

# Helpful Advice on Computerized Data Handling in Social Surveys

Fritz Scheuren

In a February 2003 exchange on SRMSNET, a LISTSERV sponsored by the Section on Survey Research Methods, there was a thread on computerized data analysis, "entitled *Respect Your Data*. Ben Earnhart, Department of Sociology, College of Liberal Arts, University of Iowa, began the exchange with about 20 items. One of the other contributors suggested that the thread be published in *AMSTAT News*. Ben agreed to summarize his material, integrating it with some of what he got from others. I asked him to be brief for space reasons and to keep to his informal "e-style." What he came up with are the 23 ideas shown below. Even though aimed at graduate students the items may be helpful reminders to all of us. Incidentally, by a nice coincidence the February 2003 issue of *The American Statistician* has a companion piece by Vardeman and Morris, entitled *Statistics and Ethics; Some Advice to Young Statisticians*.

## Respect Your Data

Ben Earnhart

1. *Use syntax*. It is OK to use point-and-click to create syntax in programs such as SPSS that allow it, but always run the syntax, and save it. You should also read the manual to understand what the syntax means. It may seem quicker to rename a variable by point-and-click, or do a simple re-code this way, but if you have to do it repeatedly, or worse, have to replicate what you did, it is *not* quicker. Reliable and replicable point-and-click manipulation of data would require that you are always right and have a 100% perfect memory. Only your advisor and departmental secretary have these abilities
2. *Use comments liberally in your syntax*. You should have a comment at the beginning of each syntax file explaining when you wrote it, what it is supposed to do, and what files are related; each time you switch to a new basic task within a given syntax file, you should make a comment. When doing particularly complicated manipulations, you may even want to put a comment for each line of code. Refer to specific pages of the codebook whenever possible so that you are never stuck having to say to yourself (or worse, to your advisor) "I don't know why I did that." A good rule of thumb for how verbose your comments should be is to ask yourself "What if I were to set this down right now and not think about it for six months? What would I need to say here, so I can quickly figure out what I was doing?"
3. *Have a naming convention for your files*. You normally want the data, syntax, and log files to have the same name, or similar enough you know at a glance what goes with what. Relating the names to the purpose can be helpful (mergekids, factorscores, maritalcoding, etc.) but whatever convention you use, stick with it consistently.
4. *Save your log files*. Save your log files and the output you generate, as well as the syntax used to generate it. If you ever have to say "I don't know how I got these numbers" to yourself (or worse, your advisor), you have a big, big problem.
5. *Use smaller files if you can*. Related to the naming convention, when doing things in

multiple steps, it is generally best to have things in smaller files, so if you goof up, you do not endanger all your work. Generally each step might have the same name but a different number on the end (recodes1, recodes2, recodes3, etc.) so you can easily backtrack. Also, this can greatly speed up your jobs -- it is *not* necessary to read the data in from ASCII and do 100 pages of recodes every time you do an analysis.

6. ***Analysis and data manipulation should be separate steps.*** Analysis and data manipulation should not occur in the same step -- at the end of the data manipulation phase, save to a systems file; then read it in for analysis. Not only will this speed up your jobs, it will prevent you from inadvertently doing analysis on data that has changed.

7. ***Discard unnecessary cases early.*** When working with subsets of a large file (for example, the cumulative General Social Survey (GSS), which is a repeated cross-section (although most people only use a single year for a given project), or Census data (for which most people only care about a single level of analysis), discard as many cases and variables as early as you can and save as a separate file. There is no need to load up 200,000 cases and 5,000 variables every time to work with 2000 cases and 50 variables.

8. ***Keep a data diary.*** Keep a journal/diary of what files you worked with and what changes you made -- update it each day you work. That way, if you need to backtrack, you can review the whole project at a glance. A notebook is good for small projects. For larger projects or ones in which you are collaborating with others, keeping a file for logging changes can be very important. This also has the side benefit of showing your advisor how hard you are working every day -- even if the changes you make turn out to be wrong, it proves you were working hard at being wrong.

9. ***Employ a directory structure?*** For large projects, use a directory structure to stay organized. With large projects, files can multiply fast, and lead to confusion, unless you use sub-directories. Have a journal/diary for each sub-directory, although if the one at the top level is complete, it is OK to just have the one.

10. ***Network drives are slower than local drives and a better workstation will never speed up a server.*** Network drives are slower than local drives -- getting a faster processor will not help if it is starving for data to process, so do not complain the computer is too slow when it is the data access that is slowing things down. For using large datasets efficiently, copy your data over locally. When done, copy things you want to save back to the network drive. Of course, if you are running the job remotely on a server, this is not relevant -- in this case what you need to speed up your jobs is a bigger monitor, better headphones, and a more comfortable chair.

11. ***Back things up.*** Back things up! Syntax files and output files are generally very small in size, so there is **NO** excuse for not backing them up to multiple locations. This will not only save you from losses due to things like hard drive failures, but also from inadvertently over-writing an important file. Back up systematically, so you know what version of a file you are working with. A handy way to do this is to create compressed (zipped) files with the day's date as the filename, so you know what is what.

12. ***But do not go crazy with backing up large data files.*** As a caveat to the principle of backing up often, do ***not*** go “crazy” with backing up large data files -- filling up a shared drive with numerous identical or nearly identical copies of the same data is absurd and can irritate your computer techs to the point where they start imposing disk space quotas -- and this will make you rather unpopular when other users find out why quotas are being imposed. If you practice good habits with your syntax, you can quickly and easily replicate the data with the syntax, if needed. So keep a copy of the original data safe, and your few most recent versions. Keep more versions around only you anticipate going back to them often. Having the two most recent versions available is handy, because you will accidentally over-write your most recent one and have to back up a step. I almost put a section called "do not over-write the input file," but realized it was pointless, because it is just one of those accidents that is inevitable, so be ready -- since you followed good documentation and used syntax, there is no reason to panic.

13. ***Never recode into the same variable name.*** Never recode into the same variable name if you can avoid it (with the possible exception of setting missing values) -- when a variable means one thing at one stage of a project, and another thing at another stage, it can really mess things up. When creating new variables, attach variable labels unless the meaning is obvious enough from the variable name; however, note that the only test for whether it is "obvious enough" is if your advisor agrees that the name adequately reflects the variable label you attached. With dummy variables, consider putting what a value of 1 means in the variable label so you (and perhaps more importantly, others) know at a glance what direction it is coded.

14. ***Read The Manual.*** Read the data file codebook, especially be aware of skip patterns, and never assume a variable means what you think it does, based on just the name and label. Never assume a syntax command will do exactly what you expect, either, and be aware of switches/options for modifying the behavior of a command. This will not only help you to avoid outright mistakes, but can let you get things done with much less effort. It will also save you from great embarrassment for those times when you do need to get help from others.

15. ***Be careful about missing value codes.*** Be careful about missing value codes. Sometimes the reason a variable is missing can be very, very important -- be aware what the different missing value codes in your data mean. This will help you to avoid pregnant men and people who pay for the privilege of going to work (e.g., have negative hourly wages).

16. ***Always run a sanity check on a constructed variable.*** After constructing a new variable, always run a sanity check on it to make sure it did what you thought it did; cross-tabs against the original variables can be a good way to do this. Pay attention to both the meaning, and the Ns. For a series of dummy variables that are supposed to be mutually exclusive and completely exhaustive, summing them should add up to 1; this check on your coding is probably the only time a variable with a variance of zero is a good thing. Do occasional sanity checks with “descriptives” (descriptive statistics) on all the variables -- this will help you identify those pregnant men and bored rich people early on. You should comment out

these sanity checks after looking the output over carefully; otherwise you will get 100 pages of output each and every time. Still do not cut them out altogether. Keeping them allows you to go back and turn them back on if you need to diagnose a new problem. It also shows your advisor what a good researcher you are.

17. **Run “descriptives” before every analysis** As well as helping you to interpret the output from the main analysis, routinely running descriptives can act as a sanity check -- mins and maxes are good for catching missing value codes that slipped through, and can help diagnose problems with your N. In addition, a variable with a variance of 0 is going to not be very useful in your analysis. Actually, if it has a variance of zero, it is called a constant, but whatever you call it, it is just not going to work.

18. **Test untried commands on small datasets.** When doing commands that you are not sure are right, do not start by using huge datasets; use options to only access a portion of the file. Do not run it on the full N until you have reason to believe you got your syntax right -- waiting 5 minutes for each run to get the same error over and over is just plain silly (though it does give you a chance to browse the web and complain that you need a faster computer).

19. **Build in checks on mathematically impossible transformations.** Build in checks on mathematically impossible transformations. When diagnosing an error, consider what it is you are trying to do, and read the log. You definitely should check for mathematical impossibilities. "Division by 0 is impossible" you may have forgotten about since high school math, but it is still just as impossible today as it was then, and your stats package will be happy to remind you of this fact, *if* you read the log. Square roots of negatives... well, while not technically impossible, you better have a heck of a theory to explain why you can and will do that, and program your own software to do it. And no matter how good your theory is, logarithms of negative numbers are problematic as well.

20. **Use value labels.** Real researchers such as yourself do not use value labels -- they know instinctively that 0 means employed, 1 means unemployed, and 3 means out of the labor force. However, when you go to show output to people who are not blessed with your incredible instinct (such as anybody other than you), value labels can be a blessing. So use them out of consideration to others.

21. **Pick a syntax convention, and stick with it.** Pick a syntax convention, and stick with it. Sure, NE, ~=, and <>, are interchangeable in some packages, but that does not mean you should do use them that way. When in doubt whether you really need parentheses, you need parentheses. Even if the statement is constructed in such a way that the computer will understand you just fine, being consistent and using parentheses help if a human ever needs to look at your code. Indentation can also help the human eye comprehend your syntax. You may claim that your advisor is super or subhuman, but for this purpose, he or she counts as human.

22. **Does this make logical sense?** Even if your code ran without errors, and you tried to follow all the steps above, look at your output and say to yourself "Does this make logical sense?" If you have discovered that a year of education decreases income dramatically, or

that cigarette smoking causes longevity, you might be in line for a Nobel prize. Or, you might be the victim of those pregnant men and bored rich people – mistakes are really sneaky, and sometimes slip by despite all your efforts. But at least, if you followed the steps above, you will be in a much better position to backtrack and either figure out where you goofed up, or prove to the world that cigarettes really are good for you.

23. *Respect your data subjects.* Never forget that you are dealing with humans, even if they come on a CD-ROM. There are ethics that apply when working with human data. Remember that people opened up their souls to you in providing the very data you are analyzing. Some of these ethics have been codified into rules. Know those rules. Be in full compliance with your Institutional Review Board (IRB) and other applicable bodies, including the ASA.

This list is not meant to be exhaustive, but merely to help researchers early in their data encounters avoid some common errors. Graduate students using social survey data seem to learn many of these ideas the hard way. Do these tips have to be rediscovered in every generation? Let's hope not.