BROOKE ANDERSON, MICHAEL LYONS, MERCEDES GONZALEZ-JUARRERO, MARCELA HENAO-TAMAYO, AND GREGORY ROBERTSON

IMPROVING THE REPRODUCIBILITY OF EXPERIMENTAL DATA RECORDING AND PRE-PROCESSIN

Contents

1
Overview

The recent NIH-Wide Strategic Plan (U.S. Department of Health and Human Services, National Institutes of Health, 2016) describes an integrative view of biology and human health that includes translational medicine, team science, and the importance of capitalizing on an exponentially growing and increasingly complex data ecosystem (U.S. Department of Health and Human Services, National Institutes of Health, 2018). Underlying this view is the need to use, share, and re-use biomedical data generated from widely varying experimental systems and researchers. Basic sources of biomedical data range from relatively small sets of measurements, such as animal body weights and bacterial cell counts that may be recorded by hand, to thousands or millions of instrument-generated data points from various imaging, -omic, and flow cytometry experiments. In either case, there is a generally common workflow that proceeds from measurement to data recording, pre-processing, analysis, and interpretation. However, in practice the distinct actions of data recording, data pre-processing, and data analysis are often merged or combined as a single entity by the researcher using commercial or open source spreadsheets, or as part of an often proprietary experimental measurement system / software combination (Figure 1.1), resulting in key failure points for reproducibility at the stages of data recording and pre-processing.

It is widely known and discussed among data scientists, mathematical modelers, and statisticians (Broman and Woo, 2018; Krishnan et al., 2016) that there is frequently a need to discard, transform, and reformat various elements of the data shared with them by laboratory-based researchers, and that data is often shared in an unstructured format, increasing the risks of introducing errors through reformatting before applying more advanced computational methods. Instead, a critical need for reproducibility is for the transparent and clear sharing across research teams of: (1) raw data, directly from hand-recording or directly output from experimental equipment; (2) data that has been pre-processed as necessary (e.g., gating for flow cytometry data, feature identification for metabolomics data), saved in a consistent, structured format, and (3) a clear and repeatable description of how the pre-processed data was

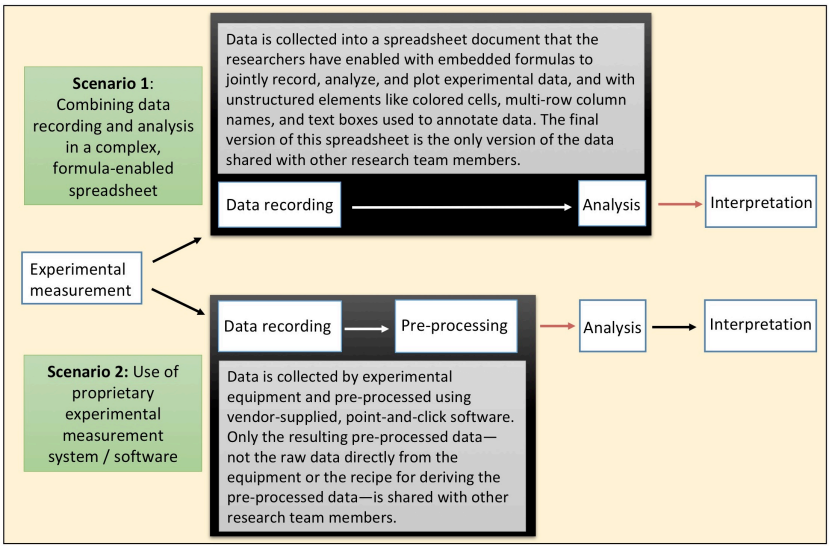

Figure 1.1: Two scenarios where 'black boxes' of non-transparent, non-reproducible data handling exist in research data workflows at the stages of data recording and pre-processing. These create potential points of failure for reproducible research. Red arrows indicate where data is passed to other research team members, including statisticians / data analysts, often within complex or unstructured spreadsheet files.

generated from the raw data (Broman and Woo, 2018; Ellis and Leek, 2018).

To enhance data reproducibility, it is critical to create a clear separation among data recording, data pre-processing, and data analysis—breaking up commonly existing "black boxes" in data handling across the research process. Such a rigorous demarcation requires some change in the conventional understanding and use of spreadsheets and a recognition by biomedical researchers that recent advances in computer programming languages, especially the R programming language, provide user-friendly and accessible tools and concepts that can be used to extend a transparent and reproducible data workflow to the steps of data recording and pre-processing. Among our team, we have found that there are many common existing practices—including use of spreadsheets with embedded formulas that concurrently record and analyze experimental data, problematic management of project files, and reliance on proprietary, vendor-supplied point-and-click software for data pre-processing—that can interfere with the transparency, reproducibility, and efficiency of laboratory-based biomedical research projects, problems that have also been identified by others as key barriers to research reproducibility (Broman and Woo, 2018; Bryan, 2018; Ellis and Leek, 2018; Marwick et al., 2018). In these training modules, we have choosen topics that tackle barriers to reproducibility that have straightforward, easy-to-teach solutions, but which are still very common in biomedical laboratory-based research programs.

1.1 License

Click on the **Next** button (or navigate using the links at the top of the page) to continue.

2
Experimental Data Recording

2.1 Separating data recording and analysis

Many biomedical laboratories currently use spreadsheets—with formulas creating underlying connections between spreadsheet cells—to jointly record, visualize, and analyze experimental data (Broman and Woo, 2018). This practice impedes the transparency and reproducibility of both data recording and data analysis. When a research group develops and uses an evolving spreadsheet template with embedded formulas, it leads to a data recording / analysis process that can become extraordinarily opaque and complex. To improve the computational reproducibility of a research project, it is critical for biomedical researchers to learn the importance of maintaining recorded experimental data as "read-only" files, separating data recording from any data pre-processing or data analysis steps (Broman and Woo, 2018; Marwick et al., 2018). Statisticians have outlined specific methods that a laboratory-based scientist can take to ensure that data shared in an Excel spreadsheet are shared in a reliable and reproducible way, including avoiding macros or embedded formulas, using a separate Excel file for each dataset, recording descriptions of variables in a separate code book rather than in the Excel file, avoiding the use of color of the cells to encode information, using "NA" to code missing values, avoiding spaces in column headers, and avoiding splitting or merging cells (Ellis and Leek, 2018; Broman and Woo, 2018). In this module, we will describe this common practice and will outline alternative approaches that separate the steps of data recording and data analysis.

Objectives. After this module, the trainee will be able to:

- Explain the difference between data recording and data analysis
- Understand why collecting data on spreadsheets with embedded formulas impedes reproducibility
- List alternative approaches to improve reproducibility

2.1.1 Data recording versus data analysis

Many scientific laboratories use spreadsheets within their data collection process, both to record data and to clean and analyze the data. One survey of over 250 biomedical researchers at the University of Washington found that most respondants used general-purpose applications like spreadsheets (Anderson et al., 2007), while a survey of neuroscience researchers at the University of Newcastle similarly found that most respondents used spreadsheets and other general-purpose software in their research (AlTarawneh and Thorne, 2017). A working group on bioinformatics and data-intensive science similarly found spreadsheets were the most common tool used across attendees (Barga et al., 2011).

Spreadsheets have long been an extremely popular tool, in part because they allow people without programming experience to conduct a range of standard computations and statistical analyses through a visual interface that is more immediately user-friendly to non-programmers than programs with command line interfaces. An early target for spreadsheet programs in terms of early users was business executives, and so the programs were designed to be very simple and easy to use—just one step up in complexity from crunching numbers on the back of an envelope (Campbell-Kelly, 2007). Spreadsheet programs in fact became so popular within businesses that many attribute these programs with driving the uptake of personal computers (Campbell-Kelly, 2007).

Spreadsheets were innovative and rapidly adapted in part because they allowed users to combine data recording and analysis—while previously, in business settings, any complicated data analysis tasked needed to be outsourced to mainframe computers and data processing teams, the initial spreadsheet program (VisiCalc) allowed one person to quickly apply and test different models or calculations to recorded data (Levy, 1984). These spreadsheet programs allowed non-programmers to engage with data, including data processing and analysis tasks, that previously required programming expertise (Levy, 1984).

In some cases, a spreadsheet is used solely to record data, as a simple type of database (Birch et al., 2018). However, biomedical researchers often use spreadsheets to both record and analyze experimental data (Anderson et al., 2007). In this case, data processing and analysis is implemented through the use of formulas and macros embedded within the spreadsheet. When a spreadsheet has formulas or macros within it, the spreadsheet program creates an internal record of how cells are connected through these formulas. For example, if the value in a specific cell is converted from Fahrenheit to Celsius to fill a second cell, and then that value is combined with other values in a column to calculate the mean temperature across several observations, then the spreadsheet program has internally saved how the later cells depend on the earlier ones. When you change the value recorded in a cell of a spreadsheet, the spreadsheet program queries this record and only recalculates the cells that depend on that cell. This process allows the program to quickly "react" to any change in cell inputs,

immediately providing an update to all downstream calculations and analyses (Levy, 1984). Starting from the spreadsheet program Lotus 1-2-3, spreadsheet programs also included *macros*, "a single computer instruction that stands for a sequence of operations" (Creeth, 1985).

Spreadsheets have become so popular in part because so many people know how to use them, at least in basic ways, and so many people have the software on their computers that files can be shared with the virtual guarantee that everyone will be able to open the file on their own computer (Hermans et al., 2016). Spreadsheets uses the visual metaphore of a traditional gridded ledger sheet (Levy, 1984), providing an interface that is easy for users to immediately understand and create a mental map of (Birch et al., 2018; Barga et al., 2011). This visually clear interface also means that spreadsheets can be printed or incorporated into other documents (Word files, PowerPoint presentations) "as-is", as a workable and understandable table of data values. In fact, some of the most popular plug-in software packages for the early spreadsheet program Lotus 1-2-3 were programs for printing and publishing spreadsheets (Campbell-Kelly, 2007). This "What You See Is What You Get" interface was a huge advance from previous methods of data analysis for the first spreadsheet program, VisiCalc, providing a "window to the data" that was accessible to business executives and others without programming expertise (Creeth, 1985). Several surveys of researchers have found that spreadsheets were popular because of their simplicity and ease-of-use (Anderson et al., 2007; AlTarawneh and Thorne, 2017; Barga et al., 2011). By contrast, databases and scritped programming lanugages can be perceived as requiring a cognitive load and lengthly training that is not worth the investment when an easier tool is available (Hermans et al., 2016; Anderson et al., 2007; Myneni and Patel, 2010; Barga et al., 2011; Topaloglou et al., 2004).

2.1.2 Hazards of combining recording and analysis

Raw data often lost.

One of the key tenets of ensuring that research is computationally reproducible is to always keep a copy of all raw data, as well as the steps taken to get from the raw data to a cleaned version of the data through to the results of data analysis. However, maintaining a easily accessible copy of all original raw data for a project is a common problem among biomedical researchers (Goodman et al., 2014), especially as team members move on from a laboratory group (Myneni and Patel, 2010).

The use of spreadsheets to jointly record and analyze data can contribute to this problem. Spreadsheets allow for the immediate and embedded processing of data. As a result, it may become very difficult to pull out the raw data originally recorded in a spreadsheet. At the least, the combination of raw and processed data in a spreadsheet makes it hard to identify which data points within a spreadsheet make up the raw data and which are the result of process-

ing that raw data. One study of operational spreadsheets noted that:

> "The data used in most spreadsheets is undocumented and there is no practical way to check it. Even the original developer would have difficulty checking the data." (Powell et al., 2009)

Further, data in a spreadsheet is typically not saved as "read-only", so it is possible for it to be accidentally overwritten. In situations where spreadsheets are shared among multiple users, without "read-only" protection, original cell values can easily be accidentally written over, and it may not be clear who last changed a value, when it was changed, or why (AlTarawneh and Thorne, 2017).

Finally, many spreadsheets use a proprietary format. In the development of spreadsheet programs, this use of proprietary, binary file formats helped a software program keep users, increasing barriers for a user to switch to a new program (since it wouldn't be able to read their old files) (Campbell-Kelly, 2007). However, this file format may be hard to open in the future, as software changes and evolves (Michener, 2015); by comparison, plain text files should be widely accessible through general purpose tools regardless of changes to proprietary software like Microsoft Excel.

Opacity of analysis steps and potential for errors.

Previous studies have found that errors are very common within spreadsheets (Hermans et al., 2016). For example, one study of 50 operational spreadsheets found that about 90% contained at least one error (Powell et al., 2009). In part, it is easier to make errors in spreadsheets and harder to catch errors in later work with a spreadsheet because the formulas and connections between cells aren't visible when you look at the spreadsheet—they're behind the scenes (Birch et al., 2018). This makes it very hard to get a clear and complete view of the pipeline of analytic steps in data processing and analysis within a spreadsheet, as well as to discern how cells are connected within and across sheets of the spreadsheet. As one early article on the history of spreadsheet programs notes:

> "People tend to forget that even the most elegantly crafted spreadsheet is a house of cards, ready to collapse at the first erroneous assumption. The spreadsheet that looks good but turns out to be tragically wrong is becoming a familiar phenomenon." (Levy, 1984)

Some characteristics of spreadsheets may heighten chances for errors. These include high conditional complexity (i.e., lots of branching of data flow through if / else structures), formulas that depend on a large number of cells or that incorporate many functions (Hermans et al., 2016). Following the logical chain of spreadsheet formulas can be particularly difficult when several calculations are chained in a row (Hermans and Murphy-Hill, 2015). Very long chains of dependent formulas across spreadsheet cells may in some case requiring sketching out by hand the flow of information through the spreadsheet to understand what's going on (Nardi and Miller, 1990). The use of macros can

also make it particularly hard to figure out the steps of an analysis and to diagnose and fix any bugs in those steps (Nash, 2006; Creeth, 1985). One study of spreadsheets in use in real life applications noted that, "Many spreadsheets are so chaotically designed that auditing (especially of a few formulas) is extremely difficult or impossible." (Powell et al., 2009)

In some cases, formula dependences might span across different sheets of a spreadsheet file. For the example given above of a spreadsheet that converts temperature from one unit to another and then averages across observations, for example, the original temperature might be recorded in one sheet while the converted temperature value is calculated and shown in a second sheet. These cross-sheet dependencies can make the analysis steps even more opaque (Hermans et al., 2016), as a change in the cell value of one sheet might not be immediately visible as a change in another cell on that sheet (the same is true for spreadsheets so large that upstream and downstream cells are not concurrently visible on screen). Other common sources of errors included incorrect references to cells inside formulas and incorrect use of formulas (Powell et al., 2009) or errors introduced through the common practice of copying and pasting when developing spreadsheets (Hermans et al., 2016).

To keep analysis steps clear, whether in scripted code or in spreadsheets or pen-and-paper calculations, it is important to document what is being done at each step and why (Goodman et al., 2014). Scripted languages allow for code comments, which are written directly into the script but not evaluated by the computer, and so can be used to document steps within the code without changing the operation of the code. Further, the program file itself often presents a linear, step-by-step view of the pipeline, stored separated from the data itself (Creeth, 1985). Calculations done with pen-and-paper (e.g., in a laboratory notebook) can be annotated with text to document the steps. However, there is evidence that spreadsheets are often poorly documented, or documented in ways that are hard to keep track of. Before spreadsheets,

> "The formulas appeared in one place and the results in another. You could see what you were getting. That cannot be said of electronic spreadsheets, which don't display the formulas that govern their calculations. As Mitch Kapor explained, with electronic spreadsheets, 'You can just randomly make formulas, all of which depend on each other. And when you look at the final results, you have no way of knowing what the rules are, unless someone tells you.'" (Levy, 1984)

Within spreadsheets, the logic and methods behind the pipeline of data processing and analysis is often not documented, or only documented with cell comments (hard to see as a whole) or in emails, not the spreadsheet file. One study that investigated a large collection of spreadsheets found that most do not include documentation explaining the logic or implementation of data processing and analysis implemented within the spreadsheet (Hermans et al., 2016). A survey of neuroscience researchers at a UK institute found that about a third of respondents included no documentation for spreadsheets used in their research laboratories (AlTarawneh and Thorne, 2017).

When spreadsheet pipelines are documented, it is often through methods that are hard to find and interpret later. One study of scientific researchers found that, when research spreadsheets were documented, it was often through "cell comments" added to specific cells in the spreadsheet, which can be hard to interpret inclusively to understand the flow and logic of a spreadsheet as a whole (AlTarawneh and Thorne, 2017). In some cases, teams discuss and document functionality and changes in spreadsheets through email chains, passing different versions of the spreadsheet file as attachments of emails with discussion of the spreadsheet in the email body. One research team investigated over 700,000 emails from employees of Enron that were released during legal proceedings and investigated the spreadsheets attached to these emails (over 15,000 spreadsheets) as well as discussion of the spreadsheets within the emails themselves (Hermans and Murphy-Hill, 2015). They found that the logic and methods of calculations within the spreadsheets were often documented within the bodies of emails that team members used to share and discuss spreadsheets. This means that, if someone needs to figure out why a step was taken or identify when an error was introduced into a spreadsheet, they may need to dig through the chain of old emails documenting that spreadsheet, rather than being able to find the relevant documentation within the spreadsheet's own file.

Often spreadsheets are designed, and their structure determined, by one person, and this is often done in an *ad hoc* fashion, rather than designing the spreadsheet to follow a common structure for the research field or for the laboratory group (Anderson et al., 2007). Often, data processing and analysis pipelines for spreadsheets are not carefully designed; instead, it's more typically for spreadsheet user to start by directly entering data and formulas without a clear overall plan (AlTarawneh and Thorne, 2017). Often, the person who created the spreadsheet is the only person who fully knows how it works (Myneni and Patel, 2010), particularly if the spreadsheet includes complex macros or a complicated structure in the analysis pipeline (Creeth, 1985).

This practice creates a heavy dependence on the person who created that spreadsheet anytime the data or results in that spreadsheet need to be interpreted. This is particularly problematic in projects where the spreadsheet will be shared for collaboration or adapted to be used in a future project, as is often done in scientific research groups. One survey of neuroscience researchers at a UK institute, for example, found that "on average, 2–5 researchers share the same spreadsheet". (AlTarawneh and Thorne, 2017) In this case, it can be hard to "onboard" new people to use the file, and much of the work and knowledge about the spreadsheet can be lost when that person moves on from the business or laboratory group (Creeth, 1985; Myneni and Patel, 2010). If you share a spreadsheet with numerous and complex macros and formulas included to clean and analyze the data, it can take an extensive amount of time, and in some cases may be impossible, for the researcher you share it with to decipher what is being done to get from the original data input in some cells to the final results

shown in others and in graphs. Further, if others can't figure out the steps being done through macros and formulas in a spreadsheet, they will not be able to check it for problems in the logic of the overall analysis pipeline or for errors in the specific formulas used within that pipeline. They also will struggle to extend and adapt the spreadsheet to be used for other projects. These problems come up not only when sharing with a collaborator, but also when reviewing spreadsheets that you have previously created and used (as many have noted, your most frequent collaborator will likely be "future you"). In fact, one survey of biomedical researchers at the University of Washington noted that,

> "The profusion of individually created spreadsheets containing overlapping and inconsistently updated data created a great deal of confusion within some labs. There was little consideration to future data exchange of submission requirements at the time of publication." (Anderson et al., 2007)

There are methods that have been brought from more traditional programming work into spreadsheet programming to try to help limit errors, including spreadsheet assertions to enable testing of spreadsheets (Hermans et al., 2016). However, these are often not implemented, in part perhaps because many spreadsheet users see themselves as "end-users", creating spreadsheets for their own personal use rather than as something robust to future use by others, and so don't seek out strategies adopted by "programmers" when creating stable tools for others to use (Hermans et al., 2016). In practice, though, often a spreadsheet is used much longer, and by more people, than originally intended. Often, the spreadsheet in this case was not designed for robust, long-term use. From early in the history of spreadsheet programs, users have shared spreadsheet files with interesting functionality with other users (Levy, 1984), and the lifespan of a spreadsheet can be much longer than originally intended—a spreadsheet created by one user for their own personal use can end up being used and modified by that person or others for years (Hermans et al., 2016).

Subpar software for analysis.

While spreadsheets serve as a widely-used tool for data recording and analysis, in many cases spreadsheets programs are poorly suited to clean and analyze scientific data compared to other programs. As tools and interfaces continue to develop that make other software more user-friendly to those new to programming, scientists may want to reevaluate the costs and benefits, in terms of both time required for training and aptness of tools, for spreadsheet programs compared to using scripted programming languages like R and Python.

Several problems have been identified with spreadsheet programs in the context of recording and, especially, analyzing scientific data. First, some statistical methods may be inferior to those available in other statistical programming language. Since the most popular spreadsheet program (Excel) is closed source, it is hard to identify and diagnose such problems, and there is likely less of an incentive for problems in statistical methodology to be fixed (rather than using development time and funds to increase easier-to-see functionality in the program). Many statistical operations require computations that cannot be

perfectly achieved with a computer, since the computer must ultimately solve many mathematical problems using numerical approximations rather than continuous methods (e.g., calculus). The choice of the algorithms used for these approximations heavily influence how closely a result approximates the true answer.

A series of papers examined the quality of statistical methods in several statistical software programs, including Excel, starting in the 1990s (McCullough and Wilson, 1999; McCullough, 1999; McCullough and Wilson, 2002, 2005; McCullough and Heiser, 2008; Mélard, 2014). In the earliest studies, they found some concerns across all programs considered (McCullough and Wilson, 1999; McCullough, 1999). One of the biggest concerns, however, was that there was little evidence over the years that the identified problems in Excel were resolved, or at least improved, over time (McCullough, 2001; McCullough and Heiser, 2008). The authors note that there may be little incentive for checking and fixing problems with algorithms for statistical approximation in closed source software like Excel, where sales might depend more on the more immediately evident functionality in the software, while problems with statistical algorithms might be less evident to potential users (McCullough, 2001).

Open source software, on the other hand, offers pathways for identifying and fixing any problems in the software, including for statistical algorithms and methods implemented in the software's code. Since the full source code is available, researchers can closely inspect the algorithms being used and compare them to the latest knowledge in statistical computing methodology. Further, if an inferior algorithm is in use, most open source software licenses allow a user to adapt and extend the software, for example to implement better statistical algorithms.

Second, spreadsheet programs can include automated functionality that's meant to make something easier for most users, but that might invisibly create problems in some cases. A critical problem, for example, has been identified when using Excel for genomics data. When Excel encounters a cell value in a format that seems like it could be a date (e.g., "Mar-3-06"), it will try to convert that cell to a "date" class. Many software programs save date as this special "date" format, where it is printed and visually appears in a format like "3-Mar-06" but is saved internally by the program as a number (for Microsoft Excel, the number of days since January 1, 1900 (Willekens, 2013)). By doing this, the software can more easily undertake calculations with dates, like calculating the number of days between two dates or which of two dates is earlier. Bioinformatics researchers at the National Institutes of Health found that Excel was doing this type of automatic and irreversible date conversion for 30 gene names, including "MAR3" and "APR-4", resulting in these gene names being lost for further analysis (Zeeberg et al., 2004). Other automatic conversion problems caused the lost of clone identifiers with composed of digits and the letter "E" (Zeeberg et al., 2004; Welsh et al., 2017), which were assumed to be expressing a number using scientific notation and so automatically and irreversibly

converted to a numeric class. Further automatic conversion problems can be caused by cells that start with an operator (e.g., "+ control") or with leading zeros in a numeric identifier (e.g., "007") (Welsh et al., 2017).

Avoiding this automatic date conversion required specifying that columns with columns susceptible to these problems, including columns of gene names, should be retained in a "text" class in Excel's file import process. While this problem was originally identified and published in 2004 (Zeeberg et al., 2004), along with tips to identify and avoid the problem, a study in 2016 found that approximately a fifth of genomics papers investigated in a large-scale review had gene name errors resulting from Excel automatic conversion, with the rate of errors actually increasing over time (Ziemann et al., 2016).

Finally, spreadsheet programs can be limited as analysis needs become more complex or large (Topaloglou et al., 2004). For example, spreadsheets can be problematic when integrating or merging large, separate datasets (Birch et al., 2018). This can create barriers, for example, in biological studies seeking to integrate measurements from different instruments (e.g., flow cytometry data with RNA-sequencing data). Further, while datasets continue to expand in their capacity for data, for very large datasets they continue to face limits that may be reached in practical applications (Birch et al., 2018), and their efficiency of running data processing and analysis pipelines across large datasets can be slow compared to code implemented with other programming languages.

Difficulty collaborating with statisticians.

Modern biomedical researchers requires large teams, with statisticians and bioinformaticians often forming a critical part of the team to enable sophisticated processing and analysis of experimental data. However, the process of combining data recording and analysis of experimental data, especially through the use of spreadsheet programs, can create barriers in working across disciplines. One group defined these issues as "data friction" and "science friction"—the extra steps and work required at each interface where data passes, for example, from a machine to analysis or from a collaborator in one discipline to one in a separate discipline (Edwards et al., 2011). From a survey of scientific labs, for example, one respondent said:

> "I can give data that I think are appropriate to answer a question to a biostatistician, but when they look at it, they see it from a different point of view. And that spreadsheet does not really encapsulate where it came from very well, how was it generated, was it random, how was this data collected. You would run a series of queries that you think are pertinent to what this biostatistician would want to know. They become a part of the exploration and not just a receiver of whatever I decided to put in my spreadsheet on that day. What I get back is almost never fully documented in any way that I can really understand and add more to the process." (Myneni and Patel, 2010)

When collaborating with statisticians or bioinformaticians, one of the key sources of this "data friction" can result from the use of spreadsheets to jointly record and analyze experiemental data. First, spreadsheets are easy to print or

copy into another format (e.g., PowerPoint presentation, Word document), and so researchers often design spreadsheets to be immediately visually appealing to viewers. For example, a spreadsheet might be designed to include hierarchically organized headers (e.g., heading and subheading, some within a cell merged across several columns), or to show the result of a calculation at the bottom of a column of observations (e.g., "Total" in the last cell of the column) (Teixeira and Amaral, 2016). Multiple separate small tables might be included in the same sheet, with empty cells used for visual separation, or use a "horizontal single entry" design , where the headers are in the leftmost column rather than the top row (Teixeira and Amaral, 2016).

These spreadsheet design choices make it much more difficult for the contents of the spreadsheet to be read into other statistical programs. These types of data require several extra steps in coding, in some cases fairly complex coding, with regular expressions or logical rules needed to parse out the data and convert it to the needed shape, before the statistical work can be done for the dataset. This is a poor use of time for a collaborating statistician, especially if it can be avoided through the design of the data recording template. Further, it introduces many more chances for errors in cleaning the data.

Further, information embedded in formulas, macros, and extra formatting like color or text boxes is lost when the spreadsheet file is input into other programs. Spreadsheets allow users to use highlighting to represent information (e.g., measurements for control animals shown in red, those for experiment animals in blue) and to include information or documentation in text boxes. For example, one survey study of biomedical researchers at the University of Washington included this quote from a respondent: "I have one spreadsheet that has all of my chromosomes … and then I've gone through and color coded it for homozygosity and linkage." (Anderson et al., 2007) All the information encoded in this sheet through color will be lost when the data from the spreadsheet is read into another statistical program.

2.1.3 Approaches to separate recording and analysis

In the remaining modules in this section, we will present and describe techniques that can be used to limit or remove these problems. First, in the module on "Structure data", we will walk through techniques to design data recording formats so that data is saved in a consistent format across experiments within a laboratory group, and in a way that removes "data friction" for collaboration with statisticians or later use in scripted code. These techniques can be immediately used to design a better spreadsheet to be used solely for data collection.

In later modules, we will discuss the use of R projects to coordinate data recording and analysis steps within a directory, while using separate files for data recording versus data processing and analysis. These more advanced formats will enable the use of quality assurance / control measures like testing of data entry and analysis functionality, better documentation of data analysis

pipelines, and easy use of version control to track projects and collaborate transparently and with a recorded history.

[We will probably want to flesh this section out as we write later modules.]

2.1.4 Discussion questions

:w

2.2 Principles and power of structured data formats

The format in which experimental data is recorded can have a large influence on how easy and likely it is to implement reproducibility tools in later stages of the research workflow. Recording data in a "structured" format brings many benefits. In this module, we will explain what makes a dataset "structured" and why this format is a powerful tool for reproducible research.

Every extra step of data cleaning is another chance to introduce errors in experimental biomedical data, and yet laboratory-based researchers often share experimental data with collaborators in a format that requires extensive additional cleaning before it can be input into data analysis (Broman and Woo, 2018). Recording data in a "structured" format brings many benefits for later stages of the research process, especially in terms of improving reproducibility and reducing the probability of errors in analysis (Ellis and Leek, 2018). Data that is in a structured, tabular, two-dimensional format is substantially easier for collaborators to understand and work with, without additional data formatting (Broman and Woo, 2018). Further, by using a consistent structured format across many or all data in a research project, it becomes much easier to create solid, well-tested code scripts for data pre-processing and analysis and to apply those scripts consistently and reproducibly across datasets from multiple experiments (Broman and Woo, 2018). However, many biomedical researchers are unaware of this simple yet powerful strategy in data recording and how it can improve the efficiency and effectiveness of collaborations (Ellis and Leek, 2018).

Objectives. After this module, the trainee will be able to:

- List the characteristics of a structured data format
- Describe benefits for research transparency and reproducibility
- Outline other benefits of using a structured format when recording data

2.2.1 Data recording standards

For many areas of biological data, there has been a push to create standards for how data is recorded and communicated. Standards can clarify both the *content* that should be included in a dataset, the *format* in which that content is stored, and the *vocabulary* used within this data. One article names these three facets of a data standard as the **minimum information**, **file formats**, and **ontologies** (Ghosh et al., 2011).

> "It is important to distinguish between standards that specify how to actually do experiments and standards that specify how to describe experiments. Recommendations such as what standard reporters (probes) should be printed on microarrays or what quality control steps should be used in an experiment belong to the first category. Here we focus on the standards that specify how to describe and communicate data and information." (Brazma et al., 2006)

Many people and organizations (including funders) are excited about the idea of developing and using data standards, especially at the community level. Good standards, that are widely adapted by researchers, can help in making sure that data submitted to data repositories are used widely and that software can be developed that is *interoperable* with data from many research group's experiments. There are also many advantages, if there are not community-level standards for recording a certain type of data, to develop and use local data standards for recording data from your own experiments. This section describes the elements that go into a data standard, discusses some choices to be made when defininining a data standard (especially choices on data structure and file formats), and some of the advantages and disadvantages of developing and using data recording standards at both the research group and community levels.

> The first of four root causes for irreproducibility in biomedical research: "First, a lack of standards for data generation leads to problems with the comparability and integration of data sets." (Waltemath and Wolkenhauer, 2016)

Ontology standards. Although it has the most complex name, an *ontology* (sometimes called a *terminology* (Sansone et al., 2012)) might be the easiest and quickest to adapt in recording data. An ontology helps define a vocabulary that is controlled and consistent to use that researchers can use to refer to concepts and concrete things within an area of research. It helps researchers, when they want to talk about an idea or thing, to use one word, and just one word, and to ensure that it will be the same word used by other researchers when they refer to that idea or thing. Ontologies also help to define the relationships between ideas or concrete things in a research area (Ghosh et al., 2011), but here we'll focus on their use in provided a consistent vocabulary to use when recording data.

For example, when recording a dataset, what do you call a small mammal that is often kept as a pet and that has four legs and whiskers and purrs? Do you record this as "cat" or "feline" or maybe, depending on the animal, even "tabby" or "tom" or "kitten"? Similarly, do you record tuberculosis as "tuberculosis" or "TB" or or maybe even "consumption"? If you do not use the same word consistently in a dataset to record an idea, then while a human might be able to understand that two words should be considered equivalent, a computer will not be able to immediately tell that "TB" should be treated equivalently to "tuberculosis".

At a larger scale, if a research community can adapt an ontology they agree to use throughout their studies, it will make it easier to understand and inte-

grate datasets produced by different research laboratories. If every research group uses the term "cat", then code can easily be written to extract and combine all data recorded for cats across a large repository of experimental data. On the other hand, if different terms are used, then it might be necessary to first create a list of all terms used in datasets in the respository, then pick through that list to find any terms that are exchangeable with "cat", then write script to pull data with any of those terms.

Several onotologies already exist or are being created for biological and other biomedical research (Ghosh et al., 2011). Existing community-level ontologies in the biological sciences include [Gene Ontology and Systems Biology Ontology are listed in the Ghosh et al. paper as two examples]. For biomedical science, practice, and research, the BioPortal website (`http://bioportal.bioontology.org/`) provides access to almost 800 ontologies, including several versions of the International Classification of Diseases, the Medical Subject Headings (MESH), the National Cancer Institute Thesaurus, the Orphanet Rare Disease Ontology and the National Center for Biotechnology Information (NCBI) Organismal Classification. For each ontology in the BioPortal website, the website provides a link for downloading the ontology in several formats. If you download the ontology as a "CSV", you can open it in your favorite spreadsheet program and explore how it defines specific terms to use for each idea or thing you might need to discuss within that topic area, as well as synonyms for some of the terms. To use an ontology when recording your own data, just make sure you use the ontology's suggested terms in your data. For example, if you'd like to use the Ontology for Biomedical Investigations (`http://bioportal.bioontology.org/ontologies/OBI`) and you are recording how many children a woman has had who were born alive, you should name that column of the data "number of live births", not "# live births" or "live births (N)" or anything else. Other collections of ontologies exist for fields of scientific research, including the Open Biological and Biomedical Ontology (OBO) Foundry (`http://www.obofoundry.org/`).

If there are community-wide ontologies in your field, it is worthwhile to use them in recording experimental data in your research group. Even better is to not only consistently use the defined terms, but also to follow any conventions with capitalization. While most statistical programs provide tools to change capitalization (for example, to change all letters in a character string to lower case), this process does require an extra step of data cleaning and an extra chance for confusion or for errors to be introduced into data.

Minimum information standards. The next easiest facet of a data standard to bring into data recording in a research group is *minimum information*. Within a data recording standard, *minimum information* (sometimes also called *minimum reporting guidelines* (Sansone et al., 2012) or *reporting requirements* (Brazma et al., 2006)) specify *what* should be included in a dataset (Ghosh et al., 2011). Using minimum information standards help ensure that data within a laboratory, or data posted to a repository, contain a number of required elements.

This makes it easier to re-use the data, either to compare it to data that a lab has newly generated, or to combine several posted datasets to aggregate them for a new, integrated analysis, considerations that are growing in importance with the increasing prevalence of research repositories and research consortia in many fields of biomedical science (Keller et al., 2017).

> "Minimum information is a checklist of required supporting information for datasets from different experiments. Examples include: Minimum Information About a Microarray Experiment (MIAME), Minimum Information About a Proteomic Experiment (MIAPE), and the Minimum Information for Biological and Biomedical Investigations (MIBBI) project." (Ghosh et al., 2011)

Standardized file formats. While using a standard ontology and a standard for minimum information is a helpful start, it just means that each dataset has the required elements *somewhere*, and using a consistent vocabulary—it doesn't specify where those elements are in the data or that they'll be in the same place in every dataset that meets those standards. As a result, datasets that all meet a common standard can still be very hard to combine, or to create common data analysis scripts and tools for, since each dataset will require a different process to pull out a given element.

Computer files serve as a way to organize data, whether that's recorded datapoints or written documents or computer programs (Kernighan and Pike, 1984). As the programmer Paul Ford writes,

> "Data is just stuff, or rather, structured stuff: The cells of a spreadsheet, the structure of a Word document, computer programs themselves—all data." (Ford, 2015)

A *file format* defines the rules for how the bytes in the chunk of memory that makes up a certain file should be parsed and interpreted anytime you want to meaningfully access and use the data within that file (Murrell, 2009). There are many file formats you may be familiar with—a file that ends in ".pdf" must be opened with a Portable Document Format (PDF) Reader like Adobe Acrobat, or it won't make much sense (you can try this out by trying to open a ".pdf" file with a text editor, like TextEdit or Notepad). The Reader has been programmed to interpret the data in a ".pdf" file based on rules defining what data is stored where in the section of computer memory for that file. Because most ".pdf" files conform to the same *file format* rules, powerful software can be built that works with any file in that format.

For certain types of biomedical data, the challenge of standardizing a format has similarly been addressed through the use of well-defined rules for not only the content of data, but also the way that content is *structured*. This can be standardized through *standardized file formats* (sometimes also called *data exchange formats* (Brazma et al., 2006)) and often defines not only the upper-level file format (e.g., use of a ".csv" file type), but also how data within that file type should be organized. If data from different research groups and experiments is recorded using the same file format, researchers can develop software tools

that can be repeatedly used to interpret and visualize that data; on the other hand, if different experiments record data using different formats, bespoke analysis scripts must be written for each separate dataset. This is a blow not only to the efficiency of data analysis, but also a threat to the accuracy of that analysis. If a set of tools can be developed that will work over and over, more time can be devoted to refining those tools and testing them for potential errors and bugs, while one-shot scripts often can't be curated with similar care. One paper highlights the dangers that come with working with files that don't follow a defined format:

> "Beware of common pitfalls when working with *ad hoc* bioinformatics formats. Simple mistakes over minor details like file formats can consume a disproportionate amount of time and energy to discover and fix, so mind these details early on." (Buffalo, 2015)

Some biomedical data file formats have been created to help smooth over the transfer of data that's captured by complex equipment into software that can analyze that data. For example, many immunological studies need to measure immune cell populations in experiments, and to do so they use piece of equipment called a flow cytometer that probes cells in a sample with lasers and measures resulting intensities to determine characteristics of that cell. The data created by this equipment is large (often measurements from [x] or more lasers are taken for [x] cells in a single run) and somewhat complex, with a need to record not only the intensity measurements from each laser, but also some metadata about the equipment and characteristics of the run. If every company that manufactured flow cytometers used a different file format for saving the resulting data, then a different set of analysis software would need to be developed to accompany each piece of equipment. For example, a laboratory at a university with flow cytometers from two different companies would need licenses for two different software programs to work with data recorded by flow cytometers, and they would need to learn how to use each software package separately. There is a chance that software could be developed that used shared code for data analysis, but only if it also included separate sets of code to read in data from all types of equipment and to reformat them to a common format.

This isn't the case, however. Instead, there is a commonly agreed on file format that flow cytometers should use to record the data they collect, called the the *FCS file format*. This format has been defined through a series of papers [refs], with several separate versions as the file format has evolved over the years. It provides clear specifications on where to save each relevant piece of information in the block of memory devoted to the data recorded by the flow cytometer (in some cases, leaving a slot in the file blank if no relevant information was collected on that element). As a result, people have been able to create software, both proprietary and open-source, that can be used with any data recorded by a flow cytometer, regardless of which company manufacturer the piece of equipment that was used to generate the data. Other

types of biomedical data also have standardized file formats, including [example popular file formats for biomedical data]. In some cases these were defined by an organization, society, or initiative (e.g., the Metabolomics Standards Initiative) (Ghosh et al., 2011), while in some cases the file format developed by a specific equipment manufacturer has become popular enough that it's established itself as the standard for recording a type of data (Brazma et al., 2006).

For an even simpler example, thing about recording dates. The *minimum information standard* for a date might always be the same—a recorded value must include the day of the month, month, and year. However, this information can be *structured* in a variety of ways. In many scientific data, it's common to record this information going from the largest to smallest units, so March 12, 2006, would be recorded "2006-03-12". Another convention (especially in the US) is to record the month first (e.g., "3/12/06"), while another (more common in Europe) is to record the day of the month first (e.g., "12/3/06").

If you are trying to combine data from different datasets with dates, and all use a different structure, it's easy to see how mistakes could be introduced unless the data is very carefully reformatted. For example, March 12 ("3-12" with month-first, "12-3" with day-first) could be easily mistaken to be December 3, and vice versa. Even if errors are avoided, combining data in different structures will take more time than combining data in the same structure, because of the extra needs for reformatting to get all data in a common structure.

> "Vast swathes of bioscience data remain locked in esoteric formats, are described using nonstandard terminology, lack sufficient contextual information, or simply are never shared due to the perceived cost or futility of the exercise." (Sansone et al., 2012)

2.2.2 Defining data standards for a research group

If some of the data you record from your experiments comes from complex equipment, like flow cytometers [or?], you may be recording much of that data in a standardized format without any extra effort, because that format is the default output format for the equipment. However, you may have more control over other data recorded from your experiments, including smaller, less complex data recorded directly into a laboratory notebook or spreadsheet. You can derive a number of benefits from defining and using a standard for collecting this data, as well.

As already mentioned, for many of the complex types of biological data, standardized file formats exist. For example, flow cytometry data is typically collected and recorded in *.fcs* files. Every piece of flow cytometry equipment can then be built to output data in this format, and every piece of software to analyze flow cytometry data can be built to read in this input. The *.fcs* file format species how both raw data and metadata (e.g., compensation information, equipment details) can be saved within the file—everyone who uses that file format knows where to store data and where to find data of a certain type.

Much of the data collected in a laboratory is smaller, less complex, or less structured than these types of data, data that is recorded "by hand", often into a laboratory notebook or a spreadsheet. One paper describes this type of data as the output of "traditional, low-throughput bench science" (Wilkinson et al., 2016). For this data recording, the data may be written down in an *ad hoc* way—however the particular researcher doing the experiment thinks makes sense—and that format might change with each experiment, even if many experiments have similar data outputs. As a result, it becomes harder to create standardized data processing and analysis scripts that work with this data or that integrate it with more complex data types. Further, if everyone in a laboratory sets up their spreadsheets for data recording in their own way, it is much harder for one person in the group to look at data another person recorded and immediately find what they need within the spreadsheet.

As a step in a better direction, the head of a research group may designate some common formats (e.g., a spreadsheet template) that all researchers in the group should use when recording the data from a specific type of experiments. This provides consistency across the recorded data for the laboratory, making easier for one lab member to quickly understand and navigate data saved by another lab member. It also opens the possibility to create tools or scripts that read in and analyze the data that can be re-used across multiple experiments with minor changes. This helps improve the efficiency and reproducibility of data analysis, visualization, and reporting steps of the research project.

This does require some extra time commitment (Brazma et al., 2006). First, time is needed to design the format, and it does take a while to develop a format that is inclusive enough that it includes a place to put all data you might want to record for a certain type of experiment. Second, it will take some time to teach each laboratory member what the format is and how to make sure they comply with it when they record data.

On the flip side, the longer-term advantages of using a defined, structured format will outweigh the short-term time investments for many laboratory groups for frequently used data types. By creating and using a consistent structure to record data of a certain type, members of a laboratory group can increase their efficiency (since they do not need to re-design a data recording structure repeatedly). They can also make it easier for downstream collaborators, like biostatisticians and bioinformaticians, to work with their output, as those collaborators can create tools and scripts that can be recycled across experiments and research projects if they know the data will always come to them in the same format. These benefits increase even more if data format standards are created and used by a whole research field (e.g., if a standard data recording format is always used for researchers conducting a certain type of drug development experiment), because then the tools built at one institution can be used at other insitutions. However, this level of field-wide coordination can be hard to achieve, and so a more realistic immediate goal might be formalizing data recording structures within your research group or department,

while keeping an eye out for formats that are gaining popularity as standards in your field to adopt within your group.

One key advantage to using standardized data formats even for recording simple, "low-throughput" data is that everyone in the research group will be able to understand and work with data recorded by anyone else in the group—data will not become impenetrable once the person who recorded it leaves the group. Also, once a group member is used to the format, the process of setting up to record data from a new experiment will be quicker, as it won't require the effort of deciding and setting up a *de novo* format for a spreadsheet or other recording file. Instead, a template file can be created that can be copied as a starting point for any new data recording.

Finally, there are huge benefits further down the data analysis pipeline that come with always recording data in the same format. If your group is working with a statistician or data analyst, it becomes much easier for that person to quickly understand a new file if it follows the same format as previous files. Further, if you work with a statistician or data analyst, he or she probably creates code scripts to read in, re-format, analyze, and visualize the data you've shared. If you always record data using the same format, these scripts can be reused with very little modification. This saves valuable time, and it helps make more time for more interesting statistical analysis if your collaborator can trim time off reading in and reformatting the data in their statistical programming language.

One paper suggests that the balance can be found, in terms of deciding whether the benefits of developing a standard outweigh the costs, by considering how often data of a certain type is generated and used:

> "To develop and deploy a standard creates an overhead, which can be expensive. Standards will help only if a particular type of information has to be exchanged often enough to pay off the development, implementation, and usage of the standard during its lifespan." (Brazma et al., 2006)

2.2.3 Two-dimensional structured data format

So far, this module has explored *why* you might want to use standardized data formats for recording experimental data. The rest of the module aims to give you tips for how to design and define your own standardized data formats, if you decide that is worthwhile for certain data types recorded within your research group.

Once you commit to creating a defined, structured format, you'll need to decide what that structure should be. There are many options here, and it's very tempting to use a format that is easy on human eyes (Buffalo, 2015). For example, it may seem appealing to create a format that could easily be copied and pasted into presentations and Word documents and that will look nice in those presentation formats. To facilitate this use, a laboratory might set up a recording format base on a spreadsheet template that includes multiple tables of different data types on the same sheet, or multi-level column headings.

Unfortunately, many of the characteristics that make a format attractive to human eyes will make it harder for a computer to make sense of. For example, if you include two tables in the same spreadsheet, it might make it easier for a person to get a one-screen look at two small data tables. However, if you want to read that data into a statistical program (or work with a collaborator who would), it will likely take some complex code to try to tell the computer how to find the second table in the spreadsheet. The same applies if you include some blank lines at the top of the spreadsheet, or use multi-level headers, or use "summary" rows at the bottom of a table. Further, any information you've included with colors or with text boxes in the spreadsheet will be lost when the data's read into a statistical program. These design elements in a data format make it much harder to read the data embedded in a spreadsheet into other computer programs, including programs for more complex data analysis and visualization, like R and Python.

> "Data should be formatted in a way that facilitates computer readability. All too often, we as humans record data in a way that maximizes its readability to us, but takes a considerable amount of cleaning and tidying before it can be processed by a computer. The more data (and metadata) that is computer readable, the more we can leverage our computers to work with this data." (Buffalo, 2015)

For most statistical programs, data can be easily read in from a spreadsheet if the computer can parse it in the following way: first, read in the first row, and assign each cell in that row as the *name* of a column. Then, read in the second row, and put each cell in the column the corresponds with the name of the cell in the same position in the first row. Also, set the data type for that column (e.g., number, character) based on the data type in this cell. Then, keep reading in rows until getting to a row that's completely blank, and that will be the end of the data. If any of the rows has more cell than the first row, then that means that something went wrong, and should result in stopping or giving a warning. If any of the rows have fewer cells than the first row, then that means that there are missing data in that row, and should probably be recorded as missing values for any cells the row is "short" compared to the first row.

One of the easiest format for a computer to read is therefore a two-dimensional "box" of data, where the first row of the spreadsheet gives the column names, and where each row contains an equal number of entries. This type of two-dimensional tabular structure forms the basis for several popular "delimited" file formats that serve as a *lingua franca* across many simple computer programs, like the comma-separated values (CSV) format, the tab-delimited values (TSV) format, and the more general delimiter-separated values (DSV) format, which are a common format for data exchange across databases, spreadsheet programs, and statistical programs (Janssens, 2014; Raymond, 2003; Buffalo, 2015).

> "Tabular plain-text data formats are used extensively in computing. The basic format is incrediably simple: each row (also known as a record) is kept on its

> own line, and each column (also known as a field) is separate by some delimiter." (Buffalo, 2015)

If you think of the computer parsing a spreadsheet as described above, hopefully it clarifies why some spreadsheet formats would cause problems. For example, if you have two tables in the same spreadsheet, with blank lines between them, the computer will likely either think it's read all the data after the first table, and so not read in any data from the second table, or it will think the data from both tables belong in a single table, with some rows of missing data in the center. To write the code to read in data from two tables into two separate datasets in a statistical program, it will be necessary to write some complex code to tell the computer how to search out the start of the second table in the spreadsheet.

Similar problems come up if a spreadsheet diverges from a regular, two-dimensional format, with a single row of column names to start the data. For example, if the data uses multiple rows to create multi-level column headers, anyone reading it into another program will need to either skip some of the rows of the column headers, and so lose information in the original spreadsheet, or write complex code to parse the column headers separately, then read in the later rows with data, and then stick the two elements back together. "Summary" rows at the end of a dataset (for example, the sums or means of all values in a column) will need to be trimmed off when the data is read into other programs, since most of the analysis and visualization someone would want to do in another program will calculate any summaries fresh, and will want each row of a dataset to represent the same "type" and level of data (e.g., one measurement from one animal).

For anything in a data format that requires extra coding when reading data into another program, you are introducing a new opportunity for errors at the interface between data recording and data analysis. If there are strong reasons to use a format that requires these extra steps, it will still be possible to create code to read in and parse the data in statistical programs, and if the same format is consistently used, then scripts can be developed and thoroughly tested to allow this. However, do keep in mind that this will be an extra burden on any data analysis collaborators who are using a program besides a spreadsheet program. The extra time this will require could be large, since this code should be vetted and tested thoroughly to ensure that the data cleaning process is not introducing errors. By contrast, if the data is recorded in a two-dimensional format with a single row of column names as the first row, data analysts can likely read it quickly and cleanly into other programs, with low risks of errors in the transfer of data from the spreadsheet.

> "Cleaning data is a short-term solution, and preventing errors is promoted as a permanent solution. The drawback to cleaning data is that the process never ends, is costly, and may allow many errors to avoid detection." (Keller et al., 2017)

2.2.4 Saving two-dimensional structured data in plain text file formats

If you have recorded data in a two-dimensional structured format, you can choose to save it in either a *plain text* format or a *binary* format. With a plain text format, a file is "human readable" when it's opened in a text editor (Hunt et al., 2000; Janssens, 2014), because each byte that encodes the file translates to a single character (Murrell, 2009), usually using an ASCII or Unicode encoding. Common plain text file formats used for biomedical research include CSV and TSV files (these are distinguished only by the character used as a delimiter—commas for CSV files versus taabs for TSV files) (Buffalo, 2015), other more complex file formats like SAM and XML are also typically saved in plain text.

A binary file format, on the other hand, encodes data within the file using an encoding system that differs from ANSCII or Unicode. To extract the data in a meaningful way, a computer program must know and use rules for the encoding and structure of that file format, and those rules will be different for each different binary file format (Murrell, 2009). Some binary file formats are "open", with all the information on these rules and encodings available for anyone to read. On the otherhand, other binary file formats are proprietary, without available guidance on how to interpret or use the data stored in them when creating new software tools. Binary files, because they don't follow the restrictions of plain text encoding and format, can encode and organize data in a way that's often much more compressed, because it's optimized to suit a specific type of data. This means that binary file formats can often store more data within a certain amount of computer memory compared to plain text file formats. Binary files can also be designed so that the computer can find and read a specific piece of data, rather than needing to read data in linearly from the start to the end of a file as with plain text formats. This means that programs can often access specific bits of data much more quickly from a binary file format that from a plain text format, making computation processing run much faster.

However, even with the speed and size advantages of many binary file formats, it is often worthwhile to record and save experimental data in a plain text, rather than binary, file format. There are a number of advantages to using a plain text format. A plain text format may take more space (in terms of computer memory) and take longer to process within other programs; however, its benefits typically outweigh these limitations (Hunt et al., 2000). Advantages include: (1) humans can read the file directly (Hunt et al., 2000; Janssens, 2014), and should always be able to, regardless of changes in and future obsolescence of computer programs; (2) almost all software programs for analyzing and processing files can input plain-text files, while binary file formats often require specialized software (Murrell, 2009); (3) the Unix system, which has influenced many existing software programs, especially open-source programs for data analysis and command-line tools, are based on inputting and outputtin

line-based plain-text files (Janssens, 2014); and (4) plain-text files can be easily tracked with version control (Hunt et al., 2000). These advantages might become particularly important in cases where researchers need to combine and integrate heterogeneous data, for example data coming from different instruments.

Another advantage of storing data in a plain text format is that it makes version control, which we'll discuss in a later module, a much more powerful tool. With plain text files, you can use version control to see the specific changes to a file. With binary files, you can typically see if a file was changed, but it's much harder to see exactly what within the file was changed.

The book *The Pragmatic Programmer* highlights some of the advantages of plain text:

> "Human-readable forms of data, and self-describing data, will outlive all other forms of data and the applications that created them. Period. As long as the data survives, you will have a chance to be able to use it—potentially long after the original application that wrote it is defunct. … Even in the future of XML-based intelligent agents that travel the wild and dangerous Internet autonomously, negotiating data interchange among themselves, the ubiquitous text file will still be there. In fact, in heterogeneous environments the advantages of plain text can outweight all of the drawbacks. You need to ensure that all parties can communicate using a common standard. Plain text is that standard." (Hunt et al., 2000)

Paul Ford, by contrast, describes some of the disadvantages of a binary file format, using the Photoshop file format as an example:

> "A Photoshop file is a lump of binary data. It just sits there on your hard drive. Open it in Photoshop, and there are your guides, your color swatches, and of course, the manifold pixels of your intent. But outside of Photoshop that file is an enigma. There is not 'view source'. You can, if you're passionate, read the standard on the web, and it's all piled in there, the history of pictures on computers. That's when it becomes clear: only Photoshop's creator Adobe can understand this thing." (Ford, 2014)

Structuring data in a gridded, two-dimensional format, as described in the last section, will be helpful even if it is in a file format that is binary, like Excel. However, there are added benefits to saving the structured data in a plain text format. Older Excel spreadsheets are typically saved in a proprietary file format (".xls"), while more recently Excel has saved files to an open binary format based on packaging XML files with the data (".xlsx" file format) (Janssens, 2014). While the open proprietary format is preferable, since tools can be developed to work with them by people other than the Microsoft team, both file formats still face some of the limitations of binary file formats as a way of recording experimental data. However, even if you have used a spreadsheet program like Excel to record data, it's very easy to still save that data in a plain text file format (Murrell, 2009). In most spreadsheet programs, you can choose to save a file "As CSV".

2.2.5 Occassions for more complex data structures and file formats

There are some cases where a two-dimensional data format may not be adequate for recording experimental data, despite this format's advantages in improving reproducibility through later data analysis steps. Similarly, there may be cases where a binary file format, or use of a database, will outweigh the benefits of saving data to a plain text format. Being familiar with different file formats can also be helpful when you need to integrate data stored in different formats (Murrell, 2009).

Non-tabular plain-text formats. First, some data has a linked or hierarchical nature, in terms of how data points are connected through the dataset. For example, data on a family try might have a hierarchical structure, where different numbers of children are recorded for each parent. As another example, if you were building a dataset describing how scientists have collaborated together as coauthors, that data might form a network. In many cases, it is possible to structure datasets with these types of "non-tabular" structure using the "tidy data" tabular format described in the next section. However, in very complex cases, it may work better to use a non-tabular data format (Raymond, 2003). Popular data formats that are non-tabular include the eXtensible Markup Language (XML) and JavaScript Object Notation (JSON) formats, both of which are well-suited for hierarchically-structured data. You may also have data you would like to use in XML or JSON formats if you are using web services to pull datasets from online repositories, as open data application programming interfaces (APIs) often return data in these formats (Janssens, 2014).

Another use of file formats that are plain text but meant to be streamed, rather than read in as a whole. When reading in data stored in a delimited plain text file, like a CSV file, a statistical program like R will typically read in all the data and then operate on the dataset as a whole. If a data file is very large, then reading in all the data at once might require so much memory that it slows down processing, or even exceed the program's memory cap [?]. One strategy is to design a data format so that the program can read in a small amount of the file, process that piece of the data, write the result out, and remove that bit of data from the program's memory before moving into the next portion of data (Buffalo, 2015). This *streaming* approach is sometimes used with some file formats used for biomedical research, including FASTA and FASTQ files.

Databases. When research datasets include not only data that can be expressed in plain text, but also data like images, photographs, or videos, it may be worth considering using a database to store the data (Murrell, 2009). Relational database managment system software, like [examples. MySQL? PostgreSQL?] can be used to organize data in a way that records connections (*relations*) between different pieces of data and allows you to access different combinations of that data quickly using Structured Query Language, or *SQL* (Ford, 2015). Further, some statistical programming languages, including R, now have tools that allow you to directly access and work with data from a database

from within the statistical program, and in some cases using scripts that are very similar or identical to the code that would be used if you'd read the data into the program from a plain text file.

> "The database is the unsung infrastructure of the world, the shared memory of every corporation, and the foundation of every major web site. And they are everywhere. Nearly every host-your-own-web-site package comes with access to a database called MySQL; just about every cell phone has SQLite3, aa tiny, pocket-sized database, built in." (Ford, 2015)

It will be more complicated to set up a database for recording experimental data, and so it's often preferable to instead save data in plain text files within a file directory, if the data is simple enough to allow that. However, there are some fairly simple database solutions that are now available, including SQLite (Buffalo, 2015).

Binary file formats.

There are cases where it may not be best to store laboratory-generated data in a plain text format. For example, the output from a flow cytometer is large and would take up a lot (more) computer memory if stored in a plain text format, and it would take much longer to read and work with the data in analysis software if it were in that format. For very large datasets like this, it may be necessary to use a binary data format, either for size or speed or both (Kernighan and Pike, 1984; Hunt et al., 2000). For very large biomedical datasets, binary file formats are sometimes designed for *out-of-memory approaches* (Buffalo, 2015), where a file format is designed in a way that allows computer programs to find and read only specific pieces of data in a file through a process called *random access*, rather than needing to read the full file into memory before a specific piece of data in the file can be accessed (a.k.a., *sequential access*) (Murrell, 2009).

2.2.6 Levels of standardization—research group to research community

Standards can operate both at the level of individual research groups and at the level of the scientific community as a whole. The potential advantages of community-level standards are big: they offer the chance to develop common-purpose tools and code scripts for data analysis, as well as make it easier to re-use and combine experimental data from previous research that is posted in open data repositories. If a software tool can be reused, then more time can be spent in developing and testing it, and as more people use it, bugs and shortcomings can be identified and corrected. Community-wide standards can lead to databases with data from different experiments, and from different laboratory groups, structured in a way that makes it easy for other researchers to understand each dataset, find pieces of data of interest within datasets, and integrate different datasets (Lynch, 2008). Similarly, with community-wide standards, it can become much easier for different research groups to collaborate with each other or for a research group to use data generated by

equipment from different manufacturers (Schadt et al., 2010).

> "Without community-level harmonization and interoperability, many community projects risk becoming data silos." (Sansone et al., 2012)

> "Solutions to integrating the new generation of large-scale data sets require approaches akin to those used in physics, climatology and other quantitative disciplines that have mastered the collection of large data sets." (Schadt et al., 2010)

However, there are important limitations to community-wide standards, as well. It can be very difficult to impose such standards top-down and community-wide, particularly for low-throughput data collection (e.g., laboratory bench measurements), where research groups have long been in the habit of recording data in spreadsheets in a format defined by individual researchers or research groups. One paper highlights this point:

> "The data exchange formats PSI-MI and MAGE-ML have helped to get many of the high-throughput data sets into the public domain. Nevertheless, from a bench biologist's point of view benefits from adopting standards are not yet overwhelming. Most standardization efforts are still mainly an investment for biologists." (Brazma et al., 2006)

Further, in some fields, community-wide standards have struggled to remain stable, which can frustrate community members, as scripts and software must be revamped to handle shifting formats (Buffalo, 2015; Barga et al., 2011). In some cases, a useful compromise is to follow a general data recording format, rather than one that is very prescriptive. For example, committing to recording data in a format that is "tidy" (which we discuss extensively in the next module) may be much more flexible—and able to meet the needs of a large range of experimental designs—than the use of a common spreadsheet template or a more proscriptive standardized data format.

2.2.7 Applied exercise

2.3 The 'tidy' data format

The "tidy" data format is an implementation of a structured data format popular among statisticians and data scientists. By consistently using this data format, researchers can combine simple, generalizable tools to perform complex tasks in data processing, analysis, and visualization. We will explain what characteristics determine if a dataset is "tidy" and how use of the "tidy" implementation of a structure data format can improve the ease and efficiency of "Team Science".

Objectives. After this module, the trainee will be able to:

- List characteristics defining the "tidy" structured data format
- Explain the difference between the a structured data format (general concept) and the 'tidy' data format (one popular implementation)

In the previous module, we explained the benefits of saving data in a structured format, and in particular using a two-dimensional format saved to a plain text file when possible. In this section, we'll talk about the "tidy text" format—a set of principles to use when structuring two-dimensional tabular data. These principles cover some basic rules for ordering the data, and the resulting datasets can be very easily worked with, including to further clean, model, and visualize the data, and to integrate the data with other datasets, using a series of open-source tools on the R platform called the "tidyverse". These characteristics mean that, if you are planning to use a standardized data format for recording experimental data in your research group, you may want to consider creating on that adheres to the "tidy data" rules.

Since a key advantage of the "tidy data" format is that it works so well with R's "tidyverse" tools, we'll also talk a bit in this section about the use of scripting languages like R, and how using them to analyze and visualize the data you collect can improve the overall reproducibility of your research.

We'll next describe what rules a dataset's format must follow for it to be "tidy", and try to clarify how you can set up your data recording to follow these rules. In a later part of this module, we'll talk more about the tidyverse tools that you can use with this data, as well as give some resources for finding out more about the tidyverse and how to use its tools.

2.3.1 What makes data "tidy"?

The "tidy" data format describes one way to structure tabular data. The name follows from the focus of this data format and its associated set of tools—the "tidyverse"—on preparing and cleaning ("tidying") data, in contrast to sets of tools more focused on other steps, like data analysis [Tidy data]. The word "tidy" is not meant to apply that other formats are "dirty", or that they include data that is incorrect or subpar. In fact, the same set of datapoints could be saved in a file in a way that is either "tidy" (in the sense of [Tidy data]) or untidy, depending only on how the data are organized across columns and rows.

> "The development of tidy data has been driven by my experience from working with real-world datasets. With few, if any, constraints on their organization, such datasets are often constructed in bizarre ways. I have spent countless hours struggling to get such datasets organized in a way that makes data analysis possible, let alone easy." [Tidy data]

The rules for making data "tidy" are pretty simple, and they are defined in detail, and with extended examples, in the journal article that originally defined the data format [Tidy data]. Here, we'll go through those rules, with the hope that you'll be able to understand what makes a dataset follow the "tidy" data format. If so, you'll be able to set up your data recording template to follow this template, and you'll be able to tell if other data you get is in this format and, if it's not, restructure it so that it does. In the next part of this module, we'll

explain why it's so useful to have your data in this format.

Tidy data, first, must be in a tabular (i.e., two-dimensional, with columns and rows, and with all rows and columns of the same length—nothing "ragged"). If you recorded data in a spreadsheet using a very basic strategy of saving a single table per spreadsheet, with the first row giving the column names (as described in the previous module), then your data will be in a tabular format. It should not be saved in a hierarchical structure, like XML (although there are now tools for converting data from XML to a "tidy" format, so you may still be able to take advantage of the tidyverse even if you must use XML for your data recording). In general, if your recorded data looks "boxy", it's probably in a two-dimensional tabular format.

There are some additional criteria for the "tidy" data format, though, and so not every structured, tabular dataset is in a "tidy" format. The first of these rules are that each row of a "tidy" dataset records the values for a single observation, and that each column provides characteristics or measurements of a certain type, in the order of the observations given by the rows [Tidy data].

> "Most statistical datasets are rectangular tables made up of rows and columns … [but] there are many ways to structure the same underlying data. … Real datasets can, and often do, violate the three precepts of tidy data in almost every way imaginable." [Tidy data]

To be able to decide if your data is tidy, then, you need to know what forms a single observation in data you're collecting. The *unit of observation* of a dataset is the unit at which you take measurements [Sedgewick 2014 Unit]. This idea is different than the *unit of analysis*, which is the unit that you're focusing on in your study hypotheses and conclusions (this is sometimes also called the "sampling unit" or "unit of investigation") [Altman and Bland]. In some cases, these two might be equivalent (the same unit is both the unit of observation and the unit of measurement), but often they are not [Sedgewick 2014 Unit].

> "The unit of observation and unit of analysis are often confused. The unit of observation, sometimes referred to as the unit of measurement, is defined statistically as the 'who' or 'what' for which data are measured or collected. The unit of analysis is defined statistically as the 'who' or 'what' for which information is analysed and conclusions are made." [Sedgewick 2014 Unit]

For example, say you are testing how the immune system of mice responds to a certain drug over time. You may have several replicates of mice measured at several time points, and those mice might be in separate groups (for example, infected with a disease versus uninfected). In this case, if a separate mouse (replicate) is used to collect each observation, and a mouse is never measured twice (i.e., at different time points, or for a different infection status), then the unit of measurement is the mouse. There should be one and only one row in your dataset for each mouse, and that row should include two types of information: first, information about the unit being measured (e.g., the time point, whether the mouse was infected, and a unique mouse identifier) and,

second, the results of that measurement (e.g., the weight of the mouse when it was sacrificed, the levels of different immune cell populations in the mouse, a measure of the extent of infection in the mouse if it was infected, and perhaps some notes about anything of note for the mouse, like if it appeared noticably sick). In this case, the *unit of analysis* might be the drug, or a combination of drug and dose—ultimately, you may want to test something like if one drug is more effective than another. However, the *unit of observation*, the level at which each data point is collected, is the mouse, with each mouse providing a single observation to help answer the larger research question.

As another example, say you conducted a trial on human subjects, to see how the use of a certain treatment affects the speed of recovery, where each study subject was measured at different time points. In this case, the unit of observation is the combination of study subject and time point (while the unit of analysis is the study subject, if the treatments are randomized to the study subjects). That means that Subject 1's measurement at Time 1 would be one observation, and the same person's measurement at Time 2 would be a separate observation. For a dataset to comply with the "tidy" data format, these two observations would need to be recorded on separate lines in the data. If the data instead had different columns to record each study subject's measurements at different time points, then the data would still be tabular, but it would not be "tidy".

In this second example, you may initially find the "tidy" format unappealing, because it seems like it would lead to a lot of repeated data. For example, if you wanted to record each study subject's sex, it seems like the "tidy" format would require you to repeat that information in each separate line of data that's used to record the measurements for that subject for different time points. This isn't the case—instead, with a "tidy" data format, different "levels" of data observations should be recorded in separate tables [Tidy data]. So, if you have some data on each study subject that does not change across the time points of the study—like the subject's ID, sex, and age at enrollment—those form a separate dataset, one where the unit of observation is the study subject, so there should be just one row of data per study subject in that data table, while the measurements for each time point should be recorded in a separate data table. A unique identifier, like a subject ID, should be recorded in each data table so it can be used to link the data in the two tables. If you are using a spreadsheet to record data, this would mean that the data for these separate levels of observation should be recorded in separate sheets, and not on the same sheet of a spreadsheet file. Once you read the data into a scripting language like R or Python, it will be easy to link the larger and smaller "tidy" datasets as needed for analysis, visualizations, and reports.

Once you have divided your data into separate datasets based on the level of observation, and structured each row to record data for a single observation based on the unit of observation within that dataset, each column should be used to measure a separate characteristic or measurement (a *variable*) for each

measurment [Tidy data]. A column could either give characteristics of the data that were pre-defined by the study design—for example, the treatment assigned to a mouse, or the time point at which a measurement was taken if the study design defined the time points when measurements would be taken. These types of column values are also sometimes called *fixed variables* [Tidy data]. Other columns will record observed measurements—values that were not set prior to the experiment. These might include values like the level of infection measured in an animal and are sometimes called *measured variables* [Tidy data].

> "While the order of variables and observations does not affect analysis, a good ordering makes it easier to scan the raw values. One way of organizing variables is by their role in the analysis: are values fixed by the design of the data collection, or are they measured during the course of the experiment? Fixed variables describe the experimental design and are known in advance. … Measured variables are what we actually measure in the study. Fixed variables should come first, followed by measured variables, each ordered so that related variables are contiguous. Rows can then be ordered by the first variable, breaking ties with the second and subsequent (fixed) variables." [Tidy data]

2.3.2 Why make your data "tidy"?

This may all seem like a lot of extra work, to make a dataset "tidy", and why bother if you already have it in a structured, tabular format? It turns out that, once you get the hang of what gives data a "tidy" format, it's pretty simple to design recording formats that comply with these rules. What's more, when data is in a "tidy" format, it can be directly input into a collection of tools in R that belong to something called the "tidyverse". This collection of tools is very straightforward to use and so powerful that it's well worth making an effort to record data in a format that works directly with the tools, if possible. Outside of cases of very complex or very large data, it should be possible.

> "A standard makes initial data cleaning easier because you do not need to start from scratch and reinvent the wheel every time. The tidy data standard has been designed to facilitate initial exploration and analysis of the data, and to simplify the development of data analysis tools that work well together." [Tidy data]

> "Tidy data is great for a huge fraction of data analyses you might be interested in. It makes organizing, developing, and sharing data a lot easier. It's how I recommend most people share data." [Toward tidy analysis]

The "tidyverse" is a collection of tools united by a common philosophy: **very complex things can be done simply and efficiently with small, sharp tools that share a common interface**.

> "The philosophy of the tidyverse is similar to and inspired by the "unix philosophy", a set of loose principles that ensure most command line tools play well together. … Each function should solve one small and well-defined class of problems. To solve more complex problems, you combine simple pieces in a standard way." [Declutter]

The tidyverse isn't the only popular system that follows this philosophy—one other favorite is Legos. Legos are small, plastic bricks, with small studs on top and tubes for the studs to fit into on the bottom. The studs all have the same, standardized size and are all spaced the same distance apart. Therefore, the bricks can be joined together in any combination, since each brick uses the same *input format* (studs of the standard size and spaced at the standard distance fit into the tubes on the bottom of the brick) and the same *output format* (again, studs of the standard size and spaced at the standard distance at the top of the brick).

This is true if you want to build with bricks of different colors or different heights or different widths or depths. It even allows you to include bricks at certain spots that either don't require input (for example, a solid sheet that serves as the base) or that don't give output (for example, the round smooth bricks with painted "eyes" that are used to create different creatures). With Legos, even though each "tool" (brick) is very simple, the tools can be combined in infinite variations to create very complex structures.

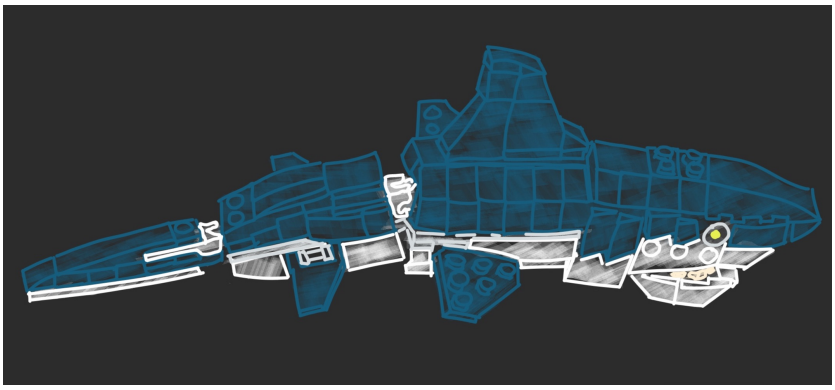

Figure 2.1: When simple tools have a common interface that allows them to be combined in different ways, like Legos, they can be combined to create complex structures, like this Lego shark.

The tools in the "tidyverse" operate on a similar principle. They all input one of a few very straightforward data types, and they (almost) all output data in the same format they input it. For most of the tools, their required format for input and output is the "tidy data" format [Tidy data], called a tidy *dataframe* in R—this is a dataframe that follows the rules detailed earlier in this section.

Some of the tools require input and output of *vectors* instead of tidy dataframes [Tidy data]; a *vector* in R is a one-dimensional string of values, all of which are of the same data type (e.g., all numbers, or all character strings, like names). In a tidy dataframe, each column is a vector, and the dataframe is essentially several vectors of the same length stuck together to make a table. Having functions that input and output vectors, then, means that you can use those functions to make changes to the columns in a tidy dataframe.

A few functions in the "tidyverse" input a tidy dataframe but output data in a different format. For example, visualizations are created using a function called `ggplot`, as well as its helper functions and extensions. This function inputs data

in a tidy dataframe but outputs it in a type of R object called a "ggplot object". This object encodes the plot the code created, so in this case the fact that the output is in a different format from the endpoint is similar to with the "eye" blocks in Legos, where it's meant as a final output step, and you don't intend to do anything further in the code once you move into that step.

This common input / output interface, and the use of small tools that follow this interface and can be combined in various ways, is what makes the tidyverse tools so powerful. However, there are other good things about the tidyverse that make it so popular. One is that it's fairly easy to learn to use the tools, in comparison to learning how to write code for other R tools [Teach the tidyverse; Teaching R to new users]. The developers who have created the tidyverse tools have taken a lot of effort to try to make sure that they have a clear and consistent *user interface* across tools [Tidy tools manifesto; Three ring circus]. So far, we've talked about the interface between functions, and how a common *input / output interface* means the functions can be chained together more easily. But there's another interface that's important for software tools: the rules for how a computer users employ that tool, or the *user interface*.

To help understand a user interface, and how having a consistent user interface across tools is useful, let's think about a different example—cars. When you drive a car, you get the car to do what you want through the steering wheel, the gas pedal, the break pedal, and different knobs and buttons on the dashboard. When the car needs to give you feedback, it uses different gauges on the dashboard, like the speedometer, as well as warning lights and sounds. Collectively, these ways of interacting with your car make up the car's *user interface*. In the same way, each function in a programming language has a collection of parameters you can set, which let you customize the way the function runs, as well as a way of providing you output once the function has finished running and the way to provide any messages or warnings about the function's run. For functions, the software developer can usually choose design elements for the function's user interface, including which parameters to include for the function, what to name those parameters, and how to provide feedback to the user through messages, warnings, and the final output.

If a collection of tools is similar in its user interfaces, it will make it easier for users to learn and use any of the tools in that collection once they've learned how to use one. For cars, this explains how the rental car business is able to succeed. Even though different car models are very different in many characteristics—their engines, their colors, their software—they are very consistent in their user interfaces. Once you've learned how to drive one car, when you get in a new car, the gas pedal, brake, and steering wheel are almost guaranteed to be in about the same place and to operate about the same way as in the car you learned to drive in. The exceptions are rare enough to be memorable—think how many movies have a laughline from a character trying to drive a car with the driver side on the right if they're used to the left or vice versa.

The tidyverse tools are similarly designed so that they all have a very similar user interface. For example, many of the tidyverse functions use a parameter named ".data" to refer to the tidy dataframe to input into the function, and this parameter is often the first listed for functions. Similarly, parameters named ".vars" and ".funs" are repeatedly used over tidyverse functions, with the same meaning in each case. What's more, the tidyverse functions are typically given names that very clearly describe the action that the function does, like `filter`, `summarize`, `mutate`, and `group`. As a result, the final code is very clear and can almost be "read" as a natural language, rather than code.

> "Another part of what makes the Tidyverse effective is harder to see and, indeed, the goal is for it to become invisible: conventions. The Tidyverse philosophy is to rigorously (and ruthlessly) identify and obey common conventions. This applies to the objects passed from one function to another and to the user interface each function presents. Taken in isolation, each instance of this seems small and unimportant. But collectively, it creates a cohesive system: having learned one component you are more likely to be able to guess how another different component works." [Three ring circus]

> "The goal of [the tidy tools] principles is to provide a uniform interface so that tidyverse packages work together naturally, and once you've mastered one, you have a head start on mastering the others." [Tidy tools manifesto]

As a result, the tidyverse collection of tools is pretty easy to learn, compared to other sets of functions in scripting languages, and pretty easy to expand your knowledge of once you know some of its functions. Several people who teach R programming now focus on first teaching the tidyverse, given these characteristics [Teach the tidyverse; Teaching R to New Users], and it's often a first focus for online courses and workshops on R programming. Since it's main data structure is the "tidy data" structure, it's often well worth recording data in this format so that all these tools can easily be used to explore and model the data.

> "All our code is underpinned by the principles of tidy data, the grammar of data manipulation, and the tidyverse R packages developed by Wickham. This deliberate philosophy for thinking about data helped bridge our scientific questions with the data processing required to get there, and the readability and conciseness of tidyverse operations makes our data analysis read more as a story arc. Operations require less syntax—which can mean fewer potential errors that are easier to identify—and they can be chained together, minimizing intermediate steps and data objects that can cause clutter and confusion. The tidyverse tools for wrangling data have expedited our transformation as coders and made R less intimidating to learn." [Our path to better science]

2.3.3 Using tidyverse tools with data in the "tidy data" format

The tidyverse includes tools for many of the tasks you might need to do while managing and working with experimental data. When you download R, you get what's called *base R*. This includes the main code that drives anything you

do in R, as well as functions for doing many core tasks. However, the power of R is that, in addition to base R, you can also add onto R through what are called *packages* (sometimes also referred to as *extensions* or *libraries*). These are kind of like "booster packs" that add on new functions for R. They can be created and contributed by anyone, and many are collected through a few key repositories like CRAN and Bioconductor.

All the tidyverse tools are included in R extension packages, rather than base R, so once you download R, you'll need to download these packages as well to use the tidyverse tools. The core tidyverse functions include functions to read in data (the `readr` package for reading in plain text, delimited files, `readxl` to read in data from Excel spreadsheets), clean or summarize the data (the `dplyr` package, which includes functions to merge different datasets, make new columns as functions of old ones, and summarize columns in the data, either as a whole or by group), and reformat the data if needed to get it in a tidy format (the `tidyr` package). The tidyverse also includes more precise tools, including tools to parse dates and times (`lubridate`) and tools to work with character strings, including using regular expressions as a powerful way to find and use certain patterns in strings (`stringr`). Finally, the tidyverse includes powerful functions for visualizing data, based around the `ggplot2` package, which implements a "grammar of graphics" within R.

You can install and load any of these tidyverse packages one-by-one using the `install.packages` and `library` functions with the package name from within R. If you are planning on using many of the tidyverse packages, you can also install and load many of the tidyverse functions by installing a package called "tidyverse", which serves as an umbrella for many of the tidyverse packages.

In addition to the original tools in the tidyverse, many people have developed *tidyverse* extensions—R packages that build off the tools and principles in the tidyverse. These often bring the tidyverse conventions into tools for specific areas of science. For example, the `tidytext` package provides tools to analyze large datasets of text, including books or collections of tweets, using the tidy data format and tidyverse-style tools. Similar tidyverse extensions exist for working with network data (`tidygraph`) or geospatial data (`sf`). Extensions also exist for the visualization branch of the tidyverse specifically. These include *ggplot extensions* that allow users to create things like calendar plots ([package]), gene arrow maps (`gggene`), network plots (`igraph`), phytogenetic trees (`ggtree`) and anatogram images (`gganatogram`). These extensions all allow users to work with data that's in a "tidy data" format, and they all provide similar user interfaces, making it easier to learn a large set of tools to do a range of data analysis and visualization, compared to if the set of tools lacked this coherence.

2.3.4 Practice quiz

2.4 Designing templates for "tidy" data collection

This module will move from the principles of the "tidy" data format to the practical details of designing a "tidy" data format to use when collecting experimental data. We will describe common issues that prevent biomedical research datasets from being "tidy" and show how these issues can be avoided. We will also provide rubrics and a checklist to help determine if a data collection template complies with a "tidy" format.

Objectives. After this module, the trainee will be able to:

- Identify characteristics that keep a dataset from being 'tidy'
- Convert data from an "untidy" to a "tidy" format

It is usually very little work to record data in a structure that follows the "tidy data" principles, especially if you are planning to record the data in a two-dimensional, tabular format already, and following these principles can bring some big advantages. We explain these rules and provide examples of biomedical datasets that both comply and don't comply with these principles, to help make it clearer how you could structure a "tidy-compliant" structure for recording experimental data for your own research.

If the data is the same regardless of whether it's "tidy" or not, then why all the fuss about following the "tidy" principles when you're designing the format you'll use to record your data? The magic here ix this—if you follow these principles, then your data can be immediately input into a collection of powerful tools for visualizing and analyzing the data, without further cleaning steps (as discussed in the previous module). What's more, all those tools (the set of tools is calld the "tidyverse") will typically *output* your data in a "tidy" format, as well.

Once you have tools that input and output data in the same way, it becomes very easy to model each of the tools as "small, sharp tools"—each one does one thing, and does it really well. That's because, if each tool needs the same type of input and creates that same type of output, those tools can be chained together to solve complex problems. The alternative is to create large software tools, ones that do a lot to the input data before giving you some output. "Big" tools are harder to understand, and more importantly, they make it hard to adapt your own solutions, and to go beyond the analysis or visualization that the original tool creators were thinking of when they created it. Think of it this way—if you were writing an essay, how much more can you say when you can mix and match words to create your own sentences versus if you were made to combine pre-set sentences?

2.4.1 Subsection 1

"Or maybe your goal is that your data is *usable* in a wide range of applications? If so, consider adopting standard formats and metadata standards early on. At

the very least, keep track of versions of data and code, with associated dates." (Goodman et al., 2014)

"Standards for data include, for example, data formats, data exchange protocols, and meta-data controlled vocabularies." (Barga et al., 2011)

"Software systems are transparent when they don't have murky corners or hidden depths. Transparency is a passive quality. A program is passive when it is possible to form a simple mental model of its behavior that is actuaally predictive for all or most cases, because you can see through the machinery to what is actually going on." (Raymond, 2003)

"Software systems are discoverable when they include features that are designed to help you build in your mind a correct mental model of what they do and how they work. So, for example, good documentation helps discoverability to a programmer. Discoverability is an active quality. To achieve it in your software, you cannot merely fail to be obscure, you have to go out of your way to be helpful." (Raymond, 2003)

"Elegant code does much with little. Elegant code is not only correct but visibly, *transparently* correct. It does not merely communicate an algorithm to a computer, but also conveys insight and assurance to the mind of a human that reads it. By seeking elegance in our code, we build better code. Learning to write transparent code is a first, long step toward learning how to write elegant code—and taking care to make code discoverable helps us learn how to make it transparent. Elegant code is both transparent and discoverable." (Raymond, 2003)

"To design for transparency and discoverability, you need to apply every tactic for keeping your code simple, and also concentrate on the ways in which your code is a communication to other human beings. The first questions to ask, after 'Will this design work?' are 'Will it be reaadable to other people? Is it elegant?' We hope it is clear … that these questions are not fluff and that elegance is not a luxury. These qualities in the human reaction to software are essential for reducing its bugginess and increasing its long-term maintainability." (Raymond, 2003)

"Software is maintainable to the extent that people who are not its author can successfully understand and modify it. Maintainability demands more than code that works; it demands code that follows the Rule of Clarity and communicates successfully to human beings as well as the computer." (Raymond, 2003)

2.4.2 Applied exercise

2.5 Example: Creating a template for "tidy" data collection

We will walk through an example of creating a template to collect data in a "tidy" format for a laboratory-based research project, based on a research project on drug efficacy in murine tuberculosis models. We will show the initial "untidy" format for data recording and show how we converted it to a "tidy" format. Finally, we will show how the data can then easily be analyzed and visualized using reproducible tools.

Objectives. After this module, the trainee will be able to:

- Understand how the principles of "tidy" data can be applied for a real, complex research project;
- List advantages of the "tidy" data format for the example project

2.5.1 Subsection 1

2.5.2 Subsection 2

2.5.3 Discussion questions

2.6 Power of using a single structured 'Project' directory for storing and tracking research project files

To improve the computational reproducibility of a research project, researchers can use a single 'Project' directory to collectively store all research data, meta-data, pre-processing code, and research products (e.g., paper drafts, figures). We will explain how this practice improves the reproducibility and list some of the common components and subdirectories to include in the structure of a 'Project' directory, including subdirectories for raw and pre-processed experimental data.

Objectives. After this module, the trainee will be able to:

- Describe a 'Project' directory, including common components and subdirectories
- List how a single 'Project' directory improves reproducibility

2.6.1 Subsection 1

One study surveyed over 250 biomedical researchers at the University of Washington. They noted that, "a common theme surrounding data management and analysis was that may researchers preferred to utilize their own individual methods to organize data. The varied ways of managing data were accepted as functional for most present needs. Some researchers admitted to having no organizational methodology at all, while others used whatever method best suited their individual needs." (Anderson et al., 2007) One respondent answered, "They're not organized in any way—they're just thrown into files under different projects," while another said "I grab them when I need them, they're not organized in any decent way," and another, "It's not even organized—a file on a central computer of protocols that we use, common lab protocols but those are just individual Word files within a folder so it's not searchable per se." (Anderson et al., 2007)

> "In general, data reuse is most possible when: 1) data; 2) metadata (information describing the data); and 3) information about the process of generating those data, such as code, are all provided." (Goodman et al., 2014)

> "So far we have used filenames without ever saying what a legal name is, so it's time for a couple of rules. First, filenames are limited to 14 characters. Second,

although you can use almost any character in a filename, common sense says you should stick to ones that are visible, and that you should avoid characters that might be used with other meanings. … To avoid pitfalls, you would do well to use only letters, numbers, the period and the underscore until you're familiar with the situation [i.e., characters with pitfalls]. (The period and the underscore are conventionally used to divide filenames into chunks…) Finally, don't forget that case distinctions matter—junk, Junk, and JUNK are three different names." (Kernighan and Pike, 1984)

"The [Unix] system distinguishes your file called 'junk' from anyone else's of the same name. The distinction is made by grouping files into *directories*, rather in the way that books are placed om shelves in a library, so files in different directories can have the same name without any conflict. Generally, each user haas a personal or *home directory*, sometimes called login directory, that contains only the files that belong to him or her. When you log in, you are 'in' your home directory. You may change the directory you are working in—often called your working or *current directory*—but your home directory is always the same. Unless you take special action, when you create a new file it is made in your current directory. Since this is initially your home directory, the file is unrelated to a file of the same name that might exist in someone else's directory. A directory can contain other directories as well as ordinary files … The natural way to picture this organization is as a tree of directories and files. It is possible to move around within this tree, and to find any file in the system by starting at the root of the tree and moving along the proper branches. Conversely, you can start where you are and move toward the root." (Kernighan and Pike, 1984)

"The name '/usr/you/junk' is called the *pathname* of the file. 'Pathname' has an intuitive meaning: it represents the full name of the path from the root through the tree of directories to a particular file. It is a universal rule in the Unix system that wherever you can use an ordinary filename, you can use a pathname." (Kernighan and Pike, 1984)

"If you work regularly with Mary on information in her directory, you can say 'I want to work on Mary's files instead of my own.' This is done by changing your current directory with the `cd` command… Now when you use a filename (without the /'s) as an argument to `cat` or `pr`, it refers to the file in Mary's directory. Changing directories doesn't affect any permissions associated with a file—if you couldn't access a file from your own directory, changing to another directory won't alter that fact." (Kernighan and Pike, 1984)

"It is usually convenient to arrange your own files so that all the files related to one thing are in a directory separate from other projects. For example, if you want to write a book, you might want to keep all the text in a directory called 'book'." (Kernighan and Pike, 1984)

"Suppose you're typing a large document like a book. Logically this divides into many small pieces, like chapters and perhaps sections. Physically it should be divided too, because it is cumbersome to edit large files. Thus you should type the document as a number of files. You might have separate files for each chapter, called 'ch1', 'ch2', etc. … With a systematic naming convention, you can tell at a glance where a particular file fits into the whole. What if you want to print the whole book? You could say `$ pr ch1.1 ch1.2 ch 1.3 ...`, but you

> would soon get bored typing filenames and start to make mistakes. This is where filename shorthand comes in. If you say `$ pr ch*` the shell takes the `*` to mean 'any string of characters,' so ch* is a pattern that matches all filenames in the current directory that begin with ch. The shell creates the list, in alphabetical order, and passes the list to `pr`. The `pr` command never sees the `*`; the pattern match that the shell does in the current directory generates aa list of strings that are passed to `pr`." (Kernighan and Pike, 1984)
>
> "The current directory is an attribute of a process, not a person or a program. … The notion of a current directory is certainly a notational convenience, because it can save a lot of typing, but its real purpose is organizational. Related files belong together in the same directory. '/usr' is often the top directory of a user file system… '/usr/you' is your login directory, your current directory when you first log in. … Whenever you embark on a new project, or whenever you have a set of related files … you could create a new directory with `mkdir` and put the files there." (Kernighan and Pike, 1984)
>
> "Despite their fundamental properties inside the kernel, directories sit in the file system as ordinary files. They can be read as ordinary files. But they can't be created or written as ordinary files—to preserve its sanity and the users' files, the kernel reserves to itself all control over the contents of directories." (Kernighan and Pike, 1984)
>
> "A file has several components: a name, contents, and administrative information such as permissions and modifications times. The administrative information is stored in the inode (over the years, the hyphen fell out of 'i-node'), along with essential system data such as how long it is, where on the disc the contents of the file are stored, and so on. … It is important to understand inodes, not only to appreciate the options on `ls`, but because in a strong sense the inodes *are* the files. All the directory hierarchy does is provide convenient names for files. The system's name for a file is its *i-number*: the number of the inode holding the file's information. … It is the i-number that is stored in the first two bytes of a directory, before the name. … The first two bytes in each directory entry are the only connection between the name of a file and its contents. A filename in a directory is therefore called a *link*, because it links a name in the directory hierarchy to the inode, and hence to the data. The same i-number can appear in more than one directory. The `rm` command does not actually remove the inodes; it removes directory entries or links. Only when the last link to a file disappears does the system remove the inode, and hence the file itself. If the i-number in a directory entry is zero, it means that the link has been removed, but not necessarily the contents of the file—there may still be a link somewhere else." (Kernighan and Pike, 1984)

2.6.2 Subsection 2

2.6.3 Practice quiz

2.7 Creating 'Project' templates

Researchers can use RStudio's 'Projects' can facilitate collecting research files in a single, structured directory, with the added benefit of easy use of version control. Researchers can gain even more benefits by consistently structuring all

their 'Project' directories. We will demonstrate how to implement structured project directories through RStudio, as well as how RStudio enables the creation of a 'Project' for initializing consistently-structured directories for all of a research group's projects.

Objectives. After this module, the trainee will be able to:

- Be able to create a structured `Project` directory within RStudio
- Understand how RStudio can be used to create 'Project' templates

2.7.1 Subsection 1

2.7.2 Subsection 2

2.7.3 Discussion questions

2.8 Example: Creating a 'Project' template

We will walk through a real example, based on the experiences of one of our Co-Is, of establishing the format for a research group's 'Project' template, creating that template using RStudio, and initializing a new research project directory using the created template. This example will be from a laboratory-based research group that studies the efficacy of tuberculosis drugs in a murine model.

Objectives. After this module, the trainee will be able to:

- Create a 'Project' template in RStudio to initialize consistently-formatted 'Project' directories
- Initialize a new 'Project' directory using this template

2.8.1 Subsection 1

2.8.2 Subsection 2

2.8.3 Applied exercise

2.9 Harnessing version control for transparent data recording

As a research project progresses, a typical practice in many experimental research groups is to save new versions of files (e.g., 'draft1.doc', 'draft2.doc'), so that changes can be reverted. However, this practice leads to an explosion of files, and it becomes hard to track which files represent the 'current' state of a project. Version control allows researchers to edit and change research project files more cleanly, while maintaining the power to 'backtrack' to previous versions, messages included to explain changes. We will explain what version control is and how it can be used in research projects to improve the transparency and reproducibility of research, particularly for data recording.

Objectives. After this module, the trainee will be able to:

- Describe version control
- Explain how version control can be used to improve reproducibility for data recording

2.9.1 Subsection 1

"Or maybe your goal is that your data is *usable* in a wide range of applications? If so, consider adopting standard formats and metadata standards early on. At the very least, keep track of versions of data and code, with associated dates." (Goodman et al., 2014)

Email attachments in lieu of common access files.

…

For example, one group of researchers investigated a large collection of emails from Enron (Hermans and Murphy-Hill, 2015). They found that passing Excel files through email attachements was a common practice, and that messages within emails suggested that spreadsheets were stored locally, rather than in a location that was accessible to all team members (Hermans and Murphy-Hill, 2015), which meant that team members might often be working on different versions of the same spreadsheet file. They note that "the practice of emailing spreadsheets is known to result in serious problems in terms of accountability and errors, as people do not have access to the latest version of a spreadsheet, but need to be updated of changes via email." (Hermans and Murphy-Hill, 2015)

"Team members regularly pass data files back and forth by hand, by email, and by using shared lab or project servers, websites, and databases." (Edwards et al., 2011)

Version control for spreadsheets

"Recent versions of spreadsheets now incorporate a 'Traack Changes' functionality which enables highlighting of changes made by different users along with a comment and review system. Such tools are a start toward this but more robust version control systems are required particularly in the context of increasingly online and collaborative method of working where large teams interact with a single document concurrently." (Birch et al., 2018)

2.9.2 Subsection 2

2.9.3 Discussion questions

2.10 Enhance the reproducibility of collaborative research with version control platforms

Once a researcher has learned to use *git* on their own computer for local version control, they can begin using version control platforms (e.g., *GitLab*, *GitHub*) to collaborate with others under version control. We will describe how a research team can benefit from using a version control platform to work

collaboratively.

Objectives. After this module, the trainee will be able to:

- List benefits of using a version control platform to collaborate on research projects, particularly for reproducibility
- Describe the difference between version control (e.g., *git*) and a version control platform (e.g., *GitLab*)

2.10.1 Subsection 1

VC platforms as a form of back-up.

One study surveyed neuroscience researchers at a UK institute. "The backup 'rule of three' states that for a file to be sufficiently backed up it should be kept in three separate locations using two different types of media with one offsite backup. A lack of an adequate backup solution could mean permanently lost data, effort and time. In this research, more than 82% of the respondents seemed to be unaware of suitable backup procedures to protect their data. Some respondents kept a single backup of work on external hard disks. Others used the Universities local networked servers as their means of backup." (AlTarawneh and Thorne, 2017)

"A good approach is to store at least three copies in at least two geographically distributed locations (e.g., original location such as a desktop computer, an external hard drive, and one or more remote sites) and to adopt a regular schedule for duplicating the data (i.e., backup)." (Michener, 2015)

2.10.2 Subsection 2

2.10.3 Discussion questions

2.11 Using git and GitLab to implement version control

For many years, use of version control required use of the command line, limiting its accessibility to researchers with limited programming experience. However, graphical interfaces have removed this barrier, and RStudio has particularly user-friendly tools for implementing version control. In this module, we will show how to use *git* through RStudio's user-friendly interface and how to connect from a local computer to *GitLab* through RStudio.

Objectives. After this module, the trainee will be able to:

- Understand how to set up and use *git* through RStudio's interface
- Understand how to connect with *GitLab* through RStudio to collaborate on research projects while maintaining version control

2.11.1 Subsection 1

"When the system prints the prompt $ and you type commands that get executed, it's not the kernel that is talking to you, but a go-between called the

command interpreter or *shell.* The shell is just an ordinary program like `date` or `who`, although it can do some remarkable things. The fact that the shell sits between you and the facilities of the kernel has real benefits, some of which we'll talk about here. There are three main ones: (1) Filename shorthands: you can pick up a whole set of filenames as arguments to a program by specifying a pattern for the names—the shell will find the filenames that fit your pattern; (2) Input-output redirection: you can arrange for the output of any program to go into a file instead of onto the terminal, and for the input to come from a file instead of the terminal. Input and output can even be connected to other programs. (3) Personalizing the environment: you can define your own commands and shorthands." (Kernighan and Pike, 1984)

"Suppose you're typing a large document like a book. Logically this divides into many small pieces, like chapters and perhaps sections. Physically it should be divided too, because it is cumbersome to edit large files. Thus you should type the document as a number of files. You might have separate files for each chapter, called 'ch1', 'ch2', etc. … With a systematic naming convention, you can tell at a glance where a particular file fits into the whole. What if you want to print the whole book? You could say `$ pr ch1.1 ch1.2 ch 1.3 ...`, but you would soon get bored typing filenames and start to make mistakes. This is where filename shorthand comes in. If you say `$ pr ch*` the shell takes the `*` to mean 'any string of characters,' so ch* is a pattern that matches all filenames in the current directory that begin with ch. The shell creates the list, in alphabetical order, and passes the list to `pr`. The `pr` command never sees the `*`; the pattern match that the shell does in the current directory generates aa list of strings that are passed to `pr`. The crucial point is that filename shorthand is not a property of the `pr` command, but a service of the shell. Thus you can use it to generate a sequence of filenames for *any* command." (Kernighan and Pike, 1984)

"One of the virtues of the Unix system is that there are several ways to bring it closer to your personal taste or the conventions of your local computing environmentl. … If there is a file named '.profile' in your login directory, the shell will execute the commands in it when you log in, before printing the first prompt. So you can put commands into '.profile' to set up your environment as you like it, and they will be executed every time you log in. … Some of the properties of the shell are actually controlled by so-called *shell variables*, with values that you can access and set yourself. For example, the prompt string, which we have been showing as $, is acually stored in a shell variable called 'PS1', and you can set it to anything you like, like this `PS1='Yes dear?'`. … Probably the most useful shell variable is the one that controls where the shell looks for commands. Recall that when you type the name of a command, the shell normally looks for it first in the current directory, then in '/bin', and then in '/usr/bin'. This sequence of directories is called the *search path*, and is stored in a shell variable called 'PATH'. If the default search path isn't what you want, you can change it, again usually in your '.profile'. … It is also possible to use variables for abbreviation. If you find yourself frequently referring to some directory with a long name, it might be worthwhile adding a line like `d=/horribly/long/directory/name` to your profile, so that you can say things like `$cd $d`. Personal variables like `d` are conventionally spelled in lower case to distinguish them from those used by the shell itself, like `PATH`." (Kernighan and Pike, 1984)

"The culmination of your login efforts is a *prompt*, usually a single character, indicating that the system is ready to accept commands from you. The prompt is

most likely to be a dolloar sign or a percent sign, but you can change it to anything you like… The prompt is actually printed by a program called the *command interpreter* or *shell*, which is your main interface to the system. … Once you receive the prompt, you can type *commands*, which are requests that the system do something. We will use *program* as a synonym for command." (Kernighan and Pike, 1984)

"While checksums are a great method to check if files are different, they don't tell us *how* the files differ. One approach to this is to compute the *diff* between two files using the Unix tool *diff*. Unix's *diff* works line by line, and outputs blocks (called *hunks*) that differ between files (resembling Git's *git diff* command)." (Buffalo, 2015)

2.11.2 Subsection 2

2.11.3 Applied exercise

3

Experimental Data Preprocessing

3.1 Principles and benefits of scripted pre-processing of experimental data

The experimental data collected for biomedical research often requires pre-processing before it can be analyzed (e.g., gating of flow cytometry data, feature finding / quantification for mass spectrometry data). Use of point-and-click software can limit the transparency and reproducibility of this analysis stage and is time-consuming for repeated tasks. We will explain how scripted pre-processing, especially using open source software, can improve transparency and reproducibility.

Objectives. After this module, the trainee will be able to:

- Define 'pre-processing' of experimental data
- Describe an open source code script and explain how it can increase reproducibility of data pre-processing

3.1.1 Subsection 1

For bioinformatics, "all too often the software is developed without thought toward future interoperability with other software products. As a result, the bioinformatics software landscape is currently characterized by fragmentation and silos, in which each research group develops and uses only the tools created within their lab." (Barga et al., 2011)

"The group also noted the lack of agility. Although they may be aware of a new or better algorithm they cannot easily integrate it into their analysis pipelines given the lack of standards across both data formats and tools. It typically requires a complete rewrite of the code in order to take advntge of a new technique or algorithm, requiring time and often funding to hire developers." (Barga et al., 2011)

"The benefit of working with a programming language is that you have the code in a file. This means that you can easily reuse that code. If the code has parameters it can even be applied to problems that follow a similar pattern." (Janssens, 2014)

"Data exploration in spreadsheet software is typically conducted via menus and dialog boxes, which leaves no record of the steps taken." (Murrell, 2009)

> "One reason Unix developers have been cool toward GUI interfaces is that, in their designers' haste to make them 'user-friendly' each one often becomes frustratingly opaque to anyone who has to solve user problems—or, indeed, interact with it anywhere outside the narrow range predicted by the user-interface designer." (Raymond, 2003)

> "Many operating systems touted as more 'modern' or 'user friendly' than Unix achieve their surface glossiness by locking users and developers into one interface policy, and offer an application-programming interface that for all its elaborateness is rather narrow and rigid. On such systems, tasks the designers have anticipated are very easy—but tasks they have not anticipated are often impossible or at best extremely painful. Unix, on the other hand, has flexibility in depth. The many ways Unix provides to glue together programs means that components of its basic toolkit can be combined to produce useful effects that the designers of the individual toolkit parts never anticipated." (Raymond, 2003)

3.1.2 Subsection 2

3.1.3 Discussion questions

3.2 Introduction to scripted data pre-processing in R

We will show how to implement scripted pre-processing of experimental data through R scripts. We will demonstrate the difference between interactive coding and code scripts, using R for examples. We will then demonstrate how to create, save, and run an R code script for a simple data cleaning task.

Objectives. After this module, the trainee will be able to:

- Describe what an R code script is and how it differs from interactive coding in R
- Create and save an R script to perform a simple data pre-processing task
- Run an R script
- List some popular packages in R for pre-processing biomedical data

3.2.1 Subsection 1

3.2.2 Subsection 2

3.2.3 Applied exercise

3.3 Simplify scripted pre-processing through R's 'tidyverse' tools

The R programming language now includes a collection of 'tidyverse' extension packages that enable user-friendly yet powerful work with experimental data, including pre-processing and exploratory visualizations. The principle behind the 'tidyverse' is that a collection of simple, general tools can be joined together to solve complex problems, as long as a consistent format is used for the input and output of each tool (the 'tidy' data format taught in other modules). In this module, we will explain why this 'tidyverse' system is so powerful and how it

can be leveraged within biomedical research, especially for reproducibly pre-processing experimental data.

Objectives. After this module, the trainee will be able to:

- Define R's 'tidyverse' system
- Explain how the 'tidyverse' collection of packages can be both user-friendly and powerful in solving many complex tasks with data
- Describe the difference between base R and R's 'tidyverse'.

3.3.1 Subsection 1

"There is a now-old trope in the world of programming. It's called the 'worse is better' debate; it seeks to explain why the Unix operating systems (which include Mac OS X these days), made up of so many little interchangeable parts, were so much more successful in the marketplace than LISP systems, which were ideologically pure, based on a single languagae (again, LISP), which itself was exceptionally simple, a favorite of 'serious' hackers everywhere. It's too complex to rehash here, but one of the ideas inherent within 'worse is better' is thata systems made up of many simple pieces that can be roped together, even if those pieces don't share a consistent interface, are likely to be more successful than systems that are designed with consistency in every regard. And it strikes me that this is a fundamental drama of new technologies. Unix beat out the LISP machines. If you consider mobile handsets, many of which run descendants of Unit (iOS and Andriod), Unix beat out Windows as well. And HTML5 beat out all of the various initiatives to create a single unified web. It nods to accessibility: it doesn't get in the way of those who want to make something huge and interconnected. But it doesn't enforce; it doesn't seek to change the behavior of page creators in the same way that such lost standards as XHTML 2.0 (which eremged from the offices of the World Wide Web Consortium, and then disappeared under the weight of its own intentions) once did. It's not a bad place to end up. It means that there is no single framework, no set of easy rules to lear, no overarching principles that, once learned, can make the web appear like a golden statue atop a mountain. There are just components: HTML to get the words on the page, forms to get people to write in, videos and images to put up pictures, moving or otherwise, and JavaScript to make everything dance." (Ford, 2014)

"One of the fundamental contributions of the Unix system [is] the idea of a *pipe*. A pipe is a way to connect the output of one program to the input of another program without any temporary file; a *pipeline* is a connection of two or more programs through pipes. … Any program that reads from a terminal can read from a pipe instead; any program that writes on the terminal can write to a pipe. … The programs in a pipeline actually run at the same time, not one after another. This means that the programs in a pipeline can be interactive; the kernel looks after whatever scheduling and synchronization is needed to make it all work. As you probably suspect by now, the shell arranges things when you ask for a pipe; the individual programs are oblivious to the redirection." (Kernighan and Pike, 1984)

"Even though the Unix system introduces a number of innovative programs and techniques, no single program or idea makes it work well. Instead, what makes it effective is an approach to programming, a philosophy of using the computer.

> Although that philosophy can't be written down in a single sentence, at its heart is the idea that the power of a system comes more from the relationships among programs than from the programs themselves. Many Unix programs do quite trivial things in isolation, but, combined with other programs, become general and useful tools." (Kernighan and Pike, 1984)
>
> "What is 'Unix'? In the narrowest sense, it is a time-sharing operating system *kernel*: a program that controls the resources of a computer and allocates them among its users. It lets users run their programs; it controls the peripheral devices (discs, terminals, printers, and the like) connected to the machine; and it provides a file system that manages the long-term storage of information such as programs, data, and documents. In a broader sense, 'Unix' is often taken to include not only the kernel, but also essential programs like compiles, editors, command languages, programs for copying and printing files, and so on. Still more broadly, 'Unix' may even include programs develpoed by you or others to be run on your system, such as tools for document preparation, routines for statistical analysis, and graphics packages." (Kernighan and Pike, 1984)

3.3.2 Subsection 2

3.3.3 Practice quiz

3.4 Complex data types in experimental data pre-processing

Raw data from many biomedical experiments, especially those that use high-throughput techniques, can be very large and complex. Because of the scale and complexity of these data, software for pre-processing the data in R often uses complex, 'untidy' data formats. While these formats are necessary for computational efficiency, they add a critical barrier for researchers wishing to implement reproducibility tools. In this module, we will explain why use of complex data formats is often necessary within open source pre-processing software and outline the hurdles created in reproducibility tool use among laboratory-based scientists.

Objectives. After this module, the trainee will be able to:

- Explain why R software for pre-processing biomedical data often stores data in complex, 'untidy' formats
- Describe how these complex data formats can create barriers to laboratory-based researchers seeking to use reproducibility tools for data pre-processing

3.4.1 Subsection 1

3.4.2 Subsection 2

3.4.3 Practice quiz

3.5 Complex data types in R and Bioconductor

Many R extension packages for pre-processing experimental data use complex (rather than 'tidy') data formats within their code, and many output data in

complex formats. Very recently, the *broom* and *biobroom* R packages have been developed to extract a 'tidy' dataset from a complex data format. These tools create a clean, simple connection between the complex data formats often used in pre-processing experimental data and the 'tidy' format required to use the 'tidyverse' tools now taught in many introductory R courses. In this module, we will describe the 'list' data structure, the common backbone for complex data structures in R and provide tips on how to explore and extract data stored in R in this format, including through the *broom* and *biobroom* packages.

Objectives. After this module, the trainee will be able to:

- Describe the structure of R's 'list' data format
- Take basic steps to explore and extract data stored in R's complex, list-based structures
- Describe what the *broom* and *biobroom* R packages can do
- Explain how converting data to a 'tidy' format can improve reproducibility

3.5.1 Subsection 1

> "Object-oriented design doesn't have to be over-complicated design, but we've observed that too often it is. Too many OO designs are spaghetti-like tangles of is-a and has-a relationships, or feature thick layers of glue in which many of the objects seem to exist simply to hold places in a steep-sided pyramid of abstractions. Such designs are the opposite of transparent; they are (notoriously) opaque and difficult to debug." (Raymond, 2003)

> "Unix programmers are the original zealots about modularity, but tend to go about it in a quiter way [that with OOP]. Keeping glue layers thin is part of it; more generally, our tradition teaches us to build lower, hugging the ground with algorithms and structures that are designed to be simple and transparent." (Raymond, 2003)

3.5.2 Subsection 2

3.5.3 Applied exercise

3.6 Example: Converting from complex to 'tidy' data formats

We will provide a detailed example of a case where data pre-processing in R results in a complex, 'untidy' data format. We will walk through an example of applying automated gating to flow cytometry data. We will demonstrate the complex initial format of this pre-processed data and then show trainees how a 'tidy' dataset can be extracted and used for further data analysis and visualization using the popular R 'tidyverse' tools. This example will use real experimental data from one of our Co-Is research on the immunology of tuberculosis.

Objectives. After this module, the trainee will be able to:

- Describe how tools like *biobroom* were used in this real research example

to convert from the complex data format from pre-processing to a format better for further data analysis and visualization
- Understand how these tools would fit in their own research pipelines

3.6.1 Subsection 1

3.6.2 Subsection 2

3.6.3 Applied exercise

3.7 Introduction to reproducible data pre-processing protocols

Reproducibility tools can be used to create reproducible data pre-processing protocols—documents that combine code and text in a 'knitted' document, which can be re-used to ensure data pre-processing is consistent and reproducible across research projects. In this module, we will describe how reproducible data pre-processing protocols can improve reproducibility of pre-processing experimental data, as well as to ensure transparency, consistency, and reproducibility across the research projects conducted by a research team.

Objectives. After this module, the trainee will be able to:

- Define a 'reproducible data pre-processing protocol'
- Explain how such protocols improve reproducibility at the data pre-processing phase
- List other benefits, including improving efficiency and consistency of data pre-processing

3.7.1 Subsection 1

3.7.2 Subsection 2

3.7.3 Discussion questions

3.8 RMarkdown for creating reproducible data pre-processing protocols

The R extension package RMarkdown can be used to create documents that combine code and text in a 'knitted' document, and it has become a popular tool for improving the computational reproducibility and efficiency of the data analysis stage of research. This tool can also be used earlier in the research process, however, to improve reproducibility of pre-processing steps. In this module, we will provide detailed instructions on how to use RMarkdown in RStudio to create documents that combine code and text. We will show how an RMarkdown document describing a data pre-processing protocol can be used to efficiently apply the same data pre-processing steps to different sets of raw data.

Objectives. After this module, the trainee will be able to:

- Define RMarkdown and the documents it can create

- Explain how RMarkdown can be used to improve the reproducibility of research projects at the data pre-processing phase
- Create a document in RStudio using
- Apply it to several different datasets with the same format

3.8.1 Subsection 1

"WordPerfect was always the best word processor. Because it allowed for insight into its very structure. You could hit a certain key combination and suddenly the screen would split and you'd reveal the codes, the bolds and italics, and so forth, that would define your text when it was printed. It was beloved of legal secretaries and journalists alike. Because when you work with words, at the practical, everyday level, the ability to look under the hood is essential. Words are not simple. And WordPerfect acknowledged that. Microsoft Word did not. Microsoft kept insisting that what you saw on your screen was the way things *were*, and if your fonts just kept sort of randonmly changing, well, you must have wanted it that way. Then along came HTML, and what I remember most was that sense of being back inside the file. Sure, HTML was a typographic nightmare, a bunch of unjustified Times New Roman in 12 pt on screens with chiclet-size pixels, but under the hood you could see all the pieces. Just like WordPerfect. That transparency was a wonderful thing, and it renewed computing for me." (Ford, 2014)

"TeX was created by Donald E. Knuth, a professor at Stanford University who has achieved international renown as a mathematician and computer scientist. Knuth also has an aesthetic sense uncommon in his field, and his work output is truly phenomenal. TeX is a happy byproduct of Knuth's mammoth enterprise, *The Art of Computer Programming*. This series of reference books, designed to cover the whole gamut of programming concepts and techniques, is a *sine qua non* for all computer scientists." (Seroul, 2012)

"Roughly speaking, text processors fall into two categories: (1) WYSIWYG systems: what you see is what you get. You see on the screen at all times what the printed document will look like, and what you type has immediate effect on the appearance of the document. (2) markup systems, where you type your text interspersed with formatting instructions, but don't see their effect right away. You must run a program to examine the resulting image, whether on paper or on the screen. In computer science jargon, markup systems must compile the source file you type. WYSIWYG systems have the obvious advantage of immediate feedback, but they are not very precise: what is acceptable at a resolution of 300 dots per inch, for an ephemeral publication such as a newsletter or flier, is no longer so for a book that will be phototypeset at high resolution. The human eye is extraordinarily sensitive: you can be bothered by the appearance of a text without being able to pinpoint why, just as you can tell when someone plays the wrong note in an orchestra, without being able to identify the CUlprit. One quickly leams in typesetting that the beauty, legibility and comfortable reading of a text depend on minute details: each element must be placed exactly right, within thousandths of an inch. For this type of work, the advantage of immediate feedback vanishes: fine details of spacing, alignment, and so on are much too small to be discernible at the screen's relatively low resolution, and even if it such were not the case, it would still be a monumental chore to find the right place

for everything by hand. For this reason it is not surprising that in the world of professional typesetting markup systems are preferred. They automate the task of finding the right place for each character with great precision. Naturally, this approach is less attractive for beginners, since one can't see the results as one types, and must develop a feeling for what the system will do. But nowadays, you can have the best of both worlds by using a markup system with a WYSIWYG *front end*; we'll talk about such front ends for TEX later on. TEX was developed in the late seventies and early eighties, before WYSIWYG systems were widespread. But were it to be redesigned now, it would still be a markup language. To give you an idea of the precision with which TEX operates: the internal unit it uses for its calculations is about a hundred times smaller than the wavelength of visible light! (That's right, a hundred times.) In other words, any round-off error introduced in the calculations is invisible to the naked eye." (Seroul, 2012)

"You should be sure to understand the difference between a text editor and a text processor. A text processor is a text editor together with formatting software that allows you to switch fonts, do double columns, indent, and so on. A text editor puts your text in a file on disk, and displays a portion of it on the screen. It doesn't format your text at all. We insist on the difference because those accustomed to WYSIWYG systems are often not aware of it: they only know text processors. Where can you find a text editor? Just about everywhere. Every text processor includes a text editor which you can use. But if you use your text processor as a text editor, be sure to save your file using a 'save ASCII' or 'save text only' option, so that the text processor's own formatting commands are stripped off. If you give TEX a file created without this precaution, you'll get garbage, because TEX cannot digest your text processor's commands." (Seroul, 2012)

"TeX enabled authors to encode their precise intent into their manuscripts: This block of text is a computer program, while this word is a keyword in that program. The language it used, called TeX markup, formalized the slow, error-prone communication that is normally carried out with the printer over repeated galley proofs." (Apte, 2019)

"The idea of writing markup inside text wasn't especially novel; it has been used from 1970's runoff (the UNIX family of printer-preparation utilities) to today's HTML tags. TeX was new in that it captured key concepts necessary for realistic typesetting and formalized them." (Apte, 2019)

"With these higher-level commands, the free TeX engine, and the LaTeX book, the use of TeX exploded. The macro file has since evolved and changed names, but authors still typically run the program called latex or its variants. Hence, most people who write TeX manuscripts know the program as LaTeX and the commands they use as LaTeX commands." (Apte, 2019)

"The effect of LaTeX on scientific and technical publishing has been profound. Precise typesetting is critical, particularly for conveying concepts using chemical and mathematical formulas, algorithms, and similar constructs. The sheer volume of papers, journals, books, and other publications generated in the modern world is far beyond the throughput possible via manual typesetting. And TeX enables automation without losing precision. Thanks to LaTeX, book authors can generate camera-ready copy on their own. Most academic and journal publishers

> accept article manuscripts written in LaTeX, and there's even an open archive maintained by Cornell University where authors of papers in physics, chemistry, and other disciplines can directly submit their LaTeX manuscripts for open viewing. Over 10,000 manuscripts are submitted to this archive every month from all over the world." (Apte, 2019)
>
> "For many users, a practical difficulty with typesetting using TeX is preparing the manuscripts. When TeX was first developed, technical authors were accustomed to using plain-text editors like WordStar, vi, or Emacs with a computer keyboard. The idea of marking up their text with commands and running the manuscript through a typesetting engine felt natural to them. Today's typesetters, particularly desktop publishers, have a different mental model. They expect to see the output in graphical form and then to visually make edits with a mouse and keyboard, as they would in any WYSIWYG program. They might not be too picky about the quality of the output, but they appreciate design capabilities, such as the ability to flow text around curved outlines. Many print products are now produced with tools like Microsoft Word for this very reason. TeX authors cannot do the same work as easily." (Apte, 2019)

3.8.2 Subsection 2

3.8.3 Applied exercise

3.9 Example: Creating a reproducible data pre-processing protocol

We will walk through an example of creating a reproducible protocol for the automated gating of flow cytometry data for a project on the immunology of tuberculosis lead by one of our Co-Is. This data pre-processing protocol was created using RMarkdown and allows the efficient, transparent, and reproducible gating of flow cytometry data for all experiments in the research group. We will walk the trainees through how we developed the protocol initially, the final pre-processing protocol, how we apply this protocol to new experimental data.

Objectives. After this module, the trainee will be able to:

- Explain how a reproducible data pre-processing protocol can be integrated into a real research project
- Understand how to design and implement a data pre-processing protocol to replace manual or point-and-click data pre-processing tools

3.9.1 Subsection 1

3.9.2 Subsection 2

3.9.3 Practice quiz

4

References

5
Bibliography

AlTarawneh, G. and Thorne, S. (2017). A pilot study exploring spreadsheet risk in scientific research. *arXiv preprint arXiv:1703.09785*.

Anderson, N. R., Lee, E. S., Brockenbrough, J. S., Minie, M. E., Fuller, S., Brinkley, J., and Tarczy-Hornoch, P. (2007). Issues in biomedical research data management and analysis: needs and barriers. *Journal of the American Medical Informatics Association*, 14(4):478–488.

Apte, P. (2019). The lingua franca of latex. *Increment*.

Barga, R., Howe, B., Beck, D., Bowers, S., Dobyns, W., Haynes, W., Higdon, R., Howard, C., Roth, C., Stewart, E., et al. (2011). Bioinformatics and data-intensive scientific discovery in the beginning of the 21st century. *Omics: a journal of integrative biology*, 15(4):199–201.

Birch, D., Lyford-Smith, D., and Guo, Y. (2018). The future of spreadsheets in the big data era. *arXiv preprint arXiv:1801.10231*.

Brazma, A., Krestyaninova, M., and Sarkans, U. (2006). Standards for systems biology. *Nature Reviews Genetics*, 7(8):593.

Broman, K. W. and Woo, K. H. (2018). Data organization in spreadsheets. *The American Statistician*, 72(1):2–10.

Bryan, J. (2018). Excuse me, do you have a moment to talk about version control? *The American Statistician*, 72(1):20–27.

Buffalo, V. (2015). *Bioinformatics data skills: Reproducible and robust research with open source tools*. " O'Reilly Media, Inc.".

Campbell-Kelly, M. (2007). Number crunching without programming: The evolution of spreadsheet usability. *IEEE Annals of the History of Computing*, 29(3):6–19.

Creeth, R. (1985). Microcomputer spreadsheets: their uses and abuses. *Journal of Accountancy (pre-1986)*, 159(000006):90.

Edwards, P. N., Mayernik, M. S., Batcheller, A. L., Bowker, G. C., and Borgman, C. L. (2011). Science friction: Data, metadata, and collaboration. *Social Studies of Science*, 41(5):667–690.

Ellis, S. E. and Leek, J. T. (2018). How to share data for collaboration. *The American Statistician*, 72(1):53–57.

Ford, P. (2014). On file formats, very briefly. *The Manual*.

Ford, P. (2015). I dreamed of a perfect database. *New Republic*.

Ghosh, S., Matsuoka, Y., Asai, Y., Hsin, K.-Y., and Kitano, H. (2011). Software for systems biology: from tools to integrated platforms. *Nature Reviews Genetics*, 12(12):821.

Goodman, A., Pepe, A., Blocker, A. W., Borgman, C. L., Cranmer, K., Crosas, M., Di Stefano, R., Gil, Y., Groth, P., Hedstrom, M., et al. (2014). Ten simple rules for the care and feeding of scientific data.

Hermans, F., Jansen, B., Roy, S., Aivaloglou, E., Swidan, A., and Hoepelman, D. (2016). Spreadsheets are code: An overview of software engineering approaches applied to spreadsheets. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 5, pages 56–65. IEEE.

Hermans, F. and Murphy-Hill, E. (2015). Enron's spreadsheets and related emails: A dataset and analysis. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 2, pages 7–16. IEEE.

Hunt, A., Thomas, D., and Cunningham, W. (2000). *The Pragmatic Programmer: From Journeyman to Master*. Addison-Wesley Professional.

Janssens, J. (2014). *Data Science at the Command Line: Facing the Future with Time-tested Tools*. " O'Reilly Media, Inc.".

Keller, S., Korkmaz, G., Orr, M., Schroeder, A., and Shipp, S. (2017). The evolution of data quality: Understanding the transdisciplinary origins of data quality concepts and approaches. *Annu. Rev. Stat. Appl*, 4:85–108.

Kernighan, B. W. and Pike, R. (1984). *The UNIX programming environment*, volume 270. Prentice-Hall Englewood Cliffs, NJ.

Krishnan, S., Haas, D., Franklin, M. J., and Wu, E. (2016). Towards reliable interactive data cleaning: A user survey and recommendations. In *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, page 9. ACM.

Levy, S. (1984). A spreadsheet way of knowledge. *Harpers*, 269:58–64.

Lynch, C. (2008). Big data: How do your data grow? *Nature*, 455(7209):28.

Marwick, B., Boettiger, C., and Mullen, L. (2018). Packaging data analytical work reproducibly using R (and friends). *The American Statistician*, 72(1):80–88.

McCullough, B. (2001). Does microsoft fix errors in excel? In *Proceedings of the 2001 joint statistical meetings*.

McCullough, B. D. (1999). Assessing the reliability of statistical software: Part ii. *The American Statistician*, 53(2):149–159.

McCullough, B. D. and Heiser, D. A. (2008). On the accuracy of statistical procedures in microsoft excel 2007. *Computational Statistics & Data Analysis*, 52(10):4570–4578.

McCullough, B. D. and Wilson, B. (1999). On the accuracy of statistical procedures in microsoft excel 97. *Computational Statistics & Data Analysis*, 31(1):27–37.

McCullough, B. D. and Wilson, B. (2002). On the accuracy of statistical procedures in microsoft excel 2000 and excel xp. *Computational Statistics & Data Analysis*, 40(4):713–721.

McCullough, B. D. and Wilson, B. (2005). On the accuracy of statistical procedures in microsoft excel 2003. *Computational Statistics & Data Analysis*, 49(4):1244–1252.

Mélard, G. (2014). On the accuracy of statistical procedures in microsoft excel 2010. *Computational statistics*, 29(5):1095–1128.

Michener, W. K. (2015). Ten simple rules for creating a good data management plan. *PLoS computational biology*, 11(10):e1004525.

Murrell, P. (2009). *Introduction to data technologies*. Chapman and Hall/CRC.

Myneni, S. and Patel, V. L. (2010). Organization of biomedical data for collaborative scientific research: A research information management system. *International journal of information management*, 30(3):256–264.

Nardi, B. A. and Miller, J. R. (1990). The spreadsheet interface: A basis for end user programming. In *Proceedings of the IFIP TC13 Third Interational Conference on Human-Computer Interaction*, pages 977–983. North-Holland Publishing Co.

Nash, J. (2006). Spreadsheets in statistical practice—another look. *The American Statistician*, 60(3):287–289.

Powell, S. G., Baker, K. R., and Lawson, B. (2009). Errors in operational spreadsheets: A review of the state of the art. In *2009 42nd Hawaii International Conference on System Sciences*, pages 1–8. IEEE.

Raymond, E. S. (2003). *The art of Unix programming*. Addison-Wesley Professional.

Sansone, S.-A., Rocca-Serra, P., Field, D., Maguire, E., Taylor, C., Hofmann, O., Fang, H., Neumann, S., Tong, W., Amaral-Zettler, L., et al. (2012). Toward interoperable bioscience data. *Nature genetics*, 44(2):121.

Schadt, E. E., Linderman, M. D., Sorenson, J., Lee, L., and Nolan, G. P. (2010). Computational solutions to large-scale data management and analysis. *Nature reviews genetics*, 11(9):647.

Seroul, R. (2012). *A Beginner's Book of TEX*. Springer Science & Business Media.

Teixeira, R. and Amaral, V. (2016). On the emergence of patterns for spreadsheets data arrangements. In *Federation of International Conferences on Software Technologies: Applications and Foundations*, pages 333–345. Springer.

Topaloglou, T., Davidson, S. B., Jagadish, H., Markowitz, V. M., Steeg, E. W., and Tyers, M. (2004). Biological data management: Research, practice and opportunities. In *Proceedings of the Thirtieth international conference on Very large data bases-Volume 30*, pages 1233–1236. VLDB Endowment.

U.S. Department of Health and Human Services, National Institutes of Health (2016). NIH-Wide Strategic Plan, Fiscal Years 2016-2020: Turning Discovery Into Health. Accessed: 2018-06-24.

U.S. Department of Health and Human Services, National Institutes of Health (2018). NIH Strategic Plan for Data Science. Accessed: 2018-06-24.

Waltemath, D. and Wolkenhauer, O. (2016). How modeling standards, software, and initiatives support reproducibility in systems biology and systems medicine. *IEEE Transactions on Biomedical Engineering*, 63(10):1999–2006.

Welsh, E. A., Stewart, P. A., Kuenzi, B. M., and Eschrich, J. A. (2017). Escape excel: A tool for preventing gene symbol and accession conversion errors. *PloS one*, 12(9):e0185207.

Wilkinson, M. D., Dumontier, M., Aalbersberg, I. J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.-W., da Silva Santos, L. B., Bourne, P. E., et al. (2016). The fair guiding principles for scientific data management and stewardship. *Scientific data*, 3.

Willekens, F. (2013). Chronological objects in demographic research. *Demographic research*, 28:649–680.

Zeeberg, B. R., Riss, J., Kane, D. W., Bussey, K. J., Uchio, E., Linehan, W. M., Barrett, J. C., and Weinstein, J. N. (2004). Mistaken identifiers: gene name errors can be introduced inadvertently when using Excel in bioinformatics. *BMC Bioinformatics*, 5(1):80.

Ziemann, M., Eren, Y., and El-Osta, A. (2016). Gene name errors are widespread in the scientific literature. *Genome biology*, 17(1):177.