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)

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 (citation). Sharing datasets upon publication of a research paper has benefits to individual researchers, often through increased visibility of the work. 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 (@McKiernan2016). 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, 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 datasets 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 [@Gentleman2004], 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" [@Peng2011].

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 theoretical ideas such as the FAIR data principles [@Wilkinson2016] 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

3. Data

The first two of these challenges have largely been solved.

Although code sharing in science had a rocky start [@Barnes2010], 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 [@Ram2013]. The bigger driver for this may also be connected to the rise in popularity of data science as a discipline distinct from statistics [@Donoho2017]. 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 (https://pypi.org/), ~ 14k packages (https://cran.r-project.org/) and ~2k packages (https://pkg.julialang.org/) respectively. These packages form the building blocks of a data scientists daily workflow.

A typical data scientist loads a dozen or two of these open source libraries at the top of a notebook and then rely on existing routines to rapidly read, transform, visualize, and model data. Each of these packages

individually depend on a complex web of other packages, building upon existing work rather than reimplementing 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 [@Boettiger2015; @Jupyter2018]. 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; @github], paired with archiving such as Zenodo provide access to code (particularly model implementations)[@zenodo]. 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.

Although there are numerous public and private data repositories, none of them function as package managers. Datasets are also far more diverse in complexity, size, and formats, making them particularly challenging to standardize or easily "install" where the code is running. As a result, data used in an analysis is often read from various locations (local, network, or cloud), various formats, varying levels of tidiness [@Wickham2014]. 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 (covered elsewhere in various 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?

Challenges in documenting your dataset

Providing a well documented dataset alongside research requires meeting several challenges. This includes understanding all the pieces and features of documentation such as metadata, licenses, and raw data; where to literally share the data; how to ensure long term provenance of the data.

One challenge is that better documentation requires more time, and that higher impact data needs better documentation.

Sharing data with colleagues: binary data is quick to share, but not as good as something like a simpler format like CSV (ask Laura DeCicco re this)

Providing the right amount of documentation for your data is challenging. Too much is overwhelming; too little and you don't know anything. To illustrate this, imagine two datasets. The first is contained in a folder with dozens of data files and several papers on the topic. The second contains only a single csv file. The first is too much; understanding it requires a high time cost. The second is too little; we don't know anything about what the data is or why it was collected. The challenge is to provide the right level of detail, somewhere between these two extremes.

It takes more effort to prepare data to make it easy to understand and share. The challenge here is understanding how much time to spend to make it easier to understand before the gains in understanding are outweighed by the time taken to improve them.

There is also a challenge in understanding the impact of data. Data with high potential for impact should be made easier to understand and share. This can be seen in the research created from the 50 hectare plot.

There are many features to consider when documenting data, such as a README file with human-readable descriptions of data and variables, data licenses, whether to include raw data, data cleaning scripts, and machine readable metadata. Together these all help make the data easier to understand. These features are not clearly described in the scientific literature in the context of including them alongside research.

To frame discussion around the challenges of data documenting, we can think of two features: "Effort to prepare", and "Ease of understanding". We can plot these on two axes and place datasets along each to indicate the ease of understanding and preparation (see Figure @ref(fig:fig-effort-underrstanding)).

Challenges

Challenges: higher potential impact means more time

To improve data impact it needs more time to prepare so it is easier to understand. Accordingly, data with higher potential for impact and reuse should be made as easy to understand as possible. Therefore data with high potential impact requires more time and effort to prepare. For example, data from medical randomized control trials (RCTs) on cancer treatment can have high impact, so requires a lot of time and effort document. Comparatively, a small survey on simple demographics can have low impact. So it might not have a worthwhile tradeoff for ease of understanding and ease of preparation. This means effort put into preparing data documentation for a small survey to be made easy to use should be kept relatively simple and not overcomplicated.

Tradeoffs may change over time as the cost of making them easy to prepare and understand change. If it becomes technologically cheaper and easier to document data more rigorously in the same time as it takes to do poorly now, then there is 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.

Data that is easy to use should avoid being in a proprietary format that requires special software to read. For example, a CSV should be preferred over proprietary data formats, such as SPSS or SAS custom data types.

Understanding tradeoffs

To help explain these descriptive qualities of the data sharing, we discuss three datasets and their relationship of **Effort to prepare**, and **Ease of understanding** in section 5.

One of the easiest solutions to making data available alongside an analysis is to include it in the same version controlled repository as the analysis scripts. This works well with the data file sizes are in the order of bytes to megabytes. When a reader clones the repository, the scripts and notebooks will be able to read all data files from relative paths with no further modifications to the code itself. Assuming the data would be useful beyond the context of the analysis, it would be beneficial to also archive it in a long-term repository.

Releasing data in R

One low cost and easy way to distribute data alongside compute would be 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. One exemplar is the `nycflights13` package by Hadley Wickham (https://cran.r-project.org/web/packages/nycflights13/index.html). 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. Advanced users can simply 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_Anderson2017]

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) 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.

Data Sharing in Julia and Python

Sharing data in a programming language happens to some extent in other languages. We discuss how Julia and Python share and distribute data in the following section.

Julia

Julia was established in 2012, and is a high level language with great performance. Searching METADATA.jl [@METADATA-jl] for packages mentioning data revealed three main categories of data packages in Julia:

1. Database access [@data-access] for services like MySQL, Hive, and ODBC, similar to R's DBI organization [@R-DBI]

2. Accessing data online via APIs, such as accessing the weather source NOAA [@NOAA-jl]

3. Data is provided directly. For example, accessing Econometric datasets [@econ-data-jl], face datasets [@face-data-jl], Market data [@market-data-jl], and even R datasets [@r-data-jl]. It is telling that the datasets provided in R are useful, as there is even a Julia package, RDatasets that provides over 700 datasets from R packages. There are also packages for Accessing datasets used in vega [@vega-data-jl], for accessing common machine learning datasets [@ml-data-jl]. There are also packages for generating fake data [@fake-data-jl], for which there are at least two R packages that do this, charlatan [@charlatan-data-r], and [@wakefield-data-r].

Python

Python is a popular general purpose language, used in engineering, statistics, and computer science. One language agnostic solution is to use the Open Knowledge Foundations Frictionless data spec. This is implemented in both R and Python.

Including data in a python package: https://python-packaging.readthedocs.io/en/latest/non-code-files.html A user can typically include data (or any other arbitrary files) inside a python package by declaring it in the manifest.

Dealing with medium to large data

Another common situation that researchers face is in dealing with data that fall somewhere between small (tabular data such as csvs that are in the order of bytes to megabytes to gigabytes that can be easily compressed) to large (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 (https://ropensci.github.io/arkdb/index.html) one could easily chunk large data from flat text files to these lite databases without running into memory limitations.

As for making 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` allows for uploading and downloading such files to GitHub releases, which would make it easy for anyone to access data files wherever the script is being run. It's worth emphasizing again that this is a short-term solution that is dependent on GitHub maintaining this service.

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 by placing it in a repository, packaging it with methods, or by 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, or a in serverless databaseses. This approach **gets the data somewhere rather than nowhere**. It's 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), Genetic data can be hosted at GenBank (@GenBank), and so on. Some other places to put data include the open science framework: https://osf.io/, Dataverse, https://dataverse.org/, and data.world, https://data.world/. 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 organisations 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 (https://github.com/the-pudding/data), or organisations such as data.world, or may self host their data (https://data.fivethirtyeight.com/).

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 (ref section 2). 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

Publishing data in a data repository

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). 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.

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

```
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 XX. 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.

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
- **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.

Codebook: Human readable dictionary of data contents

Codebooks provide human readable description of the variables in the data and contain information on:

- **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" [@McGill-codebook]. **Variable codes** apply to categorical (factor) variables, and are the values for their contents. For example, 0 = no, 1 = yes, and 0 = male and 1 = female. 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 [@White2013; @Broman2017].

Below is an example data dictionary table taken from the tidy tuesday repository on incarceration trends. 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)

variable	class	Description
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 (see Figure XX.).

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)

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-comp-bio-data]. For a brief overview of the CC BY, suitable to include in a README, see @ccby-short. For the full license, see @ccby-long.

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 https://creativecommons.org/choose/zero/. For a brief overview of the CC0, see @cc0-short, and for the full license, see @cc0-long.

Other licenses or notices to be aware of are **copyrighted data**, and **data embargos**. 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.

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.

Machine readable metadata

The 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. Say for a dataset called "demographics" with three variables:

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

Table XX Example demographics table of age, height, and nationality.

```
{
  "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 XX 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, an extended `csv` format, which has additional front matter in a YAML format using Table Schema.

Raw data: The original/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.

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.

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 [@Wickham2014], where tidy data contains variables in columns, and observations in rows. Contrasting `raw data`, `tidy data/analysis 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.

Tooling for packaging data

NOTE: not sure if this quite fits here, tooling is important, but at the moment we are not sure if we should describe all tooling, as I will have a lot of examples for R, and not for Python/Julia

Tooling for producing these this information speeds up the process of sharing data. To help create codebooks, there are R packages such as `dataMeta`, `memisc`. and `codebook`. 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, which provides tools to wrap up data into an R package, while prioviding helpers with MD5sum checks that track versioning. Note that is different to Frictionlessdata's tabular data package spec.

Example datasets

Dataset 1: Forest Census Plot on Barro Colorado Island

The "Forest Ceneus Plot on Barro Colorado Island" provides an example of data that is time consuming to document, but also robust. It contains information on trees from a 50-hectare tree plot. Here they state:

> Censuses have been carried out in 1981-1983, 1985, 1990, 1995, 2000, 2005, 2010, and 2015. In each census, all free-standing woody stems at least 10 mm diameter at breast height were identified, tagged, and mapped. Over 350,000 individual trees have been censused over 35 years.

This is an enormous, ongoing research effort. Brief context of the data is provided at the link, along with a link to the census plot data from 2012 at this paper. This provides an abstract and description of the data, along with co-authors, and a recommended acknowledgement.

This dataset falls in the top right of the graph, at high effort to prepare, and high ease of understanding. It is high effort to prepare because it is a large research effort that not only involves expensive research in terms of time. It is high ease of understanding because

The impact of this kind of research is huge, as it provides information on biodiversity …, …, … .

Dataset 2: Sensor data

Datasets obtained from sensors such as meteorological data are typically easy to prepare and understand. This is because the sensors have to be very specific about what they are measuring, so a lot of the description

of the type of data collected happens upstream at the instrument-design level. This flows down to when data is collected. The telescope data from the Sloan Digital Sky Survey (SDSS) is a great example of this. It contains:

> … the most detailed three-dimensional maps of the Universe ever made, with deep multi-color images of one third of the sky, and spectra for more than three million astronomical objects. –(https://www.sdss.org/)

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

- Good metadata in SCHEMA, a machine readable standard
- Data in multiple formats:
- Released data in archives
- Data is available in a database
- Entire dataset available for download
- Interface to the data covers all skills
- The interface provides tutorials from school children to professional astronomers on how to read it
- Raw data before processed into database is also available

The SDSS provides different degrees of understanding, for different understanding. It is a huge project involving many people, and so would have provided enormous effort to prepare, but has a high degree of understanding. It would be in the top right hand corner of the DARECO.

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.

Summary

TODO: Needs a succint summary here

Ten simple rules for publishing data

1. **Decide whether publishing is appropriate for your data** It is important for everyone to publish their data in a research data archive to ensure long-term availability, curated metadata, and that they are preserved in the future. However this step might not be necessary depending on the stage and purpose of your project. In cases where this step is cumbersome, ensuring that the data are readily available by other means.

2. **Include a README** file with your data archive. README files have a long history dating back to the 1970s in the context of free software and are a requirement of the GNU standard. In a more generic sense, README files are a form of simple documentation that describe the contents of a directory, along with brief descriptions, the type of license used and information on how to cite the collection.

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 that describes what each column contains. This can be a json schema file, which also helps websites like Google Dataset Search automatically discover your data.

5. **Provide the data in its rawest form** in a folder called "data-raw". This ensures provenance of your data by keeping a master copy from which all other tidied up data can be created.

6. **Provide [open source?] scripts** used to clean data from rawest form into tidy/analysis ready format. These 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 CDNs make access and reuse practical, but fall back to Zenodo or similar in case these services were to go away.

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 functions not only remove redundancy by removing the need to download the same data again, but also help ensure that the data are valid.

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

10. Store a copy of your data in simple formats such as csv: CSV files follow a flat and extremely simple schema and can be opened by any text editor. These files can also be easily parsed by any data science programming language. Modern software packages allow for fast reads without consuming too much memory and these tools also allow for easy modification of the data.

Conclusions

TBD

Literature Cited