Below is the same as the *AMSTAT* version as edited by Fritz Scheuren, but annotated and credit given to the many people who made helpful suggestions. The only change to the body is that the final item about ethics has been re-worded per the suggestion of Michael Oakes. I also added a section that includes recommendations that did not make it into the body of the piece for various reasons.

1. USE SYNTAX.

2. USE COMMENTS LIBERALLY IN YOUR SYNTAX.

3. HAVE A NAMING CONVENTION FOR YOUR FILES.

4. SAVE YOUR LOG FILES.

5. USE SMALLER FILES IF YOU CAN.

6. ANALYSIS AND DATA MANIPULATION SHOULD BE SEPARATE STEPS. **2**

7. DISCARD UNNECESSARY CASES EARLY.

8. KEEP A DATA DIARY.

9. EMPLOY A DIRECTORY STRUCTURE?

10. *NETWORK DRIVES ARE SLOWER THAN LOCAL DRIVES AND A BETTER WORKSTATION WILL NEVER SPEED UP A SERVER.*

11. BACK THINGS UP.

12. BUT DO NOT GO CRAZY WITH BACKING UP LARGE DATA FILES.

13. NEVER RECODE INTO THE SAME VARIABLE NAME.

14.READ THE MANUAL.

15. BE CAREFUL ABOUT MISSING VALUE CODES.

16. ALWAYS RUN A SANITY CHECK ON A CONSTRUCTED VARIABLE.

17. RUN "DESCRIPTIVES" BEFORE EVERY ANALYSIS

18. TEST UNTRIED COMMANDS ON SMALL DATASETS.

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.[1]

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?[2]"

3. Have a naming convention for your files.[3]

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.)[4] but whatever convention you use, stick with it consistently.

4. Save your log files.[5]

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[6].

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[7]. 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 up8.

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,"**9** 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**10**.

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*11*.

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**12**. 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?*13*

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 may be dealing with information about real people (e.g., Moms, Grandfathers, Sisters), even if such information comes on a CD-ROM or is downloadable. The fact is that people opened up their souls to you in providing the very data you are analyzing and there are ethical principles that apply when working with such data. For example, it's best to minimize risk to subjects by de-linking personal identifier information from other data; data containing personal identifiers should always be encrypted. In general, know the rules and be in full compliance with your Institutional Review Board (IRB) (see http://ohrp.osophs.dhhs.gov/polasur.htm) and other applicable bodies, including the ASA (see http://www.amstat.org/profession/ethicalstatistics.html#guide)

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 it in every generation? Let's hope not.

Additional suggestions and comments.

I have received much feedback on the piece with much good material that I intent to bring into the body of a later iteration of this site. In the meanwhile, here are ideas that I didn't integrate yet. I apologize if my summary mangles the intent of the emails I received – please contact me if you feel this is the case.

- There were numerous people (including Dennis Affholter and Lise DeShea) who suggested the use of graphical representations of the data, both as an exploratory tool, and as a way of checking for bad data such as missing value codes.

- Susan Carol Losh made an excellent point – that "there is a tendency to reify secondary data sets and to forget that all the problems that were there when the data were collected are STILL there when they are archived. In fact, it's worse because archives vary in giving you information about how the data were collected in the first place." She also provided a link to some detailed discussion she had made in one of her courses of numerous issues regarding secondary data, and research in general.

- Wendy Watkins and several others noted the importance of the proper use of sampling weights.

- Jean Norris made the pithy statement that “Being brilliant is no guarantee against being stupid!” Nice general summary of everything!

- Steve Simon provided links to his own pages that discuss many issues related to treatment of data, as well as many helpful tips for the use of SPSS and statistics in general. He also made the additional suggestions below:

  1. Never, ever let blank fields creep into your data. Blank fields are ambiguous, as they sometime represent missing values and sometimes represent incomplete data entry. If information is missing, put in a code for missing and distinguish among the different reasons for missing. Blank fields are especially tricky if you are importing from another system. During conversion blanks are sometimes converted to zeros. Even worse, sometimes blank fields end up being replaced by the number directly to the right throwing off all of the remaining data in that row.

  (This could be incorporated in 15. Be careful about missing value codes)

  2. Always enter and display dates using four digit years, and check any date calculations carefully. The Y2K bug was grossly overhyped, but we statisticians will be haunted with residual problems from Y2K for several decades. It is easy for the computer to put a date in the wrong century, and displaying four digit years will catch a lot of surprising errors. Be especially careful when you import dates from a different program, as every system seems to have a different set of assumptions relating to dates. Also look for outliers, like infants who are 101 years old or -99 years old.

  3. Revise your tables and graphs at least as often as you rewrite your text. I call this the "fault of default" principle. Don't take the first graph that you draw; tinker with the axes, scaling, symbols, etc. For crosstabs, consider swapping the rows and columns and see whether the row percents or the column percents make more sense. Using rounding liberally.

  4. Don't jump in with the most complicated model first. Know the frequencies for all of your important categorical variables, and the ranges of all your continuous variables. Look at crosstabs, boxplots, and scatterplots before you do any inferential analyses. Your first statistical models should be as simple single variable models, and gradually make the models more complex. Look for consistency between the graphs and your simple models and explore any discrepancies. Then look for consistency between your simple models and your more complex models and explore any discrepancies. If there are no discrepancies, that's good news, because the complex models are consistent with what the simple models and the graphs told you. And if there are discrepancies, that's also

good news. Something interesting is going on, such as one variable suppressing the effect of another--this is the process of discovery.

(This could be incorporated into 18. Run descriptives before every analysis.)

Here are some suggestions for your existing text.

In SPSS, an alternative to running syntax is to save the syntax as part of the output. Click on EDIT | OPTIONS from the menu, and click on the VIEWER tab. Make sure that the DISPLAY COMMANDS IN THE LOG option box is check.

Your points about backing up are well taken, and you might want to stress the cost of a couple of floppy disks versus an hour of their time (even at minimum wage, there is no comparison). Make an extra copy of any really important file and keep it off-site. Having three copies of a file on the same hard drive does nothing to protect you against disk failure. Finally, it helps to know the backup policy on network drives and who to contact to get files restored.

- John Weisberg made the following points about being careful during the data gathering stage:

  1. Think about analysis during questionnaire design. Part of pre-testing a questionnaire should be pre-testing the analysis. That helps reveal such problems as ambiguous questionnaire wording or response categories, unnecessary questions (i.e., that produce data you can't or won't analyse), and similar problems. Testing analysis at this stage produces a more relevant and accuarate questionnaire for both the respondent and the analyst.

  2. Track all changes to questionnaire design! Few things muck up analysis more than not being able to accurately identify when changes in the questionnaire took place, and what they were. Adamantly record theses changes when they occur - it may seem a trivial thing to remember but you won't be able to 3 or 6 months later when you're trying to figure out what the data means and how to use it! Relates to the issue you mentioned in your last post about how much to document the project.

- Arthur Kendall had an over-arching conception of errors arising out of the research process because of “human factors” and felt that it was important to manage projects with that in mind. For now, I’m evading any sort of “theory” or way of integrating things, but when it comes time to do so, his ideas seem very good. Here are some of additional suggestions he made that didn’t make it into the piece:

The team that is going to use the results should concur with the dictionary before analyses are done.

Someone should do a complete quality assurance review of the syntax.

"Pretty" your syntax. Don't be afraid to use more lines so that the logic of what you are trying to do is apparent from the shape of the syntax. e.g., left indent the first line of a procedure unless it is nested, use + in column 1 to allow nesting of statements.

Full univariate and extensive bivariate exploratory analysis should be done before the final analyses.

Remember data prep and cleaning can easily be 90% of all the people time.

Always leave menus via <paste> rather than <ok>.

Remember that building your application will be an iterative process of trial and error. This also facilitates documentation, debugging, and quality review.

- Gene Shackman forwarded the list to several other list-serves, producing much additional feedback. He also has his own site, which is full of additional resources on related topics.

- Rossi Hassad mentioned an additional resource, a presentation paper he made that covered some of the issues: "Link & Think - A Model For Enhancing The Teaching And Learning Of Introductory Statistics In The Behavioral Sciences" (ASA - JSM2002 CD-ROM). I haven't had a chance to see it and bring his ideas in, but intend to do so.

---

1 Johannes Hüsing made the additional comment that "Plus, it is much easier to get remote help if you email your code to your helping hand than to describe the sequence of submenus you iterated through." Conversely, Stephen Baker noted that for many circumstances a GUI is *better* than syntax. Among other things, he stated: "I've benchmarked complex analyses using both methods, the GUI was more than 10 times faster with [fewer] errors than syntax." I'll agree that when GUI tools are available, they can be a boon, either for preparing syntax or for carrying out tasks like data extraction. If somebody can do the same job and document it just as rigorously using a GUI, then more power to them – I'm not advocating we go back to the days when all manipulations were made on text-only mainframes (much as I sometimes miss those days). So maybe it should read something like "Use syntax unless a GUI tool can provide a similar or higher level of data integrity and documentation."

2 Thanks to Thomas E. Nichols for this wonderful heuristic for deciding how much documentation is appropriate.

3 Johannes Hüsing made a suggestion that the directory structure is as important or more important than the file names – relying on file names for meaning could actually make things more confusing. I can imagine that on really large projects this might be the case. Main point I think, though, is to have *some* sort of way to keep track of what file does what, and whatever system you use, use it consistently and make sure it makes sense to your advisor or others who might be involved in the project.

4 Suzanne Marie Dorinski made this suggestion, along with a great example from her own experience. "One project that I had to take over had programs named a.sas, b.sas, c.sas, aa.sas, ab.sas, etc. I never did figure out what order the programs were supposed to run, and ended up after several frustrating hours starting over from scratch.

5 Wooksoo Kim added a caveat to this – that while saving syntax is immensely important, saving output can at times be more trouble and space than it's worth, and can always be generated again if you have the data and syntax.

6 Johannes Hüsing made the concrete suggestion that "In SAS, I recommend to save versions of data files into different directories with different version numbers, then on an new version rephrase the libname statement to make the 'common' libname always point to the most recent version. That way, the libname statement is the only piece of code you have to change."

7 Stephen P. Baker pointed out that this and some of the other items were only relevant to some projects under some circumstances.

8 Johannes Hüsing made a comment that combined this and several other points into some general advice. Backups on a network system should be the responsibility of the network admin, so the user should not have to worry about them; related to this, my advice to copy things over locally because they run faster that way should carry the warning that the local copy does *not* normally get backed up. In comparing items 5, 7, 11, and 12, he states "it is very hard for an inexperienced user to devise a strategy of keeping files from these single rules, which are all right but contradict each other when taken beyond a certain point." This is quite true, and I'm not sure exactly how to handle it – to keep things simple and yet reasonably complete.

9 Martin Levin and Arthur Kendall both suggested that students should be warned not to over-write their original data, and that SELECT IF in SPSS can be dangerous.

10 Arthur Kendall suggested this, along with many other tips.

11 Another from Arthur Kendall. I'd add the caveat that depending on the software and how one goes about using the software, value labels can be a blessing or a curse; they obscure the actual codes in SPSS output, and in SAS can make moving and storing data more complicated. But for data that is meant for showing others, they make interpretation much, much easier, so they certainly should be applied at some stages.

12 Another from Arthur Kendall. He actually advocated that users stick with the alphabetic comparison operators (NE, GT, LT, etc) rather than the symbolic ones, such as <>, >=. I have tended to prefer the symbolic ones, so I left the point as simply be consistent. He later gave some very good reasons for his preference, the two most compelling to my mind being that dyslexics have an easier time with them, and that it avoids confusion with the assignment operator (setting a value, rather than making a comparison). Above, I'm leaving it in the same, but will likely go with his position in a later version.

13 Thanks to Jennifer Popovic and Dennis Affholter.