we understand the past, present and future of the universe itself. This impact is surely due to the care and thought put into making the data accessible and public. It is a statement of what is possible with good data sharing practices.

9.3 Dataset 3: Most other datasets

Many datasets are "data dumped" into repositories with a paper. Phrases such as the following might be familiar:

> Researcher 1: "What do these columns mean?" Researcher 2: "Sorry, I created the data 14 years ago and I don't remember".

** Need to get an example of this.

9.4 Summary

TODO: Needs a succinct summary here

10 Ten simple rules for publishing data

1. **Decide whether publishing is appropriate for your data** It is critical to publish your data if they are the basis for a peer-reviewed research publication. To be broadly useful, your data must be deposited in a permanent archive, to ensure that it does not disappear from a ephemeral location such as your university website. It shoudl also contain useful metadata so that someone outside the project team can understand what the data mean. However, this level of effort is not always critical for all data science efforts. For small-scale projects such as ones where one might generate simulated datasets, this level of curation is highly unnecessary. Here making data available in a transient location like GitHub or a website is sufficient.

2. **Include a README** file with your data archive. README files have a long history in software (https://medium.com/@NSomar/readme-md-history-and-components-a365aff07f10) and are named so that the ASCII systems capital letters filename would show this file first, making it the obvious place to put relevant information. In the context of datasets, READMEs are particularly useful when there are no reliable standards. The best practice here is to have one README per dataset, with names that easily associate with the corresponding data files. In addition to metadata about the data such as variable names, units of measurement and code books, the README should also contain information on licenses, authors/copyright, a title, and dates and locations where the data were collected. Additionally keywords, author identifiers such as ORCID, and funder information would be useful.

3. **Provide a codebook** or glossary for each variable. Codebooks provide a free-form way to document the data, their structure and underlying details of how they were collected (https://www.emgo.nl/kc/codebook/). More importantly they provide a way for future consumers of the data to understand how the variables were named, how missing data were documented, along with any additional information that might impact how the data are interpreted.

4. **Provide a machine readable format** for your data. When possible, provide machine readable metadata that map on to the open standards of schema.org and JSON-LD. These metadata provide all of the information described in the README best practices (rule 2), but in a machine readable

way and include much of the same information such as name, description, spatial/temporal coverage etc. One way to test if your metadata are machine readable is to use Google's structured testing data (https://search.google.com/structured-data/testing-tool/u/0/) to verify the integrity of the metadata.

5. **Provide the data in its rawest form** in a folder called "data-raw". Keeping your data raw (also sometimes referred to as the sushi principle) is the safest way to ensure that your analysis can be reproduced from scratch. This approach not only lets you trace the provenance of any analysis but it also ensures further use of the data that a derived dataset may prevent.

6. **Provide [open source?] scripts** used to clean data from rawest form into tidy/analysis ready format. Raw data is often unusable without further processing (data munging or data cleaning), which are the steps necessary to detect and clean inaccurate, incomplete records, and missing values from a dataset. While datasets can be cleaned interactively, this approach is often very difficult to reproduce. It is a better practice to use scripts to batch process data cleaning. Scripts can be verified, version controlled and rerun without much overhead when mistakes are uncovered. These scripts describe the steps taken to prepare the data, which helps explain and document any decisions taken during the cleaning phase. These should ideally operate on some raw data stored in the "data-raw" folder (rule 5).

7. **Keep additional copies in more accessible locations**: Even if you archive the data somewhere long-term, keep a second (or additional copies) in locations that are more readily and easily accessible for data analysis workflows. These include services like GitHub, GitHub LFS and others where fast content delivery networks (CDNs) make access and reuse practical. In the event that these services shut down or become unavailable, a long-term archival copy can still be accessed from a permanent repository such as Zenodo or Dryad and populated into a new data hosting service or technology.

8. **Use a hash function like MD5 checksum** to ensure that the data you deposited are the same that are being made available to others. Hash values such as MD5 are short numeric values that can serve as digital signatures for large amounts of data. By sharing the hash in a README, authors can ensure that the data, particularly the version of data being used by the reader is the same.

9. **Only add a citation if your data has a DOI**. A citation only makes sense when your data has a DataCite compliant DOI, which is automatically provided when data is published in repositories like Zenodo and Dryad. Although a citation may not accrue references, without a DOI it is guaranteed not to.

10. **Stick with simple data formats** to ensure long-term usefulness of datasets in a way that is not tied to transient technologies (such as ever changing spreadsheet standards), store the data as plain text (e.g., CSV) where possible. This can take the form of comma/tab separated files in most cases. This will ensure transparency and future proof your data.

11 Conclusions

The open science literature has been advocating the importance of data sharing for a long time. Many of these articles appeal to the broader benefits of data sharing, but rarely talk about the practical considerations of making data available alongside code. Trivial reproducibility has always relied upon the idea that as long as code exists on a platform such as GitHub, and clearly lists open source dependencies, one should be able to run the code at a future time. Containerization efforts such as Docker have made it even easier to capture system dependencies, and rapidly launch containers. But true reproducibility requires not just code, but also data.

Over the years researchers have made data available in a multitude of unreliable ways, including offering downloads from university websites, FTP repositories, and password protected servers, all of which are prone to bit rot. In rare cases where the data are deposited in a permanent archive, insufficient metadata and missing data processing scripts have made it harder and time intensive to map these to code. Our goal here is to describe a variety of ways in which a researcher can ship data, analogous to how easy it has become to push code to services like GitHub. The choice of technology and method all depends on the nature of the data, and its size and complexity. Once the issue of access is solved, we also discuss how much metadata to include in order to make the data useful. Ultimately there is a tradeoff to consider on a case by case basis.

Long-term studies with complex protocols require structured metadata, while more standardized datasets such as those coming from sensors require far less effort.

The key message for the reader is that accessing data for reproducibility should come with minimal barriers. Having users jump through data usage forms or other barriers is enough of a roadblock to dissuade users, which over time will make it hard to track the provenance of critical studies. For data with broad impact, one should further invest effort in documenting the metadata (either unstructured or structured depending on the complexity) and also focus on ensuring that at least one archival copy exists in a permanent archive. Following this continuum can ensure that more and more computational work becomes readily reproducible without unreasonable effort.

Literature Cited

Arslan, Ruben C. 2019. "How to Automatically Document Data with the Codebook Package to Facilitate Data Reuse." *Advances in Methods and Practices in Psychological Science* 2 (2): 169–87.

Barnes, Nick. 2010. "Publish Your Computer Code: It Is Good Enough." *Nature* 467 (7317): 753–53. https://doi.org/10.1038/467753a.

Benson, Dennis A, Ilene Karsch-Mizrachi, David J Lipman, James Ostell, and David L Wheeler. 2005. "GenBank." *Nucleic Acids Research* 33 (Database issue): D34–8.

Bezanson, Jeff, Stefan Karpinski, Viral B Shah, and Alan Edelman. 2012. "Julia: A Fast Dynamic Language for Technical Computing." *arXiv Preprint arXiv:1209.5145*.

Blanton, Michael R, Matthew A Bershady, Bela Abolfathi, Franco D Albareti, Carlos Allende Prieto, Andres Almeida, Javier Alonso-García, et al. 2017. "Sloan Digital Sky Survey IV: Mapping the Milky Way, Nearby Galaxies, and the Distant Universe," February. http://arxiv.org/abs/1703.00052.

Boettiger, Carl. 2015. "An Introduction to Docker for Reproducible Research." *ACM SIGOPS Operating Systems Review* 49 (1): 71–79. https://doi.org/10.1145/2723872.2723882.

———. 2018. *Arkdb: Archive and Unarchive Databases Using Flat Files.* https://CRAN.R-project.org/package=arkdb.

Broman, Karl W, and Kara H Woo. 2017. "Data Organization in Spreadsheets." e3183v1. PeerJ Preprints; PeerJ Inc.

Brooke Anderson, G, and Dirk Eddelbuettel. 2017. "Hosting Data Packages via Drat: A Case Study with Hurricane Exposure Data." *The R Journal* 9 (1): 486–97.

"Cc0-Long." n.d. https://creativecommons.org/publicdomain/zero/1.0/legalcode.

"Cc0-Short." n.d. https://creativecommons.org/publicdomain/zero/1.0/.

"Ccby-Long." n.d. https://creativecommons.org/licenses/by/4.0/legalcode.

"Ccby-Short." n.d. https://creativecommons.org/licenses/by/4.0/.

Cherinka, Brian, Brett H Andrews, José Sánchez-Gallego, Joel Brownstein, María Argudo-Fernández, Michael Blanton, Kevin Bundy, et al. 2018. "Marvin: A Toolkit for Streamlined Access and Visualization of the SDSS-IV MaNGA Data Set," December. http://arxiv.org/abs/1812.03833.

"Choose-Cc0." n.d. https://creativecommons.org/choose/zero/.

"Codebook Cookbook: A Guide to Writing a Good Codebook for Data Analysis Projects in Medicine." n.d. Accessed December 18, 2018. http://www.medicine.mcgill.ca/epidemiology/joseph/pbelisle/CodebookCookbook/CodebookCookbook.pdf.

"Cran." n.d. https://cran.r-project.org/.

"Csvy." n.d. http://csvy.org/.

"Data-538." n.d. https://data.fivethirtyeight.com/.

"Dataverse." n.d. https://dataverse.org/.

"Dataworld." n.d. https://data.world/.

Donoho, David. 2017. "50 Years of Data Science." *Journal of Computational and Graphical Statistics: A Joint Publication of American Statistical Association, Institute of Mathematical Statistics, Interface Foundation of North America* 26 (4): 745–66.

"Dryad." n.d. https://datadryad.org/.

"Econ-Data-Jl." n.d. https://juliafinmetrix.github.io/EconDatasets.jl/#sec-1.

Erin D. Foster, Ariel Deardorff. 2017. "Open Science Framework (OSF)." *Journal of the Medical Library Association: JMLA* 105 (2): 203.

"Face-Data." n.d. https://github.com/dfdx/FaceDatasets.jl.

"Frictionlessdata-Data-Pkg." n.d. https://frictionlessdata.io/docs/tabular-data-package.

"Fritionless-Table-Schema." n.d. https://frictionlessdata.io/specs/table-schema/.

Gentleman, Robert C, Vincent J Carey, Douglas M Bates, Ben Bolstad, Marcel Dettling, Sandrine Dudoit, Byron Ellis, et al. 2004. *Genome Biology* 5 (10): R80. https://doi.org/10.1186/gb-2004-5-10-r80.

Gentleman, Robert, and Duncan Temple Lang. 2007. "Statistical Analyses and Reproducible Research." *Journal of Computational and Graphical Statistics* 16 (1): 1–23.

"Git." n.d. https://git-scm.com/about.

"Github." n.d. https://github.com/.

Greg Finak. 2019. *DataPackageR: Construct Reproducible Analytic Data Sets as R Packages.* https://doi.org/10.5281/zenodo.2620378.

"Include-Data-Py-Pkg." n.d. https://python-packaging.readthedocs.io/en/latest/non-code-files.html.

"Julia-Pkgman." n.d. https://pkg.julialang.org/.

Jupyter, Project, Matthias Bussonnier, Jessica Forde, Jeremy Freeman, Brian Granger, Tim Head, Chris Holdgraf, et al. 2018. "Binder 2.0 - Reproducible, Interactive, Sharable Environments for Science at Scale." In *Proceedings of the 17th Python in Science Conference.* SciPy. https://doi.org/10.25080/majora-4af1f417-011.

Kirk, Rebecca, and Larry Norton. 2019. "Supporting Data Sharing." *NPJ Breast Cancer* 5 (February): 8.

"Market-Data-Jl." n.d. https://marketdata.readthedocs.io/en/latest/.

McKiernan, Erin C, Philip E Bourne, C Titus Brown, Stuart Buck, Amye Kenall, Jennifer Lin, Damon McDougall, et al. 2016. "How Open Science Helps Researchers Succeed." *eLife* 5 (July). https://doi.org/10.7554/elife.16800.

"Metadata for Registered Julia Packages." n.d. https://github.com/JuliaLang/METADATA.jl.

"NOAA-Jl." n.d. https://github.com/pazzo83/NOAAData.jl.

NotComplete, Author. 2018. *Dataspice: Create Lightweight Schema.org Descriptions of Dataset.* https://github.com/ropenscilabs/dataspice.

Peng, R. D. 2011. "Reproducible Research in Computational Science." *Science* 334 (6060): 1226–7. https://doi.org/10.1126/science.1213847.

"Piggyback." n.d. https://github.com/ropensci/piggyback.

"PLOS Computational Biology." n.d. *PLOS ONE*. Public Library of Science. https://journals.plos.org/ploscompbiol/s/data-availability.

Poisot, Timothée, Anne Bruneau, Andrew Gonzalez, Dominique Gravel, and Pedro Peres-Neto. 2019. "Ecological Data Should Not Be so Hard to Find and Reuse." *Trends in Ecology & Evolution* 34 (6): 494–96. https://doi.org/10.1016/j.tree.2019.04.005.

Popkin, Gabriel. 2019. "Data Sharing and How It Can Benefit Your Scientific Career." *Nature* 569 (7756): 445–47.

"Pudding-Data." n.d. https://github.com/the-pudding/data.

"Pypi." n.d. https://pypi.org/.

Ram, Karthik. 2013. "Git Can Facilitate Greater Reproducibility and Increased Transparency in Science." *Source Code for Biology and Medicine* 8 (1). https://doi.org/10.1186/1751-0473-8-7.

"R-Data-Jl." n.d. https://github.com/johnmyleswhite/RDatasets.jl.

Rowhani-Farid, Anisa, and Adrian G Barnett. 2016. "Has Open Data Arrived at the British Medical Journal (BMJ)? An Observational Study." *BMJ Open* 6 (10): e011784.

"Sdss-Website." n.d. https://www.sdss.org/.

Stodden, Victoria, Jennifer Seiler, and Zhaokun Ma. 2018. "An Empirical Analysis of Journal Policy Effectiveness for Computational Reproducibility." *Proceedings of the National Academy of Sciences of the United States of America* 115 (11): 2584–9.

"Tidy-Tuesday-Incarcerate." n.d. https://github.com/rfordatascience/tidytuesday/tree/master/data/2019/2019-01-22.

White, Ethan P, Elita Baldridge, Zachary T Brym, Kenneth J Locey, Daniel J McGlinn, and Sarah R Supp. 2013. "Nine Simple Ways to Make It Easier to (Re)use Your Data." *Ideas in Ecology and Evolution* 6 (2).

Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software* 59 (10). https://doi.org/10.18637/jss.v059.i10.

———. 2018. *Nycflights13: Flights That Departed Nyc in 2013.* https://CRAN.R-project.org/package=nycflights13.

Wilkinson, Mark D., Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, et al. 2016. "The FAIR Guiding Principles for Scientific Data Management and Stewardship." *Scientific Data* 3 (March): 160018. https://doi.org/10.1038/sdata.2016.18.

York, D G. 2000. "The Sloan Digital Sky Survey: Technical Summary," June. http://arxiv.org/abs/astro-ph/0006396.

Zenodo. 2016. "Zenodo."