

Building a Better Dataset Descriptor

Mike Tangedal, United Healthcare Corporation, Minnetonka, MN

ABSTRACT

PROC CONTENTS is a useful tool in describing the structure of your dataset. However, it does not provide any information as to the data values within. I have developed a macro which will list counts of missing observations, counts = 0, min and max, and a unique value list for the variables in the dataset, depending on formats set up by the user. This will be listed in a report similar to the standard look of PROC CONTENTS. This is accomplished through judicious use of PROC MEANS and PROC FREQ. These procedures are used but once, thus keeping the increase in processing time over using just PROC CONTENTS to a minimum. A complete description of the macro code is included.

INTRODUCTION

One of the tasks often put upon a data warehouse is data extraction into smaller datasets from which reports can be quickly compiled. In a managed health care environment such as United Healthcare's Applied Healthcare Informatics division, varying extracts of medical claims data are often needed for different research queries.

In the past, a simple PROC CONTENTS followed by a listing of a few records was performed after the data extraction. However, this does not serve as sufficient means to verify the often complex selection criteria specified in the data extraction process. Listing records specific to each selection criteria is not feasible due to the complexity of the allowed selection criteria. Therefore, a need was created for a generic dataset descriptor which could serve to quickly verify data extractions.

This paper describes the SAS macro developed for just this purpose. In addition to selected fields from the PROC CONTENTS listing, a count of number of missing values, a count of number of values = 0, a min and max, and unique values along with their frequency count for a variable can be obtained. After a data extraction is performed, this report is generated which appears as a PROC CONTENTS listing with added fields.

SCOPE

The macro presented here is generic enough so that with slight modifications, it can be applied to any dataset.

Assumptions here are that the dataset passed into the macro has a variable list which is known (at least mostly) to the user. The user also has to pre-determine desired processing for each variable in the dataset. The choices of processing are either a list of unique values, the min and max of that variable, or the number of values = 0. By default the number of records where that value is missing is listed.

COMPONENTS OF THE MACRO

The macro performs the following functions:

- 1) A temporary dataset from PROC CONTENTS is created.
- 2) For each record in this dataset....
 - a) the processing instruction category is retrieved
 - b) SAS statements are written based on these processing instructions.
- 3) These processing instructions are executed within a PROC MEANS and a PROC FREQ.
- 4) The results of the PROC MEANS and PROC FREQ are merged with the temporary dataset.
- 5) The temporary dataset is printed.

CREATE PROC CONTENTS DATASET

The only required parameter passed into this macro is the name of the file to be processed. The other parameters passed are the names of the four temporary flat file locations needed to store results. By default these are 'meantext', 'meanlist', 'freqtext', and 'freqlist'. These are flat files and must be at least 80 characters wide.

The 'type' of the dataset can be determined from the dataset name. This is the name of the user-defined format specific to the dataset being reported upon. *This example pertains to medical claims data, so the choices for this variable are '\$DOCTOR', '\$HOSP.', and '\$PHARM.'*

```
%MACRO DESCRIPT(FILEIN=,
/* Location of dataset to be analyzed */
MEANTEXT=meantext,
/* storage of PROC MEANS contents */
MEANLIST=meanlist,
/* storage of PROC MEANS results */
FREQTEXT=freqtext,
/* storage of PROC FREQ contents */
FREQLIST=freqlist
/* storage of PROC FREQ results */
);

/*****
* set FTYPE variable to proper format
* name
*****/
%LET FTYPE = %SUBSTR(&FILEIN,1,2);
%IF &FTYPE = DR %THEN
    &FTYPE = $DOCTOR.;
%IF &FTYPE = HL %THEN
    &FTYPE = $HOSP.;
%IF &FTYPE = RX %THEN
    &FTYPE = $PHARM.;
```

The first step in processing the dataset is to retrieve all needed information from the output of PROC CONTENTS. Retained are some statistics from the dataset referenced by "&FILEIN" as well as the variable name, type, length, format, and label fields. The format description is derived from manipulating the format descriptor fields in the PROC CONTENTS listing (FORMATD, FORMATL, and FORMAT).

```
/*****
* Get a list of all variables in the dataset
*****/
PROC CONTENTS DATA = &FILEIN NOPRINT
OUT = CTENT;
DATA CTENT
(DROP=TYPE FORMAT FORMATL FORMATD);
SET CTENT
(KEEP = LIBNAME MEMNAME NAME TYPE LENGTH
LABEL FORMAT FORMATL FORMATD VARNUM
NOBS CRDATE);
LENGTH TYPEVAR $4 FMT $12;
IF TYPE = 1 THEN TYPEVAR = 'NUM';
ELSE IF TYPE = 2 THEN TYPEVAR = 'CHAR';
```

```
IF FORMATD > 0 THEN FMT =
TRIM(FORMAT) ||
TRIM(LEFT(PUT(FORMATL,4.))) || '.' ||
TRIM(LEFT(PUT(FORMATD,4.)));
ELSE IF FORMAT NE '' THEN FMT =
TRIM(FORMAT) ||
TRIM(LEFT(PUT(FORMATL,4.))) || '.';
ELSE IF FORMATL > 0 THEN FMT =
TRIM(LEFT(PUT(FORMATL,4.))) || '.';
proc sort data = ctent; by name;
run;
```

PROCESS DATASET VARIABLES

The next step is to apply the desired level of processing to each of the variables in the dataset. It is not known what variables are in the dataset, but all of the variables that could be in the dataset are known. Lookup tables are needed. These lookup tables take the form of user-defined formats. Knowledge of the dataset is required before this task can be completed.

Since these format tables are specific to the dataset being analyzed, it might be best to store them outside the macro and %INCLUDE them into the code based on the value of the macro variable 'FTYPE'. However, for clarity, this code is included within the macro file in this example.

The following list shows all the possible variables that can appear in a doctor claims dataset along with the type of processing that applies to each. 'MISSING' implies that the only statistic needed from this variable is the number of missing observations. This statistic is generated for each variable, but for some variables, this is the only statistic required. For 'ZERO' variables, the number of observations = 0 is also needed. For 'MINMAX' variables, the minimum and maximum values for that variable are needed. For 'UNIQUE' variables, a listing of each value along with the count is needed.

```
proc format;
value $doctor
'PRVKEY', 'KEY', 'DOB', 'AUDIT', 'CPTCODE',
'CPTDATE', 'FROMDATE', 'ICDCODE', 'MEMBID',
'THRUDATE', 'FIN_LGL', 'FIN_PFD', 'PAY_CDE',
'POSTDATE', 'PROCESS', 'PROD_TYP', 'WHICH_CDE'
= 'MISSING'

'PROVID', 'CAPPCF', 'NBR_TIME', 'REFERID',
'AUTHORIZ', 'DIAG1', 'DIAG2', 'DIAG3', 'DIAG4',
'CPTMOD', 'PROV_CLM' = 'ZERO'

'AMTCLM', 'AMTCOPAY', 'AMTDEDC', 'AMTDISC',
'AMTPAID', 'AMTRESER', 'SITE_CDE',
'AMTDISLW', 'AMTCONTR', 'AMT_PFA'
= 'MINMAX'

'BUS_LINE', 'ASO_FLG', 'COBCODE', 'COPAYTYP',
'ENCLNTFLG', 'HMOID', 'PARCODE', 'SEX',
'DTL_NBR', 'ICD_NBR', 'SERV_PAR', 'COVERTYP',
'EPSDTIND' = 'UNIQUE'

OTHER = 'MISSING';
```

All potential variables for hospital and pharmacy claims datasets are also listed along with their associated type of processing.

```
value $hosp
'PRVKEY', 'PROVID', 'KEY', 'DOB', 'ADMIT_HR',
'AUDIT', 'BILLTYPE', 'COPAYTYP', 'DISCH_HR',
'FIN_LGL', 'FROMDATE', 'MEMBID', 'REVCODE',
'THRUDATE', 'FIN_PRD', 'PAY_CDE', 'POSTDATE',
'PROCESS', 'PROD_TYP', 'WHICHNBR', 'WHICHNBR'
= 'MISSING'

'AUTHORIZ', 'CAPPCP', 'PROC1', 'PROC2',
'PROC3', 'PROC4', 'PROC5', 'PROC6', 'DIAG1',
'DIAG2', 'DIAG3', 'DIAG4', 'DIAG5', 'DIAG6',
'DIAG7', 'DIAG8', 'DIAG9', 'PATSTAT',
'REFERID', 'PROV_CLM', 'DRGENTRY', 'DRGSYSM'
= 'ZERO'

'AMTCLM', 'AMTCONTR', 'AMTCOPAY', 'AMTDEDC',
'AMTDISC', 'AMTPAID', 'AMTDISLW', 'SITE_CDE',
'AMTRESER', 'DTL_NBR', 'NBR_UNIT'
= 'MINMAX'

'BUS_LINE', 'ASO_FLG', 'ADMITSRC', 'COBCODE',
'ENCNTPFLG', 'PARCODE', 'PROCETH', 'SERV_PAR',
'ADMITTYP', 'COVERTYP', 'HMOID', 'SEX'
= 'UNIQUE'

OTHER = 'MISSING';

value $pharm
'DEANER', 'PRVKEY', 'PRESCDR', 'KEY',
'PROCESS', 'PROD_TYP', 'DOB', 'FILLDATE',
'PAIDDATE', 'FIN_LGL', 'FIN_PRD', 'AUDIT',
'AUTHTYPE', 'BPL', 'DAWCDE', 'DAW_MAX',
'DRUG_STR', 'MEMBID', 'NDCCODE', 'PAY_CDE',
'PHARMID', 'RX_NBR', 'THERCLS', 'AUTH_NBR',
'WHICHNBR', 'WHICHNBR' = 'MISSING'

'CAPPCP', 'DOSAGECD', 'GENCODE', 'QTY_DRUG',
'PROV_CLM', 'DAY_SPLY' = 'ZERO'

'AMTCLM', 'AMTCOPAY', 'AMTDEDC', 'AMTDISLW',
'AMTPAID', 'CAL_COST', 'DISP_FEE' = 'MINMAX'

'BUS_LINE', 'ASO_FLG', 'CMPDTYPE', 'COPAYTYP',
'GPI', 'HMOID', 'MULTCPAY', 'PARCODE',
'REFILCDE', 'SEX', 'REVCODE' = 'UNIQUE'

OTHER = 'MISSING';
```

The following formats are used within a PROC FREQ to group the number of observations which fall into certain categories. Formats for both numeric and character variables are needed.

```
/* Format to use when 'Number = 0' stat
* is required for a numerical variable */
VALUE ZNUM .='MISSING' 0='ZERO'
OTHER='OTHER';

/* Special format to use when
* 'Number = 0' stat
```

```
* is required for a numerical variable
* in which the value '88' is interpreted
* as zero */
VALUE ZSUPPLY .='MISSING' 0,88='ZERO'
OTHER='OTHER';
```

```
/* Format to use when 'Number of values
* = missing' is required for numerical
* variable */
VALUE MNUM .='MISSING' OTHER='OTHER';
```

```
/* Format to use when 'Number of values
* = 0' is required for character variable
*/
VALUE $ZCHAR
'0', '00', '000', '0000', '00000', '000000',
'0000000', '00000000', '000000000',
'0000000000' = 'ZERO'
'' = 'MISSING' OTHER='OTHER';
```

```
/* Format to use when 'Number of values
* = missing' is required for character
* variable */
VALUE $MCHAR ''='MISSING' OTHER='OTHER';
```

The following format lists the type of processing for each variable. The type of processing is the name of another format. This is a format table that looks up another format. Again, this is specific to the variable lists in the specific datasets encountered. The user should define this format before executing the macro.

The variables listed here are the variables requiring either just the 'count of missing variables' processing or the 'number of observations = 0' processing.

```
VALUE $ZFMAT
'PRVKEY', 'KEY', 'AUDIT', 'CPTCODE', 'ICDCODE',
'MEMBID', 'WHICHNBR', 'BILLTYPE', 'COPAYTYP',
'REVCODE', 'DEANER', 'PRESCDR', 'AUTHTYPE',
'BPL', 'DAWCDE', 'GPI', 'NDCCODE', 'PHARMID',
'THERCLS' = '$MCHAR.'

'DOB', 'CPTDATE', 'FROMDATE', 'THRUDATE',
'FIN_LGL', 'FIN_PRD', 'PAY_CDE', 'POSTDATE',
'PROCESS', 'PROD_TYP', 'ADMIT_HR', 'DISCH_HR',
'WHICHNBR', 'FILLDATE', 'PAIDDATE', 'DAW_MAX',
'DRUG_STR', 'RX_NBR', 'AUTH_NBR' = 'MNUM.'

'PROVID', 'CAPPCP', 'REFERID', 'AUTHORIZ',
'DIAG1', 'DIAG2', 'DIAG3', 'DIAG4', 'CPTMOD',
'PROC1', 'PROC2', 'PROC3', 'PROC4', 'PROC5',
'PROC6', 'DIAG5', 'DIAG6', 'DIAG7', 'DIAG8',
'DIAG9', 'PATSTAT', 'DRGENTRY', 'DRGSYSM',
'DOSAGECD', 'GENCODE' = '$ZCHAR.'

'NBR_TIME', 'PROV_CLM', 'QTY_DRUG' = 'ZNUM.'

/* special format where '88' = zero */
'DAY_SPLY' = 'ZSUPPLY.';
run;
```

Processing now begins on each variable in the dataset. Temporary file space is utilized to store the code that is needed to process each variable and the results of PROC FREQ and PROC MEANS. This is done so that all variables can be analyzed through one execution of PROC FREQ and PROC MEANS. The emphasis is to add as little time to the data extraction process as possible while still producing a report that is clear and concise. Multiple executions of PROC MEANS and/or PROC FREQ would add to job time. Execution time is at a premium.

Note that 'nocenter', 'nodate', and 'nolabel' options are enacted before PROC PRINTO is executed. This is to ensure that the results of PROC FREQ and PROC MEANS are left-justified and readily available in the flat file created.

```

/**** PROCESS EACH RECORD ****/
options nocenter nodate nolabel;
/* Set nocenter and nodate and nolabel
 * options because the results of
 * PROC FREQ and PROC MEANS are going
 * to be stored to a flat file via
 * PROC PRINTO and read back in */

/* process each variable in SAS dataset */
data _NULL_;
  set ctent end=last;

  length proccat fmt $8;
  /* Get processing instruction category
   * from format lookup table */
  PROCCAT = put(name,&FTYPE);

  /* Set flags for PROC MEANS and
   * PROC FREQ processing */
  length meanflag freqflag $1;
  retain meanflag freqflag;
  if _N_ = 1 then do;
    meanflag = 'N'; freqflag = 'N'; end;

  /***** Go to decision table for each
   * variable category
   *
   * MINMAX variables are processed with
   * PROC MEANS
   *
   * MISSING, ZERO, & UNIQUE variables are
   * processed with PROC FREQ
   *****/
  if proccat in ('MISSING','ZERO','UNIQUE')
    then do;

  /*****
   * If first time at decision category,
   * turn on flag for that process and write
   * header code to file.
   *****/
    if freqflag = 'N' then do;
      freqflag = 'Y';

```

An execution of PROC FREQ is now built in an external file. Since processing time would increase if multiple runs

were executed, one extended execution of PROC FREQ is built by sending SAS code out to a temporary file. This code is to be read later in the program. '&FREQTEXT' contains the SAS code used in executing the PROC FREQ statement. '&FREQLIST' contains results of running PROC FREQ.

```

file &FREQTEXT old;
put "options missing = '*'";
put "proc printto new ";
put "    print= &FREQLIST ; ";
put "    run;";
put "proc freq data = &FILEIN ;";
end;

/*****
 * Append coding statements specific to
 * each variable to
 * specific decision category file
 *****/
file &FREQTEXT mod;
if proccat in ('MISSING','ZERO')
  then do;
  fmt = put(name,$ZFMAT.);
  put "format " name fmt ";";
end;

```

A different table is created for each variable read. Whether that table is formatted in some fashion is dictated by the type of processing as noted by 'proccat' and perhaps further noted by 'fmt'.

```

put "tables " name ;
put "    / missing nopercnt;" ;
end;
* proccat in (MISSING,ZERO,UNIQUE);

```

An execution of PROC MEANS is built here. Numerical variables requiring 'min max' processing use PROC MEANS. Once again, just one execution of PROC MEANS is best, so SAS code is generated into a temporary location and read after all the variables have been processed. '&MEANTEXT' contains the SAS code used in executing the PROC MEANS statement. '&MEANLIST' contains results of running PROC MEANS.

```

if proccat = 'MINMAX' then do;
  if meanflag = 'N' then do;
    meanflag = 'Y';
    file &MEANTEXT old;
    put "proc printto new ";
    put "    print= &MEANLIST ; run;";
    put "proc means data = &FILEIN ";
    put "    nway nmiss min max; ";
    put "    var";
  end;

```

A different table is created for each variable read. No formatting is required.

```
file &MEANTEXT mod;
put name ;
end; * proccat = 'MINMAX';
```

After all the variables have been processed, the temporary files are closed with the proper SAS code to finish the execution of PROC FREQ and PROC MEANS respectively.

```
/******
* If last record, then close out each file
* with remaining code
******/
if last then do;
  file &FREQTEXT mod;
  put "options missing = '.';";
  put "proc printto; run;" ;
```

SAS code is written to the file specified by '&FREQTEXT' via PROC PRINTTO. The SAS code reads the flat text file created from the execution of PROC FREQ. PROC FREQ lists variable names, their values, and the counts of these values in a table format. Some of the values are formatted into key words such as 'OTHER', 'MISSING', and 'ZERO'.

```
put "data fsumm (keep = name missing "
put "          unique zero);";
put "  infile &FREQLIST ;";
put "  length name vartemp $8 flag $1 ";
put "          unique $30;";
put "  retain name flag unique;";
put "  retain missing 0 zero;";
put "  input vartemp $ 1-8 @;";
put "  vartemp = left(vartemp);";
```

The 'number of missing' variable, the 'unique values' variable, and the 'number of values that equal zero' variable are assigned by reading the text file and processing based on the first eight characters of each line (variable named 'vartemp').

```
put "  select(vartemp);";
put "    when('') do;";
put "      if flag = 'y' then do;";
put "        output;";
put "        name = ''; flag = 'n';";
put "        missing=0; unique='';";
put "        zero=.;";
put "      end;";
put "    else input;";
put "    end;";
put "  when('OTHER') input;";
put "  when('-----') do;";
put "    flag = 'y'; input; end;";
put "  when('MISSING','*')";
put "    input missing;";
put "  when('ZERO') input zero;";
```

If one of the keywords is not found, then the line contains a value plus a count of the variable in the table. In such cases, a descriptive text string is built

```
put "    otherwise do;";
put "      if flag='y' then do;";
put "        input count;";
put "        unique = trim(unique) || ";
put "        ' ' || trim(left(vartemp));";
put "        || '(' || ";
put "        trim(left(put(count,10)))";
put "        || ')';";
put "      end;";
put "    else do;";
put "      name = scan(vartemp,1);";
put "      input; end;";
put "    end;";
put "  end;";
put "proc sort data = fsumm; by name;";
put "run;";
```

SAS code is also written to the file specified by '&MEANTEXT' via PROC PRINTTO. The SAS code reads the flat text file created from the execution of PROC MEANS. PROC MEANS was executed giving statistics for 'number of missing' (nmiss), minimum value (min), and maximum (max) value. These values are retrieved from the flat file.

```
file &MEANTEXT mod;
put "; run; proc printto; run;" ;
put "data minmax ";
put "  (keep = name missing min max);";
put "  infile &MEANLIST missover;";
put "  length name $8 flag $1;";
put "  retain flag;";
put "  retain missing min max 0;";
put "  input name $ 1-8 @;";
```

The first eight characters of each line (variable named 'vartemp') dictates further processing. Dashes indicate the beginning and ending of tables. Within each table, the number of missing, minimum, and maximum values of that variable are read.

```
put "  if name = '-----' and ";
put "    flag ne 'y' then flag = 'y';";
put "  else if name = '-----' and ";
put "    flag = 'y' then stop;";
put "  else if flag = 'y' then do;";
put "    input missing min max;";
put "    output;";
put "  end;";
put "proc sort data = minmax; by name;";
put "run;";
end; *if last observation;
```

```
run;
```

RUN PROC MEANS & PROC FREQ

SAS code has been generated for a PROC MEANS procedure and a PROC FREQ procedure in two temporary files based on the variables found in the dataset. Now this code is executed and the results are stored in flat files.

```

/*****
* Run PROC MEANS and PROC FREQ code
*****/
%include &MEANTEXT /SOURCE2;
run;
%include &FREQTEXT /SOURCE2;
run;

```

MERGE RESULTS

The results of the PROC MEANS dataset, the PROC FREQ dataset, and the PROC CONTENTS dataset are merged.

```

/*****
* Merge these datasets with the original
* contents dataset to create a full dataset
*****/
data temp;
  set fsumm minmax;
  by name;
data cntent;
  merge cntent temp;
  by name;
run;

```

PRINT RESULTS

The dataset as an emulation of a PROC CONTENTS report is printed. Note that the system options are set back to 'center' for printing and that 'nobyline' is set to take advantage of printing results of PROC CONTENTS in the title lines.

In this case, maximum print width is 132 characters, so this modified PROC CONTENTS listing is formatted to that page width. An increase in page width would allow for more information to be stored under the 'unique values' heading for one. (This text sting is limited to 30 characters in this example.)

Also noteworthy is that the first two title lines are reserved for information from the PROC CONTENTS. Additional title information can be specified in 'TITLE' statements before the invocation of this macro.

```

/*****
* Print this dataset
*****/
PROC SORT DATA = CTENT;
  BY LIBNAME MEMNAME NOBS CRDATE NAME;
options center date nobyline label;
PROC PRINT DATA = CTENT
  NOOBS LABEL DOUBLE uniform;
  BY LIBNAME MEMNAME NOBS CRDATE;
TITLE1
  'ADJKEY RESULTS: LISTING OF #BYVAL (MEMNAME) '
  ' FROM LIBRARY #BYVAL (LIBNAME) ' ;
TITLE3 '# OBS: #BYVAL (NOBS) '
  '      CREATION DATE: #BYVAL (CRDATE) ' ;
VAR NAME TYPEVAR LENGTH FMT LABEL MISSING
ZERO MIN MAX UNIQUE;
LABEL NAME = 'VARIABLE'
  TYPEVAR = 'TYPE' FMT = 'FORMAT'
  LENGTH = 'LENGTH' LABEL = 'LABEL'
  MISSING = '# MISSING' ZERO = '# = 0'
  UNIQUE = 'UNIQUE VALUES';
RUN;

```

CONCLUSION

The power of this macro is that it provides an effective verification report for data extracts without adding significantly to the processing time. This is done through building temporary files for the execution of but one run of PROC FREQ and PROC MEANS.

As useful as this macro might be, it is not a 'plug in'. Modifications are needed in order for it to be of any use. It is most useful on datasets for which a known potential variable list exists. And for those variables, decisions have to be made on processing before the macro is invoked. Then format tables need to be built. Once this is done however, this macro can prove to be quite useful on each data extraction. If quick but effective data verification is needed on extracted datasets from a larger data warehouse where the potential variable list is known, your version of this macro may be just the thing.

Comments, suggestions, and questions are welcome. The author can be reached at the following address:

Mike Tangedal
United Healthcare Corporation
Mail Route: MN008-W109
Minnetonka, MN 55343

phone: (612) 936-7464
email: mtanged@uhc.com

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries.