Paper 030-2008

Adventures in ODS: Producing Customized Reports Using Output from Multiple SAS® Procedures

Stuart Long, Westat, Durham, NC
Jeff Abolafia, Rho Inc., Chapel Hill, NC
Lawrence Park, Chapel Hill, NC

Abstract

SAS ODS, combined with the Macro Facility, provides the seasoned programmer with the tools to customize the content of output that is available to generate summarized reports of procedural or multi-procedural listings. One of the shortcomings of statistical analysis in SAS is the amount of output that is generated from SAS procedures and dumped to the SAS Listing. This often voluminous amount of output combined with statistics extraneous to the needs of reviewers can be problematic.

SAS ODS allows the programmer to extract only the desired statistics from a procedure, storing them in a single summary data set. Numerous ODS Output data sets can be merged together and used to generate concise reports summarizing the results from the multiple SAS procedures. The automation of the creation of these summary data sets allows for iteration to propagate the analysis over hundreds or thousands of data points.

This paper will define analysis methods and detail the translation of these methods into modular macros. Each macro module will execute one main task and produce a product before processing is passed on to the next macro. A supervisory macro will be designed to allow these modules to customize the summarization of an analysis. With proper design, as demonstrated in this paper, these modules can be interchangeable within the supervisory macro, so as to allow the substitution of procedures to alter the analysis as needed. This is an advanced presentation intended for SAS programmers knowledgeable with BASE/SAS and the SAS Macro facility.

Introduction

SAS provides the seasoned programmer with two tools that can be combined to develop automated systems for summarizing statistical output from SAS procedures. The first is the SAS Output Delivery System (ODS), which allows the programmer to route selected SAS procedure output to one or more destinations. The second tool, the SAS Macro Facility, provides the means to automate the creation of these output data sets and allows for iteration of a procedure over numerous variables.

Before the introduction of ODS in version 7, many, but not all, procedures contained the option of using the OUT= statement to route statistics to SAS data sets. This continues to be available in the current SAS releases. This method provides a convenient way to save procedural statistics for additional processing. Although it remains a viable tool for the programmer in SAS Version 9.1.3, SAS ODS provides similar capabilities and has significant advantages over the OUT= statement. First, not all statistics output from SAS procedures are available using the OUT= statement, for example the likelihood estimate from PROC LOGISTIC. In addition, OUT= is not available in all procedures. PROC POWER, new to SAS Version 9, is one example. Second, ODS allows the user to keep particular output items and observations of interest and to rename the output variables, if desired. Third, ODS provides a uniform way across all SAS procedures to collect statistics into data sets, and is the way of the future, hence it is a must learn technique.

This paper will demonstrate 1) the use of a supervisory macro, modular macros and SAS ODS output data sets to generate a summary data set and report derived from several SAS procedures, 2) the ease of adapting this blueprint to alter the analysis as needed (from linear models to logistic models) by substituting macro modules calling different procedures, and 3) how to iterate an analysis over numerous variables of interest. Each macro module executes one main task, while the supervisory macro controls the execution of the modular macros in the appropriate order.

The End to the Means – A Summary Report

The aim of the examples in this paper is to produce a succinct summary report for a set of linear model covariates. Statistics for the covariates are derived from several SAS procedures, specifically descriptive statistics from Proc Means and parameter estimates derived from linear models using Proc GLM. The ODS OUTPUT statement will be used to store the output from these procedures in SAS data sets, which will subsequently be merged into a final dataset from which the report is generated. Figure 1 shows the summary to be generated.

```
 Figure 1: Example Report

Response Variable     = Body Mass Index
Class Covariates      = race gender
Continuous Covariates = age height

Independent             Mean  Standard    Model              Parameter Estimates
Variable                         Error        N         Crude   SE      Adjusted    SE
                         (a)       (a)      (a)          (b)   (b)         (c)     (c)
-----------------------------------------------------------------------------------------

Years lived on Farm    36.21      0.08     28166      0.030  0.001       0.021  0.002
Acres planted crops    47.03      0.04     28209      0.022  0.001       0.024  0.002
```

The statistics in this report come from three SAS Procedures executed independently of each other:

(a) PROC MEANS
(b) PROC GLM – main covariate only (crude)
(c) PROC GLM – multivariable (adjusted)

Overview

We will write a set of code to analyze numerous variables to be included in the above summary report. The SAS Macro facility allows us to automate data management and analytical tasks. An essential consideration in designing a manageable automated system is modularization. A well designed macro should execute one main task; overworking a macro by including multiple tasks can make it cumbersome to review, debug and update. A good macro module of this nature should run as a self-contained element, provided that the proper parameters are passed in the macro call. This also facilitates testing and debugging, offers ease of future maintenance, and insures portability.

Below is the list of the tasks needed to automate an analysis to produce the above summary report. Remember, our goal is to create a single composite data set containing the summary information derived from numerous procedures for all of the independent variables of interest. We start by identifying main tasks for generating and presenting the statistics in our example report.

1) Initialize the BASE data set.
2) Produce a summary data set of mean statistics.
3) Produce a dataset of crude model estimates with standard error.
4) Produce a dataset of adjusted model estimates and standard error.
5) Append the summary data sets to a BASE summary data set.
6) Create a supervising macro to delegate these tasks through invocations of other macro modules

In addition, there are secondary tasks that support the main tasks. These tasks warrant creation of separate macro modules for each one:

7) Validate the user supplied information.
8) Specify the covariates for the adjusted model.
9) Create an analysis dataset.
10) Generate the summary report..

Initialize the BASE Data Set

We first initialize a dataset that will accumulate the final set of statistics of interest. This is the BASE data set for the PROC APPEND in the **SUMAMRYSET()** macro, below. It is prudent for several reasons to initialize this data set prior to any other processing:

1) We can use PROC APPEND without any worry of conflict between the BASE= data set and the DATA= data set.
2) Variables will have pre-assigned attributes.
3) Variables will be arranged in the order that we assign to them.

The BASE data set will be initialized only if it does not already exist. It will contain 0 observations once returned by this macro, as well as when it is read the first time by the PROC APPEND in the **SUMMARYSET()** macro. Since we are assigning attributes to a new data set that in essence contains no variables or observations, the SAS LOG will alert us to the fact that these variables are uninitialized. To avoid this unnecessary alert in the SAS LOG, we will turn off the NOTES feature while executing this DATA step and then turn the NOTES feature back on after the DATA step finishes.

```
%MACRO initbase(base=);

  OPTION NONOTES;

  DATA &base;
    LENGTH key_var response_var $32
           key_label response_label $40
           class_vars continuous_vars $200
           mean stddev stderror n
           crude_estimate adjusted_estimate
           crude_stderr adjusted_stderr 8.;
    LABEL  key_var           = "Name of Independent Variable"
           key_label         = "Label of Independent Variable"
           response_var      = "Name of Response Variable"
           response_label    = "Label of Response Variable"
           mean              = "PROC MEANS: Mean of Independent Variable"
           n                 = "PROC MEANS: N of Independent Variable"
           stddev            = "PROC MEANS: Standard Deviation of Independent Variable"
           stderror          = "PROC MEANS: Standard Error Independent Variable"
           crude_estimate    = "Crude PROC GLM: Parameter Estimate"
           crude_stderr      = "Crude PROC GLM: Standard Error of Estimate"
           adjusted_estimate = "Adjusted PROC GLM: Parameter Estimate"
           adjusted_stderr   = "Adjusted PROC GLM: Standard Error of Estimate"
           class_vars        = “Adjusted PROC GLM: Class covariates”
           continuous_vars   = “Adjusted PROC GLM: Continuous covariates”;
    FORMAT crude_estimate adjusted_estimate PVALUE6.4
           crude_stderr adjusted_stderr ODDSR8.3 ;
  RUN;

  OPTION NOTES;

  DATA &base;
    SET &base;
    STOP;
  RUN;

%MEND initbase;
```

This module will be called by the **SUPERVISORY()** macro only if the BASE data set does not yet exist:

```
%IF %SYSFUNC(EXIST(&outfile)) NE 1 %THEN %initbase(base=&outfile);
```

Produce a Summary Data Set of Mean Statistics

Our task involves looking at the relationship between a response variable and an independent variable of interest. First we will design a macro module to produce a data set of mean statistics for the independent variable from PROC MEANS.

```
PROC MEANS DATA=analysis_set mean n stderr std;
  VAR years_on_farm;
RUN;
```

We modify this code to redirect the output to an ODS output data set, SUMMARY_MEANS:

```
ODS OUTPUT summary=summary_means;
PROC MEANS DATA=analysis_set mean n stderr std;
  VAR years_on_farm;
RUN;
ODS OUTPUT CLOSE;
```

which contains all of the statistics requested for the given variable.

Figure 2: Abbreviated Contents of initial *SUMMARY_MEANS* dataset

```
#    Variable                 Type    Len    Format     Label
-----------------------------------------------------------------
1    years_on_farm_Mean       Num       8    BEST12.    Mean
2    years_on_farm_N          Num       8    BEST5.     N
7    years_on_farm_StdDev     Num       8    BEST12.    Std Dev
6    years_on_farm_StdErr     Num       8    BEST12.    Std Error
```

Ultimately, we will append together these descriptive statistics for numerous variables of interest. To facilitate this processing, we need to rename the variables containing the summary statistics to have a common, generic name for each execution of the macro:

```
  ODS OUTPUT summary=summary_means (RENAME = (years_on_farm_mean   = mean
                                              years_on_farm_n      = n
                                              years_on_farm_stderr = stderror
                                              years_on_farm_stddev = stddev));
  PROC MEANS DATA= analysis_set mean n stderr std;
    VAR years_on_farm;
  RUN;
  ODS OUTPUT CLOSE;
```

As shown in Figure 2, the SAS ODS Object contains the statistics requested but does not identify the variable on which the information is generated, except as part of the name of the statistic. Since we are generating statistics for a variable using several SAS procedures and need to combine them later, we must create a common key variable on these data sets to allow sorting and merging. We see in Figure 2 that the name and label for the independent variable are not available without further processing of the SUMMARY_MEANS data set. However, if we provide more than one variable to the VAR statement, SAS adds variable names and labels to the ODS Object SUMMARY. Since variable labels are not retrievable from the PROC GLM ODS Objects, we will provide the response variable as our second variable to be processed by the PROC MEANS, thus generating the names and labels for both the response and independent variables in the ODS Object SUMMARY:

```
  ODS OUTPUT summary=summary_means (DROP = bmi_mean bmi_n bmi_stderr bmi_stddev
                                    RENAME = (vname_bmi            = response_var
                                              Label_bmi            = response_label
                                              vname_years_on_farm  = key_var
                                              label_years_on_farm  = key_label
                                              years_on_farm_mean   = mean
                                              years_on_farm_n      = n
                                              years_on_farm_stderr = stderror
                                              years_on_farm_stddev = stddev));
  PROC MEANS DATA= analysis_set mean n stderr std;
    VAR bmi years_on_farm;
  RUN;
  ODS OUTPUT CLOSE;
```

Now let’s write a macro module to automate the generation of this summary data set. We will add one step to the macro, which is to suppress the SAS LISTING as an output destination. This destination must then be reset after the execution of the PROC MEANS:

```
%MACRO procmeans(data   = ,
                 var1   = ,
                 var2   =);

    ODS LISTING CLOSE;
    ODS OUTPUT summary=summary_means (DROP = &var1._mean &var1._n &var1._stderr &var1._stddev
                                      RENAME = (VNAME_&var1    = response_var
                                                LABEL_&var1    = response_label
                                                VNAME_&var2    = key_var
                                                LABEL_&var2    = key_label
                                                &var2._mean    = mean
                                                &var2._n       = n
                                                &var2._stderr = stderror
                                                &var2._stddev = stddev) );
    PROC MEANS DATA= &data mean n stderr std;
      VAR &var1 &var2;
    RUN;
    ODS OUTPUT CLOSE;
    ODS LISTING;
%MEND procmeans;
```

The following invocation of the **PROCMEANS()** macro module produces the data set in Figure 3.

```
%procmeans(data   = analysis_set ,
           var1   = bmi ,
           var2   = years_on_farm)
```

Figure 3: Abbreviated Contents of finalized *SUMMARY_MEANS* dataset

```
 #    Variable    Type      Len    Format     Label
-----------------------------------------------------------------------------
 1    response_var   Char   (32)             (Name of Response Variable)
 2    response_label Char   (40)             (Label of Response Variable)
 3    key_var        Char   (32)             (Name of Independent Variable)
 4    key_label      Char   (40)             (Label of Independent Variable)
 5    mean           Num      8    BEST12.   (PROC MEANS: Mean)
 6    n              Num      8    BEST5.    (PROC MEANS: N)
 7    stddev         Num      8    BEST12.   (PROC MEANS: Standard Deviation)
 8    stderror       Num      8    BEST12.   (PROC MEANS: Standard Error)
```

Attributes appearing in parenthesis in Figure 3 were assigned to the BASE data set in the **INITBASE()** macro.

Produce a Summary Data Set of Crude and Adjusted Model Estimates with Standard Errors

Two of our main tasks are similar enough that we can design one macro module to process them both: the execution of the crude and adjusted PROC GLM. Our crude PROC GLM code is as follows:

```
PROC GLM DATA=analysis_set;
   MODEL bmi = years_on_farm ;
QUIT;
```

A macro to run this procedure is trivial.

```
%MACRO procglm (data            = ,
                response        = ,
                independent     =);

  PROC GLM DATA=&data;
    MODEL &response = &independent ;
  QUIT;

%MEND procglm;
```

Just as the hard code of the PROC GLM can be modified to accommodate an adjusted model:

```
PROC GLM DATA=analysis_set;
  CLASS race gender;
  MODEL bmi = race gender height age years_on_farm;
QUIT;
```

so can then **PROCGLM()** macro:

```
%MACRO procglm (data            = ,
                response        = ,
                independent     = ,
                class_vars      = ,
                continuous_vars = );

  PROC GLM DATA=&data NAMELEN=32;
    %IF (NOT %LENGTH(%SCAN(&class_vars,1)))
    %THEN CLASS &class_vars ; ;
    MODEL &response = &continuous_vars &class_vars &independent;
  QUIT;

%MEND procglm;
```

Now we have a model that conditionally adds a CLASS statement if categorical covariates are passed into the macro, as well as adding all class and continuous covariates to the MODEL statement. However, the task is larger than just running the model. We must redirect the desired statistics into a SAS data set and RENAME the Parameter to be the Key_Var so that it will be compatible with the data set produced by the PROC MEANS macro:

```
%MACRO procglm (test             = ,
                data             = ,
                response         = ,
                independent      = ,
                class_vars       = ,
                continuous_vars = );

  ODS LISTING CLOSE;
  ODS OUTPUT parameterestimates = summary_&test._glm
                               (DROP = dependent probt tvalue
                                RENAME = (Parameter = Key_var
                                          Estimate  = &test._Estimate
                                          stderr    = &test._stderr)
                                WHERE  = (UPCASE(key_var)=UPCASE("&independent"))) ;
  PROC GLM DATA=&data NAMELEN=32;
    %IF (NOT %LENGTH(%SCAN(&class_vars,1))) %THEN
    CLASS &class_vars ; ;
    MODEL &response = &continuous_vars &class_vars &independent;
  QUIT;
  ODS OUTPUT CLOSE;
  ODS LISTING;

%MEND procglm;
```

In the above macro, we pass one additional parameter called “test”, which identifies the model as crude or adjusted. This parameter is used as the statistics are renamed in the ODS OUTPUT statement to identify them as crude or adjusted. Finally, we use the WHERE= data set option in the ODS OUTPUT statement to keep only the observations for the independent variable of interest.

We will invoke the **PROCGLM()** macro module twice, once for the crude model and once for the adjusted model:

```
%procglm (test            = Crude,
          data            = analysis_set,
          response        = bmi,
          independent     = years_on_farm)

%procglm (test            = Adjusted,
          data            = analysis_set,
          response        = bmi,
          independent     = years_on_farm,
          class_vars      = race gender,
          continuous_vars = age height)
```

The above two invocations of the PROCGLM macro module will produce the data sets seen in Figures 4 and 5:

Figure 4: Abbreviated Contents of *SUMMARY_CRUDE_GLM* data set

```
 #  Variable        Type  Len  Format      Label
-------------------------------------------------------------------------------------
 1   key_var        Char  (32)             (Name of Independent Variable)
15   crude_estimate  Num    8  PVALUE6.4   (Crude PROC GLM: Parameter Estimate)
16   crude_stderr    Num    8  ODDSR8.3    (Crude PROC GLM: Standard Error of Estimate)
```

Figure 5: Abbreviated Contents of *SUMMARY_ADJUSTED_GLM* data set

```
 #  Variable           Type  Len  Format      Label
------------------------------------------------------------------------------------------
 1  key_var            Char  (32)             (Name of Independent Variable)
15  adjusted_estimate  Num     8  PVALUE6.4   (Adjusted PROC GLM: Parameter Estimate)
16  adjusted_stderr    Num     8  ODDSR8.3    (Adjusted PROC GLM: Standard Error of Estimate)
```

Append the Summary Data Sets to a BASE Summary Data Set

The next macro, **SUMMARYSET()**, merges the three datasets of summary statistics by the key variable, key_var, which is passed into the macro module with the “merge_var” parameter.

```
  %MACRO summaryset(merge_var       = ,
                    base            = ,
                    setlist         = ,
                    delsets         = ,
                    class_vars      = ,
                    continuous_vars = );

    %LOCAL i dsets;
```

First, we identify all the summary data sets that were created by the macro modules. These are in the WORK library and were named with a common prefix SUMMARY_ , which we pass into the macro as the parameter “setlist”.

```
    PROC SQL NOPRINT ;
      SELECT TRIM(LIBNAME) || "." || MEMNAME
        INTO :dsets SEPARATED BY " "
        FROM DICTIONARY.TABLES
        WHERE LIBNAME = "WORK" AND
              MEMNAME LIKE UPCASE("&setlist%");
    QUIT ;
```

Next, we sort each of these data sets by the merge variable passed as a macro parameter. In our example, this variable is key_var. This step may appear extraneous since each data set in this example contains only one observation, however, it has been included to facilitate proper merging of data sets with a BY statement using sorted data.

```
    %LET i= 1;

    %DO %UNTIL (NOT %LENGTH(%SCAN(&dsets,&i, %STR( ))));
        PROC SORT DATA=%SCAN(&dsets,&i, %STR( ));
          BY &merge_var;
        RUN;
        %LET i = %EVAL(&i+1);
    %END;
```

Now, we merge all of the data sets into one summary data set by the merge variable.

```
    DATA summary_;
      MERGE &dsets;
        BY &merge_var;
      class_vars=”&class_vars”;
      continuous_vars=”&continuous_vars”;
    RUN;
```

Finally, we append the summary data set with all other summary observations created by previous executions of the macro.

```
    PROC APPEND BASE=&base DATA=&setlist FORCE;  RUN;
```

We will include one more minor task in this macro, a cleanup procedure using PROC DATASETS to remove all data sets created within the macro modules which will no longer be required after this macro has completed execution:

```
    PROC DATASETS;
      DELETE &setlist: &delsets / MEMTYPE = DATA;
    QUIT;

  %MEND summaryset;
```

The following invocation of the macro **SUMMARYSET()** will produce the data set seen in Figure 6.

```
  %summaryset(merge_var       = key_var,
              base            = GLM_summary,
              setlist         = SUMMARY_,
              delsets         = analysis_set,
              class_vars      = race gender
              continuous_vars = age height)
```

Figure 6: Abbreviated Contents of *GLM_SUMMARY* data set

```
 #  Variable          Type   Len   Format      Label
------------------------------------------------------------------------------------------
 1  key_var           Char   (32)             (Name of Independent Variable)
 2  key_label         Char   (40)             (Label of Independent Variable)
 3  response_var      Char   (32)             (Name of Response Variable)
 4  response_label    Char   (40)             (Label of Response Variable)
 5  mean              Num      8              (PROC MEANS: Mean of Independent Variable)
 6  n                 Num      8              (PROC MEANS: N of Independent Variable)
 7  stddev            Num      8              (PROC MEANS: Standard Deviation of Independent Variable)
 8  stderror          Num      8              (PROC MEANS: Standard Error Independent Variable)
 9 crude_estimate     Num      8   PVALUE6.4  (Crude PROC GLM: Parameter Estimate)
10 crude_stderr       Num      8   ODDSR8.3   (Crude PROC GLM: Standard Error of Estimate)
11 adjusted_estimate  Num      8   PVALUE6.4  (Adjusted PROC GLM: Parameter Estimate)
12 adjusted_stderr    Num      8   ODDSR8.3   (Adjusted PROC GLM: Standard Error of Estimate)
```

Validate the User Supplied Information

This will be the largest module since exhaustive checks must be performed on the data supplied by the user to the supervisory macro. When a complex macro such as this fails during compilation or execution, it is quite often due to omitted or miss-specified parameters. Our validation module will check the user supplied information as follows:

1) Is the input dataset specified.
2) Does the input dataset exist.
3) Is the response variable specified.
4) Does the response variable exist within the user supplied dataset.
5) Is the response variable numeric.
6) Is the independent variable of interest specified.
7) Does the independent variable exist within the user supplied dataset.
8) Is the independent variable numeric.
9) Do each of the base co-variables exist within the user supplied dataset.
10) Are each of the base model co-variables numeric.
11) Is the final output dataset specified.

If any of these 11 checks fail, an error message will be written to the log to identify the item that failed in the call of the supervisory macro and the macro processing will be halted. The appendix contains the code for the **VALIDATE()** macro module.

Specify the Base Covariates for the Adjusted Model

Our example task includes running an adjusted PROC GLM with a pre-determined set of base covariates. For the purposes of the first example in this paper, we require that the main independent variable be continuous. The base covariates included in the adjusted model can be either categorical or continuous data. In the GLM models, the categorical variables must appear in both the CLASS and MODEL statements, while the continuous variables only appear in the MODEL statement. To distinguish between the two types of variables, the user must attach an identifying character as a prefix to the name of the continuous covariates in the call to the supervisory macro. Here, we use the "@" symbol as the control character. In the invocation to the supervisory macro, we will supply the list of covariate in the parameter "covariates":

```
covariates = @height race @age gender
```

The covariates height and age are continuous variables while the rest are categorical. We use a macro, **SPLIT(),** to parse this string of covariates based on whether each variable has the "@" prefix. This macro module is unique from all the other modules presented in this paper in that it generates no SAS code. All the code is macro code which is never actually compiled. For this reason, we will elect to create what is termed as a user defined macro function. The main advantage of a user defined macro function is that we will avoid generating global macro variables. A minor advantage is that this structure will make our program more readable. Below demonstrates the usage of this user defined macro in our example:

```
%LET class_vars      =%split(vars=@height race @age gender,control=);
%LET continuous_vars =%split(vars=@height race @age gender,control=@);
```

The above assignments using this user define macro function produce the following equivalent statements:

```
%LET class_vars      = race gender ;
%LET continuous_vars = height age;
```

The code for this macro module can be found in the Appendix.

Create an Analysis Data Set

It is not uncommon for studies to have incomplete data and missing variables on some observations. This means that procedures run on the variables in a dataset may have different numbers of observations. Meaningful comparisons between crude and adjusted parameter estimates derived from statistical models can only be made if the models are run on the same set of observations; otherwise, it is impossible to determine if differences in the estimates are due to adjustment for the covariates or the differences in the observations included in the models. For this reason, the macro **ANALYSISSET()** will create an analysis data set containing only records with complete data for the variables included in the adjusted logistic regression model. The four parameters passed to this macro are:

1) the data set to be analyzed
2) the response variable
3) the independent variable of interest
4) the additional categorical and continuous covariates

```
%MACRO analysisset(data      = ,
                   var1      = ,
                   var2      = ,
                   variables = ) ;

   DATA analysis_set ;
     SET &data(KEEP = &var1 &var2 &variables);
     IF NMISS(OF _numeric_) = 0 THEN OUTPUT;
   RUN;

%MEND analysisset;
```

The following invocation of the **ANALYSISSET()** macro will produce the data set shown in Figure 7. The code above keeps only those variables required for the analysis and removes any observations that contain missing data.

```
%analysisset(data      = mydata,
             var1      = bmi,
             var2      = years_on_farm,
             variables = race gender height age) ;
```

Figure 7: Abbreviated Contents of *ANALYSIS_SET* data set

```
 #  Variable            Type   Len   Format      Label
------------------------------------------------------------------------------
1   years_on_farm       Num     3                Years lived on farm
2   bmi                 Num     8                Body Mass Index
3   gender              Num     8    _SEX.       Person's Gender
4   height              Num     8                Person's Height in inches
5   race                Num     8    _RACE.      Person's race
6   age                 Num     8                Person’s Age
```

Create a Supervisory Macro to Delegate These Tasks Through Invocations of Other Macro Modules

We have now written six macro modules, each working independently to complete the main and secondary tasks for the analysis. Since these modules must be executed in proper order, we create a supervisory macro that will pass the correct parameters to each module in order. In this example, the **SUPERVISOR()** macro acts as the supervisory macro. This macro runs the analysis for one set of variables. It is called repeatedly to run analyses for different response, independent, or base co-variables. This macro takes the following six parameters:

1) The dataset to be analyzed
2) The response variable
3) The main independent variable of interest
4) The base covariates to appear in the CLASS and MODEL statements
5) The name of the BASE data set which will contain the accumulated statistics
6) Whether the generation of a summary report has been requested

```
%MACRO supervisor(dataset     = ,
                  response    = ,
                  independent = ,
                  covariates  = ,
                  outfile     = ,
                  report      = );

%LOCAL echeck;
```

To insure proper processing, force the names of the response and independent variables to caps.

```
  %LET response    = %UPCASE(&response);
  %LET independent = %UPCASE(&independent);
```

The first macro invoked will be the **VALIDATE()** module, with five parameters:

1) The data set to be analyzed
2) The response variable
3) The independent variable
4) The list of covariates
5) The BASE data set

```
  %validate(dset   = &dataset,
            var1   = &response,
            var2   = &independent ,
            basevs = &covariates,
            out    = &outfile  )
```

If an error was found in any of the parameters supplied by the user, then the macro variable ECHECK will be assigned the value of “1”. If this is the case, then the macro will be instructed to cease processing with the %RETURN statement.

```
  %IF &echeck = 1 %THEN %RETURN;
```

If the BASE data set does not already exist, then the second macro invoked will be the **INITBASE()** module:

```
  %IF %SYSFUNC(EXIST(&outfile)) NE 1 %THEN %initbase(base=&outfile);
```

Next, the user defined macro function, **SPLIT(),** is used to assign covariates as either class or continuous:

```
  %LET class_vars      =%split(vars=&covariates,control=);
  %LET continuous_vars =%split(vars=&covariates,control=@);
```

The **ANALYSISSET()** macro module creates the analysis data set before any of the procedural modules are invoked:

```
  %analysisset(data      = &dataset ,
               var1      = &response ,
               var2      = &independent ,
               variables = &class_vars &continuous_vars )
```

The **PROCMEANS()** macro module creates our first summary data set, ***SUMMARY_MEANS***. This macro takes three parameters:

```
  %procmeans(data   = analysis_set,
             var1   = &response ,
             var2   = &independent )
```

The next two macro invocations both call the **PROCGLM()** module. The first invocation of the **PROCGLM()** macro module runs the crude model and creates our second summary data set**, *SUMMARY_CRUDE_GLM*.** The *class_vars* and *continuous_vars* parameters do not appear in the first invocation of the **PROCGLM()** macro module since they are null.

```
    %procglm (test             = Crude,
              data             = analysis_set,
              response         = &response,
              independent      = &independent )
```

The second invocation of the **PROCGLM()** macro runs the adjusted model and creates the data set, ***SUMMARY_ADJUSTED_GLM***. In this invocation of the **PROCGLM()** macro, the “class_vars” and “continuous_vars” parameters are used to pass the names of the covariates for this model.

```
    %procglm (test             = Adjusted,
              data             = analysis_set,
              response         = &response,
              independent      = &independent ,
              class_vars       = &class_vars,
              continuous_vars = &continuous_vars )
```

The last macro module, **SUMMARYSET(),** assembles the final data set of summary statistics and removes temporary data sets created during the process.

```
    %summaryset(merge_var       = key_var,
                outset          = GLM_summary,
                setlist         = SUMMARY_,
                delsets         = analysis_set,
                class_vars      = &class_vars ,
                continuous_vars = &continuous_vars)
```

One final task is to print a summary report if one was requested in the invocation of the **SUPERVISOR()** macro. Example code for the **PRINTREPORT()** macro module can be found in the appendix:

```
    %IF %UPCASE(&report) = YES %THEN %PrintReport(dset=&outfile);

  %MEND supervisor;
```

Two calls to **SUPERVISOR()** produce the data set used to create the results shown in Figure 8 .

```
%supervisor(dataset     = mydata,
            response    = bmi,
            independent = years_on_farm,
            covariates  = @age race gender @height,
            outfile     = glm_summary,
            report      = NO)

%supervisor(dataset     = mydata,
            response    = bmi,
            independent = acres,
            covariates  = @age race gender @height,
            outfile     = glm_summary,
            report      = YES)
```

```
 Figure 8: Example Report

Response Variable     = Body Mass Index
Class Covariates      = Race Gender
Continuous Covariates = Age Height

Independent             Mean  Standard    Model           Parameter Estimates
Variable                         Error        N       Crude     SE    Adjusted     SE
-------------------------------------------------------------------------------------
Years lived on Farm    36.21      0.08    28166       0.030  0.001       0.021  0.002
Acres planted crops   147.03      0.14    28209       0.022  0.001       0.024  0.002
```

Flexibility of the Design of the Supervisory Macro

The supervisory macro gathers output for statistics produced for a continuous variable in PROC MEANS, a crude PROC GLM and an adjusted PROC GLM using stand alone macro modules to execute each of the tasks to complete the overall analysis.

The design allows us to selectively replace modules to change the analysis, for example, to do an analysis of binary response data and accommodate categorical independent variables.

In this section, we will alter the methods, moving from linear to logistic regression models. To do this we will make five main changes.

1) The BASE data set must be adjusted to handle the new analysis.
2) The PROCMEANS module will be replaced by a PROCFREQ module.
3) The PROCGLM will be replaced by a PROCLOGISTIC module.
4) We will add a parameter passed to the supervisory macro to identify the referent level of the independent variable.
5) Our key variable will change from the name of the independent variable (Key_Var) to the value of the independent variable (Key_Value) .

In addition, we will add checks to the validation macro to verify that the referent level exists and that the response variable contains exactly two values as is required for a logistical regression.

Let’s first build the macro module for the PROC FREQ:

```
PROC FREQ DATA=mydata;
  TABLES farmsize * asthma;
RUN;
```

As in our previous examples, we will output our data to a SAS data set while temporarily suppressing the SAS listing destination. We will add a WHERE statement to the data set options in the ODS OUTPUT statement which will keep only the observations of interest.

```
ODs LISTING CLOSE;
ODS OUTPUT CrossTabFreqs = summary_freq
                           (KEEP = farmsize asthma _TYPE_ FREQUENCY COLPERCENT
                            WHERE = ( _TYPE_=”11”));

PROC FREQ DATA=mydata;
  TABLES farmsize * asthma;
RUN;

ODS OUTPUT CLOSE;
ODS LISTING;
```

We now create a macro module which will generate and update a SAS data set containing the desired statistics and descriptive information. Unlike the task in the **PROCMEANS()** macro, the output data set from the PROC FREQ will require additional processing in a DATA step within the **PROCFREQ()** macro to attain the desired final ***SUMMARY_FREQ*** data set.

```
%MACRO procfreq(data     = ,
                col_var  = ,
                row_var  = );

  ODS LISTING CLOSE;
  ODS OUTPUT CrossTabFreqs = Summary_Freq
                           (KEEP = &col_var &row_var _TYPE_ FREQUENCY COLPERCENT
                            WHERE = ( _TYPE_=”11”) );

  PROC FREQ DATA=&data;
    TABLES &row_var * &col_var; RUN;

  ODS OUTPUT CLOSE;

  ODS LISTING;

  DATA summary_freq ( KEEP = response_var response_label response_value1 response_fmt1
                             response_value2 response_fmt2 Key_var Key_Label Key_value
                             Key_format n_col1 p_col1 n_col2 p_col2 ) ;
    SET summary_freq;
      BY &row_var &col_var;
    RETAIN response_value1 response_fmt1 n_col1 p_col1;
    IF FIRST.&row_var THEN DO;
       n_col1          = frequency;
       p_col1          = colpercent;
       response_value1 = &col_var;
       response_fmt1   = VVALUE(&col_var);
    END; ELSE
    IF LAST.&row_var THEN DO;
       n_col2          = frequency;
       p_col2          = colpercent;
       response_value2 = &col_var;
       response_fmt2   = VVALUE(&col_var);
       response_var    = "&col_var";
       response_label  = VLABEL(&col_var);
       Key_var         = "&row_var ";
       Key_value       = &row_var;
       Key_format      = VVALUE(&row_var);
       Key_label       = VLABEL(&row_var);
       OUTPUT;
    END;
  RUN;
%MEND ProcFreq;
```

The following invocation of the **PROCFREQ()** macro will produce the data set in Figure 9:

```
  %procfreq(data     = analysis_set,
            col_var  = asthma,
            row_var  = farmsize)
```

Figure 9: Contents of *SUMMARY_FREQ* data set

```
   #  Variable            Type   Len  Format     Label
----------------------------------------------------------------------------------------------
 1   response_label     Char    (40)           (Label of Response Variable)
 2   response_value1    Char    (40)           (Value for Level 1 of Response variable)
 3   response_fmt1      CHAR    (40)           (Format value for Level 1 of Response variable)
 4   response_value2    Char    (40)           (Value for Level 2 of Response variable)
 5   response_fmt2      CHAR    (40)           (Format value for Level 2 of Response variable)
 6   response_name      Char    (32)           (Name of Response Variable)
 7   Key_label          Char    (40)           (Label of Independent Variable)
10   Key_var            Char    (32)           (Name of Independent Variable)
11   Key_value          Num       8            (Numeric Value of Independent Variable)
12   Key_format         Char    (40)           (Encoded text value of Independent Variable)
13   n_col1             Num       8            (PROC FREQ Frequency of Column 1)
14   p_col1             Num       8            (PROC FREQ Percent of Column 1)
13   n_col2             Num       8            (PROC FREQ Frequency of Column 2)
14   p_col2             Num       8            (PROC FREQ Percent of Column 2)
```

Next, we create a **PROCLOGISTIC()** macro. As with the **PROCGLM()** macro, this module will be designed to run either a crude or adjusted logistic model. However, our desired statistics for the crude and adjusted models must be output to two different ODS Objects, which requires us to output, combine and perform additional processing on two data sets within the **PROCLGISTIC()** macro to create the final ***SUMMARY_CRUDE_LOGISTIC*** and ***SUMMARY_ADJUSTED_LOGISTIC*** data sets.

```
  %MACRO ProcLogistic(test            = ,
                      data            = ,
                      response        = ,
                      independent     = ,
                      ref             = ,
                      class_vars      = ,
                      continuous_vars = );

    ODS LISTING CLOSE;

    ODS OUTPUT ParameterEstimates = ParameterEstimates
                                    (KEEP = variable classval0 probchisq estimate StdErr
                                     RENAME = (ProbChiSq = &test._p_value)
                                     WHERE = (UPCASE(variable)="&independent"))
               OddsRatios         = OddsRatios
                                    (KEEP = EFFECT OddsRatioEst LowerCL UpperCL
                                     RENAME = ( OddsRatioEst = &test._OddsRatio
                                                LowerCL      = &test._LowerCL
                                                UpperCL      = &test._UpperCL )
                                     WHERE = ( UPCASE(SCAN(EFFECT,1))= "&independent"));
   PROC LOGISTIC DATA = &data DESCENDING ORDER=INTERNAL NAMELEN=32;
      CLASS &class_vars
            &independent (PARAM=REF REF="&ref");
      MODEL &response = &class_vars &continuous_vars &independent ;
         FORMAT &independent;
    RUN;

    ODS OUTPUT CLOSE;
    ODS LISTING;

    DATA ParameterEstimates ( KEEP = key_value &test._p_value );
      SET ParameterEstimates ;
     Key_Value = INPUT(classval0,8.);
    RUN;

    DATA OddsRatios (KEEP = key_value &test._OddsRatio &test._LowerCL &test._UpperCL);
      SET OddsRatios ;
      Key_Value = input((SCAN(EFFECT,2)),2.);
    RUN;

    PROC SORT DATA=ParameterEstimates;
      BY key_value; RUN;

    PROC SORT DATA=OddsRatios;
      BY key_value; RUN;

    DATA summary_&test._logistic ;
      MERGE ParameterEstimates OddsRatios ;
        BY key_value;
    RUN;
```

The ***ParameterEstimates*** and ***OddsRatios*** data sets will not be used outside of the **PROCLOGISTIC()** module. Since **PROCLOGISTIC()** is the only module that knows these data sets exist, they should be deleted before exiting this macro:

```
     PROC DATASETS;
       DELETE ParameterEstimates OddsRatios;
     QUIT;
     RUN;

   %MEND proclogistic;
```

The following invocation of the PROCLOGISTIC() macro will produce the data set in Figure 10:

```
   %ProcLogistic(test            = Crude,
                 data            = &dataset,
                 response        = &response,
                 independent     = &independent,
                 ref             = &referent);
```

Figure 10: Contents of finalized *SUMMARY_CRUDE_LOGISTIC* dataset

```
 #   Variable           Type   Len   Format      Label
-------------------------------------------------------------------------------------
 1   Key_var            Char  (32)               (Name of Independent Variable)
 2   Key_value          Num     8                (Value of Independent Variable)
 3   Crude_p_value      Num     8     PVALUE6.4  (PROC LOGISTIC Crude p-value)
 4   Crude_OddsRatio    Num     8     ODDSR8.3   (PROC LOGISTIC Crude Odds Ratio Estimate)
 5   Crude_LowerCL      Num     8     ODDSR8.3   (PROC LOGISTIC Crude Lower 95% Limit)
 6   Crude_UpperCL      Num     8     ODDSR8.3   (PROC LOGISTIC Crude Upper 95% Limit)
```

Although we have introduced a completely different analysis, the changes to the supervisory macro and its modules are trivial. We use two new modules, **PROCFREQ()** and **PROCLOGISTIC()**, and make some additional, minor changes to the supervisor macro. The changes to the **SUPERVISORY()** macro are highlighted in gray in the code below. Additional changes to the **VALIDATE()** and **INITBASE()** modules are also minor and can be supplied by the authors upon request.

```
%MACRO supervisor(dataset     = ,
                  response    = ,
                  independent = ,
                  referent    = ,
                  covariates  = ,
                  outfile     = ,
                  report      = );

  %LOCAL echeck;

  %LET response    = %UPCASE(&response);
  %LET independent = %UPCASE(&independent);

  %validate(dset   = &dataset,
            var1   = &response,
            var2   = &independent ,
            ref    = &referent ,
            basevs = &covariates,
            out    = &outfile  )

  %IF &echeck = 1 %THEN %RETURN;

  %IF %SYSFUNC(EXIST(&outfile)) NE 1 %THEN %initbase(base=&outfile);

  %LET class_vars      =%split(vars=&covariates,control=);
  %LET continuous_vars =%split(vars=&covariates,control=@);

  %analysisset(data      = &dataset ,
               var1      = &response ,
               var2      = &independent ,
               variables = &class_vars &continuous_vars )

  %procfreq(data     = analysis_set,
            col_var  = &response,
            row_var  = &independent)

  %ProcLogistic(test            = Crude,
                data            = &dataset,
                response        = &response,
                independent     = &independent,
                ref             = &referent);

  %ProcLogistic(test            = Adjusted,
                data            = &dataset,
                response        = &response,
                independent     = &independent,
                ref             = &referent,
                class_vars      = &class_vars,
                continuous_vars = &continuous_vars )

  %summaryset(merge_var       = key_value,
              base            = &outfile,
              setlist         = SUMMARY_,
              delsets         = analysis_set,
              class_vars      = &class_vars,
              continuous_vars = &continuous_vars )

  %IF %UPCASE(&report) = YES %THEN %PrintLogistic(dset=&outfile);

%MEND supervisor;
```

The following invocation of the summary macro will produce the data set found in Figure 11:

```
%supervisor(dataset     = mydata ,
            response    = asthma ,
            independent = farmsize ,
            referent    = 1 ,
            covariates  = @age state race ,
            outfile     = logistic_summary ,
            report      = YES  )
```

Figure 11: Example Summary Report for Logistical Analysis

```
Model adjusted for age state race.

                       Ever Diagnosed with asthma
Variable                     Yes         No            P-Value           Adjusted
Label/Format                N    %       N    %      Crude Adjusted     OR       95% CI
-----------------------------------------------------------------------------------------
Size of Farm
  No Farm                  346 46.9    6658 39.8     .       .
  < 500 acres              278 37.7    6358 38.0     0.0361  0.7049    0.949  0.722 1.246
  500+ acres               113 15.3    3697 22.1     <.0001  0.5378    1.143  0.747 1.747
```

The Process of Iteration

The macro set demonstrated in this paper returns summary results for one set of models. We carefully designed this system to allow for iteration of the process. Each time the supervisory macro is invoked, the summary results are appended to the accumulated summary results produced by previous invocations of the macro. This allows the programmer to design an iteration macro to generate composite data sets with summaries from multiple sets of models. Each item that is passed as a parameter to the supervisory macro can be dynamic in the iteration macro, however, for ease of demonstration purposes, we will develop an iteration macro where all parameters are static except the independent variable. The CALL EXECUTE routine offers a method to iterate the invocation of macros from within a DATA Step. Below is an example of how to iterate the macro demonstrated in this paper using CALL EXECUTE:

```
%MACRO callexecute(arrayvars = );

  DATA _NULL_;
    LENGTH independent $ 32 print_report $3;
    IF 0 THEN SET mydata;
    ARRAY i_var (*) &array_vars;
    DO i = 1 TO DIM ( i_var );
       independent = VNAME ( i_var[i] );
       /* ask for a report to be printed when the last item of the array is read */
       IF i = DIM(i_var) THEN print_report="Yes "; ELSE
                              print_report="No ";
      CALL EXECUTE (CATS('%NRSTR('
                   ,'%supervisor(dataset     = mydata'
                   ,",response    = asthma"
                   ,",independent = ", independent
                   ,",referent    = 1"
                   ,",covariates  = @age_group bmi_group"
                   ,",outfile     = logistic_summary "
                   ,",report      = ", print_report
                   ,")"
                   ,")"
                   )
              );
    END;
  RUN;

%MEND callexecute;

%runsum(arrayvars = farmsize –- grain_years)
```

This macro generates the report seen in Figure 12. More information on iterating the invocation of macros using CALL EXECUTE can be found in paper 013 of the SUGI 30 proceedings[1]. Macro Arrays can also be used to iterate these coding systems; see paper 105 of the 2008 SAS Global Forum proceedings[2].

Figure 12: Example of summary report for Logistic Regression macro after iteration

```
Model adjusted for age state race.

                       Ever Diagnosed with asthma
Variable                      Yes         No           P-Value         OR      95% CI
Label/Format               N    %       N    %      Crude Adjusted       Adjusted
-----------------------------------------------------------------------------------------
Size of Farm
  No Farm                 346 46.9    6658 39.8     .       .
  < 500 acres             278 37.7    6358 38.0     0.0361  0.7049     0.949  0.722 1.246
  500+ acres              113 15.3    3697 22.1     <.0001  0.5378     1.143  0.747 1.747

Frequency of Milking Cows
  Never                   215  6.8    1275  7.8     .       .          .      .     .
  Monthly                 225  7.1    1104  6.8     0.0679  0.0445     1.238  1.005 1.525
  Weekly                 1396 43.9    7195 44.0     0.0768  0.0007     1.325  1.125 1.560
  Daily                  1341 42.2    6773 41.4     0.0434  <.0001     1.426  1.205 1.689

Frequency of Tractor Use
  Never                  2250 71.1   12203 74.9     .       .          .      .     .
  Monthly                 744 23.5    3465 21.3     0.0010  0.0042     1.146  1.044 1.259
  Weekly                  138  4.4     528  3.2     0.0004  0.0040     1.341  1.098 1.637
  Daily                    33  1.0      95  0.6     0.0015  0.0186     1.642  1.086 2.481

Years Grown Grain
  None                     51 17.9    1262 21.7     .       .          .      .      .
  < 5 years                47 16.5     917 15.7     0.2507  0.2305     1.287  0.852 1.943
  5-9 years                90 31.6    1539 26.5     0.0393  0.0477     1.435  1.004 2.052
  10-19 years              50 17.6    1005 17.3     0.3071  0.2669     1.258  0.839 1.887
  20+ years                46 16.1    1083 18.6     0.8107  0.6627     1.096  0.725 1.657
```

Discussion

We have shown how a blueprint for automating an analysis can be designed so that it can be modified with reasonable ease to accommodate a completely different analysis. The authors have altered this system to handle numerous different analyses. In addition to the example shown here, we have adapted this system to run a survival analysis by substituting PROC TPHREG for PROC LOGISTIC. Further, the PROC LOGISTIC as well as the PROC TPHREG versions of this program handle modeling of both continuous and categorical variables . The TPHREG version of this program is available upon request.

For clarity in this presentation, we selected a small set of output statistics from the regression models. In a real analysis, any additional statistics needed can be added easily, for example obtaining the Likelihood Estimates from PROC LOGISTIC to compare models is a fairly simple alteration to the data sets created in the **PROCLOGISTIC()** macro module. Programmers should refer to the ODS documentation provided for each SAS procedure to determine what statistics are available and how to address these elements.

We have chosen to manage the iteration here using CALL EXECUTE, but there are other means to run this iteration. A notable alternative to CALL EXECUTE is the use of Macro Arrays. For a detailed presentation of the use of Macro Arrays for iteration of these types of analyses, see paper 105 of the 2008 SAS Global Forum proceedings[2].

Caution must be used when embedding these types of macros within other SAS programs. In our case, conflicts will occur if the macro is placed within code that has opened additional ODS OUTPUT destinations, or has closed the ODS LISTING destination. If NOTES has been turned off in the OPTIONS statement, it will be left active upon exiting this system of macros. We have pointed out that all variables should be labeled and formatted before processing in these analyses. Certain system settings can cause missing labels or formats to generate fatal errors, while others might only produce warnings. The validation macro can alert the user to any situation that might cause the macro system to fail. In our examples, the supervisory macro will cease operations if such a situation is found, while generating a warning to the SAS LOG.

Conclusion

SAS ODS provides an efficient method for creating summary datasets from voluminous output generated by SAS procedures. Combined with the SAS Macro Facility, ODS provides the programmer with the tools to automate this process. The methods presented are designed for advanced application development by programmers who have a reasonably good understanding of the SAS Macro facility. However, once written, these applications can be used by an end user who does not have knowledge of the techniques required to develop these macros.

Further development rests on the knowledge that forward compatibility will rely on the ODS Output dataset since SAS has committed this facility to be an integral part of upgrades in the SAS software.

DISCLAIMER: The contents of this paper are the work of the authors and do not necessarily represent the opinions, recommendations, or practices of Westat and Rho Inc. The examples presented in this paper are derived from analyses of the Agricultural Health Study by the National Cancer Institute, and the National Institutes of Environmental Health Sciences. All statistics included in this paper have been altered for presentation purposes and do not represent actual statistical associations from the study.

Acknowledgements

The authors would like to thank Ian Whitlock, Ed Heaton, David Shore and Ronald Fehd for their assistance or review of the methods and coding in this paper.

References

1. Long, S., Park, L. (2005). Contrasting programming techniques for summarizing voluminous SAS® output using the SAS Output Delivery System (ODS) (PROC FREQ as an example). *Proceedings of the 30th Annual SAS Users Group International Conference.*

2. Long, S., Heaton, E.(2008). Using the SAS® DATA Step and PROC SQL to Create Macro Arrays. *Proceedings of the 2008 SAS Global Forum Conference.*

Contact Information

Stuart Long (long3@niehs.nih.gov)
Westat
1009 Slater Road, Suite 110
Durham, NC 27703

Appendix:

```
%MACRO validate(dset = ,
                var1 = ,
                var2 = ,
                ref  = ,
                basevs = ,
                out  = );
  %LOCAL z vnum indflag;
  %LET echeck=0 ;
  %LET indflag=0 ;

  %* check that DATA exists *;
  %IF &dset = %STR( ) %THEN %DO;
    %let ex=NO ;
    %put Summary-> THE DSET PARAMETER WAS NOT SPECIFIED SUMMARY WILL STOP EXECUTING;
    %let echeck=1 ;
  %end ;
  %else %do ;
     %if %sysfunc(exist(&dset)) %then %let ex=YES;
     %else %let ex=NO ;
  %end ;
  %if &ex=NO %then %do ;
    %put Summary-> THE INPUT DATASET [ &dset ] DOES NOT EXIST SUMMARY WILL STOP EXECUTING;
    %let echeck=1 ;
  %end;

  %* check input parameters *;
  %let dsnid = %sysfunc(open(&dset)) ;

  %* check that response variable exists *;
  %if &var1 = %str( ) %then %do;
     %put Summary-> THE VAR1 PARAMETER WAS NOT SPECIFIED SUMMARY WILL STOP EXECUTING;
    %let echeck=1 ;
  %end ;
  %else %if %sysfunc(varnum(&dsnid,&var1))=0 %then %do;
     %let echeck=1 ;
     %put Summary-> THE VARIABLE &var1 NOT FOUND IN DATASET &dset Execution terminating.;
  %end;

  %* check that response variable is numeric *;
  %else %do ;
    %let vnum =  %sysfunc(varnum(&dsnid,&var1)) ;
    %if %sysfunc(vartype(&dsnid,&vnum)) ne N %then %do;
      %let echeck=1 ;
      %put  Summary-> THE VARIABLE &var1 is not Numeric Execution terminating. ;
    %end;
  %END;
  %* check that independent variable exists *;
  %if &var2 = %str( ) %then %do;
     %put Summary-> THE VAR2 PARAMETER WAS NOT SPECIFIED SUMMARY WILL STOP EXECUTING;
    %let echeck=1 ;
  %end ;
  %else %if %sysfunc(varnum(&dsnid,&var2))=0 %then %do;
     %let echeck=1 ;
     %let indflag=1 ;
     %put  Summary-> THE VARIABLE &var2 NOT FOUND IN DATASET &dset  Execution terminating. ;
  %end;

  %* check that independent variable is numeric *;
  %else %do ;
    %let vnum =  %sysfunc(varnum(&dsnid,&var2)) ;
    %if %sysfunc(vartype(&dsnid,&vnum)) ne N %then %do;
     %let echeck=1 ;
     %let indflag=1 ;
     %put  Summary-> THE VARIABLE &var2 is not Numeric Execution terminating. ;
    %end;
  %end;

  %* check that covariates exist *;
  %if  &basevs ne %str( ) %then %do ;
     %let covars = %sysfunc(compress(&basevs,@)) ;
     %let z =1 ;
     %do %until (%scan(&covars,&z)= ) ;
       %let tvar = %scan(&covars,&z,' ') ;
       %if %sysfunc(varnum(&dsnid,&tvar))=0 %then %do;
          %let echeck=1 ;
          %put  Summary-> THE VARIABLE &tvar NOT FOUND IN DATASET &dset  Execution
terminating. ;
       %end;

       %* check that variable is numeric *;
       %else %do ;
          %let vnum =  %sysfunc(varnum(&dsnid,&tvar)) ;
          %if %sysfunc(vartype(&dsnid,&vnum)) ne N %then %do;
            %let echeck=1 ;
            %put  Summary-> THE VARIABLE &tvar is not Numeric Execution terminating. ;
          %end;
       %end;
     %let z = %eval(&z + 1);
     %end;
  %end;



   %* check if OUT macro variable specified *;
   %if &out. = %str( ) %then %do;
      %let echeck=1 ;
      %put Summary-> THE OUT PARAMETER was not specified. Execution terminating. ;
   %end;
%MEND validate;


  %MACRO initbase(base=);

    OPTION NONOTES;

    DATA &base;
      LENGTH key_var response_var $32
             key_label response_label $40
             class_vars continuous_vars $200
             mean stddev stderror n
             crude_estimate adjusted_estimate
             crude_stderr adjusted_stderr 8.;
      LABEL  key_var           = "Name of Independent Variable"
             key_label         = "Label of Independent Variable"
             response_var      = "Name of Response Variable"
             response_label    = "Label of Response Variable"
             mean              = "PROC MEANS: Mean of Independent Variable"
             n                 = "PROC MEANS: N of Independent Variable"
             stddev            = "PROC MEANS: Standard Deviation of Independent Variable"
             stderror          = "PROC MEANS: Standard Error Independent Variable"
             crude_estimate    = "Crude PROC GLM: Parameter Estimate"
             crude_stderr      = "Crude PROC GLM: Standard Error of Estimate"
             adjusted_estimate = "Adjusted PROC GLM: Parameter Estimate"
             adjusted_stderr   = "Adjusted PROC GLM: Standard Error of Estimate"
             class_vars        = “Adjusted PROC GLM: Class covariates”
             continuous_vars   = “Adjusted PROC GLM: Continuous covariates”;
      FORMAT crude_estimate adjusted_estimate PVALUE6.4
             crude_stderr adjusted_stderr ODDSR8.3 ;
    RUN;

    OPTION NOTES;

    DATA &base;
      SET &base;
      STOP;
    RUN;

  %MEND initbase;
```

```
%MACRO split(vars=, control=);
    %LOCAL  count varlist chars;
    %IF &vars NE %THEN %DO;
       %IF &control NE %THEN %DO;
          %LET count=1 ;
          %DO %UNTIL (%QSCAN(&vars,&count)= ) ;
             %LET var = %QSCAN(&vars,&count);
             %IF %SUBSTR(&var,1,1)=&control %THEN
                 %SYSFUNC(COMPRESS(%QSCAN(&vars,&count),&control))  ;
             %LET count = %EVAL(&count+1);
          %END ;
       %END;
       %ELSE %DO;
          %LET count=1 ;
          %DO %UNTIL (%QSCAN(&vars,&count)= ) ;
             %LET var = %QSCAN(&vars,&count);
             %IF %INDEX(ABCDEFGHIJKLMNOPQRSTUVWXYZ_,%UPCASE(%SUBSTR(&var,1,1))) NE 0 %THEN
             %QSCAN(&vars,&count) ;
             %LET count = %EVAL(&count+1);
          %END ;
       %END;
    %END;
  %MEND split;

  %MACRO analysisset(data      = ,
                     var1      = ,
                     var2      = ,
                     variables = ) ;

     DATA analysis_set ;
       SET &data(KEEP = &var1 &var2 &variables);
       IF NMISS(OF _numeric_) = 0 THEN OUTPUT;
     RUN;

  %MEND analysisset;

  %MACRO procmeans(data   = ,
                   var1   = ,
                   var2   =);

      ODS LISTING CLOSE;
      ODS OUTPUT summary=summary_means (DROP = &var1._mean &var1._n &var1._stderr
                                               &var1._stddev
                                        RENAME = (VNAME_&var1   = response_var
                                                  LABEL_&var1   = response_label
                                                  VNAME_&var2   = key_var
                                                  LABEL_&var2   = key_label
                                                  &var2._mean   = mean
                                                  &var2._n      = n
                                                  &var2._stderr = stderror
                                                  &var2._stddev = stddev) );
      PROC MEANS DATA= &data mean n stderr std;
        VAR &var1 &var2;
      RUN;
      ODS OUTPUT CLOSE;
      ODS LISTING;

  %MEND procmeans;

  %MACRO procglm (test            = ,
                  data            = ,
                  response        = ,
                  independent     = ,
                  class_vars      = ,
                  continuous_vars = );

    ODS LISTING CLOSE;
    ODS OUTPUT parameterestimates = summary_&test._glm
                                   (DROP = dependent probt tvalue
                                    RENAME = (Parameter = Key_var
                                              Estimate  = &test._Estimate
```

```
                                            stderr    = &test._stderr)
                                  WHERE  = (UPCASE(key_var)=UPCASE("&independent"))) ;
  PROC GLM DATA=&data NAMELEN=32;
    %IF (NOT %LENGTH(%SCAN(&class_vars,1))) %THEN
    CLASS &class_vars ; ;
    MODEL &response = &continuous_vars &class_vars &independent;
  QUIT;
  ODS OUTPUT CLOSE;
  ODS LISTING;

%MEND procglm;

%MACRO summaryset(merge_var       = ,
                  base            = ,
                  setlist         = ,
                  delsets         = ,
                  class_vars      = ,
                  continuous_vars = );

  %LOCAL i datasets;

  PROC SQL NOPRINT ;
    SELECT TRIM(LIBNAME) || "." || MEMNAME
      INTO :datasets SEPARATED BY " "
      FROM DICTIONARY.TABLES
      WHERE LIBNAME = "WORK" AND
            MEMNAME LIKE UPCASE("&setlist%");
  QUIT ;

  %LET i= 1;

  %DO %UNTIL (NOT %LENGTH(%SCAN(&datasets,&i,%STR( ))));
      PROC SORT DATA=%SCAN(&datasets,&i,%STR( ));
        BY &merge_var;
      RUN;
      %LET i = %EVAL(&i+1);
  %END;


  DATA summary_;
    MERGE &datasets;
      BY &merge_var;
    class_vars="&class_vars";
    continuous_vars="&continuous_vars";
  RUN;


  PROC APPEND BASE=&base DATA=&setlist FORCE;  RUN;


  PROC DATASETS;
    DELETE &setlist: &delsets / MEMTYPE = DATA;
  QUIT;

%MEND summaryset;

%MACRO PrintReport(data=);

  /* Insert code here for writing a report using any SAS reporting tool that is preferred
     by the programmer. */

%MEND PrintReport;


%MACRO supervisor(dataset     = ,
                  response    = ,
                  independent = ,
                  covariates  = ,
                  outfile     = ,
                  report      = );
```

```
   %LOCAL echeck;


   %LET response    = %UPCASE(&response);
   %LET independent = %UPCASE(&independent);


   %validate(dset   = &dataset,
             var1   = &response,
             var2   = &independent ,
             basevs = &covariates,
             out    = &outfile  )


   %IF &echeck = 1 %THEN %RETURN;


   %IF %SYSFUNC(EXIST(&outfile)) NE 1 %THEN %initbase(base=&outfile);


   %LET class_vars      =%split(vars=&covariates,control=);
   %LET continuous_vars =%split(vars=&covariates,control=@);


   %analysisset(data      = &dataset ,
                var1      = &response ,
                var2      = &independent ,
                variables = &class_vars &continuous_vars )

    %procmeans(data   = analysis_set,
                 var1   = &response ,
               var2   = &independent )


   %procglm (test            = Crude,
             data            = analysis_set,
             response        = &response,
             independent     = &independent )


   %procglm (test            = Adjusted,
             data            = analysis_set,
             response        = &response,
             independent     = &independent ,
             class_vars      = &class_vars,
             continuous_vars = &continuous_vars )


   %summaryset(merge_var       = key_var,
               base            = GLM_summary,
               setlist         = SUMMARY_,
               delsets         = analysis_set,
               class_vars      = &class_vars ,
               continuous_vars = &continuous_vars)

   %IF %UPCASE(&report) = YES %THEN %PrintReport(data=&outfile);

 %MEND supervisor;
```