Paper 98-27

Bulletproofing Your SAS® Results

Vanessa Hayden, Fidelity Investments, Boston, 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 with 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. SAS code will be presented at a beginner level, but all SAS users may benefit from the organizational ideas.

INTRODUCTION

Self-documenting reports will save you the time and aggravation of explaining apparent discrepancies between different versions of output, rechecking numbers, etc. In addition, the code you write to create self-documenting reports is also easier to maintain and update. These programming strategies are especially helpful when you program for a number of projects and are frequently asked to run additional analyses on projects that had been set aside for some months. While the strategies recommended below do add to your initial programming time, the effort you invest can pay back many times over.

The techniques used in this paper use macro variables and simple stored macros. Throughout the paper, user-defined macro names (names of macro variables and macro routines) will be printed in bold-italic font. Appendix A provides a brief overview of macro programming. Appendix B incorporates all these techniques into a single example. To see additional examples and related topics, see the author's other SUGI 2002 paper in the Data Warehousing and Enterprise Solutions section: Paper 160-27, *Unit of Analysis Programming*, in the Data Warehousing section. All examples in this paper use fictional data. The opinions expressed herein are solely the author's and do not necessarily reflect the opinions of her employer.

FOLLOW BOOLEAN CONVENTIONS

It is a truism that you should use descriptive variable names wherever possible. In the case of boolean (or binary) variables, however, use of a simple convention makes output much easier to interpret. For example, try to interpret the regression output in Figure 1. What does the parameter estimate of 3.1 mean for the *gender* variable? Either men made an average of 3.1 more purchases than women at the pro shop YTD, or vice versa (i.e., the intercept for one group is higher than that for the other). You must refer to a codebook – or your overtaxed memory – to find out which is the case.

```
The REG Procedure
Model: MODEL1
Dependent Variable: purchases (YTD Purchases at Pro
Shop)

                Parameter Estimates

                Parameter  Standard
Variable  DF    Estimate     Error t Value  Pr > |t|

Intercept  1    12.84211   0.93008   13.81    <.0001
gender     1     3.08097   1.45922    2.11    0.0432
```

Figure 1. Sample regression output with uninformative variable name 'gender.'

If, on the other hand, you name the boolean variable *maleflg* and code it as 1=male, 0=female, then the results are readily interpretable, to you and to others. The key is to give the variable an intuitive name and follow the standard programming convention for boolean variables (Figure 2).

Boolean Notation:
1 = Yes
0 = No

Figure 2. Standard boolean notation. Used in conjunction with descriptive variable names, boolean notation helps make reports self-explanatory.

LINK TITLES TO REPORT CRITERIA

It's frustrating to pick up a report and wonder, "Was this from the full population, or only study completers (or only women, or only patients aged 55+, etc.)?" Much worse is using data from a report labeled "Study Completers" and realizing later that the title was incorrect (and you've given the client incorrect numbers). The best way to ensure accurate titles is to use macro variables that link your report titles to your report criteria.

For example, the following two code blocks will produce the same report. If you change the WHERE statement in the first block without changing the title statement, you get a mislabeled report. The second block is more foolproof. It's well worth the extra effort of typing out the second example. Whenever you type a WHERE statement, consider substituting a simple macro variable, even when (you think) you're just running a "one off."

```
/* WITHOUT MACRO CODE */
proc print data=WORK.june;
  where loanamt ge 25000;
  title "June loans > $25,000";
run; title;

/* WITH MACRO CODE */
%let wherclause = loanamt ge 25000;
proc print data=WORK.june;
  where &wherclause;
  title "June loans with &wherclause";
run; title;
```

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 -- 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 – and you may not even realize it. In addition, when you are faced with a stack of old results, you will have the usual problem 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. Use a single macro variable to ensure consistency between program code and corresponding titles, footnotes, labels, and user formats.

The following example shows how to apply macro-coded constants in a hypothetical incentive program where employees receive Incentive pay based on their sales dollars. Employees who meet the goal of $1 MM in sales receive a commission at 25% per thousand dollars in sales.

```
%let goal = $1,000,000;
%let incrate = .00025;

data _null_;
  numgoal = compress("&goal",',$');
  call symput('goalval',numgoal);
  incpct = put(&incrate,percent8.3);
  call symput('incpct',incpct);
run;

/* Display macro values in SAS log */
%put GOALVAL:  &goalval;
%put INCPCT:   &incpct;

proc format;
 value salesf
   low - <&goalval = "BELOW GOAL (<&goal)"
   &goalval - high = "MET GOAL (>=&goal)";
run;

data WORK.empsales2;
  set WORK.empsales;
  if sales ge &goalval
    then incpay = sales * &incrate;
    else incpay = 0;
  saleslvl = put(sales,salesf.);
  format sales dollar14. incpay dollar6.;
run;

proc report nowd headskip spacing=0;
  column saleslvl empid incpay;
  define saleslvl / group 'Sales Level';
  define empid / n format=comma6. 'Count';
  define incpay / mean format=dollar9.
    "Average Bonus Pay at &incpct";
run;
```

Sales Level	Count	Average Bonus Pay at 0.025%
BELOW GOAL (<$1,000,000)	12	$0
MET GOAL (>=$1,000,000)	219	$946

Figure 3. Sample report showing constants incorporated into formats and labels.

While the program requires two forms of the goal level (with dollar signs for the labels; without for the numeric computations), the user keys the value only once. The DATA _NULL_ step removes the dollar sign and commas from the macro variable *&goal* to generate the parallel macro variable *&goalval*. Having one set of macro variables generate the other ensures that the computed values always match labels and titles.

DOCUMENT THE DENOMINATOR

In addition to report titles, the sample size is a very useful way to match up reports with their original purpose. Whenever possible, include the sample N somewhere in the printed report. The code below demonstrates how macro variables can be used to store sample sizes and display them via user formats and titles (Figure 4). (The code assumes a data set called PERM.surveys that is already unique by survey respondent.)

```
proc freq data=PERM.surveys noprint;
  tables trmtgrp / out=WORK.smpl;
run;

data _null_;
  set WORK.smpl end=eof;
  retain tot;
  if _n_ eq 1 then tot = count;
    else tot = tot + count;
  if trmtgrp eq 0 then
    call symput('placN',put(count,comma6.));
    else if trmtgrp eq 1 then
      call symput('trmtN',put(count,comma6.));
  if eof then
    call symput('totN',put(tot,comma6.));
run;

proc format;
  value grpNf
    0 = "Placebo (n=&placN)"
    1 = "Miracle Drug (n=&trmtN)";
run;
title2 "Total Sample Size: &totN Subjects";
```

Total Sample Size: 228 Subjects

	How do you rate Miracle Drug overall?		
	POOR/ FAIR	GOOD	V GOOD /XNT
	Pct	Pct	Pct
MEDICATION			
Placebo (n=162)	45.6	26.6	27.8
Miracle Drug (n=66)	31.8	22.7	45.5

Figure 4. Survey results displaying sample sizes.

It's also very useful to document the denominator when running statistical procedures, many of which can exclude records with missing values (depending on the options used in the procedure). The statistical procedure will display the degrees of freedom, but may not display the number of observations excluded due to missing data. Note in Appendix B that the source data contain demographic and claims data for 107 patients, but 3 patients are eliminated from the regression due to missing fields[1].

AUTOMATE DATES AND TIMES

Date constants often seem to be the weak link in production applications. A miskey can produce, for example, a report labeled "November" that contains October data. Linking titles to report criteria helps you spot the error, but it's even better if you eliminate the handkeying entirely.

In production systems, it is usually possible to read the system date and compute all the required date variables off that value. Whether you are programming a production system or a one-off report, you should never require more than one hand-keyed date. From that date you should be able to compute any other date constants and display formats the program requires.

The following example shows how you can use the system date to generate reports for the previous month. The program requires no manual monthly changes, handles the year-end appropriately and places a descriptive title in the report.

```
data _null;
  begdt = intnx('MONTH',today(),-1);
  enddt = intnx('MONTH',today(),+1-1)-1;
  call symput('begdt',put(begdt,date9.));
  call symput('enddt',put(enddt,date9.));
```

[1] The purpose of reporting the database N (as opposed to the statistical N) is for keeping your reports organized. Obviously, you report the degrees of freedom for statistical purposes. Having both numbers on the report allows you to determine how many observations were excluded due to missing data.

```
  monthnm = put(begdt,monname12.);
  yr = put(year(begdt),4.);
  length titledt $ 16;
  titledt = compbl(monthnm || ' ' || yr);
  call symput('titledt',titledt);
run;
```

The report below (Figure 5) came from a TABULATE procedure that used macro variables in the selection critieria and title statement.

```
title "Miracle Drug Sales for &titledt";
where date between "&begdt"d and "&enddt"d;
```

At the time the report was run (during September of 2000), these statements resolved to

```
title "Miracle Drug Sales for August 2000";
where date between "01Aug2000"d and
      "31Aug2000"d;
```

Note that while the formats of macro-variable dates are different, the logic in the DATA step forces them to refer to the same month, thereby ensuring consistency.

MIRACLE DRUG SALES FOR **August 2000** 1

	SALES
REGION	
Midwest	$189,500
Northeast	$217,000
Southern	$741,500
All	$1,485,200

Figure 5. Self-generating sales report. The title and report criteria were both generated from the system date.

UPDATE THE DATE AND TIME ON OUTPUT

By default SAS 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. It is dangerous, however, to simply reset the page number to 1, because you lose valuable information about which report was run first. For example, if you find a data error after lunch, you would want to toss all reports you produced that morning.

To clearly document the output sequence, update the date/time and restart page numbering at one for each separate report or procedure. Use the DATA step functions TODAY and TIME to access the current date/time, then use the CALL SYMPUT routine to store the values as macro variables. The macro *%getdttm* turns off the default date title, resets the page number at one, and inserts the current date/time into a footnote (see the macro definition in Appendix B, programming block 1000).

TURN OFF TITLES AND FOOTNOTES

As mentioned above, incorrect headers and footers are infinitely worse than missing titles. SAS software leaves titles and footnotes on until they are explicitly turned off, or the session is ended. To avoid printing off reports with incorrect titles, make it a habit to turn off titles & footnotes immediately after they are used. For the cleanest code, put the most general information in the top titles (e.g., name of project) and switch on & off the lower titles (see, e.g., Appendix B).

CONCLUSION

As a SAS programmer, you are often expected to track and explain the myriad report changes 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 macros offer a relatively simple way to document your output with the criteria used to produce your output, including dates, subsetting conditions, sample size, and specific constants. In addition, self-descriptive boolean variables and diligent use of intelligent titles/footnotes make your output much more self-explanatory.

As a final note, attach to any printed results a copy of the log file used to generate them. (Alternatively, if you are distributing 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 database logic and report specifications used.

APPENDIX A: BRIEF INTRODUCTION TO MACRO PROGRAMMING

The SAS macro facility is a very powerful tool that can be used to streamline and automate programming. The word *macro* can refer to macro variables (a form of symbol substitution) and stored, compiled macros (executable routines). I will refer to the former as *macro variables* and the latter as *macro routines*. The section below will introduce you to the basic usage of macro variables and macro routines, but for complete syntax refer to SAS Institute documentation.

MACRO VARIABLES

Macro variables are programming variables that represent strings of text. The macro variable names and their associated strings are stored in a *symbol tables*, as in Figure 6. Macro variables in the *global* symbol table can be accessed throughout the program; those in *local* symbol tables are specific to macro routines.

Macro Variable	Associated Text Value
&goal	$1,000,000
&goalval	1000000
&wherclause	loanamt ge 25000
&sjN	107
&realdate	11JAN2002
&realtime	21:40:33
&wherstmt	where claimtyp='P'
&z	2 + 3

Figure 6. Example global symbol table.

Macro variable references are resolved *before code execution*, so they can be used for *code substitution*. Whenever you refer to a macro variable in your code, the macro compiler will substitute the text stored under that macro variable name. For example, the compiler will substitute the text "loanamt ge 25000" wherever it sees the macro variable reference *&wherclause*.

To create a macro variable use the %LET assignment statement, as in the following examples. To set the value to null, place a semi-colon directly after the equal sign.

```
%let goal = $1,000,000;
%let wherstmt = where claimtyp = 'P';
%let z = &x + &y;
%let goal=;
```

It is possible to create data-dependent macro variables, which are assigned during program execution. The value may come from a data set (e.g., the maximum number of diagnosis codes associated with a single claim ID) or from the source system (e.g., the current time). To assign a data-dependent macro

variable, use the CALL SYMPUT routine within a DATA step. 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.) The code below creates a *data set variable* dt, converts the numeric date value to text using the date9 format, and stores the resulting text string in the *macro variable* realdate.

```
data _null_;
  dt=today();
  call symput('realdate',put(dt,date9.));
run;
title "Today's date is &realdate";
```

To reference a macro variable, use the "&" symbol followed by the macro variable name. If the reference occurs within quoted text, you must use double quotation marks for the macro reference to resolve. In the example below, notice the difference in macro resolution between title1 and title2:

```
/* Raw SAS code */
proc print data=WORK.claimsum (obs=5);
  &wherestmt;
  title1 "&wherestmt";
  title2 '&wherestmt';
run; title;

/* SAS code with macro references resolved */
proc print data=WORK.claimsum (obs=5);
  where claimtyp='P';
  title1 "where claimtyp='P'";
  title2 '&wherestmt';
run; title;
```

In some situations, SAS needs extra information to parse a macro variable reference from the text surrounding it. In these cases, use a period to end the macro variable reference.

```
%let path = /adhoc/sugi
filename mydata "&path./sugidata.csv";
```

When resolved, the filename mydata will store the value "/adhoc/sugi/sugidata.csv" (without the period).

Since macro variables are character data, they do not 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;
```

If necessary, you can get macros to perform numeric computations either by using special macro functions %SYSEVAL and %SYSFUNC, or by working through a DATA step.

MACRO ROUTINES

Macro routines are executable blocks of code that are stored in a macro *catalog*. They are typically used to streamline repetitive sections of code and to ensure consistency among related sections.

To create a macro routine, embed a block of code between the %MACRO and %MEND statements. On the %MACRO statement, assign a name to the routine. This will store the code block in the macro catalog under that name. To invoke the macro routine, precede the macro name with the "%" symbol. In the example below, the *%expire* call will execute the SETINIT procedure.

```
%macro expire;
proc setinit;
run;
%mend expire;

%expire;
```

The power of macro programming reveals itself when you parameterize parts of the code so that the coding block can be customized for multiple situations. You pass parameters (macro variables that are local to the macro routine) by enclosing them within parentheses. The following macro routine can be used to print the first 10 records of any data set:

```
%macro print10(dsn);
proc print data=&dsn (obs=10) noobs uniform;
run;
%mend;

%print10(WORK.sales);
%print10(PERM.pat);
```

The two macro calls will print the top ten records of two different data sets.

MACRO TOOLS

For debugging – and for documenting the logic used in a job – you will want to write extra information to the SAS log. SAS provides you with several tools that write parts of the macro logic to the log file, although it still takes time to learn how to interpret the macro log file.

The %PUT statement displays the resolved value of macro variables in the SAS log. You can also use this statement to write specific text to the log file. This is analogous to the PUT statement in a DATA step; however, when using %PUT you do not enclose the text within quotes.

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

The following three system options print additional information to the log, and can be turned on (first option) or off (second option).

```
options symbolgen | nosymbolgen
options mprint | nomprint
options mlogic | nomlogic
```

The SYMBOLGEN option shows the resolved value of every macro variable, every time it changes. The MPRINT option displays the code within a macro every time it is invoked, with the local macro variables resolved. The MLOGIC option displays information about macro execution. The excerpt below shows lines from a log with each of these three options turned on[2].

LOG FILE WITH SYMBOLGEN OPTION

```
13      options symbolgen nomprint nomlogic;
14      %print10(PERM.pat);
SYMBOLGEN:  Macro variable DSN resolves to
PERM.pat

NOTE: There were 10 observations read from the
data set PERM.PAT.
```

LOG FILE WITH MPRINT OPTION

```
17      options nosymbolgen mprint nomlogic;
18      %print10(PERM.pat);
MPRINT(PRINT10):   proc print data=PERM.pat
(obs=10) noobs uniform;
MPRINT(PRINT10):   run;

NOTE: There were 10 observations read from the
data set PERM.PAT.
```

LOG FILE WITH MLOGIC OPTION

```
21      options nosymbolgen nomprint mlogic;
22      %print10(PERM.pat);
MLOGIC(PRINT10):  Beginning execution.
MLOGIC(PRINT10):  Parameter DSN has value
PERM.pat
```

[2] I find it most helpful to read the log with the MPRINT option turned on, the SYMBOLGEN option turned off and strategic %PUT statements throughout the code to document the values of certain macro variables.

```
NOTE: There were 10 observations read from the
data set PERM.PAT.
NOTE: The PROCEDURE PRINT printed page 3.
NOTE: PROCEDURE PRINT used:
      real time           0.00 seconds
      cpu time            0.01 seconds
MLOGIC(PRINT10):  Ending execution.
```

APPENDIX B: EXAMPLE

This example assumes a very simple claims database with a patient-level data set and a claim-level data set. The top five records of these data sets are shown in Figures 7 and 8.

dxdt	ptid	trtflg	dissev	age	maleflg
01/01/1999	001	1	5	61	0
02/22/1999	002	1	4	35	1
02/07/1999	003	1	3	60	1
03/23/1999	004	1	4	65	1
06/19/1999	005	1	5	67	0

Figure 7. Sample records from patdat (patient-level data set). See also the LABEL statement for PERM.pat in code block 2000 below.

claimdt	ptid	claimamt
01/16/1999	001	$169
06/07/1999	002	$769
06/21/1999	003	$8
05/08/1999	004	$682
11/16/1999	005	$5

Figure 8. Sample records from claimdat (claim-level data set). See also the LABEL statement for PERM.claims in code block 2000 below.

```
/*********************************************/
/* Program: SUGI2002-CC-AppendixB.sas        */
/* Author:  Vanessa Hayden                   */
/* Date:    01.11.2002                       */
/*                                           */
/* Inputs:  patdat.csv (patient-level data)  */
/*          claimdat.csv (claims-level data) */
/*                                           */
/* Outputs: SAS listing                      */
/*********************************************/

/*-------------------------------------------*/
/* MAINLINE                                  */
/* 1000-HOUSEKEEPING                         */
/* 2000-READ-EXTERNAL-DATA                   */
/* 3000-SUMMARIZE-CLAIMS-DATA                */
/* 4000-MERGE-WITH-PATIENT-DATA              */
/* 5000-RUN-REGRESSIONS                      */
/*-------------------------------------------*/

/*-------------------------------------------*/
/* 1000-HOUSEKEEPING                         */
/*-------------------------------------------*/
options nocenter ls=64 ps=40;

/* root unix dir stored as a macro variable */
/* used in filenames and libname statements */
%let root=/apm/adhoc5/vhayden/sugi/bullet;

filename PATDAT "&root./patdat.csv";
filename CLAIMDAT "&root./claimdat.csv";

libname PERM "&root./data";

/* assign value to clmdays near the top of   */
/* program so it's easier to find and update */
%let clmdays = 180;

title1 "SUGI 2001: Bulletproofing";
title2 "Appendix B Example";

/*+++++++++++++++++++++++++++++++++++++++++++*/
/* GETDTTM (stored macro)                    */
/* Purpose: Update date & time in footnote */
/*          and reset page number to 1       */
/* Inputs:  None                             */
/* Outputs: &realdate: current system date */
/*          &realtime: current system time */
/*+++++++++++++++++++++++++++++++++++++++++++*/
%macro getdttm;
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 nodate;
%mend getdttm;

/*+++++++++++++++++++++++++++++++++++++++++++*/
/* GETUNIQN (stored macro)                   */
/* Purpose: Counts unique values of a key  */
/*          field in a data set              */
/* Inputs:  &ds:  Name of data set           */
/*          &key: Key variable               */
/* Outputs: &keyvalN: Count of unique        */
/*           values of &key in &ds           */
/*+++++++++++++++++++++++++++++++++++++++++++*/
%macro getuniqN(ds,key);
proc sql;
  create table WORK.uniqval as
    select distinct &key
    from &ds;
quit;

%global keyvalN;

%let dsid=%sysfunc(open(WORK.uniqval));
%let keyvalN=%sysfunc(attrn(&dsid,nobs));
%let rc=%sysfunc(close(&dsid));
%mend getuniqN;

/*+++++++++++++++++++++++++++++++++++++++++++*/
/* PRINT5 (stored macro)                     */
/* Purpose: Prints top 5 obs of data set   */
/* Inputs:  &ds:       Name of data set      */
/*          &titlenbr: Title number to use */
/* Outputs: SAS listing                      */
/*+++++++++++++++++++++++++++++++++++++++++++*/
%macro print5(dsn,titlenbr);
proc print data=&dsn (obs=5) noobs uniform;
  title&titlenbr "Top 5 observations in &dsn";
run;
%mend print5;

/*-------------------------------------------*/
/* 2000-READ-EXTERNAL-DATA                   */
/*-------------------------------------------*/
data PERM.pat (label='Patient-Level Data');
  infile PATDAT dlm=',' dsd firstobs=2;
  informat dxdt mmddyy8.;
  input ptid dxdt trtflg dissev age maleflg;

  format ptid z3.
         dxdt mmddyy10.;
  label ptid    = 'Patient ID'
        dxdt    = 'Date of Initial Diagnosis'
        trtflg  = 'Treatment Indicator'
        dissev  = 'Disease Severity (1-5)'
        age     = 'Patient Age'
        maleflg = 'Male Indicator';
run;
%getdttm;
```

```
%print5(dsn=PERM.pat,titlenbr=3);

data PERM.claims (label="Health Care Claims");
  infile CLAIMDAT dlm=',' dsd firstobs=2;
  informat claimdt mmddyy8.;
  input ptid claimdt claimamt;

  format ptid z3.
         claimdt mmddyy10.
         claimamt dollar8.;
  label ptid    = 'Patient ID'
        claimdt = 'Claim Date'
        claimamt= 'Claim Amount';
run;
%print5(dsn=PERM.claims,titlenbr=3);

/*-------------------------------------------*/
/* 3000-SUMMARIZE-CLAIMS-DATA                */
/*-------------------------------------------*/
proc sql;
  create table WORK.totals as
    select a.ptid, sum(claimamt) as claimtot
    from PERM.pat as a, PERM.claims as b
    where a.ptid=b.ptid and
       0<=intck('day',dxdt,claimdt)<=&clmdays
    group by a.ptid
    order by a.ptid;
quit;

/*-------------------------------------------*/
/* 4000-MERGE-WITH-PATIENT-DATA              */
/*-------------------------------------------*/
data WORK.totals2;
  merge PERM.pat WORK.totals;
  by ptid;
  label claimtot="(Claim Total Within &clmdays
                  Days of Initial Diagnosis)";
run;

/*-------------------------------------------*/
/* 5000-RUN-REGRESSIONS                      */
/*-------------------------------------------*/
%getdttm;
%getuniqn(ds=WORK.totals2,key=ptid);
%put keyvalN: &keyvalN;  /* show value in log */

proc reg data=WORK.totals2;
  model claimtot = trtflg dissev age maleflg;
  title3 "Regression on Total Claim Dollars";
  title4 "N=&keyvalN Patients";
run; title3;      /* turns off titles 3 and 4 */

/*********************************************/
/*                   END                     */
/*********************************************/
```

ACKNOWLEDGMENTS

Many thanks to Alexis Hayden and Kathleen Kane for their review and comments, and to Fidelity Investments for sponsoring my attendance at SUGI.

CONTACT INFORMATION

Your comments and questions are valued and encouraged.

Vanessa Hayden
Director, Marketing Research
Fidelity Investments
82 Devonshire Street, R3D
Boston, MA 02109-3614
Phone: (617) 563-3178
Email: vanessa.hayden@fmr.com


```
                          Regression on Total Claim Dollars                    1
                                  N=107 Patients

                                The REG Procedure
                                  Model: MODEL1
        Dependent Variable: claimtot (Claim Total Within 180 Days of Initial Diagnosis)

                              Analysis of Variance

                                          Sum of           Mean
          Source                 DF       Squares         Square    F Value    Pr > F

          Model                   4     487627761      121906940       5.95    0.0002
          Error                  99    2030022268       20505275
          Corrected Total       103    2517650029

                    Root MSE          4528.27511    R-Square     0.1937
                    Dependent Mean         17951    Adj R-Sq     0.1611
                    Coeff Var            25.22568

                              Parameter Estimates

                                                Parameter       Standard
  Variable    Label                       DF     Estimate          Error   t Value   Pr > |t|

  Intercept   Intercept                    1   9427.66125    2777.46327      3.39     0.0010
  trtflg      Treatment Indicator          1   -1931.99000     891.97922     -2.17     0.0327
  dissev      Disease Severity (1-5)       1    870.97744     320.82930      2.71     0.0078
  age         Patient Age                  1    125.58939      41.09823      3.06     0.0029
  maleflg     Male Indicator               1   -1546.41054     902.78957     -1.71     0.0899

                              12JAN2002 at 20:20:17
```