14

Best Practices in Data Management

At the end of our journey through different tools and techniques for data management, what have we learned? How can we use these skills to make our research better? As I emphasized throughout the book, good data management means thinking about two different aspects: the *structure* of your data, and the *manipulation* of the data content.

14.1 TWO GENERAL RECOMMENDATIONS

Thinking about data structure is usually straightforward in the social sciences, since the vast majority of our data is stored in tables or can be fit into a tabular format. A tabular data format is one where we have information on separate cases (rows), each of which consists of the same set of variables (columns). Since this tabular data structure is so common in the social sciences, we hardly reflect on it. Still, there is considerable variation in how different tools deal with tables. In spreadsheets, for example, a table is simply a two-dimensional container of information, without necessarily imposing a standardized structure of variables and cases. Rows can have different lengths, cells can be merged, and the type of data stored in a column can vary between cases. This means that spreadsheet software offers little support to ensure the correct format of our data and to reduce errors, which (among many other problems) can lead to trouble when processing spreadsheet data further in statistical software. Hence, it is up to the user to check for errors and inconsistencies in the data.

https://doi.org/10.1017/9781108990424.020 Published online by Cambridge University Press

Other tools take data structure much more seriously. R, for example, explicitly uses a rectangular tables as its standard data structures, such as data frames or tibbles. While the columns in a data frame always have types, these types can change silently when you re-assign values to them. This is called "dynamic typing," and it is a standard feature of the R programming language. Therefore, while R at least maintains typed columns, it also does not prevent you from certain errors, such as setting a string value in what is supposed to be a numeric variable. Relational databases follow a stricter approach here. As you recall, in a relational database, data definition is a separate step in your workflow, where you set up the structure of tables before adding content. This involves specifying the columns and their types, and the database system later makes sure that only valid data is entered into the respective columns. Changing column types is an explicit step and needs to be done using the appropriate commands in SQL.

Data structure, however, also relates to the question of long vs. wide tables, or the splitting of data across several tables. Recall that "wide" tables are those where we have a dataset with two dimensions (e.g., countries and years), and where the columns are used to represent one of these dimensions. In a "long" table, the two dimensions are simply stored in different columns. We discussed that wide tables are usually not a good choice, as the addition of more data requires changes to the table structure (the addition of more columns). Also, as a general rule of thumb, tables should contain data on exactly one type of entity only. In order to avoid redundant data, we distributed our data across several tables, and linked them to each other by means of unique identifiers.

In sum, when working on an empirical research project, it is a good idea to reflect on the structure of your data. In some cases, this will be easy, in particular if your research project involves a single table only. For other projects, it may be worth spending some time to figure out a suitable structure underlying the data, as this will make your work easier. In the following, I list a number of questions that can help you do this.

RECOMMENDATION 1: THINK ABOUT THE STRUCTURE OF YOUR DATA.

- What variables does your dataset contain, and what are their types?
- Can you avoid redundant data? For example, can one variable simply be generated by transforming one or more other variable(s)? If so, there is no need to keep the former in the main dataset.

https://doi.org/10.1017/9781108990424.020 Published online by Cambridge University Press

- Does your table grow down when adding data? Do you keep your data in a "long" format, such that the addition of new data and new variables remains easy?
- Do you only have data on one type of entity in the dataset? If not, you may consider dividing your data into several tables, making sure that unique identifiers exist to link the records.

The second main aspect to consider for an empirical research project is the workflow in which you process your data. As discussed in Chapter 1, data management involves the creation of datasets for analysis, starting from one or more raw input datasets. One of the most important requirements for us is the transparency and replicability of the data processing workflow. In other words, we need to make sure to document every step we take in getting from the raw input datasets to the analysis dataset. The way to do this is to save this workflow as a script, for example, using the R programming language. Manual "point-and-click" operations such as editing and reformatting data in a spreadsheet should be avoided if at all possible (but can of course be useful when creating a new, manually coded dataset from scratch). If you follow the approach taken throughout the book, all your data management work will be done in R and possibly SQL, which is why it is transparent and can be replicated.

Another important question is whether you need a file-based workflow, or whether you can benefit from using a dedicated database for your data. A purely file-based approach is technically easier to implement, since you do not need an additional database server and the separate steps to import/export the data. However, databases can be useful if your datasets are large and/or involve many interlinked tables. In theses cases, specialized functionality in a relational DBMS can help you keep these tables consistent, while being able to speed up operations involving large numbers of records. Also, databases are designed for a multi-user environment with different levels of access. For example, this makes it possible to keep a database available such that some users can update it, while others have read-only access.

Of course, your workflow and the tools you choose for it are also strongly determined by the type of data you deal with. Simple tables with text and numbers can be processed in almost any type of statistical toolkit or database system. If you need to process more specialized types such as spatially referenced or textual data, this will affect your choice

of software. I presented various extension libraries for R, but we also discussed how PostgreSQL can be extended to manage data beyond the traditional tabular model.

Overall, the choice of a particular processing pipeline is closely related to the above questions around data structure. Once you know what type of data you deal with and what its structure should be, you can select the suitable software tools for processing it. Again, here is a list of possible questions you should consider when doing so.

RECOMMENDATION 2: THINK ABOUT THE WORKFLOW TO PROCESS YOUR DATA.

- What is the amount of data you have? R can deal with small to medium-sized datasets well, but it may find large ones difficult to process. In these cases, databases provide the necessary features such as indexing, which allow you to deal with large amounts of data efficiently.
- Do you require the software to ensure the correctness and consistency of your data? If so, it may be useful to opt for a relational database with explicitly defined table structures and support for interlinked tables.
- Do you need to deal with specialized types of data? R has many extension libraries that can handle spatial data, text data, or networks. Some of this functionality is also available through PostgreSQL's extensions, but this is much more limited.
- Does the complexity of the technical setup matter? Purely file-based data storage with data processing in R is easier to set up, and requires less technical overhead for others replicating this work.
- Do several collaborators work on the data at the same time? If there is concurrent access to data by several users, file-based data storage is often not ideal. For these cases, a distributed setup with data stored on a separate database server should be preferred.

To conclude the book, I give some recommendations on how to handle two challenges that often arise in data management and coding: the collaborative work on research projects, and the public dissemination of research data.

14.2 COLLABORATIVE DATA MANAGEMENT

More and more work in the social sciences is conducted collaboratively, which means that several scholars work together to produce an article or a book. Oftentimes, this also means that empirical work on a project is

carried out by different researchers, and data and code must be shared between them. What is the best way to organize this process?

A simple and very common solution is to use a shared drive (such as Dropbox) for the exchange of data and code. In practice, however, this can lead to several problems. First, with multiple users accessing the same shared files, there is a huge risk of someone overwriting somebody else's changes. Imagine User A and User B editing an R script on the shared folder at the same time. If User A saves their changes first, User B will overwrite these changes when saving their edits. The same can happen to data stored on a shared drive. The second (and related) problem is that most cloud storage services do not keep histories of files, unless you explicitly enable (and pay for) this functionality. That is, once you save a file, only the latest version remains available, and you no longer have access to earlier ones. For these reasons, I recommend not to use simple shared drives for collaboration; they only work if there is a clear division of labor such that at any given point in time, there is *only one user writing to the shared drive*. This may be difficult to implement in practice and can still result in data loss, which is why it is preferable to use a version control system (VCS).

VCS have been developed for computer programming, so they can also be used for data management as long as all your operations are documented in code (which after reading this book, they should be). A VCS can be helpful for your work in two ways. First, you can save *different versions* of your source code as you continue to work on it. These versions represent different stages of your project, and you typically add a short summary to each version to describe what it does. Overall, this transparent approach to code development is also very useful when working on a project alone (without collaborators), since it allows you to go back to particular versions and track the changes you made since then – for example, to check for errors in your code. Second, for collaborative work, the VCS allows you to combine and merge changes made by different users into a single code base. Figure 14.1 illustrates this.

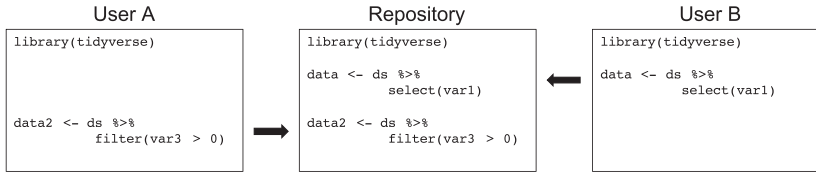

FIGURE 14.1. A version control system for collaborative coding.

https://doi.org/10.1017/9781108990424.020 Published online by Cambridge University Press

If two researchers (User A and User B) each work on the same script, each of them can push their changes into a main repository maintained by the VCS. This repository serves as the storage facility for the different revisions that the users make to the code. It allows each user to pull changes made by the others and merge them into their local copy of the script, without overwriting their own edits. There exist different types of VCS. By now, *distributed* VCS such as Git are the most popular, and there exist numerous introductions and quick-start guides online (see, e.g., Blischak et al., 2016).

What type of content should be managed by a VCS? Systems of this kind are tailored to the management of source code. That is, they are designed to track changes in text files, but not in binary files (such as Stata's `.dta` files). For the purpose of data management, this usually means that only your data management *scripts* should be placed under version control, but not the data files themselves, if you can avoid it. For collaborative projects, I recommend that your code include download commands to fetch the input datasets from their original location, rather than storing these datasets in your repository. This way, each collaborator can generate the output datasets simply by running the project code, without the need to manually obtain copies of the input datasets. This, however, is only possible if the original input datasets do not change and can always be obtained from a given location. If this is not the case, adding a copy to your repository is usually the best option. Of course, if your data resides in a relational database and needs to be accessed by the different collaborators in the project, this is even easier, since you simply connect to the same database and perform much of your data processing there. The code to do this should of course be located in a file under version control.

A detailed introduction to version control is beyond the scope of this book. My goal in this section was to make you aware of these systems and provide some intuition on what they do. If you frequently work on projects with several collaborators, it is definitely worth learning more about these systems, even if the initial learning curve may be steep. However, as mentioned above, VCS are designed to manage code, not data. If you need to distribute data files, we take a brief look at data dissemination in the following section.

14.3 DISSEMINATING RESEARCH DATA AND CODE

The data management and processing that we described in this book is done by you as a researcher, and by the other members of your team.

https://doi.org/10.1017/9781108990424.020 Published online by Cambridge University Press

However, at some point during the project, you usually need to make your procedures public, to ensure transparency of your scientific approach and to give others the possibility to replicate your work. At the latest, this should happen when a research article is published, and many journals by now require the publication of replication data and code along with the article. Typically, this includes material for the analysis only; in the terminology introduced in Chapter 1, this is the analysis dataset(s) and the analysis code.

However, at a time when empirical datasets in the social sciences are becoming increasingly complex and large, there are good reasons to also share data and code that were used to generate the analysis data in the first place. In particular, if several input datasets are involved, mistakes can happen when merging them. While sharing *code* is straightforward since the files are small and there is usually no sensitive content involved, this may be different for research *data*. Datasets can contain sensitive information and attributes that identify individuals, which is why they often cannot be shared in public. Other data are prevented from public dissemination by legal constraints, for example, if they were purchased from a commercial provider. While sharing data is preferred from a scientific point of view, it is essential that this happens within the given ethical and legal limits. To learn about new approaches for preserving privacy in research data while still making analysis possible, see, for example, Evans and King (2022). Also, all your data and code should be properly documented.

When all legal and ethical requirements for data sharing are met, you can use one of several ways to disseminate your research data and code. The easiest one is again to post these materials on a publicly available website, as many researchers and journals do for their replication materials. However, I recommend using one of the freely available, specialized portals for this purpose. Many institutions use the *Dataverse* software to create their own research dissemination infrastructure, but you can use Harvard's Dataverse at https://dataverse.harvard.edu to set up a free account and post your code and data. An even more comprehensive research infrastructure is provided by the *Open Science Foundation* at https://osf.io. While it contains numerous other useful tools to facilitate collaboration among researchers, it also allows you to share your data and code publicly.

These two portals have several features that make them particularly suitable for data dissemination. First, they maintain different versions of files. We already saw that this is useful for code files, but is also necessary when distributing research data publicly. Using the file history offered

by Dataverse or OSF, you can release new versions of your data that fix mistakes or extend coverage of your data. Since these versions can be tagged with a short description, you can document the evolution of a file or a dataset for users outside your project. Second, the data portals allow you to make your data files accessible under a fixed link, regardless of the current version of the file. That is, when you distribute a link to your file, this link remains the same even if you update the file. Older versions remain accessible in the data repository, but you need to explicitly request the respective version. This makes the development of datasets, but also the access to them, flexible and transparent, without the need to use your own website or a simple cloud storage facility.

14.4 SUMMARY AND OUTLOOK

At the beginning of this book, I asked whether it is useful to spend an entire book on data management. I gave a number of reasons for why this would be helpful to quantitative researchers working with increasingly complex and large amounts of data. A key learning I emphasized in the book is the need to *document* every data processing step in getting from the raw input data to the dataset(s) used for analysis. This makes data management as *convenient* as possible for you as a researcher, but also ensures that your research is *replicable and transparent* for others. All this is possible when you perform data management in code, such that each operation is included in your script. It becomes much more difficult when using manual operations, for example, in a spreadsheet software. The tools and techniques we covered in this book also allow you to make your data management process *scalable* and *versatile*, such that the same approaches can be used regardless of the size of your dataset or the types of data that you intend to use for your research project. Here, in particular, I introduced relational databases as a dedicated data management tool, which can handle large amounts of data, but can also be extended to cover more specialized types such as spatial or textual data.

The tabular data model has been of central importance throughout this book. Tables come naturally to social scientists – all the applied examples we teach in introductory methods classes involve data formatted as tables. In the book, I showed how tables are useful for many applications, including those that go beyond the conventional format of a single, rectangular table. Data can be distributed across several tables, but the traditional structure can also be extended to include more specialized columns such as those with spatial coordinates. Despite the continued

importance of structured, tabular data, unstructured data will become increasingly relevant also in the social sciences. In this book, we only scratched the surface of this topic when dealing with text data. Therefore, if you want to continue to expand your data analysis skills beyond the topics covered in this book, learning more about text analysis and related topics would be a promising way to go. I hope that this book can serve as a useful and comprehensive preparation as you continue this journey.

https://doi.org/10.1017/9781108990424.020 Published online by Cambridge University Press