USING MACROS TO RECODE SENSITIVE DATA
Mary-Elizabeth Eddlestone, Information Maximizers

The Problem: I was given a child abuse incidence data set containing sensitive information, such as the Social Security Numbers of victims, parents and suspects. Before anyone was allowed to examine and analyze the data, the sensitive variables needed to be replaced to ensure the privacy of the individuals involved. I was asked to perform these recodes, without looking at the data, in the least amount of time possible.

The Solution: I designed a set of simple macros that will (a) produce a unique list of all values of each sensitive variable, (b) create a key for each value, (c) merge the key into the original data, (d) replace the sensitive variable with its key, and (e) preserve a separate data set which maps which key is matched to each value.

The macros presented below can be adapted for a number of situations. They can be used to recode many variables in a single data set, variations on a single variable in one data set, or a single variable in many data sets. This paper shows how to recode many variables in a single data set and many similar variables in the same data set.

First, let's look at the simplest example, recoding one variable in one data set. The variable SSN contains the Social Security Numbers of people in the study. A person can appear in the data set more than once.

%macro values (var=);
proc sort noduplicates
  data=ssd.master
  (keep=&var)
  out=&var;
by &var;
%mend;

The first macro, %values, simply uses PROC SORT to create a temporary data set containing unique values of a particular variable. The temporary data set is then used in the next macro to make a key.

%macro makekey (start=, out=, var=, miss=);
data ssd.&out;
  set &var;
  if _n_ = 1 then key = &start;
  else key +1;
  if &var = . then key = &miss;
%mend;

The macro, %makekey, creates the key by beginning with a start value specified by the macro variable &start and increments by one. A missing value will be assigned a key specified by the macro variable &miss, such as 9999. The key and the original variable are preserved in a separate data set.

The last step entails merging the key into the original data set to replace the sensitive variable.
%macro merge (in=, var=);
proc sort data=ssd.master;
  by &var;
data ssd.master;
  merge ssd.master (in=a)
      ssd.&in;
  by &var;
  if a;
  drop &var;
  rename key=&var;
%mend;

The macro, %merge, sorts the master data set by the variable to be replaced. It then merges the key values into the master data set and replaces the variable with the key.

To create a key for SSN, the macros are invoked as follows:
%values (var=SSN);
%makekey (start=100000, out=SSN, var=SSN, miss=999999);
%merge (in=SSN, var=SSN);

In a scenario as simple as this example, the separate macros could conceivably be combined into one. The benefit of keeping them as modules, however, is that they can be expanded and combined to allow for more flexibility.

Let's suppose that there are three variables containing Social Security Numbers, one each for the child (VSSN), a parent (PSSN) and the suspect(SSSN). A person's SSN could show up in multiple records and, an SSN can be in more than one of the three variables. In order to create a key that captures every possible value of an SSN, all three variables need to be used. The macros, %makekey and %merge, can be modified as %makekey3 and %merge2 to accomplish this.
Coders' Corner

% makekey3 (start=, out=, varl=, var2=, var3=, miss=);

data &out;
  set &var1
    &var2 (rename=(&var2=&var1))
    &var3 (rename=(&var3=&var1));

proc sort data=&out noduplicates;
  by &var1;

data ssd.&out;
  set &out;
  if _n_ = 1 then key = &start;
  else key+1;
  if &var1 = . then key = &miss;
%mend;

%macro merge2 (in=, varl=, var2=);
  proc sort data=ssd.master;
    by &var2;
  data ssd.master;
    merge ssd.master (in=a)
      ssd.&in (rename(&var1=&var2));
    by &var2;
    if a;
    drop &var2;
    rename key=&var2;
%mend;

When combined with the original macros, these macros can create a unique key for every Social Security Number that appears in the data set. The macros would be invoked as follows:

%values(var=PSSN);
%values(var=VSSN);
%values(var=SSSN);
%makekey3 (start=100000, out=SSN, var1=PSSN, var2=VSSN, var3=SSSN, miss=999999);
%merge (in=SSN, var=PSSN);
%merge2 (in=SSN, varl=PSSN, var2=VSSN);
%merge2 (in=SSN, varl=PSSN, var2=VSSN);

Looking at this step by step it can be seen that each invocation of %values creates a data set of unique SSNs for each variable.

These are not mutually exclusive data sets, so they need to be combined and then sorted once more, as seen in %makekey3. The rename option on the set statement creates a new temporary data set with a single variable. Duplicate values are removed and the key is made as in %makekey.

Note that even if the three variables were mutually exclusive, the additional sort in %makekey3 has a built-in safeguard of removing the observation that could be in more than one data set -- that of a missing value.

The invocation of %merge replaces the parent's SSN (PSSN) with the key. The first invocation of %merge2 replaces the child's SSN (VSSN), and the second invocation replaces the suspect's SSN (SSSN).

The end result is a data set containing no sensitive variables that can be passed safely to the primary investigators, and a set of data sets containing the keys which can be passed to the data collection agency. Each can then refer to specific individuals by their unique key and thereby avoid any problems with privacy. If additional safeguards are necessary, the data sets could be encrypted before they change hands.

The obvious drawback to these macros is that they are all I/O intensive, since multiple sorts are required.

Trademarks
SAS is a registered trademark of SAS Institute, Inc., Cary, NC.

For further information, the author can be reached at eddlestone@maximizers.com.