CLEAN UP YOUR ACT: DATA CLEANING WITH SAS
Mel Widawski, Brettmel Development

ABSTRACT
There is usually no such thing as a clean data set, and this includes publicly available datasets that have been used for years. There are three steps in data cleaning: 1) Discover the problem, 2) Locate the item in error, 3) Correct the error. We will explore each of these areas to discover some short cuts, guidelines, and hints.

INTRODUCTION
The one thing in data analysis that no one pays enough, attention to, allows enough time for, or does as thoroughly as they would like is data cleaning. And yet, if attention has not been paid to data cleaning, often the data analysis will be worthless.

Discovering the Problem
It often saves time to use existing procedures to determine if there are problems requiring attention. Procedures such as FREQ, UNIVARIATE, and CORR can be useful in determining whether problems exist. These are used to determine which variables need closer examination.

Locate the Items in Error
Once you have determined that there is an error for a variable it is necessary to locate the observation that has the problem on that variable. While PROC PRINT can be used for this, I find that PUT statements work better. I also find that a single pass for all variables is preferable to multiple lists of errors. In any case the least efficient method of locating the cases in error is scanning the data.

Fixing the Problems
Now that you have a list of the errors in the data, you will need to correct the errors. There are two main methods of making corrections to the data: 1) correcting the raw data directly, or 2) writing a program to make the corrections in the SAS data set. We will discuss the pros and cons of each of these solutions. In any case be sure to BIUS, Back It Up St_you know the rest). Always make a copy of the data set so you can find the original values before making any changes.

Along the way I will endeavor to share some tricks and hints to make the process easier.

DISCOVERING THE PROBLEM
Use standard procedures for this task and don’t ignore graphics. At times graphics are the only way to see some of the problems in data. Most people have discovered UNIVARIATE now and know how useful it is for examining data. Extreme cases pointed out by UNIVARIATE should be examined more closely.

PROC FREQ is another procedure for examining data and determining data out of range. But don’t settle for simple frequencies as sometimes the problems with data will not be revealed until you look at the inconsistencies between variables.

With continuous variables often PROC FREQ can be as bad PROC PRINT for revealing problems with data. I actually saw someone do a PROC FREQ on income when working with census data. Now this was the numeric variable income, rather than a grouped variable with income categories. It was in the old days of printed output and the fanfold output reached across campus and would have taken years to scan.

PROC PLOT can show unusual values with continuous variable much like cross-tabulating variables in PROC FREQ does with categorical variables.

What are the types of problems that we will be searching for?

- Data out of range
- Data gaps
- Inconsistent data
- Extreme cases
- Bivariate outliers

There are certain errors that are almost impossible to find such as data in range that is simply randomly mistyped. That is the reason that important data should be entered in a double entry system. It is also possible to do a sample of forms with a separate entry to determine the rate of miss-entered data.
We will review some SAS procedures and discuss what data problems they can detect.

**PROC UNIVARIATE**

This procedure can be used to detect data out of range for both continuous data and numeric nominal data.

It automatically gives you extreme values for example the following:

```sas
PROC UNIVARIATE PLOT;
   ID subid ;
   VAR birthyr;
RUN;
```

May produce the following output:

<table>
<thead>
<tr>
<th>Extremes</th>
<th>Lowest</th>
<th>subid</th>
<th>Highest</th>
<th>subid</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1901(</td>
<td>3</td>
<td>1969(</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>1945(</td>
<td>1</td>
<td>1972(</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>1956(</td>
<td>11</td>
<td>1975(</td>
<td>9</td>
</tr>
<tr>
<td></td>
<td>1958(</td>
<td>13</td>
<td>1977(</td>
<td>5</td>
</tr>
<tr>
<td></td>
<td>1959(</td>
<td>18</td>
<td>1998(</td>
<td>4</td>
</tr>
</tbody>
</table>

It helps to know how and when the data was collected, for in this case it was known that the earliest birthyr was 1945 so subid 3’s value of 1901 is impossible for this dataset. Also, since the study was done in 1997 and the minimum age was 20, a value of 1998 is impossible for two reasons. Thus we would want to look at the questionnaire results for subid 4 as well.

It is also useful to examine the quantiles information to see how wildly different the extreme values are from the rest.

<table>
<thead>
<tr>
<th>Quantiles(Def=5)</th>
<th>100% Max</th>
<th>99%</th>
<th>95%</th>
<th>90%</th>
<th>50% Med</th>
<th>95%</th>
<th>90%</th>
<th>5%</th>
<th>1%</th>
<th>Range</th>
<th>Q3-Q1</th>
<th>Mode</th>
</tr>
</thead>
</table>

Skew can also be an indication of problems. Notice how the case with the birthyr of 1901 stands out in the plot below. This is also useful to get a handle on any transformations that might need to be made to continuous variables for adequate analysis.

```
```

<table>
<thead>
<tr>
<th>Stem Leaf</th>
<th>#</th>
<th>198</th>
<th>8</th>
<th>1</th>
</tr>
</thead>
<tbody>
<tr>
<td>196 0001224459257</td>
<td>13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>194 56899</td>
<td>5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>192</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>190 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Multiply Stem.Leaf by 10**+1

**PROC MEANS**

This procedure is useful for checking consistency between categorical variables and continuous variables. It is especially useful to insure that the continuous variables fall within the range of the categories that they should after creating a grouping variable such as birthyr. It should be a matter of course that each transformation is checked against the original values. This implies that when you create a transformed variable you retain the original variable.

Examine the following code:

```sas
DATA samplef2;
SET samplef;
/* Four Groups */
IF birthyr>=1967
   THEN birthyrg=4;
```
ELSE IF birthyr>=1961
    THEN birthyrg=3;
ELSE IF birthyr>=1959
    THEN birthyrg=2;
ELSE IF birthyr>1940
    THEN birthyrg=1;

/* Decades */
birthyrd=floor((birthyr-1900)/10);

RUN;

Title2 "Checking Transform";
PROC MEANS MIN MAX;
    CLASS birthyrg;
    VAR birthyr;
RUN;

PROC MEANS with the CLASS statement and specifying the minimum and maximum is ideal for determining that all of the values of the original variable have been recoded properly. Notice that you can tell at a glance that each category contains the correct range of values.

<table>
<thead>
<tr>
<th>N</th>
<th>birthyrg</th>
<th>Obs</th>
<th>Minimum</th>
<th>Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>3</td>
<td>3</td>
<td>1945.00</td>
<td>1958.00</td>
</tr>
<tr>
<td>2</td>
<td>5</td>
<td>5</td>
<td>1959.00</td>
<td>1960.00</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>7</td>
<td>1961.00</td>
<td>1965.00</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>5</td>
<td>1969.00</td>
<td>1989.00</td>
</tr>
</tbody>
</table>

PROC PLOT
This procedure has some of the advantages of using the minimum and maximum with ranges of a continuous variable. But it gives the additional advantage of being able to see many data values at a time. It is a little harder to tell if the cutoffs have been missed.

The following PROC PLOT applies to the same transformations as the previous PROC MEANS:

Title "Transformations";
Title2 "Checking Grouping";
PROC PLOT;
    PLOT birthyrg*birthyr=birthyrg;
RUN;

You can see at a glance whether or not the recode worked.
Proc plot is very useful in pointing up anomalies in continuous data that might have gone unnoticed. Consider the following:

```
Title "Checking Relationship";
PROC PLOT;
   PLOT scale2*scale1;
RUN;
```

The plot of the two variables is very revealing. You can easily see that there is an outlier data value. It would be advisable to check the original forms to see if this was mistyped, if we knew which case to look for.

```
Plot of scale2*scale1. Legend: A = 1 obs, B = 2 obs, etc.
```

Notice the outlier with a score under 40 for scale2 and a score around 70 for scale1. This is the case that should be further examined.

**PROC FREQ**

If you have established value labels (formats) for your categorical variables, then this aids in locating values out of range. Consider the following frequency tables:

```
Title "Detecting the Problem";
Title2 "Nominal Variables";
PROC FREQ;
   TABLES sex race language;
RUN;
```

Notice how easy it is to see data out of range when the values in range are labeled.

```
Gender

<table>
<thead>
<tr>
<th>sex</th>
<th>Freq</th>
<th>Percent</th>
<th>Cumulative Freq</th>
<th>Cumulative Percent</th>
</tr>
</thead>
<tbody>
<tr>
<td>male</td>
<td>9</td>
<td>45.00%</td>
<td>9</td>
<td>45.00%</td>
</tr>
<tr>
<td>female</td>
<td>9</td>
<td>45.00%</td>
<td>18</td>
<td>90.00%</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>5.00%</td>
<td>19</td>
<td>95.00%</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>5.00%</td>
<td>20</td>
<td>100.00%</td>
</tr>
</tbody>
</table>
```

The values 3 and 5 are clearly out of range.

The mistake that is often made is to stop with one-way frequencies. When exploring data, there are often other variables, which can help identify bad data. For example, it would be rather suspicious if someone in a non-latino
ethnic category were to have taken a questionnaire in Spanish. Producing cross-tabulations between variables is one way of identifying this type of problem.

Title2 "Contingency Logic";
PROC FREQ;
   TABLES language*race/missprint norow
          nocol nopercent;
RUN;

Examining the output we see that there are indeed questionable values.

The FREQ Procedure
Table of language by race

<table>
<thead>
<tr>
<th>Language (Language of Interview)</th>
<th>Hispanic</th>
<th>Other</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spanish</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>english</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Other</td>
<td>0</td>
<td>2</td>
<td>0</td>
</tr>
<tr>
<td>Total</td>
<td>3</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

LOCATING CASES WITH BAD DATA

Now that we have identified the variables that contain questionable data we need to know which observations those problem data belong to so we can check the original forms. The first tendency is to do PROC PRINT with a where clause. This is problematic for two reasons. First, if all of the variables with problems are printed simultaneously then this can be much like scanning the data to find the errors. Second, if separate prints are done for each variable then time is wasted searching for the same forms a number of times.

Using put statements in a data step allow a single pass of the data, so that all of the problem items for a single case can be printed together. And doing the put statements as the subject of if statements can keep the output down to just the data problems for that case. There is a tendency to want to group those variables that require similar range tests together to simplify programming. Doing this can make the job of the person who has to examine the questionnaires much more difficult, and can ultimately waste time. This is especially true if the questionnaires are long and there are a large number of questionnaires. The continual shuffling back and forth also increases the likelihood of errors. If at all possible the printout of variables should be in the order that they were found in the questionnaire.

A Bad Data Report

Notice in the program below that SUBID is only written once for each subject and that each of the variables with problem values for that subject are printed after the SUBID.

By using list output the name of the variable as well as the value is output so the variable may be identified. Each variable that is output is in order as it is in the data set. The LINK is used to cut down on the amount of code. This method is the most flexible, but it requires a great deal of code for long questionnaires.

Here is a simple program that does just that, and can be used for short surveys.

Title2 "All Together";
DATA _NULL_;  
   SET sample;
   BY subid;
   FORMAT race ;
   RETAIN flag;
   IF First.subid THEN flag=.
   IF NOT(1940<birthyr< 1997) THEN DO;
      IF not flag THEN LINK SID;
      PUT @15 birthyr=;
   END;
   IF sex >2 THEN DO;

IF not flag THEN LINK SID;
PUT @15 sex=;
END;
IF race >4 THEN DO;
IF not flag THEN LINK SID;
PUT @15 race=;
END;
IF race ^=3 and language=1 THEN DO;
IF not flag THEN LINK SID;
PUT @15 race= @30 language;
END;
RETURN;
SID: PUT SUBID=;
FLAG=1;
RETURN;
RUN;
The output of this report is in the LOG, but it could easily be output to a file. The report is concise and useful for checking the original forms and for making corrections.

<table>
<thead>
<tr>
<th>subid=3</th>
<th>birthyr=1901</th>
</tr>
</thead>
<tbody>
<tr>
<td>subid=4</td>
<td>birthyr=1998</td>
</tr>
<tr>
<td>subid=6</td>
<td>sex=5</td>
</tr>
<tr>
<td>subid=12</td>
<td>sex=3</td>
</tr>
<tr>
<td>subid=13</td>
<td>race=1</td>
</tr>
<tr>
<td></td>
<td>language=Spanish</td>
</tr>
<tr>
<td>subid=16</td>
<td>race=5</td>
</tr>
<tr>
<td></td>
<td>language=Spanish</td>
</tr>
<tr>
<td>subid=18</td>
<td>race=.</td>
</tr>
<tr>
<td></td>
<td>language=Spanish</td>
</tr>
</tbody>
</table>

Generally I find that with longer questionnaires most are relatively free of errors I can find, a number have a few errors and some have a large number of errors. Lists produce in this way facilitate the task of reviewing the forms. This method is cumbersome with large numbers and varieties of questions.

Making Use of Arrays
If your questionnaire consists of instruments with a large number of questions each, and are consistent in the possible responses. Then it is possible to use arrays and do loops to drastically reduce code.
Assume that one of the instruments is a 200 question likert scale with choices 1-5 for each item. Then it is possible to do exactly what we did above with much less code.

Title2 "200 Question Likert Scale";
DATA _NULL_; 
SET sample; 
BY subid; 
RETAIN flag; 
IF First.subid THEN flag=.;
ARRAY SCALE1 (J) S1Q1-S1Q200; 
DO OVER SCALE1; 
IF NOT(1<=scale1<=5)
THEN DO; 
IF not flag 
THEN LINK SID; 
PUT @15 scale1=;
END; 
END; 
RETURN;
SID: PUT SUBID=;
       FLAG=1;
RETURN;
RUN;

With list output on put statements using array names what is printed is the name of the original variable followed by the value. This is one of the nice features of SAS. Thus it becomes possible to shorthand the reference of variables with arrays while getting the full information in the output.

<table>
<thead>
<tr>
<th>subid</th>
<th>s1q10</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>18</td>
<td>6</td>
<td></td>
</tr>
</tbody>
</table>

Mixed Up Messes Of Questions

Many times what you have is changing ranges every other question mixed in with some semi-consistent scales, you also have string variables inter mixed with your numeric variables. It is usually safe to assume that most of your data cleaning is going to involve your numeric variables. It would be nice to use arrays as a shortcut, but the variable nature of the ranges to be checked makes that difficult.

What you need is a small additional file containing the low and hi values to check for all of your numeric variables. The nice thing about this is that clerical help can accomplish it. Or in my case, work-study students can do the job. We will make use of tricks above along with the fact that a single SET additional set statement that is executed once will concatenate variables to your file. We will also use the _NUMERIC_ system variable in the array statement to create an array with all of the numeric variables in the file. The compare file contains one record with the high and low values for each numeric variable in order in two sets of variables.

```
Title2 "200 Question Likert Scale Items";
DATA _NULL_;
   RETAIN flag flag2;
   IF ^flag2 THEN SET compare;
       flag2=1;
   SET sample;
   BY subid;
   IF First.subid THEN flag=.;

   ARRAY HiVal  (J)  H1-H2000;
   ARRAY LoVal  (J)  L1-L2000;
   ARRAY SCALE1 (J) _NUMERIC_;
   DO OVER  SCALE1;
       IF NOT(LoVal<=scale1<=HiVal)
           THEN DO;
               IF not flag THEN LINK SID;
               PUT @15  scale1=;
           END;
   END;
   RETURN;
SID: PUT SUBID=;
       FLAG=1;
RETURN;
RUN;
```

The only thing to be worked out is how to handle the consistency checking based on multiple variables.

CORRECTING THE DATA
We must now decide how to make corrections. There are those who like to make the corrections directly to the raw data. There are reasons for making corrections in this way:

- The bad data is gone
- Only the corrected data is available
- No one will ever run programs on the bad data again
- There is nothing to remember, the data set is now updated

All of these are good points, but the problem I have with this is that:

- There is no record of the changes that were made
- The data is gone for good unless you back-up first
- If you back up first then you have a copy of the unfixed data that you could use by mistake
- It becomes difficult to reconstruct the original data if mistakes are made during data cleaning
- You are using the same method for cleaning that was used when the errors were made originally.

I prefer making the changes with a program if at all possible.

- This insures that a copy of the original data will be retained.
- The program itself is a record of what was done in data cleaning.
- In order to correct errors made in data cleaning all you have to do is correct the program and run it.

In either case you will have to carefully mark the original data set so that it id know that the cleaning program needs to be applied to it. One nice thing about this is that the programs that located the bad data can partially create the correction program. Let’s look at a slight modification to one of those programs to see how this can be done.

```sas
Title2 "200 Question Likert Scale Items"
DATA _NULL_
SET sample;
BY subid;
RETAIN flag;
IF First.subid THEN flag=.;
ARRAY SCALE1 (J) S1Q1-S1Q200;
DO OVER SCALE1;
   IF NOT(1<=SCALE1<=5)
      THEN DO;
         IF not flag THEN LINK SID;
         PUT @15 SCALE1= ';';
      END;
   END;
IF flag THEN PUT 'END;' ;
RETURN;
SID:
   PUT 'IF ' SUBID= ' THEN DO; '
   FLAG=1;
   RETURN;
RUN;
```

This program will produce the modified output as follows, which can be used just as we used the original report. It can also be saved as a SAS program and modified with the new correct values.

```sas
IF subid=3 THEN DO;
   s1q10=8 ;
END;
IF subid=4 THEN DO;
   s1q101=9 ;
   s1q111=7 ;
   s1q161=0 ;
   s1q141=9 ;
END;
IF subid=18 THEN DO;
   s1q51=6 ;
END;
```

This has the added advantage that it becomes unlikely that values of other variables that did not have detectable problems could accidentally be over written during the cleaning process.
CONCLUSION
There are as many ways to approach this problem as there are people involved in data cleaning activities. Some obsess more than others. I hope you have gained a concept of what is involved in data cleaning and some ideas to make it easier.

ACKNOWLEDGMENTS
I would like to thank Barbara Widawski for all her help.

CONTACT INFORMATION
Contact the author at:
E-mail Address: mel@ucla.edu

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.