Bulletproofing Your SAS® Results

Vanessa Hayden, Policy Analysis, Inc., Brookline, MA

ABSTRACT

We've all been thrown by questions like these: "Why doesn't this report foot with the one you gave me before?" and "Can you just quickly duplicate this old analysis but make one change?" In a fast-paced environment, it can be difficult to meet changing business needs but also produce internally consistent results. This talk presents tips and tools using SAS software that can help reduce the effort it takes to update your code, while also bulletproofing the results. Simply creating macro-driven titles that automatically reflect dates and changes in data definitions can save you hours of grief reconciling numbers later. Special emphasis will be placed on the use of macros and date functions. SAS® code will be presented at a beginner level, but all SAS® users may benefit from the organizational ideas presented.

INTRODUCTION

One of the most important things you can do to save yourself time and effort is to document your SAS® output as clearly and comprehensively as possible. This will save you a great amount of time in explaining apparent discrepancies between different versions of output, rechecking your numbers, and so on. More importantly, the process you go through to achieve this also makes it easier for you to maintain and update your code. These strategies can be especially useful when you program for a number of projects and are often asked to run additional analyses on projects that had been set aside for some months. While these strategies do add to your initial programming time, the effort you put in can pay back many times over.

The techniques used in this paper rely heavily on user-defined SAS macro variables, introduced next.

A BRIEF INTRODUCTION TO MACRO VARIABLES

Within the SAS System, users can define macro variables to streamline and automate their programming. The values of macro variables are character strings that are resolved before program execution; thus they can be used to change and generate SAS® code. For example, suppose you plan to run multiple PRINT procedures on different subsets of data. First you can create a macro variable named "whereclause" that contains the SAS® code you would normally type into your WHERE statement. To create this macro variable, use a %LET statement.

```
%let whereclause = loanamt ge 25000;
```

Substitute this macro variable for code in your WHERE and TITLE statements.

```
proc print data=work.june;
  where &whereclause;
  title "June loans with
&whereclause";
run;
title;
```

Before program execution the macro processor translates your code into the following:

```
proc print data=work.june;
  where loanamt ge 25000;
  title "June loans with loanamt ge
25000";
run;
title;
```

Note that when you refer to a macro variable, you use the "&" symbol plus the macro variable name. If the reference occurs within quoted text, you must use double quotation marks for the macro reference to resolve.

While macro variables are usually assigned before program execution, there are times when you need to assign a value from a data set. To accomplish this, use the CALL SYMPUT routine within a DATA step. The CALL SYMPUT routine requires two arguments: the name you wish to assign to the macro variable, enclosed in quotes, and the text string to assign. When assigning a value from a numeric variable, use the PUT function to convert the number to a text string. (Using the PUT function prevents "numeric to character" conversion notes from showing up in your SAS® log.)

```
data _null_;
  dt=today();
  call symput('dt',put(dt,date9.));
```

```
run;
title “Today’s date is &dt”;
```

Note that a macro variable can hold only one value at a time. When you assign a value from a data set with multiple observations, use the CALL SYMPUT routine only for the observation that contains the needed value. For example, use the LAST. flag if you are using a RETAIN statement to count unique subjects.

```
data work.two;
  set work.one;
  by subject;
  retain sjN;
  if _n_ eq 1 then sjN = 0;
  if first.subject then sjN + 1;
  if last.subject then
    call symput(‘sjN’,put(sjN,f5.));
run;
title “Reports Based on &sjN
Subjects”;
```

Since macro variables are character data, they do not, by default, perform numeric computations. For example, in the three %LET statements below, the macro variable z resolves to the text string “2+3”, not to the value 5.

```
%let x = 2;
%let y = 3;
%let z = &x + &y;
```

You can get macros to perform numeric computations either by using special macro functions %SYSEVAL and %SYSFUNC, or by working through a DATA step. Because numeric computations are automatic in the DATA step, it’s often easier to take the latter approach.

```
data _null_;
  z = &x + &y;
  call symput(‘z’,put(z,f1.));
run;
```

To verify the values assigned to macro variables, use the %PUT statement and double check the resolved value in the SAS® log. This technique is analogous to using the PUT statement within the DATA step.

```
%put &z;
%put &whereclause;
```

In the %PUT statement, you can add text to be displayed in the SAS® log, but do not enclose the text within quotes.

```
%put NUMBER OF SUBJECTS: &sjN;
%put SELECTION CRITERIA:
&whereclause;
```

DOCUMENT THE DENOMINATOR

Many reports do not display the effective sample size upon which they are based. For example, you might wish to see univariate statistics on claims for hospital stays, for all patients with certain characteristics. Some patients will have multiple hospitalizations during the relevant time period, so the n listed in the report is not equal to the sample size of patients meeting the study criteria. If you are asked to run this report for different sets of patient characteristics, you should set up appropriate titles or footnotes to display, at a minimum, the sample size upon which each report was based.

There are several ways to access and store the sample size in a macro variable. If you have a data set with one record per observation, you can do this in open code using the OPEN function. The following code creates a macro variable “sjN” that contains the number of observations in the data set work.subjects.

```
%let dsid=%sysfunc(open(work.subjects));
%let sjN=%sysfunc(attrn(&dsid,nobs));
%let rc=%sysfunc(close(&dsid));
title "HOSPITALIZATIONS FOR &sjN
SUBJECTS";
```

If your data set contains multiple records per subject, you can count unique subjects using the retain statement in the DATA step (see example under “A Brief Introduction to Macro Variables”), or you can create a data set that is unique by subject. The SQL procedure offers a simple way to create a data set with one record per relevant observation.

```
proc sql;
  create table work.uniquesj as
    select distinct subject
    from work.hospitalizations;
quit;

%let dsid=%sysfunc(open(work.uniquesj));
%let sjN=%sysfunc(attrn(&dsid,nobs));
%let rc=%sysfunc(close(&dsid));
title "HOSPITALIZATIONS FOR &sjN
SUBJECTS";
```

CODE ALL CONSTANTS AS MACRO VARIABLES

Hard-coded constants are just dangerous. Nine times out of ten you will need to change the value of constants at some point, whether to meet changing business definitions, to update changing values (such as year starting and ending dates), or even to adapt your program for a similar project. Updating hard-coded constants in your programs can be a nightmare when the value occurs more than once in a program, and if you miss even one change, your output will be incorrect. In addition, when you are faced with a stack of old results, you will have the usual program of identifying after the fact which hard-coded value was used for each printout. A safe strategy is to assume that all constants will be changed at some point, and set them up in macro variables to accommodate potential future changes. For values that you anticipate will change often or that help define a set of output, use the same macro variables in titles, footnotes, labels, or user formats.

The following example shows how to apply macro-coded constants in a hypothetical incentive program where employees receive different rates of incentive pay based on their sales dollars. Employees who meet the baseline goal of $1 MM in sales receive a commission at a rate of 25% per thousand dollars in sales, and those whose sales exceed $3.5 MM receive a commission at a rate of 37.5% per thousand dollars in sales.

```
%let base = $1,000,000;
%let exceed = $3,500,000;
%let rate1 = .00025;
%let rate2 = .000375;

data _null_;
  value1 = compress("&base",',$');
  value2 = compress("&exceed",',$');
  call symput('baseval',value1);
  call symput('exceedval',value2);
run;

proc format;
  value salesf
    low - <&baseval =
              "BELOW GOAL (<&base)"

    &baseval-<&exceedval =
                 "MET GOAL (&base)"
    &exceedval-high =
        "SURPASSED GOAL (>&exceed)";
run;

data work.empsales2;
  set work.empsales;
  if sales ge &exceedval
    then incpay = sales * &rate2;
    else if sales ge &baseval
      then incpay = sales * &rate1;
    else incpay = 0;
  grp = put(sales,salesf.);
  format   sales   dollar14.   incpay
dollar6.;
run;

proc report nowd headskip spacing=0;
  column grp empid incpay;
  define grp / group 'Incentive
Category';
  define empid / n format=comma6.
'Count';
  define incpay / mean format=dollar9.
    "Average Incentive Pay";
run;
```

The code above produces the following report:

```
Average

Incentive
Incentive Category          Count      Pay

BELOW GOAL (<$1,000,000)       18       $0
MET GOAL ($1,000,000)         244     $571
SURPASSED GOAL (>$3,500,000)   25   $1,424
```

Note that while the user keys values for the macro variables “&base” and “&exceed,” the DATA _NULL_ step removes the dollar sign and commas to generate the parallel macro variables “&baseval” and “&exceedval.” The first set allows the program to display a standard dollar format within the labels and the second set allows for numeric compuations. (Numeric constants in the DATA step can not contain dollar signs and commas.) Having one set of macro variables generate the other ensures that the computed values always foot with labels and titles.

AUTOMATE DATES AND TIMES

For date-sensitive reports, you can automate date production in a way that ensures the dates in the titles are the same dates that were used to extract the report data. Suppose you run a report every Monday that shows sales for the previous week (Monday through Sunday). If you could start with the macro variables “startdt” and “enddt” containing the text strings `10JUL2000` and `16JUL2000`, respectively,

you could ensure consistency between report titles and data:

```
proc print data=work.empsales;
  where saledt between
    "&startdt"d and "&enddt"d;
  title "Sales For &startdt Through
&enddt";
run; title;
```

You can easily generate macro variables for the weekly start and end dates with functions available in the DATA step. If you always run the program on Mondays, you could compute the start date as the current date minus seven days, and the end date as the current date minus one day.

```
data _null_;
  call symput('startdt',
    put(today()-7,date9.));
  call symput('enddt',
    put(today()-1,date9.));
run;
title "Sales For &startdt Through
&enddt";
```

To enhance this program, you might set it up so that it works no matter what day of the week you run the program (especially important for holiday weeks). The WEEKDAY function returns the day of the week for a given date, in the format 1=Sunday, 2=Monday, and so forth. Starting with the previous date, you can loop backwards testing the weekday one day at a time until you hit the previous Monday.

```
data _null_;
  testdt = today();
  if weekday(testdt) eq 2 then
monday=1;
  do j = 1 to 7 while (monday ne 1);
    testdt = testdt - 1;
    if weekday(testdt) eq 2 then
    monday=1;
  end;
  call symput('startdt',put(testdt-
7,date9.));
  call symput('enddt',put(testdt-
1,date9.));
run;
title "Sales For &startdt Through
&enddt";
```

Another strategy involves the date function INTNX, which will advance or reverse a date by a given number of days, weeks, months, or years. When you add or subtract a week to a date, the date returned is always a Sunday, regardless of the starting date. The INTNX function identifies the current week's Sunday as week 0 and counts forwards or backwards from there. Be careful using this function, because SAS software counts weeks as starting on Sunday, the "current week's Sunday" will always be the previous Sunday. For example, if you add one week to the date of this presentation (Monday, September 25, 2000), the INTNX function will add six days and return the date for next Sunday (October 1, 2000). If you subtract one week, the INTNX function will subtract 8 days and return not yesterday's date but the Sunday before that (September 17, 2000). You can test this by running the code below.

```
data work.foo;
  dt = '25sep2000'd;
  add1wk = intnx('week',dt,+1);
  sub1wk = intnx('week',dt,-1);
  adddays = add1wk - dt;
  subdays = sub1wk - dt;
  format dt add1wk sub1wk mmddyy8.;
run;

proc print;
run;
```

You can apply the INTNX function to our sales week example. The code below subtracts one week using the INTNX function, then adds one day to the result to find the previous Monday.

```
data _null_;
  dt = today();
  startdt = intnx('week',dt,-1)+1;
  enddt = startdt + 6;
  call symput
    ('startdt',put(startdt,date9.));
  call
symput('enddt',put(enddt,date9.));
  format startdt enddt mmddyy8.;
run;
title "Sales For &startdt Through
&enddt";
```

UPDATE THE DATE AND TIME ON OUTPUT

By default the SAS System for PCs displays the date and time that you initially opened the SAS System, and numbers pages consecutively from that time. This system does uniquely identify pages of output; however, most managers prefer output to start on page one, not page 2057. You might prefer each set of results to start on page one and display the actual date and time they were run. To access the current date and time, use the DATA step functions TODAY and TIME, and put those values into macro variables using the CALL SYMPUT routine.

```
data _null_;
  dt=today();
  tm=time();
  call symput ('RealDate',put(dt,date9.));
  call symput ('RealTime',put(tm,time.));
run;

footnote "&RealDate at &RealTime";
options pageno=1;
```

CONCLUSION

As a SAS® programmer, you are often expected to track and explain myriad changes to reports and analyses that occur over the life of a project. Even if you could remember all the changes made to different projects on different dates, your time is better spent programming than explaining and justifying old results.

SAS macro variables offer a relatively simple way to document your output with relevant criteria used to produce your output, including dates, subsetting/where conditions, sample size, and specific constants. While this paper outlined techniques for printed output, you can apply many of these techniques to electronic SAS® output.

As a final note, I recommend attaching to any printed results a copy of the log file used to generate them. (Alternatively, if you are disseminating reports electronically, have your SAS® programs save the log window output to a dated text file.) You can't document all programming points in your titles and footnotes, and there is no substitute for having the actual log for review when you get the inevitable questions about the exact report specifications that you used.

ACKNOWLEDGMENTS

The author thanks Policy Analysis, Inc. for its support in attending this conference and Alexis Hayden for editorial review.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Vanessa Hayden
Policy Analysis, Inc.
Four Davis Court
Brookline, MA 02445
Work Phone: (617) 232-4400
Fax: (617) 232-1155
Email: vhayden@pai2.com