Title: Data-specific functions: a comment on Salganik et al.

Author: Jacob C. Fisher

Author's affiliation: University of Michigan

Author's email: jakef@umich.edu

Acknowledgements:

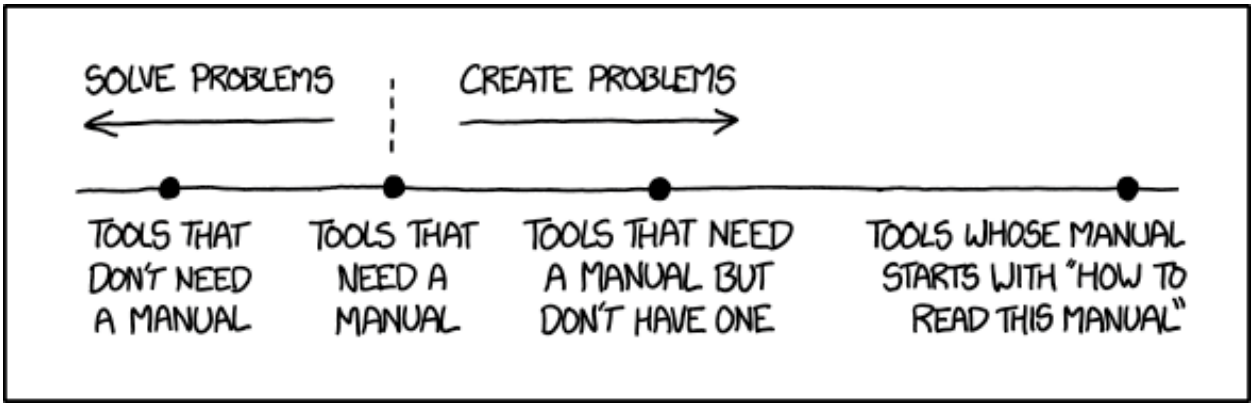
The author thanks Bryce Bartlett, Achim Edelmann, and Ashton Verdery for their helpful comments. Grants from the National Science Foundation (1535370, 1760609) and the National Institutes of Health (UL1-TR002240) supported this research.

Keywords:

Data-specific functions, survey research, data sharing, quantitative methodology, computational social science

Abstract:

In a recent article, Salganik et al. describe a new approach to managing survey data in service of the Fragile Families Challenge, which they call "treating metadata as data." Although the approach that they present is a good first step, a more ambitious proposal could improve survey data analysis even more substantially. I recommend that data collection efforts distribute an open-source set of tools for working with a particular data set that I call data-specific functions. The goal of these functions is to codify best practices for working with the data in a set of functions for commonly used statistical software. These functions would be jointly developed by the users and distributers of the data. Building such functions would both shorten the learning curve for new users, and would improve the quality of the data, by making tacit knowledge about problems with the data explicit and easy to act on.



[1]

In a recent article in this journal, Salganik et al. (forthcoming) describes a new approach to managing survey data in service of a prediction competition, the Fragile Families Challenge. They refer to their approach as "treating metadata as data" and suggest that the analysis of complex survey designs can be simplified if variable names are given a consistent, regular expression-compatible nomenclature and are placed in a dataset of metadata. They expect analysts will use the dataset of metadata to help them include many variables in a predictive model in a way that respects the structure of the data.

Although the approach that Salganik et al. (forthcoming) suggests is sound, it does not go far enough. The "metadata as data" approach builds the information from the survey's codebook – namely, response type, the survey wave, and what type of respondent answered the question – into the data, but it does not encode tacit knowledge about the dataset. In many complex surveys, analysts accumulate a large amount of tacit knowledge about characteristics of the survey. For example, in a longitudinal survey about a school where students are held back, an analyst might learn which variables to use for merging such that held back students aren't included twice. At best, the "metadata as data" approach leaves analysts to uncover that tacit

[1] Munroe, Randall. 2014. "Manuals." *XKCD*. Retrieved November 12, 2018 (https://xkcd.com/1343/).

knowledge from a large pile of ill-organized codebooks; at worst, analysts could be left with no documentation about data problems whatsoever.

To ameliorate this problem, I propose an additional tool for distributing data from complex surveys: data-specific functions. Ideally, as maintainers and users of large datasets develop approaches to dealing with unusual characteristics of the data, they would encode their approaches in a set of functions in a common statistical computing language. These functions would be provided open-source for the community of people who use the data, to be improved as new intricacies of the data are found. By jointly building functions to process the data, users and maintainers of the dataset could ensure that tacit knowledge is disseminated quickly, accurately, and without requiring users to read large amounts of documentation.

The remainder of this comment outlines principles for how data-specific functions should be constructed and shared among a research team, or among users of a dataset more broadly, and addresses possible concerns with creating data-specific functions. In general, the difference between building data-specific functions for a dataset and documenting the data in formal documentation is like the difference between putting a guard in front of a jigsaw versus putting a warning in the owner's manual. Although most people would know avoid the blade, and others would read the manual and see the warning, a guard directly on the saw would force everyone using the tool to avoid the blade. Similarly, by building precautions directly into the data analysis tools, instead of tucking them into ancillary documentation, we can ensure that everyone using a dataset uses it in a consistent and responsible way.

Principles for data-specific functions

For the purposes of this comment, a function means a reusable set of code that takes one or more inputs (called arguments), performs a set of actions, and returns an output. A data-

specific function is a function that is designed for processing a particular dataset. Although a data-specific function can be elaborate, like the RAND code distributed by the Health and Retirement Study (HRS; 2018) or the DHS.rates package in R (Elkasabi 2018), at its core, a data-specific function need not be complicated – a function that takes the dataset to be processed and returns a modification of that dataset would serve as a data-specific function. To build data-specific functions most effectively, however, I suggest a few common recoding activities that are particularly amenable to data-specific functions: common recodes and known data errors, subsetting criteria, merge criteria, commonly used summaries, and language-specific constructions.

Recoding and correcting known data errors

Recoding of data and known data errors are a useful first target for data-specific functions. In most datasets, the data contain some anomalies, which researchers modify prior to conducting analyses. These anomalies come in two varieties. The first variety are oddities specific to a given dataset, such as encoding missing values as 999, or top-coding of income values above a certain dollar amount. Such anomalies are often noted in codebooks, but not all researchers notice them. This can lead to errors that find their way into published work (e.g., Firebaugh 1980). The second variety are simply errors, retained in a dataset for archival reasons. Errors could enter the data for any number of reasons, from the use an overzealous Excel auto-complete function to typos in data entry.

Recoding and known data errors are particularly amenable to data-specific functions because they are simple to fix, but identifying all of them typically requires tacit knowledge about the data. Errors in the data are rarely well-advertised, if they are written down at all. As such, to correct them, an analyst would have find the error in the full documentation before beginning analysis, re-identify the error him- or herself in the raw data, or – more likely – be

corrected about a data problem by a member of the project staff, which is a labor intensive and unreliable solution. Each of those solutions is, at best, inefficient, and, at worst, likely to result in some of the errors falling through the cracks.

Correcting these two types of errors would likely involve two sets of data specific functions. First, one set of functions would address recoding errors that occur on more than one variable in the dataset. The recoding functions would take a variable name as an argument, and would perform a recoding action, such as converting 999 to a software-appropriate missing value. Second, a broader function would correct known errors in the dataset. The broader function might call the smaller, recoding functions to recode variables in addition to correcting errors in the data.

Subsetting criteria

A second use case for data-specific functions is subsetting criteria, by which I mean rows or columns that should be dropped from the data. In some datasets, known errors arises, and experienced analysts know to drop those cases. Most often, these are cases occur when the rows appear to contain valid data, but they are not comparable to other rows in the data. For example, in the university data collected by the Institute for Research on Innovation and Science (IRIS; Owen-Smith, Lane, and Weinberg 2017), some of the university systems provide data on both flagship and satellite campuses, which cannot be compared to single-campus universities. For comparisons between main university campuses, the non-flagship campuses are often dropped.

Although it is a simple task, subsetting the data often involves a considerable amount of tacit knowledge about the data, making it a good use case for data-specific functions. A data-specific function to subset the data would take the dataset's path as an argument and would return the subset data. The function could be combined with the recoding functions to create an omnibus "load data" function. Different projects may require different subsets of the data;

taking the example above, the non-flagship campuses in the IRIS data could still be included for studies that are not comparing main university campuses. In those cases, either an argument could be added to the function, specifying the type of analysis and therefore the types of cases that must be dropped, or separate data-specific functions could be created for each type of analysis.

Merge criteria

A third use case for data-specific functions is merging or joining two datasets. Although merges are simple in principle – two datasets are combined by matching on one or more variables that the datasets have in common – the specifics of which datasets can be merged, which variables they should be merged on, and how many values will be unmatchable between datasets are rarely documented well. For example, in the PROSPER dataset (Osgood et al. 2013; Spoth et al. 2004, 2007), a longitudinal study of two cohorts of students in the same set of schools over time, students can be held back, so analysts must merge on both student ID and cohort. In most cases, this is a piece of tacit knowledge that new analysts must learn the hard way.

Merge criteria can be encoded in two different ways. First, and ideally, the default values for merges with the data would be set correctly. Setting the defaults for merges typically means indexing the data using the correct columns. In SQL, this would mean indexing the dataset by the columns that are most likely to be used for a merge, and in R (R Core Team 2017), this would mean saving the data as data.table objects (Dowle and Srinivasan 2017), with keys set to the merge columns. Saving the correct merge defaults may not be possible, however. Some analysts may use SAS or Stata, which do not have a concept of default columns for merging, and some data sources may be merged in multiple ways, such as the IRIS data, which can be merged by unique grant numbers or unique values of university. Therefore, a second solution would be a

data-specific function that takes paths to the datasets to be joined as arguments and returns the merged dataset. Ideally, the data-specific function would also produce a message noting the match rate, meaning the number of rows that could be matched in the merge, and the correct match rate, meaning the number of rows that can be matched when the full datasets are merged correctly. The additional information about matching would notify analysts when something has gone wrong, and how many additional cases were lost because of the error – a task that often takes analysts a long time to determine by hand.

Commonly used summaries

A fourth use case for data-specific functions is calculating commonly used summaries of the data. Summaries might include new variables or summary statistics generated from the data. Although many summary statistics can be calculated by the analyst without much ado, certain summary statistics require considerable knowledge about the data. For example, imputing wealth in the HRS is an elaborate process that requires many input variables. While the variables needed may be well-documented, the complexity of the task increases the likelihood of individual analysts making errors. Similar problems arise can arise for calculations that only use a few variables but require many steps using those variables. Calculation of demographic rates, for example, is a commonly performed calculation that requires many steps. A calculation need not be complex to warrant a data-specific function, however – as long as it is routinely required by users of the data, having a function that computes it automatically will benefit users of the data.

Data-specific functions can generate both new variables and tables of summary calculations. In both cases, the function would take the path to the data and the names of the required variables as arguments. Functions that produce new variables would return the dataset with the new variable added, and functions that produce tables of summary calculations would

return a different a different dataset containing the summary statistics, similar to the operation of the MEANS procedure in SAS. Ideally, the function would default to using the variable names used in the distributed copy of the dataset, such that new users of the data do not have to look up what the variable names are.

For several datasets, data-specific functions to calculate commonly used summaries already exist. The SAS macros developed by RAND for the HRS and the DHS.rates R package serve as data-specific functions intended to calculate a new variable – wealth – and a table of summary statistics – demographic rates – respectively. These data-specific functions vary in the extent to which they are intended to be used by people who download the dataset. The RAND SAS macros are intended to serve as reference material that explain how an additional, downloadable dataset of wealth characteristics was created. By contrast, the DHS.rates package is designed to be used to calculate summary statistics from the DHS data in daily practice. When possible, functions for use in daily practice are preferable, because they allow the analyst to make corrections to the functions that can be disseminated without requiring other users to re-download and replace a pre-calculated dataset. In both cases, however, the functions encode information about the data in an immediately usable form, making use of the data more efficient.

Language-specific constructs

A fifth use case for data-specific functions is modifying the data such that it can be used with a given statistical language. The most common example of this is in R, where specific objects often must be created from the data before analysis can proceed. For example, R has a robust set of tools for analyzing network data, in the statnet (Handcock et al. 2008, 2016) and igraph (Csardi and Nepusz 2006) packages, but the data must first be put into a network object before those tools can be used. Similar, although less involved, problems arise in other

languages. For example, longitudinal data can either be stored as "long" or "wide", but it must be in long form for Stata to analyze it.[2]

A data-specific function would be helpful for constructing the objects or data structures needed to analyze the data in a specific language. With network data in R, a function might take the path to the data as an argument, and return a tibble (Müller and Wickham 2017) of network objects, for example. Or, with the longitudinal data in Stata, the command might take the path of the dataset, and reshape it from wide to long using the correct variable names. In both cases, performing these tasks requires tacit knowledge about how to group the data, and about how the data can be converted into the correct format. A set of data-specific functions would encode these tasks into a set of tools, making it much easier for new analysts to begin using the data productively.

Practical concerns

Three practical concerns, which might prevent people from using this approach, bear mention: replicability, costs, and the propagation of errors.

Replicability

The first concern, replicability, refers to the difficulty of reproducing past results calculated using data-specific functions. Data-specific functions are likely to change as people continue to improve on the functions, including adding support for new waves of data, or correcting new errors. These changes could cause existing code to produce different results than before. This problem also occurs without data-specific functions, however. New, ill-documented

[2] Interoperability between long and wide forms has been improving – e.g., at the time of this writing, Stata could use wide form data for structural equation models – but that has not obviated the need to switch between long and wide form data.

changes to a dataset can change results that analysts had produced in the past. Data-specific functions could make those changes more transparent, by putting them all in one place.

To ensure the replicability of past results, data-specific functions should be developed using a good system for version control. To make code replicable, changes – particularly those that change the returned values – should be tracked using version numbers, and people using the functions for published work should report the version of the data-specific functions that they use. With good version control, using data-specific functions can improve replicability, by allowing users of the data to track how the functions have changed since a prior use, and by assuring that users of the data always use the most up-to-date version of the data. As the transparency and openness promotion initiative grows (Nosek et al. 2015), citing a particular version of data-specific functions may present another approach to increasing the replicability of scientific results.

Costs

The second concern, costs, refers to the amount of time that users would spend coding the functions.[3] Ideally, creating these functions should reduce the amount of time collectively spent on recoding the data, because they would allow a concise way to distribute tacit knowledge about the data. Practically, however, researchers in sociology are accustomed to hoarding data analysis code. In sociology, academics are primarily rewarded for publishing papers, not writing code.[4] As a result, academic sociologists make code-writing pay by "learning a dataset" – meaning

[3] Another primary cost would be the time and money required to set up the infrastructure for hosting the functions – the version control system and distribution to new users. These costs have declined substantially with the introduction of websites like GitHub, and they are now negligible.

[4] There are exceptions to this rule. In some departments, evaluation for hiring, promotions, tenure, and raises takes code writing into account. Additionally, authors of large-scale, widely used software may benefit from their software, both formally through hiring, promotions, and tenure, and raises, and informally, through reputational benefits among their peers. However, recognition for writing code is not systematic in sociology and is most often given for code tied to one or more substantive publications.

learning the tacit knowledge and writing the code to analyze those data – and publishing from those data repeatedly. With few incentives to write or share code, academic sociologists rarely share their code with others, and may view it as sacrificing their competitive advantage.[5]

To convince academics to contribute to, and use, data-specific functions, the benefit to both the contributors and the users should be emphasized. The value for a user of data-specific functions is clear – correctly analyzing a new dataset is much easier when one takes advantage of the accumulated knowledge about that dataset, which data-specific functions encapsulate. However, a contributor of those functions will find that their value is equally clear. By saving the recoding steps in a function, an analyst frees him- or herself from having to remember all the recoding steps needed every time he or she wants to perform an analysis. This makes working with the dataset quicker and easier, even if he or she knows the dataset's limitations well.

To ensure contributions, data distributors should provide the data-specific functions with the proviso that people who use them must agree to share their code in return. This follows the arrangement in "copyleft" licenses, where anything derived from the licensed product is bound by the same license (Morin, Urban, and Sliz 2012). Ideally, people who use the data-specific functions would add to them when they find the functions lacking, but at minimum, data distributors could incorporate new changes that people build into their code without adding to the data-specific functions.

Publication of the code in a statistical software journal could also serve as an incentive for contributions. Academics publish articles that describe a suite of functions in statistical software journals both to demonstrate the use of those functions and to obtain a professional

[5] Another common reason that sociologists, and scientists of all stripes, do not share their code is that "it is not good enough" (Barnes 2010). Data-specific functions may help this crisis of confidence, by showing that most computer code is not that good. More likely, however, the problem of not sharing code for lack of confidence is one that will have to be addressed in graduate statistics courses.

reward from publishing an article. Much of the functionality for R, in particular, has been developed by academic statisticians who developed their suite of functions into a package, and published a paper or a book describing those functions (e.g., the tidyverse packages were introduced in a series of papers and books: Grolemund et al. 2013; Müller and Wickham 2017; Wickham 2007, 2009, 2011, 2014; Wickham et al. 2011; Wickham and Grolemund 2016). By bundling the data-specific functions associated with a particular dataset into a package, and then publishing the package in a journal such as the *Journal of Statistical Software*, academics who contribute to the data-specific functions could reap the rewards for publishing an additional paper as well.

Propagation of errors

The third concern, the propagation of errors through the functions, must be weighed against the alternative, individual analysts' errors. By virtue of making code easy to use, data-specific functions also make it easy for mistakes in the code to be replicated over and over in analyses.[6] The clearest example of this is the revelation that a bug in the statistical code used to analyze fMRI data likely led to errors in thousands of papers (Eklund, Nichols, and Knutsson 2016). However, this potential problem must be weighed against the alternative, where every analyst writes a different version of the code to clean the data. Different versions of the code are both more likely to contain errors, because each analyst would have to properly encode all of the tacit knowledge about the data, and are harder to correct, because each version would have to be inspected individually. The fMRI software is a case in point; a recent paper was able to suggest a correction for all of the erroneous papers *en masse*, because all of the papers built on the same code base (Kessler, Angstadt, and Sripada 2017). By centralizing the tools, data-specific

[6] I thank Bryce Bartlett for pointing out this potential concern.

functions could speed the propagation of errors, but they could equally easily propagate corrections.

Conclusion

This comment suggests an expansion on Salganik et al. (forthcoming)'s approach to improving the distribution of large survey datasets: data-specific functions. A data-specific function encodes the common recoding, subsetting, or merge criteria into a tool for managing the data. These functions are intended to be tools that don't need a manual to use (as in the Munroe 2014 comic shown at the beginning of this article) – ideally, a new user to the dataset should be able to use the data-specific functions to analyze the data right away. By storing the all of the tacit knowledge about a dataset in a single, immediately usable place, a data-specific function can make it easier to do responsible data analysis.

Works Cited

Barnes, Nick. 2010. “Publish Your Computer Code: It Is Good Enough.” *Nature* 467(7317):753–753.

Csardi, Gabor and Tamas Nepusz. 2006. “The Igraph Software Package for Complex Network Research.” *InterJournal* Complex Systems:1695.

Dowle, Matt and Arun Srinivasan. 2017. *Data.Table: Extension of `data.Frame`*.

Eklund, Anders, Thomas E. Nichols, and Hans Knutsson. 2016. “Cluster Failure: Why FMRI Inferences for Spatial Extent Have Inflated False-Positive Rates.” *Proceedings of the National Academy of Sciences* 201602413.

Elkasabi, Mahmoud. 2018. *DHS.Rates: Calculate Key DHS Indicators*.

Firebaugh, Glenn. 1980. “The Case of the Missing-Values Card, and Other Mysteries: Another Look at the Effect of Government Spending on Income Inequality.” *American Sociological Review* 45(1):137–46.

Grolemund, Garrett, Hadley Wickham, Vitalie Spinu, Imanuel Constigan, Chel Hee Lee, Richard Cotton, Ian Lyttle, and Winston Chang. 2013. *Lubridate: Make Dealing with Dates a Little Easier*.

Handcock, Mark S., David R. Hunter, Carter T. Butts, Steven M. Goodreau, Pavel N. Krivitsky, Skye Bender-deMoll, and Martina Morris. 2016. *Statnet: Software Tools for the Statistical Analysis of Network Data*. The Statnet Project (<URL: http://www.statnet.org>).

Handcock, Mark S., David R. Hunter, Carter T. Butts, Steven M. Goodreau, and Martina Morris. 2008. “Statnet: Software Tools for the Representation, Visualization, Analysis and Simulation of Network Data.” *Journal of Statistical Software* 24(1):1–11.

HRS. 2018. “Health and Retirement Study, (RAND Core Income and Wealth Imputations v.P) public use dataset.”

Kessler, Daniel, Mike Angstadt, and Chandra S. Sripada. 2017. “Reevaluating ‘Cluster Failure’ in FMRI Using Nonparametric Control of the False Discovery Rate.” *Proceedings of the National Academy of Sciences* 114(17):E3372–73.

Morin, Andrew, Jennifer Urban, and Piotr Sliz. 2012. “A Quick Guide to Software Licensing for the Scientist-Programmer.” *PLOS Computational Biology* 8(7):e1002598.

Müller, Kirill and Hadley Wickham. 2017. *Tibble: Simple Data Frames*.

Munroe, Randall. 2014. “Manuals.” *XKCD*. Retrieved November 12, 2018 (https://xkcd.com/1343/).

Nosek, B. A., G. Alter, G. C. Banks, D. Borsboom, S. D. Bowman, S. J. Breckler, S. Buck, C. D. Chambers, G. Chin, G. Christensen, M. Contestabile, A. Dafoe, E. Eich, J. Freese, R. Glennerster, D. Goroff, D. P. Green, B. Hesse, M. Humphreys, J. Ishiyama, D. Karlan, A. Kraut, A. Lupia, P. Mabry, T. Madon, N. Malhotra, E. Mayo-Wilson, M. McNutt, E. Miguel, E. Levy Paluck, U. Simonsohn, C. Soderberg, B. A. Spellman, J. Turitto, G. VandenBos, S. Vazire, E. J. Wagenmakers, R. Wilson, and T. Yarkoni. 2015. "Promoting an Open Research Culture." *Science* 348(6242):1422–25.

Osgood, D. Wayne, Mark E. Feinberg, Scott D. Gest, James Moody, Daniel T. Ragan, Richard Spoth, Mark Greenberg, and Cleve Redmond. 2013. "Effects of PROSPER on the Influence Potential of Prosocial Versus Antisocial Youth in Adolescent Friendship Networks." *Journal of Adolescent Health* 53(2):174–79.

Owen-Smith, Jason, Julia Lane, and Bruce Weinberg. 2017. "The Institute for Research on Innovation & Science (IRIS). Full Documentation for UMETRICS 2016Q3a Dataset."

R Core Team. 2017. *R: A Language and Environment for Statistical Computing*. Vienna, Austria: R Foundation for Statistical Computing.

Spoth, Richard, Mark Greenberg, Karen Bierman, and Cleve Redmond. 2004. "PROSPER Community–University Partnership Model for Public Education Systems: Capacity-Building for Evidence-Based, Competence-Building Prevention." *Prevention Science* 5:31–39.

Spoth, Richard, Cleve Redmond, Chungyeol Shin, Mark Greenberg, Scott Clair, and Mark Feinberg. 2007. "Substance-Use Outcomes at 18 Months Past Baseline: The PROSPER Community-University Partnership Trial." *American Journal of Preventive Medicine* 32(5):395–402.

Wickham, Hadley. 2007. "Reshaping Data with the Reshape Package." *Journal of Statistical Software* 21(12):1–20.

Wickham, Hadley. 2009. *Ggplot2*. New York, NY: Springer.

Wickham, Hadley. 2011. "The Split-Apply-Combine Strategy for Data Analysis." *Journal of Statistical Software* 40(1):1–29.

Wickham, Hadley. 2014. "Tidy Data." *Journal of Statistical Software* 59(10):1–23.

Wickham, Hadley, Dianne Cook, Heike Hofmann, and Andreas Buja. 2011. "Tourr: An R Package for Exploring Multivariate Data with Projections." *Journal of Statistical Software* 40(2):1–18.

Wickham, Hadley and Garrett Grolemund. 2016. *R for Data Science*. Sebastopol, CA: O'Reilly Media, Inc.