UNIVERSITY OF ILLINOIS CHICAGO

Data Management

ENT Resident Seminar

Michael L. Berbaum, Ph.D.

June 2022

The below guidance on data management was developed by Michael L. Berbaum, Ph.D., of the CCTS Biostatistics Core. For more information on this topic, including advice about how to apply it in your research, consider scheduling a consultation with a biostatistician.

Please contact us with any comments or corrections.

Introduction

We usually separate *data management* from *data analysis*. The ultimate goal is to produce a data analysis on which a report of study results can be based. For quantitative research the data analysis is almost always carried out using a statistical program or package like *SPSS*, *Stata*, *SAS*, or *R*. The goal of data management is to prepare the data for analysis by one of these packages, to make the data *readable* and *correct*.

Task 1: Data Entry

Case 1: You have data on paper and need to make it digital.

1. Need to know a target data format (CSV from spreadsheet, fixed columns with format specified) that the statistical package will be able to read. Nowadays most statistical programs can read most formats. But if you plan to use *SAS*, that sets a certain workflow; if you plan to use *R*, the workflow may be slightly different.
2. Need to have a software tool which will make typing the data less error-prone. For example, range-checking to be sure an age of 213 years cannot be entered.
    a. *SPSS* has a data entry module (if still available) that's pretty good
    b. One can use the *REDCap* system: just program it like a questionnaire, and then type in the data as if you are administering a questionnaire to yourself. Has various supports like range-checking and branching.
3. *In extremis* use a plain text editor (also called an *ASCII* editor or a *programming* editor). On Windows there's a built in one called *Notepad*; for something a bit nicer, download *Notepad++*. There are dozens of these (*Nano*, *Emacs*, . . .). What you do *not* want is a wordprocessor like *WordPerfect* or *Word*: Those put in hidden codes that will ruin data entry. Again, formats like fixed columns and CSV (comma separated values) are the simplest.

Case 2: You have a digital file of some kind.

1. If the file is in a standard format that statistical programs can read, you're done!

2. Sometimes lab devices or certain complex databases use an arcane format. Then you need to use a *programming language* to read that format and write the data out into a standard format (e.g., fixed columns, CSV). You might want to hire this done rather than learn to do it yourself, though it's not that that hard. Another option is to use a *database language* like SQL (pronounced either S-Q-L or "sequel"). Consult the CCTS Bioinformatics Core for best practice.

At the end of this step you should have a digital computer file in a standard *readable* format.

Task 2: Read the Data into Your Statistical Package [Script 1]

Script 1 syntax file reads the raw data. If you save the output, give the file a name like that of Script 1. It is ***very wise*** to include dates and times in file names, such as `ReadRawData_2022-06-15_1600.sas` or `ReadRawData_2022-06-15_1600.R`. Here one does not need to save the output.

The goal here is just to be sure the package can read your raw data file. If the package throws errors, you'll need to fiddle with the choice of format for reading the data file. You want to be able to see *within the statistical package* that the numbers in your raw data input file are displaying correctly. Check that you have the right number of rows – those are the cases, patients, respondents. Check that you have the right number of columns – those are Record IDs and variables you have measured or questionnaire responses. Almost always your data set will be *rectangular* like this.

This step is done when what's showing inside the statistical package matches the content of your raw data file. This confirms that the raw data file is *readable*.

Task 3: Label the Data [Script 2]

Script 2 contains Script 1 plus added syntax to label the data. It is ***very wise*** to include dates and times in file names, such as `LabelRawData_2022-06-16_1600.sas` or `LabelRawData_2022-06-16_1600.R`. You still have additional syntax to write, so you do not need to save the results yet. It's best if Script 2 has a different name from Script 1.

So far your data is unlabeled, or may be called *V1, V2, . . .* or *X1, X2, ...* This is terrible because you don't really know what you're looking at, and it will be hopeless to try to remember that *V29* is O2 saturation. You need to apply two kinds of labels:

1. *Variable labels* like `O2 sat` or `Age`. (These will show up all through your later statistical output: Keep them short. Capitalize. Don't use a tricky naming scheme – if you have to explain it for someone else to know what the variable is, it's *tricky*.)

2, *Value labels* like `low`, `medium`, and `high` that signify the meaning of categorical variables that might in the raw data be coded as `1`, `2`, and `3`. Generally set it so increasing number codes indicate increasing amounts. If you have an indicator or "dummy" variable that is coded with 0's and 1's, say 0 = `Male` and 1 = `Female`, apply those value labels, and adopt the variable naming convention that whatever the label for `1` is shall also be the name of the variable; so the variable name is `Female`. If you call the variable `Gender` you'll have to remember who the 0's and 1's are or look it up.

The labeling syntax differs between statistical packages. It's unavoidable that you'll have to learn how it's done from your statistical program's *Manual*.

Run the syntax for labeling on your raw data. Now check that the labels got applied as you intended. A good way to do that is to run the *frequencies* command that's built in to every package. It should show you the variable labels and the value labels. Take time to be sure nothing got twisted up: You'll be working with these labels from now on, so take time to inspect the result and be sure it's right.

Aside for *REDCap*

If you use the **R**esearch **E**lectronic **D**ata **cap**ture system (*REDCap*) to enter or to collect your data, you have already supplied the variable and value labels when you set up the data entry forms. **D**on't **R**epeat **Y**ourself

(DRY)! REDCap allows you to download your raw data (often in CSV format) and to download a file *for your chosen statistical package* that already contains the proper syntax to label the data. Use that, and skip the duplicative re-labeling step entirely!

Aside for Longitudinal Data

Longitudinal or repeated measures data can call for some special handling. Consult the CCTS Biostatistics Core or your local *guru*. There are a few techniques to use when setting up a longitudinal data collection that will save time and effort later.

Task 4: Cleaning the Data

At this point you have been able to read in your raw data, label it nicely, and check that the labels ended up where they should have. The next task is always called *cleaning*. In effect you already started doing this when you ran the *frequencies* command. Before when you did that, the focus was on correct labeling. Now we run the command again, perhaps requesting more detailed output (e.g., a selection of Min, Max, Mean, Median, Quartiles, Standard Deviation, Inter-Quartile Range or IQR). **Look for *Weirdness*. Look for the *Impossible*.** Take notes on everything you notice.

1. Look at the ***Extremes*** of each variable. With a sample of children, there should not be any aged 43; with a sample of adults, there should not be any 5-year-olds. If you have coded your rating scales 1-to-5, there should not be any 0's or 6's and 7's. If there are a lot of cases (a *spike*) piled up at the Min or Max, that's an indication of floor and ceiling effects (the allowed range of response was too narrow, or respondents adopted a *response style* of going to one extreme or the other).
2. Watch out for miscoding or mishandling of ***missing data***. Some statistical programs (*SPSS*) conventionally used `-9` for missing data, or `-97`, `-98`, and `-99` for different kinds of missing data (e.g., could not contact respondent, respondent refused to answer, etc.). If you don't declare these codes to signify missingness, the program will just assume they are valid data and use them with all the rest of the data in the analysis. The result of that is *not good!* Some statistical programs have special functions to examine the patterns of missing data (e.g., *R*) that are quite helpful.

For more tips and resources, see our Data Cleaning page.

Task 5: Repair the Data [Script 3]

Script 3 contains Script 2 plus added syntax to modify and save data values. It is ***very wise*** to include dates and times in file names, such as `RepairData_2022-06-15_1600.sas` or `RepairData_2022-06-15_1600.R` The saved data files names should again include dates and times, such as `RepairData_2022-06-15_1600.sas7bdat` or `RepairData_2022-06-15_1600.Rdata`. It's best if Script 3 has a different name from Script 2. These files can be read in by later scripts if you choose (which makes later scripts shorter). Many workers will just choose to expand Script 3 to make Script 4 (with a new name), and not save the built data set yet.

What if you find a mistake? The fix is usually to replace (selected portions of) the bad data with correct data, if you know it. If you don't know what the respondent actually said or what the correct measurement was, the replacement will likely have to be the missing data code (e.g., in *R* it's `NA`, in *SPSS* it might be `-9`, in *SAS* it's a period `.`).

At the end of Task 5 you should have a data file that is *correct*, or nearly so.

Task 6: Recode or Transform the Data and Save [Script 4]

Script 4 contains Script 3 plus added syntax to create new variables. It's best if Script 4 has a different name from Script 3. The next variables might be recodes of existing variables, such as replacing a 1-to-5 rating scale by grouping the outer values: `(1, 2), 3, (4, 5)` becomes `1, 2, 3`. The new variable should have a new name that relates to the original variable's name and shows what was done: `Depression5` is recoded as `Depression3`. ***Always keep the original variable.*** A variable might need to be log transformed to reduce skewness in its distribution: `Income` is transformed into `logIncome`. This is looking ahead to the data analysis phase.

It is the nature of this work that there will be additional additions to Script 4 as new analyses and interpretations are tried out. It's tiresome to have to keep choosing new names for scripts that are only modestly changed (i.e., added another recode). At this point you might start calling your scripts something like `MasterBuild_2022-07-04_1234.sas` and the saved data file `MasterBuild_2022-07-04_1234.sas7bdat`. (For *R* this would be `MasterBuild_2022-07-04_1234.R` and the saved data file `MasterBuild_2022-07-04_1234.Rdata`.

Incredibly Important Point about Dates in MasterBuild File Names

Putting dates and times in `MasterBuild` file names pays a big dividend when you reach the data analysis stage. When you write an analysis script, there should be some header documentation at the top: Project name, who wrote (this) syntax file, when started, purpose, short explanation of technique used, changes, etc. And then the next thing you invariably will do is read in the latest `MasterBuild` file to get at the data. To do that you have to type the name of the file to read, something like `MasterBuild_YYYY-MM-DD_HHMM`. Doing that means you know down to the minute **which version of the data was used to produce this batch of analytical results**. The syntax of this *read* statement will vary with different packages, but whichever the package, it has to find the saved data file, and if the saved data file contains dates and times, you have automatically documented the exact version of the data used for this set of analyses.

If you work with *REDCap* and download its *SAS* or *R* build files, here's a ***major tip***. Open the file with extension `.sas` or `.R` that you downloaded from *REDCap* and find the spot where the built file is saved. That file will be called `redcap`. Edit the name of the file to be `redcap_YYYY-MM-DD_HHMM` (fill in today's date). You can also change `redcap` to something more informative, like `REDCap-MasterBuild_YYYY-MM-DD_HHMM`. This guarantees that any file trying to read the saved built data ***must employ the date and time*** to read the file, thereby *automatically* ensuring this information is included in your scripts and outputs. There will be a huge payoff later when the journal editor sends a "revise and resubmit" and you have to locate which version of the data was used in the analysis.

Other Data Management Principles Not Elsewhere Mentioned

1. **Never throw anything away!** Disk space is abundant and cheap. Any time you save a file, just put a new date and time in the filename. Yes, you will have some copies of rather similar files. So what? ***You will not lose any of your work!***
2. **Always save original and changed versions of variables** This is an application of the principle **Never throw anything away!**
3. If you find your working subdirectory (folder) getting cluttered with too many old files that you're not using, make subdirectory called `Archive` or `Posted Files` and move the excess files there. Now they are out of your way, but you still have them all. *What principle would this be?*
4. Think about the file hierarchy for a project. You'll need subdirectories for `RawData` and `MasterBuilds` and `Images` and `Essays`. Plan this out. Discuss it with your workmates and be sure everyone agrees and understands what goes where. Write it down so new team members can learn it easily.
5. The **Ultimate Goal** of data management is that it's possible to rebuild everything by running one script, or a short stack of scripts. In minutes. If you can't do that, you still have to improve your workflow and its reproducibility.
6. You cannot rebuild what you do not have. **Back up everything often!** (Your system may provide automatic backups, but be sure. Most cloud storage solutions allow recovery of lost files, so consider using *Box* or *DropBox* or *GitHub*. (Be sure all HIPAA rules are complied with.)

Resources

Assessment Capacities Project (ACAPS) (2016). *Data Cleaning*. 19 pp. PDF, many links. https://web.archive.org/web/20230628185233/https://www.acaps.org/fileadmin/user_upload/acaps_technical_brief_data_cleaning_april_2016_0.pdf.

Benini, Aldo (2013a). *How to Approach a Dataset, Part 1: Data Preparation*. https://web.archive.org/web/20160318211302/https://aldo-benini.org/Level2/HumanitData/Acaps_How_to_approach_a_dataset_Part_1_Data_preparation.pdf.

Benini, Aldo (2013b). *How to Approach a Dataset, Part 2: Analysis*. https://web.archive.org/web/20160318211248/https://aldo-benini.org/Level2/HumanitData/Acaps_How_to_approach_a_dataset_Part_2_Analysis.pdf.

Bonner, Anne (2019). *The complete beginner's guide to data cleaning and preprocessing*. Python, simple. https://web.archive.org/web/20240130234159/https://towardsdatascience.com/the-complete-beginners-guide-to-data-cleaning-and-preprocessing-2070b7d4c6d.

Elgabry, Omar (2019). *The ultimate guide to data cleaning*. https://web.archive.org/web/20240130233510/https://towardsdatascience.com/the-ultimate-guide-to-data-cleaning-3969843991d4?gi=c3e850fa59c9.

Jonge, Edwin de and Mark van der Loo (2013). *An introduction to data cleaning with R*. PDF, extensive (53 pages). https://web.archive.org/web/20240130234406/https://cran.r-project.org/doc/contrib/de_Jonge+van_der_Loo-Introduction_to_data_cleaning_with_R.pdf.

Sciforce (2019). *Data cleaning and processing for beginners*. Python, simple. https://web.archive.org/web/20240130233942/https://medium.com/sciforce/data-cleaning-and-preprocessing-for-beginners-25748ee00743.

Willems, Karlijn (2017). *An introduction to cleaning data in R*. R, simple. https://web.archive.org/web/20240130235354/https://www.datacamp.com/courses/cleaning-data-in-r.