A Step-By-Step Illustration of Building a Data Analysis Tool with Macros

Diana Wobus, Westat
John Gober, Bureau of Census

ABSTRACT
Grouping observations in a data set into subgroups according to their percentiles based on a continuous variable (expense, age, or score) and then generating statistics for each subgroup can be tedious. By applying macros appropriately, however, one can achieve greater program efficiency and error-proof results. This paper illustrates a rational process of building a data analysis tool that demonstrates the power and efficiency of macros. By describing three approaches that we employed in a real life project, this paper shows how each one improves the tool as our methodology develops. Using a step-by-step illustration of macros in DATA steps and statistical procedures to solve the problem, this paper helps SAS® programmers who have just begun tapping into "mysterious" macro programming to learn about some of the use functions of macros. It also demonstrates the rationales that led us to the results we desired. We assume that readers of this paper have a basic understanding of macros (and macro variables), do-loops, and data set manipulation procedures.

INTRODUCTION
Several methods are available to generate summary statistics for subgroups of observations in a large data set. The "by" or "class" statements used in several SAS statistical procedures can do the job. If, however, observations are grouped by different percentiles of the data set based on a continuous variable, it is not a simple matter to just execute procedures with a "by" or "class" statement. The chance of lengthy coding and error-generating increases greatly.

In a project of estimating health services expenditures, we tried three different approaches for generating statistics for observations grouped by percentiles of the expenditure variable. As a result, we developed a data analysis tool that is very flexible and uses minimal coding to obtain subgroup statistics. Its main strength is macro programming, which enables analysts to accomplish common statistical analyses with greater efficiency and convenience.

OBJECTIVE
It's a well established fact in the health services financing and research that a small percentage of health services users (i.e. anyone who used services by a health care provider such as hospitals, office-based doctors, dentists, home health agencies, etc) consume a large percentage of available resources (measured in dollars). An important part of our project is to examine how much of the national top health services dollars have been used by what proportion of the users and what is the trend. To do this analysis, we need to have an analysis tool that allows us to generate the statistics repeatedly with varying parameters.

Suppose, during our preliminary analysis, we observe that users in the upper 10 percentile (above 90%) of the expenditures incurred the largest costs to the nation’s health care system. We then decide to look closer at these users, say, at every ten percentile below 90 percentile and at every one percentile between 90 and 99 percentiles and at every tenth percentile for the upper one percent. The objective, therefore, is to produce a table containing summary payment information for individuals at specified percentiles of the population based on payment.
SAMPLE DATA

The data set we will use is the 1998 person level summary data (downloaded from AHRQ's web site) from Medical Expenditure Panel Survey, a longitudinal survey producing a nationally representative estimate of health services expenditures. Each observation on the file represents a user and contains a total charge and a total payment for the person's medical care in the fiscal year of 1998 as well as the payment for each type of medical care services, such as inpatient stay, outpatient department visits, office-based doctor visits, dental care, home health care, etc. The file includes all persons who were sampled through a complex survey design. To simplify the illustration, the unweighted expenditures are used.

METHODOLOGY

To demonstrate how we developed our analysis tool, we will describe the three approaches we used, in the following three sections.

1. Univariate and Summary Approach

With the yearly summary payment as the analysis variable, the UNIVARIATE procedure allows us to specify percentiles, generating for us payment values corresponding to each specific percentile. The thought is that if we can find the dollar value, then we can group recipients within each percentile and then generate summary statistics. To find the corresponding values, we execute the following code:

Illustration 1.a

```plaintext
proc univariate data = year98.h28 noprint;
var totpay98;
output out = univ
  pctlpts = 10 to 100 by 10
  pctlpre = pct_;
run;
```

In the input data set, YEAR98.H28, the analysis variable is TOTEXP98, the yearly total payment. The "output" statement does a number of things. The "out=" option tells the procedure to output a temporary data set, UNIV; the "pctlpts=" option specifies the percentile range, each percentile resulting in a variable in the output data set; and the "pctlpre=" option assigns to each new variable in the output data set a text string as the variable name, with a prefix of our choosing. We use "pct_" as the prefix of the variable names.

As shown in Table 1., the output data set consists of one record containing 10 variables. Their names correspond to the percentiles (and dollar values) specified in PROC UNIVARIATE.

<table>
<thead>
<tr>
<th>pct_10</th>
<th>pct_20</th>
<th>pct_30</th>
<th>pct_40</th>
<th>pct_50</th>
<th>pct_60</th>
<th>pct_70</th>
<th>pct_80</th>
<th>pct_90</th>
<th>pct_100</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.00</td>
<td>45.00</td>
<td>133.00</td>
<td>256.00</td>
<td>424.00</td>
<td>690.00</td>
<td>1,133.00</td>
<td>2,113.00</td>
<td>4,488.00</td>
<td>378,588.00</td>
</tr>
</tbody>
</table>

The next step, intuitively, is to summarize yearly payments for persons within each percentile. The MEANS or SUMMARY procedure can easily accomplish this using a "where" statement to set the value boundaries for each percentile. However, in order to generate the summary statistics for persons at each percentile, we need to execute a PROC SUMMARY for as many times as there are percentiles. The result, unfortunately, is a very lengthy program, as illustrated below:

Illustration 1.b

```plaintext
proc summary data=year98.h28 n sum max min maxdec=0;
var totpay98;
where totpay98 <= 0;
output out=pct_20 n=n sum=sum max=max min=min;
```
run;

proc summary data=year98.h28 n sum max min maxdec=0;
   var totexp98;
   where totexp98 <= 45;
   output out=pct_30 n=n sum=sum max=max min=min;
run;

proc summary data=year98.h28 n sum max min maxdec=0;
   var totexp98;
   where totexp98 <= 133;
   output out=pct_40 n=n sum=sum max=max min=min;

..........
...
(more PROC steps omitted)
...

proc summary data=year98.h28 n sum max min maxdec=0;
   var totexp98;
   where totexp98 <= 2113;
   output out=pct_80 n=n sum=sum max=max min=min;
run;

proc summary data=year98.h28 n sum max min maxdec=0;
   var totexp98;
   where totexp98 <= 4488;
   output out=pct_90 n=n sum=sum max=max min=min;
run;

proc summary data=year98.h28 n sum max min maxdec = 0 ;
   var totexp98;
   output out=pct_100 n=n sum=sum max=max min=min;
run;

The final step is to concatenate all of the output data sets from PROC SUMMARY into a single data set, using either a DATA step or a DATASETS procedure. A printout of the merged data set using a DATA step is shown in Table 2.

Illustration 2.

data final;
   set pct_10  pct_20  pct_30  pct_40  pct_50
      pct_60  pct_70  pct_80  pct_90  pct_100;
run;

proc print data=final width=min;
   var n sum max min;
   title 'Table 2.';
   format n comma8. sum max min comma13.;
run;

Table 2.

<table>
<thead>
<tr>
<th>Obs</th>
<th>n</th>
<th>sum</th>
<th>max</th>
<th>min</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,718</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>5,848</td>
<td>29,978</td>
<td>45</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>8,383</td>
<td>250,615</td>
<td>133</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>10,780</td>
<td>706,070</td>
<td>256</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>13,033</td>
<td>1,459,970</td>
<td>424</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>15,229</td>
<td>2,655,427</td>
<td>690</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>17,351</td>
<td>4,550,585</td>
<td>1,133</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>19,524</td>
<td>7,935,885</td>
<td>2,113</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>21,724</td>
<td>14,784,849</td>
<td>4,488</td>
<td>0</td>
</tr>
<tr>
<td>10</td>
<td>24,072</td>
<td>48,752,178</td>
<td>378,588</td>
<td>0</td>
</tr>
</tbody>
</table>
Here we see several problems. First, we are forced to manually enter the value for each percentile into each of the SUMMARY procedures. Second, to generate non-cumulative numbers, we need to enter both the lower and upper values of each percentile, greatly increasing the amount of coding and chance for error. Third, a slight change in the percentile specification in PROC UNIVARIATE changes every value, which means we have to re-enter the values for every PROC SUMMARY all over again. With every increase in the number of percentiles, the program becomes longer, and the risk of errors becomes greater. Additionally, the result in Table 2. does not indicate for which percentile the statistics (numbers in rows) are generated. For this approach we have to remember the variables in the PROC UNIVARIATE output data set and somehow add the labels to the final data set either manually or through additional data set manipulation.

2. Ranking Approach

The approach using RANK procedure is a lot easier. We begin by specifying the number of groups. PROC RANK assigns a rank score to each of the observations in the data set in the order of a numeric variable, which in this case is the yearly payment. Once the rank order is assigned, PROC RANK generates summary statistics for observations in each rank. We execute the following code, as an example:

Illustration 3.

```
proc rank data=year98.h28 out=univ group=5;
  var totexp98;
  ranks rank;
run;

proc summary data=univ;
  class rank;
  var totexp98;
  output out=final n=n sum=sum max=max min=min;
run;
```

Here, the "group=" option in "proc rank" statement specifies the number of groups into which we want to divide the total observations. For example, an option of "group=100" breaks the observations in the data set into 100 groups, with one percent of the observations in each group. Likewise, an option of "group=20" breaks the observations into 20 groups, five percent of the total observations per group; an option of "group=25" breaks the observations into 25 groups, four percent per group; and so forth. The "ranks" statement specifies the name of the variable containing rank scores. If no "ranks" statement is used, the name of the analysis variable on the "var" statement will be used for rank scores (See SAS document for RANK PROCEDURE). A printout of the output data set is shown in Table 3.

<table>
<thead>
<tr>
<th>rank</th>
<th>n</th>
<th>sum</th>
<th>max</th>
<th>min</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>4,718</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>87</td>
<td>402</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>2,421</td>
<td>121,822</td>
<td>89</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2,401</td>
<td>328,806</td>
<td>189</td>
<td>90</td>
</tr>
<tr>
<td>4</td>
<td>2,411</td>
<td>628,818</td>
<td>341</td>
<td>190</td>
</tr>
<tr>
<td>5</td>
<td>2,405</td>
<td>1,079,420</td>
<td>575</td>
<td>342</td>
</tr>
<tr>
<td>6</td>
<td>2,410</td>
<td>1,858,598</td>
<td>1,004</td>
<td>576</td>
</tr>
<tr>
<td>7</td>
<td>2,404</td>
<td>3,375,736</td>
<td>1,943</td>
<td>1,005</td>
</tr>
<tr>
<td>8</td>
<td>2,408</td>
<td>7,128,617</td>
<td>4,405</td>
<td>1,944</td>
</tr>
<tr>
<td>9</td>
<td>2,407</td>
<td>34,229,959</td>
<td>378,588</td>
<td>4,409</td>
</tr>
</tbody>
</table>

We seem to now have a simpler program with the ranking approach, which functions in the same way as PROC UNIVARIATE in grouping cases and it provides minimum values. However, the Rank procedure lacks the flexibility we get from the UNIVARIATE procedure. We are unable to generate groups of varying percentiles in one step as in PROC UNIVARIATE. We have to execute as many PROC RANKS as there
are percentile specifications. For example, we must execute three RANK procedures in order to obtain observations at: 1) every ten percentile up to 90 percentile; 2) every one percentile between 91 and 99 percentile range; and 3) every tenth percentile for the upper one percentile. We then have to do more data manipulation and cut and paste the results to generate a single table.

3. Univariate and Summary Procedures with Macros

Ideally, we would like to write a program that is both flexible with percentile specifications and simplistic with a one-step process. We also want to avoid hard coding and have the program automatically feed the percentile values, one at a time, into as many SUMMARY procedures as there are percentiles. We find that SAS macros are the answer.

We use PROC UNIVARIATE as described in our first approach to achieve varying percentiles, such as:

Illustration 4.a

```
proc univariate data=year98.h28 noprint;
  var totals;
  output out = univ
    pctlpts = 10 to 90 by 10, 91 to 99 by 1, 99.1 to 100 by .1
    pctlpre = pct_;
run;
```

From the output data set, UNIV, we know there are 28 variables each for a percentile specified. We then write the following macro that gives us the desired performance. This macro, METHOD3, is described in segments, as follows:

Illustration 4.b

```
%macro method3;
  data _null_; set univ;
  array pct2[28] pct_10 -- pct_100;
  %do i = 1 %to 28;
    call symput("var&i", put(pct2[&i], 8.));
  %end;
run;
```

The DATA step in the macro operates as follows: it takes the output data set from the PROC UNIVARIATE (containing one observation with as many variables as specified); the do-loop scans through the variables in the array, during which its "call symput" function retains each value in a macro variable named "var&i," with "&i" being one of the array elements. At invocation, the macro variables will pass the values contained in them to where each is called for, as shown in Illustration 4.c:

Illustration 4.c

```
%do i = 1 %to 28;
  proc summary data=year98.h28 n sum min max maxdec=0;
    var totexp98;
    where totexp98 <= &&var&i;
    output out = out&i (drop = _type_ _freq_) n=n sum=sum max=max min=min;
  run;
%end;
```

```
proc datasets;
  delete final; *use DELETE statement if testing your code multiple times;
%do i = 1 %to 28;
  append base=final data=out&i force;
%end;
```
In the first do-loop, the "where" and "output" statements in PROC SUMMARY resolve "var&i" with the retained values passed on from the macro variables. The "where" statement specifies the cut-off point below which the dollar amount of all observations is summed, resulting a cumulative sum of payment. The "&&" in the macro variables scans the statement twice. The first scan resolves "&&" into "&" and "&i" into numbers, and the second scan further resolves them into actual payment values. The "output' statement outputs the summary statistics of observations at each percentile into a uniquely named data set. Then all output data sets from the SUMMARY procedures are added (concatenated) together to form a table with PROC DATASETS using a second do-loop. The macro ends with a "%mend" statement. Invoking the macro with a "%method3" call, we have the result as shown in Table 4.

Table 4.

<table>
<thead>
<tr>
<th>Obs</th>
<th>n</th>
<th>sum</th>
<th>max</th>
<th>min</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4,718</td>
<td>0.00</td>
<td>1.00</td>
<td>0.00</td>
</tr>
<tr>
<td>2</td>
<td>4,830</td>
<td>577.00</td>
<td>7.00</td>
<td>0.00</td>
</tr>
<tr>
<td>3</td>
<td>7,226</td>
<td>122,224.00</td>
<td>89.00</td>
<td>0.00</td>
</tr>
<tr>
<td>4</td>
<td>9,656</td>
<td>456,540.00</td>
<td>190.00</td>
<td>0.00</td>
</tr>
<tr>
<td>5</td>
<td>12,038</td>
<td>1,079,848.00</td>
<td>341.00</td>
<td>0.00</td>
</tr>
<tr>
<td>6</td>
<td>14,449</td>
<td>2,162,724.00</td>
<td>576.00</td>
<td>0.00</td>
</tr>
<tr>
<td>7</td>
<td>16,853</td>
<td>4,017,866.00</td>
<td>1,004.00</td>
<td>0.00</td>
</tr>
<tr>
<td>8</td>
<td>19,259</td>
<td>7,397,490.00</td>
<td>1,944.00</td>
<td>0.00</td>
</tr>
<tr>
<td>9</td>
<td>21,665</td>
<td>14,522,219.00</td>
<td>4,405.00</td>
<td>0.00</td>
</tr>
<tr>
<td>10</td>
<td>21,906</td>
<td>15,636,783.00</td>
<td>4,876.00</td>
<td>0.00</td>
</tr>
<tr>
<td>11</td>
<td>22,147</td>
<td>16,884,359.00</td>
<td>5,485.00</td>
<td>0.00</td>
</tr>
<tr>
<td>12</td>
<td>22,387</td>
<td>18,286,868.00</td>
<td>6,236.00</td>
<td>0.00</td>
</tr>
<tr>
<td>13</td>
<td>22,628</td>
<td>19,893,031.00</td>
<td>7,146.00</td>
<td>0.00</td>
</tr>
<tr>
<td>14</td>
<td>22,869</td>
<td>21,765,330.00</td>
<td>8,489.00</td>
<td>0.00</td>
</tr>
<tr>
<td>15</td>
<td>23,110</td>
<td>24,003,117.00</td>
<td>10,207.00</td>
<td>0.00</td>
</tr>
<tr>
<td>16</td>
<td>23,350</td>
<td>26,749,037.00</td>
<td>12,703.00</td>
<td>0.00</td>
</tr>
<tr>
<td>17</td>
<td>23,591</td>
<td>30,316,672.00</td>
<td>17,417.00</td>
<td>0.00</td>
</tr>
<tr>
<td>18</td>
<td>23,832</td>
<td>35,721,994.00</td>
<td>30,123.00</td>
<td>0.00</td>
</tr>
<tr>
<td>19</td>
<td>23,856</td>
<td>36,475,517.00</td>
<td>32,398.00</td>
<td>0.00</td>
</tr>
<tr>
<td>20</td>
<td>23,880</td>
<td>37,267,012.00</td>
<td>33,823.00</td>
<td>0.00</td>
</tr>
<tr>
<td>21</td>
<td>23,904</td>
<td>38,111,560.00</td>
<td>37,186.00</td>
<td>0.00</td>
</tr>
<tr>
<td>22</td>
<td>23,928</td>
<td>39,040,060.00</td>
<td>40,406.00</td>
<td>0.00</td>
</tr>
<tr>
<td>23</td>
<td>23,952</td>
<td>40,051,096.00</td>
<td>43,991.00</td>
<td>0.00</td>
</tr>
<tr>
<td>24</td>
<td>23,976</td>
<td>41,168,810.00</td>
<td>49,799.00</td>
<td>0.00</td>
</tr>
<tr>
<td>25</td>
<td>24,000</td>
<td>42,424,344.00</td>
<td>54,348.00</td>
<td>0.00</td>
</tr>
<tr>
<td>26</td>
<td>24,024</td>
<td>43,857,560.00</td>
<td>65,431.00</td>
<td>0.00</td>
</tr>
<tr>
<td>27</td>
<td>24,048</td>
<td>45,718,617.00</td>
<td>93,763.00</td>
<td>0.00</td>
</tr>
<tr>
<td>28</td>
<td>24,072</td>
<td>48,752,178.00</td>
<td>378,588.00</td>
<td>0.00</td>
</tr>
</tbody>
</table>

To take advantage of macros, we will add the following features to our program to achieve flexibility, simplicity, and convenience:

- Let macro variables perform substitution of all changes in percentile specifications.
- Let macro variables capture the names of the variables in the output data set from PROC UNIVARIATE and add the names as labels to the final data set--the final table.
- Let the macro automatically count the variables in the output data set from PROC UNIVARIATE and feed the count value into the subsequent DATA steps.
- Let macro variables obtain noncumulative summary statistics for observations at each percentile. This is a very important feature.
Our final program is shown in the following segments.

Illustration 5.a

*Specify percentile range and analysis variable;
%let range = 10 to 90 by 10, 91 to 99 by 1, 99.1 to 100 by .1;
%let totsum = totexp98;

Once the compilation begins, the first "%let" statement creates a macro variable to conveniently substitute percentile specifications in the program. The second "%let" statement is to allow for using different analysis variables. These two macro variables pass their values to the next PROC UNIVARIATE, which generates percentile values.

Illustration 5.b

%macro method3;

*Generate percentile values;
proc univariate data=year98.h28 noprint;
  var &totsum;
  output out = univ
    pctlpts = &range
    pctlpre = pct_;
run;

Note in PROC UNIVARIATE, we can request different percentile ranges with the macro variable, &range.

Illustration 5.c

*Make sure labels are in correct order;
proc contents data=univ noprint out=varname(keep=name npos);
run;

proc sort data=varname out=vname(keep = name);
  by npos;
run;

*Get names of lower and upper percentiles;
data _null_
  set vname end=last;
  if n_ = 1 then call symput ('fstvar', name);
  if last then call symput ('lstvar', name);
run;

The purpose of the CONTENTS and SORT procedures is to capture the names of variables in the PROC UNIVARIATE output data set to be used in the final table as labels. The two procedures result in a data set, VNAME, containing one variable of the labels sorted in the same order as they are specified in PROC UNIVARIATE. Another method of capturing the first and last variable names based on their position number would be to use the automatically assigned and maintained SASHELP library. Thus, the three steps in Illustration 5.e can substituted with one step below:

data _null_
  set sashelp.vcolumn(where=(libname='WORK' and memname='UNIV'));
  by npos;
  if first.npos then call symput ('fstvar', name);
  if last then call symput ('lstvar', name);
run;
This SASHELP library contains a wealth of metadata about the currently running SAS session. One of the tables in this library, VCOLUMN, contains information on every dataset associated with an open library including libnames, member names, variable names, labels, formats, relative position of variables, and much more. By querying this library with specific conditions we can reduce our three step process down to a single data step to capture the first and the last variable name. A word of warning however, if you have numerous libraries assigned which contain hundreds of datasets with hundreds of variables, this process may not save you any processing time over the PROC CONTENTS procedure.

Illustration 5.d

*Use first and last names in the array to count number of array elements;
DATA _NULL_;
  SET UNIV;
  ARRAY PCT1[*] &FSTVAR -- &LSTVAR;
  CALL SYMPUT ('VARNUM', PUT(DIM(PCT1),8.));
RUN;

*Capture percentile values;
DATA _NULL_;
  SET UNIV;
  ARRAY PCT2[*] &FSTVAR -- &LSTVAR;
  %DO I = 1 %TO &VARNUM;
    CALL SYMPUT ("VAR&I", PUT(PCT2[&I],8.));
  %END;
RUN;

Once the names of the first and the last variables are captured, we used two DATA steps (_null_) to achieve two purposes. The first DATA step takes UNIV file and replaces the macro calls with the names fstvar and lstvar to set the array boundaries. It then counts the number of the array elements to determine the array dimension, which includes all variables in UNIV and contains the count value in another macro variable, varnum. The second DATA step then uses the values from all three macro variables from the previous two DATA steps to perform the following to create, with a do-loop, a series of new macro variables, var&i, that contains the actual payment value from the array elements corresponding to the specified percentiles.

Illustration 5.e

%DO I = 1 %TO &VARNUM;
  %LET J = %EVAL(I-1);
  PROC SUMMARY DATA=YEAR98.H28 N SUM MIN MAX MAXDEC=2;
    VAR &TOTSUM;
    %IF &I = 1 %THEN %DO;
      WHERE &TOTSUM <= &&VAR&I;
    %END;
    %ELSE %DO;
      WHERE &&VAR&J <= &TOTSUM <= &&VAR&I;
    %END;
    OUTPUT OUT = OUT&I (DROP = _TYPE_ _FREQ_) N=n SUM=SUM MAX=MAX MIN=MIN;
  RUN;
%END;

Now, with the lower limit, upper limit, and number of percentile breakdowns, we are ready to group records and generate statistics. In this section, two do-loops are used to generate summary statistics using the values passed on from the macro variables. The "%do i =" statement is an outer do-loop that serves as a counter for percentiles. In this do-loop, the "%let j =" statement is to create an inner loop for capturing the lower limits. The "%eval" function converts "&i" from a text string to a number for numeric calculation.

The second do-loop consists of two parts. The "%if &i = 1" part is to process the first percentile when there is no lower limit. The second part sums the payment variable between the lower (&&var&j) and upper (&&var&i) limits of all other percentiles, resulting in non-cumulative values. At the end of every outer do-
loop, PROC SUMMARY outputs a data set, OUT&I, and the outer do-loop returns for the next PROC SUMMARY until all SUMMARY procedures are executed.

Illustration 5.F

```
proc datasets;
  %do i = 1 %to &varnum;
    append base = merged data=out&i force;
  %end;
run;

data final;
  merge vname merged;
run;

proc print data=final noobs;
  title 'Table 5.';
  format n comma8. sum max min comma13.;
run;

%mend method3;
%method3;
```

With all of the data sets output by PROC SUMMARY, PROC DATASETS concatenates them into one data set, MERGED, with an "append" statement. The last DATA step combines VNAME data set from PROC SORT with MERGED to attach a label to each observation in MERGED to indicate the specific percentile at which statistics are generated. A printout of the final data set, in Table 5., shows the results we have achieved.

```
Table 5.
Output of Final Program

<table>
<thead>
<tr>
<th></th>
<th>n</th>
<th>sum</th>
<th>max</th>
<th>min</th>
</tr>
</thead>
<tbody>
<tr>
<td>pct_10</td>
<td>4,718</td>
<td>0.00</td>
<td>0.00</td>
<td>0.00</td>
</tr>
<tr>
<td>pct_20</td>
<td>112</td>
<td>577.00</td>
<td>7.00</td>
<td>1.00</td>
</tr>
<tr>
<td>pct_30</td>
<td>2,396</td>
<td>121,647.00</td>
<td>89.00</td>
<td>8.00</td>
</tr>
<tr>
<td>pct_40</td>
<td>2,430</td>
<td>334,316.00</td>
<td>190.00</td>
<td>90.00</td>
</tr>
<tr>
<td>pct_50</td>
<td>2,382</td>
<td>623,308.00</td>
<td>341.00</td>
<td>191.00</td>
</tr>
<tr>
<td>pct_60</td>
<td>2,411</td>
<td>1,082,876.00</td>
<td>576.00</td>
<td>342.00</td>
</tr>
<tr>
<td>pct_70</td>
<td>2,404</td>
<td>1,855,142.00</td>
<td>1,004.00</td>
<td>577.00</td>
</tr>
<tr>
<td>pct_80</td>
<td>2,406</td>
<td>3,379,624.00</td>
<td>1,944.00</td>
<td>1,005.00</td>
</tr>
<tr>
<td>pct_90</td>
<td>2,406</td>
<td>7,124,729.00</td>
<td>4,405.00</td>
<td>1,945.00</td>
</tr>
<tr>
<td>pct_91</td>
<td>241</td>
<td>1,114,564.00</td>
<td>4,876.00</td>
<td>4,409.00</td>
</tr>
<tr>
<td>pct_92</td>
<td>241</td>
<td>1,247,576.00</td>
<td>5,485.00</td>
<td>4,877.00</td>
</tr>
<tr>
<td>pct_93</td>
<td>240</td>
<td>1,402,509.00</td>
<td>6,236.00</td>
<td>5,486.00</td>
</tr>
<tr>
<td>pct_94</td>
<td>241</td>
<td>1,606,163.00</td>
<td>7,146.00</td>
<td>6,240.00</td>
</tr>
<tr>
<td>pct_95</td>
<td>241</td>
<td>1,872,299.00</td>
<td>8,489.00</td>
<td>7,153.00</td>
</tr>
<tr>
<td>pct_96</td>
<td>241</td>
<td>2,237,787.00</td>
<td>10,207.00</td>
<td>8,497.00</td>
</tr>
<tr>
<td>pct_97</td>
<td>240</td>
<td>2,745,920.00</td>
<td>12,703.00</td>
<td>10,215.00</td>
</tr>
<tr>
<td>pct_98</td>
<td>241</td>
<td>3,567,635.00</td>
<td>17,417.00</td>
<td>12,706.00</td>
</tr>
<tr>
<td>pct_99</td>
<td>241</td>
<td>5,405,322.00</td>
<td>30,123.00</td>
<td>17,449.00</td>
</tr>
<tr>
<td>pct_99_1</td>
<td>24</td>
<td>753,523.00</td>
<td>32,398.00</td>
<td>30,138.00</td>
</tr>
<tr>
<td>pct_99_2</td>
<td>24</td>
<td>791,495.00</td>
<td>33,823.00</td>
<td>32,427.00</td>
</tr>
<tr>
<td>pct_99_3</td>
<td>24</td>
<td>844,548.00</td>
<td>37,186.00</td>
<td>33,832.00</td>
</tr>
<tr>
<td>pct_99_4</td>
<td>24</td>
<td>928,500.00</td>
<td>40,406.00</td>
<td>37,203.00</td>
</tr>
<tr>
<td>pct_99_5</td>
<td>24</td>
<td>1,011,036.00</td>
<td>43,991.00</td>
<td>40,540.00</td>
</tr>
<tr>
<td>pct_99_6</td>
<td>24</td>
<td>1,117,714.00</td>
<td>49,799.00</td>
<td>44,139.00</td>
</tr>
<tr>
<td>pct_99_7</td>
<td>24</td>
<td>1,255,534.00</td>
<td>54,348.00</td>
<td>50,140.00</td>
</tr>
<tr>
<td>pct_99_8</td>
<td>24</td>
<td>1,433,216.00</td>
<td>65,431.00</td>
<td>54,675.00</td>
</tr>
<tr>
<td>pct_99_9</td>
<td>24</td>
<td>1,861,057.00</td>
<td>93,763.00</td>
<td>65,960.00</td>
</tr>
<tr>
<td>pct_100</td>
<td>24</td>
<td>3,033,561.00</td>
<td>378,588.00</td>
<td>93,920.00</td>
</tr>
</tbody>
</table>
```
CONCLUSIONS

Our experience shows that, when repetitive coding is unavoidable, macros are the most appropriate programming technique to use. The analysis tool we developed for our specific project can easily be adapted to accomplish a variety of tasks. For example, for users without SAS experience, the macro can be modified in such a way that users simply supply the percentile specifications. The macros and macro variables will do the rest. The concept we used in developing the tool can be applied to analyses using other statistical procedures as well. We believe there are alternative approaches for achieving the same results with similar or greater flexibility and efficiency. We welcome ideas and suggestions. Our demonstration exemplifies the thought process by which we reached our conclusion.

REFERENCES


AUTHORS

Diana Wobus obtained her doctorate in Education and Research Methodology in 1994. She is currently involved in imputations of Medical Expenditure Panel Survey data at Westat., Inc. as an analyst. She began SAS programming in the late 1980s and has been an active SAS user in health services research since 1993. Contact at:

Westat
Survey Operations
1650 Research Boulevard, Room RE340
Rockville, MD 20850
Phone: 240-453-2967
Email: wobusd@westat.com

John Charles Gober specializes in the acquisition and manipulation of large data sets. Currently working at the US Census Bureau, he has been programming in SAS for over 20 years. His areas of expertise include improving program performance and efficiency, establishing micro-to-host links on various platforms, and creating applications to aid data migration and processing. Contact at:

U.S. Bureau of the Census
FOB 3 MS 8700
4700 Silver Hill Road
Washington, D.C. 20233-8700
Phone: 301-763-5964
Email: john.charles.gober@census.gov

DISCLAIMER: The contents of this paper are the work of the authors and do not necessarily represent the opinions, recommendations, or practices of Westat or the Census Bureau.

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 registered trademarks or trademarks of their respective companies.