

Chapter 18

Data provenance

Data provenance or data lineage refers to the detailed history of how data was created and manipulated, as well as the process of ensuring the validity of such data by documenting the details of its origins and transformations. For instance, we want to know where (by whom) a dataset was created and what was the process used to create it. Then, if there were any changes, such as fixing erroneous entries, we need to have a good record of such changes.

18.1 Why should we care?

We care because things can go terribly wrong if we don't. Imagine that your year-long analysis was based on a wrong, or simply outdated, version of the dataset! Or imagine your analysis was based on a wrong assumption on the meaning of a column in the dataset. Not ensuring data provenance can potentially lead to career-ending disasters.

Imagine a file generically named `data.txt` containing some tables of numbers but without explanation. No header names, no `readme.txt` you can find. Then imagine you discover files `data-v1.txt`, `data-v2.txt`, `data-v2-final-new.txt`. Which file should you work with? Who made the different files and when? Good research practice means we must have answers to these questions. We must both understand the provenance of the data and ensure that the provenance remains accessible moving forward.

18.2 Best practices for data provenance

The key to data provenance is record-keeping while you work. New data replaces old data. Data will be changed by your code, then the code will change. Data will be changed by your collaborators, without explanation. Having a personal practice of record-keeping is critical, and an eternal vigilance.

18.2.1 Document, document, document


The most important step is to document details about your dataset. Are you constructing your dataset? Then document each and every step of the construction process meticulously. Have you received the dataset from someone else? Then document each and every step of the data acquisition process. Who sent you the data? When? Were there multiple updates?

Then document what is in the dataset. Document each and every column of the dataset. What is the meaning of each column? What is the data type of each column? What is the range of each column? Some columns may use pre-defined *data vocabularies* such as ICD-10 (International Classification of Diseases, 10th Revision) codes; some columns may contain free text. Together, the detailed documentation that explains a dataset is known as the *data dictionary*.

Then document your processing of the dataset. A good way is *never touching the dataset by hand* and doing everything through a script. The script can then serve as a record of the data processing steps. For instance, suppose there is a typo in the dataset and you need to replace some words. First, create a separate data table that contains all the replacements to be made then write and run a script to apply those replacements. The replacements table and the script together document your changes.¹ And the script and data can be woven into a workflow using workflow tools (Ch. 20).

Documentation is eternal: when (not if) data change, the documentation will need to change with it.

Data identifiers and filenames It should never be a mystery where a file came from or what it means. Appropriate identifiers are key to good provenance. Identifiers can be URIs (uniform resource identifiers) or DOIs (Digital Object Identifiers), but at the most basic level we can think of them as filenames. Identifiers should be unique and follow a consistent naming convention. Consistency prevents confusion and forgetfulness around the data; once you design or learn the naming pattern, you can quickly derive the identifier. Choose identifies that give enough information so an interested party can figure out where the data fits into the research. We recommend identifiers that incorporate project name, author name (if appropriate) or data source name, a brief description of the data (say two–three words) and a rough indication of the file format (which can be as simple as a file name extension). We see that `inferred-social-network_bagrow_node-attributes_v20221001.json`, while long, is more informative than `node-attributes.json`. Avoid special characters and spaces in filenames, and document any abbreviations used to prevent names becoming overly long.

 Avoid generic file names, such as “f.txt,” “result.dat,” “output.csv,” etc. Even for the briefest programming session, be fastidious with filenames.

A corollary: if you encounter a codebase or dataset containing such generic filenames, be wary of problems and skeptical of the data provider’s commitment to data provenance good practices.

¹ Consider also recording metadata such as when the script was last run and why. This may also be captured automatically with a logging framework.

- ✓ Always use a unique filename such that you can determine where in your code the file was made even without running the code. Add “slugs” as unique parts of the name when building file names programmatically.

Data versioning As data are modified, you’ll need to track different versions. If not, you may wind up with different copies of the data and be unsure which to use. Even worse, the different copies may have undergone separate revisions, causing a diverging history that you will need to reconcile. We recommend three practices to help alleviate this problem. First, use data versions; we recommend a timestamp-based approach (see also Sec. 15.6.2), not version numbers. And never indicate “version new” or “version final” as you will soon have “version new new,” “version final new,” “version new final 2,” etc. That way lies madness. Second, when possible, log whatever the latest version of the data is, alongside older versions. Three, avoid discarding older versions: you never know if you will need to consult them in the future. The second and third practices interact: as you revise data, update the “latest version” metadata on *all* prior versions. This can be a burden if not automated, but it is good practice.

- ! Avoid “free-floating” version identifiers. When confronted with “data-new.txt,” “data-new-final.txt,” and “data-new-new.txt,” you will struggle to tell which to use.

- ✓ Use timestamps as version numbers for data.

- ! Don’t rely on filesystem metadata such as “last modified” dates. These can be notoriously ephemeral, one disk migration or operating system upgrade away from being rewritten.

Checksums A computational tool that can help with tracking data provenance is the *checksum*. Checksums are small blocks of data, computed by an algorithm, that describe our data. Crucially, if our data change by a tiny amount, the checksum will be completely different. Checksums are often used to check for errors when data are transmitted. If the checksum on the received data has changed compared to the original checksum, we know a change happened; conversely, if the checksums are equal, there is a very high probability the data are unchanged. Checksums are also very important for cryptographic security, but for data scientists, integrity checks are the most valuable use of checksums. We recommend recording checksums whenever you generate an important data file to help you distinguish the file and detect if errors in the data have appeared. All computer platforms come with built-in commands for computing checksums.

- ✓ Compute and save checksums of important data files. Keep them in a readme or other documentation file, along with the date when they were computed and other relevant details.

Raw data should be read-only A simple step to safeguard data is making raw data read-only. Different computer systems and database processes provide file access permissions in different ways. Flagging data as read-only forces you to never change the data by hand. You can only derive new datasets from the raw data. While not foolproof, this extra layer of permission does provide a valuable safety check, particularly against accidental modifications (such as from buggy code).

18.3 Backups

As discussed in Ch. 17, backups are critical when working with digital files. While computers are astonishingly reliable these days, it is still easy to lose information.

Backing up data presents challenges and opportunities when it comes to tracking data provenance. First, backups necessarily mean multiple copies of the data will exist. You need to keep track of and ensure all copies are synchronized, updated, and otherwise tracked.

What happens when data files are dynamic? For example, an experiment may be ongoing and data continue to be generated. Or the data come from a web service and new outputs are collected, say, daily. Follow the data versioning practices we described above as part of your backup practice. Together, good identifiers and versioning can help maintain provenance when backing up data.

Finally, consider using checksums to track different versions of a backed-up file. Checksums are again helpful here, to track the integrity of your backups. Keep logs of checksums separate from the files themselves so that they are available in the event of data loss.

18.4 Summary

Data provenance is a central challenge when working with data. Computing helps but also hinders our ability to maintain records of the work we do with the data. The best science will result when we adopt strategies to carefully and consistently record and track the origin of data and any changes made along the way. While such strategies generally take time and effort to implement, making them seem tedious in the short term, over time your research will become more reliable and you and your collaborators will be grateful.

Bibliographic remarks

The overview of record-keeping best practices by Schreier et al. [418], includes helpful information for data provenance. An influential review by Gray et al. [191] is also worth consulting, particularly the discussion of scientists choosing between files and databases. Edwards et al. [141] discuss pain points between data management and interdisciplinary collaboration.