Notes on Data Cleaning

Data quality has been defined as "fitness for use," and data quality is a key aspect of every study. Without data quality, what is the point of a study? A study could be worse than nothing if poor data leads to erroneous conclusions.

Data cleaning is a key component of data quality for most studies. Data cleaning is the process of examining your data, finding incorrect values, and dealing with those values in some way (optimally by having them corrected).

Some data cleaning scenarios:

1. In a small study, data is being entered directly into an Excel spreadsheet. To find incorrect values, the data are read into JMP and subjected to a quick examination to find extreme or inappropriate numeric values or inappropriate character values, which signal probable data entry errors. For example, a field should contain integers between 1 and 5 but a 9 exists on one record, and a field of car manufacturer values contains 10 values of "Chevrolet" and 1 value of "Chevrolett". One car weight is 42555 while all other car weights are between 1800 and 7000. Correct these errors in Excel and re-check with JMP until no obviously erroneous values remain. (PROC FREQ and PROC UNIVARIATE are SAS alternatives for this type of data checking.)

   Since this type of checking will not find NON-OBVIOUS problems in continuous numeric variables, you have two people independently enter important continuous numeric data into separate Excel spreadsheets (along with IDs, of course). When they are finished, you independently import the spreadsheets into SAS data sets and compare the data sets with PROC COMPARE. Any differences are resolved at the Excel level.

   In this scenario, the Excel spreadsheet is the *raw data*. Hopefully you have the original data on paper, and this would be considered the *source data*.

2. In a clinical trial, data are collected with a sophisticated data management system (DMS) and stored in a relational database. The DMS includes edit checks on fields to minimize bad data entry, but inevitably some incorrect values will slip through. Periodically, SAS data sets are produced from the DMS database, and

data-checking SAS programs are run to look for unexpectedly missing or obviously incorrect or inconsistent data values. Sites are notified of these values so that they can correct them (or use the option to tell you that they have no better values). In addition, monitoring visits are conducted where entered values (based on SAS reports) are compared with source documents such as hospital records. When discrepancies are found, the site is asked to make corrections in the DMS. This entire set of procedures is called a *data query process*.

The data query process might also include central statistical monitoring, as described with nice examples in Venet et al., *A statistical approach to central monitoring of data quality in clinical trials* (https://www.ncbi.nlm.nih.gov/pubmed/22684241). Central statistical monitoring might be able to catch unintentional data errors (as from a poorly calibrated instrument), data errors caused by carelessness (resulting in lots of missing values), or fabricated data, especially when such problems are occurring at a single site which thus stands out from all other sites. Results of central statistical monitoring can help to focus monitoring visits on sites where they are most needed.

Data query process steps are repeated as necessary over the life of a study so that data are clean at the database level by the time of database lock (the time when no more changes are allowed via the DMS). In all checking activities, efforts focus on the study's most important variables – record identifiers, primary and secondary endpoints, and key safety variables – to avoid spending precious study resources, both time and money, on activities that aren't critical to the study's mission.

3. In a longitudinal cohort study (say four participant visits five years apart at three sites), data are collected with increasingly sophisticated DMS systems over time, but data are still regularly imported and checked with SAS as in a clinical trial (scenario #2). Additional data checking activities:

    - SAS reports are used to check for inconsistencies across visits, such as height that changed by more than 10% or the report at visit 2 of a

pregnancy in someone reported to be a male at visit 1. Sites are informed of problems so that they can make corrections, if appropriate.

- For measured values such as height, means and standard deviations are examined across sites to check for systematic differences (is one site measuring height in a non-standard way or recording heights in the wrong units or using an incorrect formula to calculate something?). Sites are informed of problems so that they can make corrections, if appropriate.
- If incorrect values are found after the data collection phase (in the analysis phase), study or manuscript-specific derived variables incorporating "corrections" are created and used. Often, these involve bad values being set to missing, or values corrected using a formula if a systematic problem occurred (for example, multiply values of a certain lab variable by 1.07 if the measurement was made after machine calibration on a certain date).

4. In a certain longitudinal cohort study, several different types of data standardization and coding are required before all desired study analyses can be carried out. These activities could be considered data cleaning of a sort. Standardization followed by coding might be needed in areas such as the following:

    - Home address data – standardize reported address information in preparation for geocoding (assignment of longitude and latitude values so that distances can be computed)
    - Medication names – before assignment of standard medication codes, correct names typed in when options in a selection list were not comprehensive (hire a pharmacist)
    - Potential linking fields such as birthdate and city of residence - clean up in preparation for trying to link our cohort with an unrelated database collected for a different purpose

5. You are in charge of reporting on data trends to local government policymakers. Your data sources include databases prepared for other purposes and surveys administered to local target populations. Data cleaning in preparation for reporting consists of making obvious data corrections and applying other best

judgments to arrive at improved estimates of client locations, practices, and needs.

6. You are in charge of preparing a newly-received data source for combination with an existing one. The two data sources contain similar information but in different forms. You need to perform appropriate conversions to variables in the new data source before combining the data. The following are just some of the conversions needed for consistency, given the characteristics of variables in your existing data source: Convert new data source zip code values to character strings; if a four-digit zip code is found, assuming a leading 0 was lost and insert one. Convert date strings to SAS date variables. Convert state names to standard state codes (FIPS). Insert dashes into standard locations in phone numbers. Note: This is not standard data cleaning in terms of correcting data values, but it is a VERY common activity for SAS programmers. SAS functions are your best friend in doing this type of work.

   Suggested references for this type of data cleaning:

   Bahler, Caroline. Data Cleaning and Base SAS Functions.
   http://www2.sas.com/proceedings/sugi26/p056-26.pdf

   Stampfel, Caroline. Overview: SAS Data Cleaning/Standardization.
   http://www.amchp.org/programsandtopics/data-assessment/Documents/Data%20Linkage%20Training%202011/SAS_Slides/Day1B_Character%20Data%20Cleaning%20Techniques.pdf

   Cassell, David. The Basics of the PRX Functions.
   http://www2.sas.com/proceedings/forum2007/223-2007.pdf

   Examples 9.18 and 9.19 in Michele Burlew, Combining and Modifying SAS Data Sets: Examples, 2nd edition

Some data cleaning techniques

In his book on data cleaning using SAS (Cody's Data Cleaning Techniques Using SAS, Second Edition. 2008. Cary, NC: SAS Institute Inc.), Ron Cody defines data cleaning as including

- Checking that character variables include only valid values.
- Checking that numeric variables are within predetermined ranges.
- Checking for missing values for variables where complete data is necessary.
- Checking for and eliminating duplicate entries.
- Checking for presence and uniqueness of certain values, such as participant IDs.
- Checking for invalid date values and invalid date sequences.
- Verifying that complex multi-file rules have been followed.

There are many ways in SAS to do any of these tasks, but here is at least one way to do each. See his book or use your SAS toolkit to imagine other ways to do each type of checking. JMP is also a great way to do preliminary investigations, if not formal query reports.

The examples below use data similar to but not the same as the PATIENTS data from Cody's data cleaning book. Note that this data is MUCH dirtier than any real study data you are likely to encounter (I hope!), but it is useful for demonstrating techniques.

Variable name	Variable label	Type	Valid values
PatNum	Patient Number	character	Numerals only
Gender	Gender	character	'M' or 'F'
Visit	Visit Date	numeric (date in MMDDYY10. format)	Any valid date
HR	Heart Rate	numeric	Between 40 and 100
SBP	Systolic Blood Pressure	numeric	Between 80 and 200
DBP	Diastolic Blood Pressure	numeric	Between 60 and 120
Dx	Diagnosis Code	character	1 to 3 digit numeral
AE	Adverse Event?	character	'0' or '1'

A. Check that character variables include only valid values

Use PROC FREQ, scan output:

```
proc freq data=clean.patients613 /* (drop=patnum) */;
    tables _character_ / nocum nopercent missing;
run;
```

Use PROC FREQ and formats to simplify output:

```
proc format;
    value $gen 'F','M'='Valid'
                   ' '='Missing'
               other  ='Miscoded';
run;
proc freq data=clean.patients613;
    tables gender / missing;
    format gender $gen.;
run;

/* alternatively */
proc format;
    value $gen 'F','M'='Valid'
                   ' '='Missing';
            *  other  ='Miscoded';
run;
proc freq data=clean.patients613;
    tables gender / missing;
    format gender $gen.;
run;
```

Search for and print invalid values:

```
data badstrings1(keep=patnum dx gender AE);
    set clean.patients613;
    if notdigit(strip(patnum))>0 or notdigit(strip(dx))>0 or
       gender ^in ('M','F') or AE ^in ('0','1') then output;
run;
proc print data=badstrings1; run;

/* equivalent alternative */
data badstrings2(keep=patnum dx gender AE);
    set clean.patients613;
    if notdigit(strip(patno))>0 or notdigit(strip(dx))>0 or
       verify(gender,'MF')>0 or verify(AE,'01')>0 then output;
run;
proc print data=badstrings2;
run;

/* another equivalent */
proc print data=clean.patients613(keep=patnum dx gender AE);
    where notdigit(strip(patnum))>0 or notdigit(strip(dx))>0 or
       verify(gender,'MF')>0 or verify(AE,'01')>0;
run;
```

B. Check that numeric values are within predetermined ranges or, if there are not formal predetermined ranges, look for outliers

Range checks should be designed to compare numeric values with predetermined ranges or, if such ranges have not been specified, to identify values that are physiologically impossible or outside normal variation for the population under study. Basic tools that you learned about in BIOS 511 are excellent for this type of check.

```
ODS SELECT EXTREMEOBS;
proc univariate data=clean.patients613 nextrobs=10;
    id patnum;
    var _numeric_;
run;

/* look specifically for out-of-range data */
proc print data=clean.patients613;
    var patnum HR SBP DBP;
    where ^(40<=HR<=100 and 80<=SBP<=200 and 60<=DBP<=120);
run;

/* look for values more than 2 or 3 or 4 standard deviations from the mean */
/* (for highly skewed data, use UNIVARIATE, ODS SELECT EXTREMEOBS instead) */
proc means data=clean.patients613;
    var sbp;
    output out=sbpstats(drop=_type_ _freq_) mean=sbpmean std=sbpstd;
run;
data combine;
    set clean.patients613(keep=patnum sbp);
    if _N_=1 then set sbpstats;
    if (sbp<sbpmean-3*sbpstd or sbp>sbpmean+3*sbpstd) and ^missing(sbp)then
output;
run;
proc print data=combine;
run;
```

A weakness of numeric checks is that we have no way of detecting incorrect entries that are not unusual. That is, perhaps someone's heart rate is 90, but the interviewer accidentally records 80. There is no algorithmic way to detect this, which is why site monitoring is conducted. Monitors from the data coordinating center (an organization like the CSCC) visit sites and check a random sample of entered data (as contained in the DMS and imported into SAS) with values in the source documents maintained by the site. If they find differences, they inform the site, keep records of their findings, and perhaps provide retraining in the case of widespread errors. Site monitoring is really the only way to detect such subtle errors, besides requiring double entry of all data.

C. Checking for missing values for variables where complete data is necessary.

This topic is covered in the notes on missing values (in the References area of the course Sakai site). Of course, both PROC FREQ and PROC MEANS provide options for detecting and counting missing values, and you can easily locate records with missing values for specific variables with PROC PRINT or PROC SQL or a DATA step. In any context where you might be tempted to look for missing values by comparison with ., .z, or ' ', I urge you to use the MISSING function instead.

D. Checking for and eliminating duplicate entries.

Sometimes a data set contains records that are exact duplicates of other records, and you will almost certainly want to get rid of such duplicates, keeping only one copy of the record. Sometimes a data set contains multiple records for an ID (or some collection of key variables) when it should have only one record per ID. Is this because one of the IDs is in error or because multiple records were accidentally entered for the same person more than once? It is often difficult to know. In other cases, multiple records per ID are fine and expected, as in a data set that should contain unique records for ID / Visit combinations.

Use PROC SORT with the NODUPRECS option to eliminate records that are exact duplicates. Use the DUPOUT option to save eliminated records so that you can see what was found.

```
proc sort data=clean.patients613 out=nodupobs dupout=dupobs noduprecs;
   by patnum visitdate;
run;

NOTE: There were 31 observations read from the data set CLEAN.PATIENTS613.
NOTE: 1 duplicate observations were deleted.
NOTE: The data set WORK.NODUPOBS has 30 observations and 8 variables.
NOTE: The data set WORK.DUPOBS has 1 observations and 8 variables.
```

NODUPRECS eliminated the duplicate record for PatNum=002, VisitDate=11/13/2008, Dx=Y, etc. Before analyzing this data we would want to get rid of this duplicate record, which obviously should not be in our data set twice, so that our counts aren't too high and so forth.

In this particular case, using NODUPKEY in place of NODUPRECS produces the same result, at least if PatNum and VisitDate are used as the sort variables. What if we use NODUPKEY and sort only on PatNum?

```
proc sort data=clean.patients613 out=noduppats dupout=duppats nodupkey;
   by patnum;
run;
```

```
NOTE: There were 31 observations read from the data set CLEAN.PATIENTS613.
NOTE: 3 observations with duplicate key values were deleted.
NOTE: The data set WORK.NODUPPATS has 28 observations and 8 variables.
NOTE: The data set WORK.DUPPATS has 3 observations and 8 variables.
```

Here, what we have in addition to the 002 case are visits by patients 003 and 006 on different dates, and depending on our purposes, we would probably not want to get rid of these duplicates.

You can also use FIRST. and LAST. variables to find duplicate records.

```
proc sort data=clean.patients613 out=sorted;
   by patnum visitdate;
run;
data dups;
   set sorted;
   by patnum visitdate;
   if ^(first.visitdate and last.visitdate);
run;
```

The following PROC SQL code is pretty much equivalent to a PROC SORT with NODUPRECS except that you have no way to save the duplicate records into their own data set.

```
proc sql;
   create table single as
         select distinct *
         from clean.patients613;
```

A neat trick for finding IDs with multiple occurrences uses PROC FREQ.

```
proc freq data=clean.patients613;
   tables patnum / out=dup_no(keep=patnum count where=(count>1));
run;
proc print data=dup_no;
run;
```

Obs	PatNum	COUNT
1	002	2
2	003	2
3	006	2

Once you have the dup_no data set, you can use a nice PROC SQL-based sequence to examine the records with duplicate IDs.

```
proc sql noprint;
   select quote(patnum) into :dup_list separated by ' '
   from dup_no;
%put &dup_list;

proc print data=clean.patients613;
   where patnum in (&dup_list);
run;
```

The SAS log shows &dup_list to be `"002" "003" "006"`, and the resulting PROC PRINT output is below.

Obs	PatNum	Gender	VisitDate	HR	SBP	DBP	Dx	AE
2	002	F	11/13/2008	84	120	78	Y	0
3	003	X	10/21/2008	68	190	100	3	1
6	006		06/15/2009	72	102	68	6	1
16	002	F	11/13/2008	84	120	78	Y	0
17	003	M	11/12/2009	58	110	74		0
31	006	F	07/07/2009	92	148	84	1	0

You could also produce this type of information with a single SQL step, as follows.

```
proc sql;
   select patnum, visitdate
         from clean.patients613
         group by patnum
         having count(patnum)>1;
```

Patient Number	Visit Date
002	11/13/2008
002	11/13/2008
003	10/21/2008
003	11/12/2009
006	06/15/2009

Patient Number	Visit Date
006	07/07/2009

E. Checking for presence and uniqueness of certain values, such as participant IDs.

   Missing value and duplicate record techniques apply to this type of data cleaning.

   With particular reference to ID variables and avoiding data problems, when planning your study data make sure that EVERY DATA SET has an appropriate ID variable or variables for identifying records and linking with other study data sets. Also, at the CSCC we create ID values programmatically, and the last digit of every ID value is something called a *check digit*. A check digit is simply a summary digit related by an algorithm to all other digits in the ID. When our CSCC DMS system uses IDs, the system checks each one for an appropriate check digit. If an ID value's check digit is not correct according to the algorithm, the DMS knows that it is not a valid ID.

F. Checking for invalid date values and invalid date sequences.

   As you know, the SAS System has an internal calendar and keeps track of dates relative to January 1, 1960. Thus, if you tell SAS to interpret a value such as 06/05/2013 as a date, it is able to do so and stores the number of days between 06/05/2013 and 01/01/1960. However, if you tell SAS to interpret 06/31/2013 as a date, it returns a missing value since the SAS calendar knows that June only contains 30 days.

   This means that date cleaning is often difficult in SAS since what we see in our data set are missing values, not incorrect dates. A useful technique, at least with small amounts of data, is to read in date data twice, once as a SAS date variable and once as a character string, and examine the cases where the date variable value is missing. Another option is to save month, day, and year values separately and decide how to reconstruct date variables when one of the date parts is missing or invalid (perhaps substituting 1 or 15 for a missing day value, for example).

Here's an example of reading in date variables as both dates and character values.

```
data somedates;
   input @1 ID $3. @5 date mmddyy10. @5 chardate $10.;
cards;
001 11/01/2010
002 06/31/2010
003 02/29/2012
004 02/29/2011
005 01/01/2010
006 15/15/2010
007 04/30/2010
;
proc print data=somedates;
   format date date9.;
run;
```

Invalid date input is a frequent source of log messages such as the following, in this case. You can try to interpret these messages to see which input records have invalid date information, or use a PROC PRINT step as shown.

```
NOTE: Invalid data for date in line 87 5-14.
RULE:      ----+----1----+----2----+----3----+----4----+----5----+----6----+----7----+----8--
--
87         002 06/31/2010
ID=002 date=. chardate=06/31/2010 _ERROR_=1 _N_=2
NOTE: Invalid data for date in line 89 5-14.
89         004 02/29/2011
ID=004 date=. chardate=02/29/2011 _ERROR_=1 _N_=4
NOTE: Invalid data for date in line 91 5-14.
91         006 15/15/2010
ID=006 date=. chardate=15/15/2010 _ERROR_=1 _N_=6
```

Obs	ID	date	chardate
1	001	01NOV2010	11/01/2010
2	002	.	06/31/2010
3	003	29FEB2012	02/29/2012
4	004	.	02/29/2011
5	005	01JAN2010	01/01/2010
6	006	.	15/15/2010
7	007	30APR2010	04/30/2010

If a certain date variable is supposed to have values within a certain range, you can use SAS date constants and PROC PRINT to find incorrect values.

```
proc print data=clean.patients613;
   var patnum visitdate;
   where ^('01apr2008'd <= visitdate <= '31oct2009'd);
run;
```

An invalid date sequence is another possible data problem. For example, an adverse event might be reported as occurring on a date before the person's randomization date. To detect such problems you can use techniques such as described in the next section (combine data sets containing the variables of interest and use DATA step logic to compare the dates, flagging pairs that break the rule).

G. Verifying that complex multi-file rules have been followed (consistency checks)

If you are fortunate, you are working with data from a sophisticated DMS, and many simple data problems have been prevented by edit checks or selection lists in the DMS. Such a DMS might even offer cross-field validation checks to prevent data anomalies, such as a smoking questionnaire where someone answering No to the question "Do you smoke?" is allowed by a later question to report smoking 5 cigarettes per day. However, even the most sophisticated DMS probably does not offer field validation ACROSS forms. For example, one would not expect a DMS to catch a situation such as a person claiming to be a male on the Demography form but reporting a pregnancy on the Medical History form. For that type of consistency check, you'll generally need to depend on a data check performed with SAS or other reporting/analysis software, after importing the data from the database where the DMS stores the data. Here is some sample code to illustrate this approach.

```
data combine;
   merge demog(keep=id gender)
         mhx(keep=id numpreg);
   by id;
   if gender='M' and numpreg>0 then problem=1;
run;

title 'Males reporting at least one pregnancy';
proc print data=combine;
   where problem=1;
run;
```

Also, you can use data checks to find missing forms – that is, form-based records that should exist for a participant according to the study protocol, but they do not exist. For example a study might call for every randomized participant to have both a demographics form and a medical history form at visit 1. Given a data set of randomized participants and data sets of visit 1 demographics and medical history forms, a merge with IN flags could easily reveal missing (or extra!) records. Similarly, study protocol could specify that a concomitant medication form should be filled out in the case of an adverse event. Adverse event records can be matched with concomitant medication records by appropriate key variables to identify missing forms.

H. Using lack of internal consistency to detect data errors

Having repeated measures over time might enable you to detect data entry errors. For example, if a person's weight is measured over a series of months and one of the weight values is wildly different from the others, that entry or its associated unit of measurement could be in error. That suspect weight value might not stand out in comparison with all other weight values for all other study participants yet stand out in comparison with the other values for that particular person.

Also, often data problems are found when variables are used to derive other variables. For example, a weight and height could look perfectly reasonable when standing on their own. Yet combining them to produce a derived variable such as BMI (weight / height^2, where weight is in kilograms and height is in meters) might reveal that at least one of the variables height and weight (or their associated unit of measure) is probably in error. BMI is typically around 25, so a BMI value below 10 or greater than 60 might signal a problem in a constituent variable.

<u>Data correction or otherwise coping with incorrect values</u>

As implied by the scenarios beginning this chapter, how incorrect values are dealt with will be different in different situations. If the data belongs to you (you are compiling and entering it, as in scenario #1), of course you should make necessary corrections. If the data are from a clinical trial or longitudinal study and belong to the clinical sites, which have the source documents, the clinical sites should make corrections.

When data corrections are made during the data collection phase of a clinical trial or longitudinal study, good documentation should be maintained about how and why these corrections were made. With any data used for analysis, people evaluating or relying on that analysis need to have confidence in the data behind it. The idea is to maintain an audit trail so that someone evaluating the data could understand where every value came from. A good DMS system will automatically log all deletions, updates, or changes to data after initial entry. When data values are changed as a result of a data query process (data cleaning), the data center sending out the report and the site responding to the query should each keep records of their actions.

If you find problems with some data values in a mature study where data collection has ended, you might set those values to missing and then deal with them using methods described in this course's notes on missing values (in the References area of the Sakai site). A better course might be to leave the original variable as is but create a new derived variable that is identical to the original one except for some corrections including missing values where the correct value is unknown. Then use the derived variable in your analysis.

If your problem is outliers for continuous variables, it might be that these outliers are the most interesting part of your data, and you should study them to see what they can tell you! On the other hand, they can cause computational issues and keep you from learning properly from the bulk of your data. So how might you deal with outlying values? Here are some possibilities.

- Use robust regression or other non-parametric methods where the outliers won't have so much influence.
- Use rankings or data divisions such as quartiles in place of the original variable values.
- Transform the variable to reduce the degree of outlyingness. However, this can bring its own problems and complicate interpretation.
- In a kind of sensitivity analysis, report results both including and excluding the outlying values.