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

Diana Zhang Wobus, University of Maryland, Baltimore, MD
John Charles Gober, Beekeeper, Inc., Alexandria, VA

## 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 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 basics of macros and 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 in each observation, 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 managed-care rate setting project, we tried three different approaches for generating statistics for observations grouped by percentiles. 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.

## BACKGROUND

Amid the national movement toward managed care in public health services, the state of Maryland charged the Center for Health Program Development and Management at the University of Maryland Baltimore County to develop a comprehensive plan for reforming the state's Medicaid Program. The Center is also responsible for developing risk-adjusted capitation rates for the managed care organizations, a process by which payment rates are developed among traditional health maintenance organizations. The capitation rates are based on the Medicaid recipients' current fee-for-service (FFS) costs paid by the Medicaid Program.

## OBJECTIVE

An important part of our analysis of the capitation rates requires us to examine the FFS of the Medicaid population. For this project, we need to review the FFS costs at different percentiles of the population based on their yearly total payments, measured in dollars.

Suppose, during our preliminary analysis, we observe that Medicaid recipients in the upper 10 percentile (above 90%) incurred the largest costs to the Medicaid Program. We then decide to look closer at these recipients, 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 percentile. 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 a fiscal year 1995 person-level summary of Maryland Medicaid paid claims. Each observation on the file contains Medicaid eligibility and payment information for a Medicaid recipient. One of the most important variables on the file is a yearly payment amount summing all types of medical services. For a simple demonstration, we selected at random 5,000 observations from the total FY95 Medicaid population.

## 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 Medicaid 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**

```
proc univariate data = test.sample noprint;
  var totals;
  output out = caprate
         pctlpts = 10 to 100 by 10
         pctlpre = pct_;
run;
```

In the input data set, TEST.SAMPLE, the analysis variable is totals, the yearly total payment. The "output" statement does a number of things. The "out=" option tells the procedure to output a temporary data set, CAPRATE; 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 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 1.**

**Content of Univariate Output Data Set**

| PCT_10 | PCT_20 | PCT_30 | PCT_40 | PCT_50 | PCT_60 |
|---|---|---|---|---|---|
| 86.21 | 165.10 | 272.27 | 398.06 | 589.21 | 958.20 |

| PCT_70 | PCT_80 | PCT_90 | PCT_100 |
|---|---|---|---|
| 1,772.92 | 3,453.23 | 6,896.16 | 299,427.91 |

The text 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 recipients 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**

```
proc summary data=test.sample
             n sum max min maxdec=2;
  var totals;
  where totals <= 86.2;
  output out=pct_10
         n=n sum=sum max=max min=min;
run;

proc summary data=test.sample
             n sum max min maxdec=2;
  var totals;
  where totals <= 165.1;
  output out=pct_20
         n=n sum=sum max=max min=min;
run;
..........
..........
(more PROC steps omitted)
..........
..........
proc summary data=test.sample
             n sum max min maxdec=2;
  var totals;
  where totals <= 6896.2;
  output out=pct_90
         n=n sum=sum max=max min=min;
run;

proc summary data=test.sample
             n sum max min maxdec=2;
  var totals;
  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;
  format n sum max min comma10.2;
run;
```

**Table 2.**

**Output of Merged Data Set**

| N | SUM | MAX | MIN |
|---|---|---|---|
| 1,000 | 88,097.43 | 165.09 | 0.00 |
| 1,501 | 195,473.52 | 272.29 | 0.00 |
| 2,001 | 361,654.45 | 398.09 | 0.00 |
| 2,500 | 603,430.58 | 588.66 | 0.00 |
| 3,000 | 981,466.91 | 958.05 | 0.00 |
| 3,500 | 1,643,027.50 | 1,772.82 | 0.00 |
| 4,000 | 2,883,759.03 | 3,452.34 | 0.00 |
| 4,500 | 5,367,852.60 | 6,895.97 | 0.00 |
| 5,000 | 17,625,172.01 | 299,427.91 | 0.00 |

Here we see several problems. First, we are forced to manually enter the value for each percentile into each of the SUMMARY procedures. 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. Secondly, 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. Thirdly, 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=test.sample
          out=caprate group=5;
  var totals;
  ranks rank;
run;

proc summary data=caprate;
  class rank;
  var totals;
  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 3.**

**Output of RANK Procedure**

| RANK | N | SUM | MAX | MIN |
|---|---|---|---|---|
| 0 | 1,000 | 88,097.43 | 165.09 | 0.00 |
| 1 | 1,000 | 273,158.93 | 398.02 | 165.11 |
| 2 | 1,000 | 620,210.55 | 958.05 | 398.09 |
| 3 | 1,000 | 1,902,292.12 | 3,452.34 | 958.35 |
| 4 | 1,000 | 14,741,412.98 | 299,427.91 | 3,454.12 |

We seem to now have a simpler program with the ranking approach. In this case, however, 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=test.sample noprint;
  var totals;
   output out = caprate
               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, CAPRATE, 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 caprate;
    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=test.sample
                 n sum max min maxdec=0;
      var totals;
      where totals <= &&var&i;
      output out = out&i
             (drop = _type_ _freq_)
             n=n sum=sum max=max min=min;
    run;
  %end;

  proc data sets;
    %do i = 1 %to 28;
      append base=final data=out&i force;
    %end;
  run;

%mend method3;
%method3;
```

In the 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. 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.**

**Output of the Program with Macros**

| OBS | N | SUM | MAX | MIN |
|---|---|---|---|---|
| 1 | 499 | 25,016.11 | 85.79 | 0.00 |
| 2 | 999 | 87,932.34 | 164.79 | 0.00 |
| 3 | 1,498 | 194,656.77 | 272.00 | 0.00 |
| 4 | 1,999 | 360,858.34 | 397.85 | 0.00 |
| 5 | 2,500 | 603,430.58 | 588.66 | 0.00 |
| 6 | 2,999 | 980,508.86 | 957.82 | 0.00 |
| 7 | 3,500 | 1,643,027.50 | 1,772.82 | 0.00 |
| 8 | 4,000 | 2,883,759.03 | 3,452.34 | 0.00 |
| 9 | 4,500 | 5,367,852.60 | 6,895.97 | 0.00 |
| 10 | 4,550 | 5,731,394.14 | 7,638.76 | 0.00 |
| 11 | 4,600 | 6,136,129.79 | 8,652.60 | 0.00 |
| 12 | 4,650 | 6,598,773.76 | 9,869.63 | 0.00 |
| 13 | 4,700 | 7,147,863.22 | 12,332.47 | 0.00 |
| 14 | 4,750 | 7,835,705.20 | 14,874.85 | 0.00 |
| 15 | 4,800 | 8,658,765.93 | 17,986.06 | 0.00 |
| 16 | 4,850 | 9,642,829.55 | 21,974.46 | 0.00 |
| 17 | 4,900 | 10,932,994.47 | 30,046.06 | 0.00 |
| 18 | 4,950 | 12,892,122.95 | 49,262.62 | 0.00 |
| 19 | 4,955 | 13,152,827.64 | 53,713.46 | 0.00 |
| 20 | 4,960 | 13,424,150.96 | 54,512.64 | 0.00 |
| 21 | 4,965 | 13,716,865.66 | 62,645.44 | 0.00 |
| 22 | 4,970 | 14,039,711.20 | 66,004.70 | 0.00 |
| 23 | 4,975 | 14,377,996.11 | 68,923.38 | 0.00 |
| 24 | 4,980 | 14,741,180.92 | 76,767.68 | 0.00 |
| 25 | 4,985 | 15,169,361.99 | 97,922.03 | 0.00 |
| 26 | 4,990 | 15,724,317.40 | 120,333.35 | 0.00 |
| 27 | 4,995 | 16,471,774.29 | 171,051.22 | 0.00 |
| 28 | 5,000 | 17,625,172.01 | 299,427.91 | 0.00 |

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.

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 tot = totals;
```

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=test.sample noprint;
    var &tot;
    output out = caprate
                 pctlpts = &range
                 pctlpre = pct_;
  run;

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

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

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

**Illustration 5.c**

```
*******************************************;
  *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;

  *******************************************;
  *Use first and last names in the array to
   count number of array elements;
  *******************************************;
  data _null_;
  set caprate;
    array pct1[*] &fstvar -- &lstvar;
      call symput ('varnum', put(dim(pct1),8.));
  run;

*******************************************;
  *Get percentile values;
*******************************************;
  data _null_;
    set caprate;
    array pct2[*] &fstvar -- &lstvar;
      %do i = 1 %to &varnum;
        call symput ("var&i", put(pct2[&i],8.));
      %end;
  run;
```

During the macro compilation, three temporary DATA steps (_null_) are generated. In the first DATA step, the "call symput" function captures the value of the first and last variables in data set VNAME from PROC SORT and retains the two string values in two macro variables, fstvar and lstvar. The second DATA step takes CAPRATE and replaces the macro calls with the values from fstvar and lstvar. It then counts the number of the array elements, which includes all variables in CAPRATE and contains that count value in another macro variable, varnum. The third DATA step then uses the values from all three macro variables from the previous two DATA steps to perform the following: setting the array boundaries, specifying the array dimension, and creating 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.d**

```
%do i = 1 %to &varnum;
  %let j = %eval(&i-1);
  proc summary data=test.sample
               n sum max min maxdec=2;
    var &tot;
    %if &i = 1 %then %do;
      where &tot <= &&var&i;
    %end;
    %else %do;
      where &&var&j <= &tot <= &&var&i;
    %end;
    output out = out&i (drop = _type_ _freq_)
                 n=n sum=sum max=max min=min;
  run;
%end;
```

In this section, 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. The "%do j=" statement is another counter that keeps track the value of the lower percentile. The "%eval" function converts "&i" from a string to a number for numeric calculation.

In order to keep the array within its logical range, two inner macro do-loops are introduced within PROC SUMMARY. The first do-loop sums the payment variable for persons in the lowest percentile, while the second do-loop sums the payment variable for persons 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.e**

```
  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 comma13.2;
  run;

%mend method3;
%method3;
```

With all of the data sets, OUT&I, 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**

| NAME | N | SUM | MAX | MIN |
|---|---|---|---|---|
| PCT_10 | 499 | 25,016.11 | 85.79 | 0.00 |
| PCT_20 | 500 | 62,916.23 | 164.79 | 86.19 |
| PCT_30 | 499 | 106,724.43 | 272.00 | 165.09 |
| PCT_40 | 502 | 166,473.57 | 397.85 | 272.00 |
| PCT_50 | 501 | 242,572.24 | 588.66 | 398.02 |
| PCT_60 | 499 | 377,078.28 | 957.82 | 589.75 |
| PCT_70 | 501 | 662,518.64 | 1,772.82 | 958.05 |
| PCT_80 | 500 | 1,240,731.53 | 3,452.34 | 1,773.01 |
| PCT_90 | 500 | 2,484,093.57 | 6,895.97 | 3,454.12 |
| PCT_91 | 50 | 363,541.54 | 7,638.76 | 6,896.35 |
| PCT_92 | 50 | 404,735.65 | 8,652.60 | 7,648.67 |
| PCT_93 | 50 | 462,643.97 | 9,869.63 | 8,747.96 |
| PCT_94 | 50 | 549,089.46 | 12,332.47 | 9,923.33 |
| PCT_95 | 50 | 687,841.98 | 14,874.85 | 12,360.69 |
| PCT_96 | 50 | 823,060.73 | 17,986.06 | 14,924.59 |
| PCT_97 | 50 | 984,063.62 | 21,974.46 | 18,010.49 |
| PCT_98 | 50 | 1,290,164.92 | 30,046.06 | 22,032.39 |
| PCT_99 | 50 | 1,959,128.48 | 49,262.62 | 30,652.32 |
| PCT_99_1 | 5 | 260,704.69 | 53,713.46 | 50,712.47 |
| PCT_99_2 | 5 | 271,323.32 | 54,512.64 | 53,727.28 |
| PCT_99_3 | 5 | 292,714.70 | 62,645.44 | 56,128.82 |
| PCT_99_4 | 5 | 322,845.54 | 66,004.70 | 62,992.14 |
| PCT_99_5 | 5 | 338,284.91 | 68,923.38 | 66,125.84 |
| PCT_99_6 | 5 | 363,184.81 | 76,767.68 | 70,021.89 |
| PCT_99_7 | 5 | 428,181.07 | 97,922.03 | 77,398.16 |
| PCT_99_8 | 5 | 554,955.41 | 120,333.35 | 101,293.90 |
| PCT_99_9 | 5 | 747,456.89 | 171,051.22 | 130,483.05 |
| PCT_100 | 5 | 1,153,397.72 | 299,427.91 | 183,651.54 |

## 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 other statistical analyses as well. We believe there are alternative approaches for achieving the same results with similar or greater flexibility and efficiency. Our demonstration exemplifies the thought process by which we reached our conclusion.

## REFERENCES

SAS Institute, Inc. (1990), SAS Guide to Macro Processing, Version 6, Second Edition. Cary, NC: SAS Institute, Inc.

SAS Institute, Inc. (1990), SAS Language Reference, Version 6, First Edition. Cary, NC: SAS Institute.

SAS Institute, Inc. (1990), SAS Procedures, Version 6, First Edition. Cary, NC: SAS Institute.

## ACKNOWLEDGEMENT

We wish to thank Peter Wobus, US Bureau of Census, for his valuable comments on the concepts expressed in this paper.



## AUTHORS

Diana Zhang Wobus obtained her doctorate in Education and Research Methodology in 1994. She is currently involved in developing risk-adjusted capitation rates for managed care organizations participating in the Maryland Medicaid reform initiative. She began SAS programming in the late 1980s and has been an active SAS user in education and health policy research since 1993. Contact:

University of Maryland Baltimore County
Center for Health Program Development and Management
1000 Hilltop Circle
Baltimore, Maryland 21250
Phone/FAX: 410-455-6847/6850
Email: zhangwob@umbc.edu

John Charles Gober specializes in the acquisition and manipulation of large data sets. He has been programming in SAS for over seventeen 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:

8105 Carlyle Place
Alexandria, Virginia 22308-1402
Phone/FAX: 703-768-1319/3614
Email: j.gober@worldnet.att.net