Taking the Drudgery Out of Data Checking: Automatic Data Validation Using FORMATS to Validate the Data, PROC DATASETS to Drive the Process and MACROS to Hang it All Together.

David Trenery, Hoechst Marion Roussel, Denham, UK.

ABSTRACT

Checking data is a tedious but necessary stage before data analysis. This paper describes a utility that takes some of the drudgery out of the data checking process and enables one to be more confident about the quality of the data.
You often have formats that label valid data values and are permanently associated to variables in a data set. This utility takes advantage of these formats to validate the data. I use the DATASETS procedure to drive the validation process. It provides a list of variables and their formats. Modified formats validate the data. Macros, the SQL procedure and the DATA step provide the tools to link the process.
You supply the name of the data set you wish to check and optionally any identifying variable. Then the utility does the rest, listing observations with invalid values for those variables with an associated format.
This utility has been tested and run on SAS® software version 6.11 and 6.12. It should run on all operating systems. The intended audience is for SAS users of moderate or greater experience.

INTRODUCTION

Data validation is a tedious but necessary stage before data analysis. You can check small data sets by eye. With the SAS System you can use the FREQ procedure to check variables with a small range of values, or you can write a customised program to check a specific variable. Being easily bored and of lazy disposition I needed another way - something that took almost no input from myself.

CREATING FORMATS TO CHECK DATA

This paper arose because I frequently need to check variables for invalid codes. One particularly large data set for a drug trial contained adverse event data (illnesses, injuries, etc.), it included a patient identifier, the text the doctor used to describe the adverse event and it's World Health Organisation (WHO) medical dictionary code.
To find and list the invalid WHO codes, I first created a format for all valid values. Then using the FORMAT procedure's OTHER= special range I added a label for invalid values to this format. I then printed the data applying a WHERE clause using the format - printing only the observations with invalid codes.

See the following code.

```
PROC FORMAT;
   VALUE mdcode
      ‘4001.001’ = ‘ABDOMEN ENLARGED’
      ‘4002.001’ = ‘ABDOMINAL DISTENSION’
           .
           .
      ‘5267.008’ = ‘ANIMAL BITE’
      ‘5267.009’ = ‘INSET STING/BITE’
       other     = ‘!*@# Invalid !*@#’;
RUN;

PROC PRINT data=adverse(keep=patient
                    verbatim mdcode);
   WHERE put(mdcode,$mdcode4.)=‘!*@#’;
      ID patient;
      VAR mdcode verbatim;
RUN;
```

Below is an example of the output

```
Invalid values of MDCODE

PATIENT     MDCODE     VERBATIM

009-053    5061.019    SLEEP WALKING
132-055    5267.001    HEAD INJURY
159-001                DYSMETRIA
162-001    1825.001    NYSTAGMUS
```

This output lists the invalid values of the variable MDCODE in the data set, there is one missing and three that do not exist in the WHO dictionary that we were using. After further investigation I found these to have been wrongly entered.

USING AN EXISTING FORMAT

Rather than use a VALUE statement to create a format as in the previous code, a more convenient method for large formats is to use a data set and the CNTLIN= option of the FORMAT procedure. If there is an existing format that needs modifying the CNTLOUT= option of the FORMAT procedure will create a data set from a format. I use these two options to create a data set from an existing format, to which I then add an observation with the OTHER= special range and turn this into a new format.

See the following code.

```
PROC FORMAT cntlin=_fmtchk;
   SELECT $mdcode;
RUN;

DATA _fmtchk;
     .... data step code that modifies the format ....
RUN;

PROC FORMAT cntlout=_fmtchk; RUN;
```

CREATING A MACRO

After I carried out this process for one variable I realised how easy and useful it was and decided to create a utility macro to apply to any variable in any data set.
I broke up the process into two stages:

- *Create the new format.*
- *Print out the invalid codes, using the format in a WHERE clause to select invalid codes.*

The more complex of the two stages was creating the new format. The sheer versatility of formats created a few problems, but more about those later.

The following is the code that creates the new formats.

```
%MACRO chk_nfmt(fmtlib,fmtlist);

*--- Turn the formats (but not ---------------;
*--- informats) into a data set --------------;
PROC FORMAT library=&fmtlib
        cntlout=_fmtchk(
        where=(type not in('I','J')));       *1;
   SELECT &fmtlist;
RUN;

DATA _fmtchk;
   SET _fmtchk;
   *--- Get rid of LowHighs or Other ---------;
   WHERE hlo not in('LH','O');         *2 and 3;
   BY  fmtname type notsorted;             *4a;
   *--- Minimum width of 4 as '!*@#' ---------;
   *--- indicates an invalid value -----------;
   IF default<4 then default=4;             *5;
   OUTPUT;

   *--- Create a label for OTHER -------------;
   *--- non-valid codes ----------------------;
   IF last.type;                           *4b;
   hlo='O';
   length=21;
   label='!*@# Invalid !*@#';
   start=' ';
   end=' ';
   OUTPUT;
RUN;
```

```
*--- Recreate the format in WORK library ----;
PROC FORMAT cntlin=_fmtchk; RUN;

%MEND chk_nfmt;
```

The data sets created by the CNTLOUT= option and used by the CNTLIN= option include the variables FMTNAME, TYPE and HLO. FMTNAME contains the format name but without any $ signs, TYPE the format or informat type and HLO indicates whether the range includes HIGH, LOW or OTHER.

Issues and problems

There were a few issues and problems with creating the macro that I dealt with in the previous code. I discuss them here.

***1** *As well as formats the CNTLOUT data set contains informats that I identified with the TYPE variable and then deleted.*

***2** *Some formats may already have an existing OTHER category that I identified by the HLO variable and then deleted.*

***3** *Some formats (most frequently picture formats) have a single label covering the entire possible range, that is HLO = ‘HL’. These formats can not have any “invalid” values so it is pointless using these to validate the data. New formats where not created for these - I deleted them from the CNTLOUT data set using the HLO variable.*

***4a** *and* **b** *Character and numeric formats can have the same name. I used LAST.TYPE to determine the last observation of a format. With this information I added one observation per format to label invalid values.*

***5** *As the WHERE clause was creating subsets by the four character string ‘!*@#’ the default width of the new format had to be at least four. I did not want to subset by a single character as there could conceivably be format labels of ‘!’ or any other single character I might use.*

DRIVING THE PROCESS WITH THE DATASETS PROCEDURE

You can permanently associated formats with particular variables. This information is stored in the SAS data set. You can use the DATASETS procedure to find the names of variables and formats associated with them in a data set.
I decided to use the DATASETS procedure to drive the checking process. With it I obtained a list of variables along with their formats in the data set being checked. I updated each format to label invalid codes. I passed the name of each variable and its format to the Macro that checked each variable for invalid codes (%CHK_VAR). I broke the task down into the following steps:

***6** *Obtain a list of formatted variables, and their formats.*

***7** *Create some macro variables containing this information. See SAS Communications (1997), Q1, p48 for details on the SQL method.*

***8** *Create new formats with a special label for invalid data (%CHK_NFMT macro).*

**9 Point to the new format library. Note that depending on a the FMTSEARCH option in effect a LIBREF of LIBRARY can cause problems in using formats stored in the WORK library. I stored the path of the library 'LIBRARY' for future restoration (in *11). The View SASHELP.VSLIB contains allocated libraries and their paths.*

**10 Check each variable in turn for invalid data (%CHK_VAR macro).*

**11 Restore the original SAS environment.*

The following is a macro that carries out these steps:

```
%MACRO chk_dset(dset,fmtlib,idvars);

*--- Find the names of variables with --------;
*--- formats, their type and their format ----;
PROC DATASETS nolist;
   CONTENTS data=&dset
            out=_dinfo(keep =name format
                        where=(format ne ' '))
         noprint;                                *6;
QUIT;

*--- Create some macro variables -------------;
DATA _dinfo;
   SET _dinfo end=eof;
   *--- Name of variable ---------------------;
   CALL SYMPUT(
         'var'||left(put(_n_,3.)), name);
   *--- Name of format ---;
   CALL symput(name,trim(format));
   *--- Number of observations ---------------;
   IF eof then
      CALL SYMPUT('num',put(_n_,3.));        *7;
RUN;

*--- Generate list of formats ('&FMTLIST') ---;
PROC SQL noprint;
   SELECT distinct trim(left(format))
   INTO :fmtlist separated by " "
   FROM _dinfo;                               *7;
QUIT;

*--- Add label for invalid codes to format ---;
%chk_fcat(&fmtlist);                         *8;

*--- Find the current format search path  ----;
*--- Make sure new formats stored in WORK ----;
*--- library are used -----------------------;
PROC SQL noprint;
   SELECT path into :libpath
   FROM sashelp.vslib
   WHERE libname='LIBRARY';                  *9;
QUIT;

LIBNAME library clear;
OPTIONS fmtsearch=(work);

*--- Invoke the CHK_VAR macro once for each --;
*--- different variable which has a format ---;
%MACRO repeat;
   %do i=1 %to &num;
      %chk_var(&dset,&&var&i,
            &&&&&&var&i,&idvars);            *10;
   %end;
%MEND repeat;

%repeat;

*--- Restore libname LIBRARY and -------------;
*--- FMTSEARCH options -----------------------;
OPTIONS fmtsearch=&fmtsrch;
%if &libpath ne %then
   LIBNAME library "&libpath";              *11;

%MEND chk_dset;
```

When making this data checking procedure a universal utility I had to consider the added complications that:

- *You can store formats in more than one library.*
- *You can apply formats by storing them in the work library or in a library with a LIBREF of LIBRARY or a library pointed to using the FMTSEARCH= option.*
- *Formats in different libraries can have the same name.*

The first two of these may not be an issue at your site, if so then you can simplify the following code that deals with these issues.

```
%MACRO chk_fcat(fmtlist);

*--- Find format search path -----------------;
PROC SQL noprint;
   SELECT setting
   INTO :fmtsrch separated by " "
   FROM sashelp.voption
   WHERE optname='FMTSEARCH';               *12;
QUIT;
```

```
*--- Select only available format catalogs ---;
*--- in format search path LIBRARY or --------;
*--- WORK libraries --------------------------;
DATA vscatlg;
   LENGTH libname word $8.;
   SET sashelp.vscatlg;                          *13;
   WHERE memname='FORMATS';
   DO i=1 to 97;
      word=scan("&fmtsrch",i);
      IF libname=word then DO;
         OUTPUT; LEAVE;
         END;
      IF word=' ' then LEAVE;
   END;
   IF libname in('LIBRARY','WORK');
   IF index("&fmtsrch",libname)=0;
   IF libname='LIBRARY' then i=98;
   ELSE                      i=99;
   OUTPUT;
   KEEP libname i;
RUN;

*--- Order the format catalogs ---------------;
PROC SORT data=vscatlg;
   BY descending i;
RUN;                                             *14;

DATA _null_;
   SET vscatlg end=eof;
   CALL SYMPUT(
         'fmtlib'||left(put(_n_,2.)),libname);
   IF eof then
         CALL SYMPUT('count',(put(_n_,2.);
RUN;

*--- Repeat once per format catalog, ---------;
*--- last formats searched done first --------;
%DO i=1 %to &count;
   %chk_nfmt(&&&&fmtlib&i,&fmtlist);
%END;

%MEND chk_fcat;
```

**12 Gets a list of all format catalogs known to the system. The View SASHELP.VOPTIONS contains a list of the current option settings.*

**13 Gets a subset of these that are in the current format search path. SASHELP.VSCATLG contains a list of all catalogs known to the system.*

**14 I created new formats for each format catalog in reverse search order. Therefore if a format occurs in different libraries the one that would have been found and used normally will overwrite the other.*

6.12 CONSIDERATIONS

In version 6.12 of the SAS System you can now use another format as a range of a format. If this range format contains the OTHER= special range then this utility will fail to create the format to find invalid values. Hence for these variables will not be checked.

CONCLUSION

Now I have a utility that only needs a data set name to check and list invalid data values. I have always been in the habit of permanently associating my formats with variables and hence found this method of checking data particularly useful and simple to apply.
For some variables missing values are invalid, for others they are not. Since I started using this utility I have begun to add labels to formats where missing values are valid. Alternatively a line of code can remove missing values from the data checking process.

REFERENCES

“Technical Tips”, (1997), SAS communications, Q1,48

SAS is a registered trademark of the SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

David Trenery
Hoechst Marion Roussel, Broadwater Park, Denham, Uxbridge, Middx UB9 5HP,UK
(0044) 1895-837803
David.Trenery@hmrag.com