

**Stepwise Anything:
Reading in Regression Procedure Output for Further Testing**

Ted Clay, Clay Software and Statistics, Ashland, Oregon
Linda Remy, University of California, San Francisco, California

ABSTRACT

A class of standardized macros is presented for reading in SAS procedure output. These tools are used by higher level macros to accomplish the following tasks: (1) Archiving numerous regression models, (2) Bootstrapping, (3) Testing for interaction between groups of dummy variables, (4) Running any regression procedure stepwise by examining p-values, and (5) Running procs LOGISTIC and four others stepwise using groups of variables entered together, which is a feature only of PROC REG.

INTRODUCTION

In this paper we present a hierarchy of techniques, mostly implemented in the form of macros. They can be visualized as a pyramid, with four levels.

- Level 1, at the bottom, is "Reading in Proc Output". This section is about a set of macros, all beginning with "RD", whose function is to read in the output of SAS procedures, storing statistical results in the form of datasets.
- Level 2 is "Archiving Results". This section discusses techniques for storing the results of many statistical models.
- Level 3, is "Comparing and Summarizing Results", which is applied in two quite different areas. First we illustrate bootstrapping, where multiple samples are analyzed, and the results archived and summarized. The second use is in comparing one model to another. The log-likelihood ratio chi-square test is introduced as a test for comparing models. Hence the "Compare" level in the pyramid below.
- Level 4, "Stepwise Modeling", shows how all of the other three levels can work together to automatically search for a reduced set of independent variables. This technique can be applied to variables in groups (for some procedures) or singly (for all regression procedures).

The following pyramid illustrates the relationship between the levels just explained. The reader should not infer that we believe stepwise regression is the pinnacle of all statistical techniques.

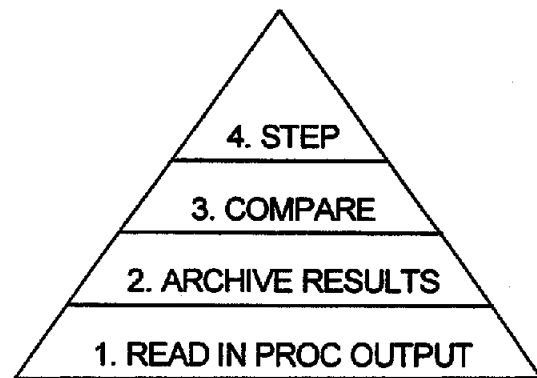


Figure 1: Stepwise regression can be implemented using model comparison based on reading in and storing statistics from regression output.

**LEVEL 1:
READING IN PROC OUTPUT**

Every SAS programmer has encountered the need to manage the statistical results generated by SAS procedures. Although many procs have features to store statistical output directly into SAS datasets, some do not, or will store some results in a dataset but write other results only to the procedure output file. Future versions of the SAS system may provide storage for some or all of these statistics, but meanwhile we need to solve the problem of reading in SAS procedure output. This section describes a standardized way of doing this which has made the task easier and the result more robust.

Analysis and Modeling

The following macros have been developed which read in SAS procedure output:

RDFREQ	reads PROC FREQ output
RDGLM	reads PROC GLM output (basic)
RDREG	reads PROC REG output, regular or stepwise.
RDLOGIST	reads PROC LOGISTIC output, regular or stepwise
RDPHREG	reads PROC PHREG output

Each "RD" macro has the following parameters:

PROCFILE	Name of the input file containing procedure output.
BY	list of By-variables (optional)
DATA	name of original input dataset (only required if BY parameter is specified)
PRINT	Y/N whether to print the input file.
RDOUT_x, RDOUT_xx, etc.	Output file(s) needed to store the statistical results, depending on the structure of output produced by a particular procedure.

The PROCFILE could be created either as the output of an entire SAS program, or it could be created by using PROC PRINTTO to direct the procedure output to a specific file from within a program. In either case, the macros require that the PROCFILE contain the output from at most one execution of the proc, although that execution may have been with a BY statement, generating several repeated sections of output. A consistent line length of 126 was used during the development and testing of these macros.

The RD macros do not require knowledge of the options used when the procedure first was executed. The goal generally was to have the macro be flexible enough to figure out what information is available simply by reading the output file, just as a person would. Therefore, the variables on the output datasets are always the same, but if some output statistics were not generated by the original execution of the PROC, these variables would contain missing values on the output datasets. Thus following programs can count on certain variables existing, and the PROC CONTENTS of the output datasets serves as the documentation of

the full range of statistics being captured by the macro.

If BY variables were used in the original execution of the PROC, we want the same BY variables on the output files containing the statistics. For the macro to have the identical characteristics of the BY variables (i.e. their type, length and labels), the original input dataset name is required.

To make development easier, each macro uses a two-step structure, creating an intermediate dataset which can be printed as an aid in debugging.

Step 1: Read each line of the PROCFILE into a dataset with one observation per line and a character variable LINE. Calculate the BY-variables, and a character variable called SECTION, which associates each line with a particular section of SAS procedure output. For example, with a simple PROC GLM execution, the sections are "Model Header", "Type I Stats" and "Type III Stats". When developing or enhancing these macros, before going on to try to pick up the statistical results within each section, it is a comfortable sanity check to print the intermediate dataset at this point, checking the by-variables and the sections. We have found that most procedure output can be broken into sections by a combination of detecting key words and counting blank lines. Lines which do not get assigned to any section are ignored.

Step 2: Within the named sections, read the lines of procedure output to pick up the statistical results. The majority of statistical results can be correctly identified by a combination of detecting key words on a given line and using the SCAN function. For example, R-square is the second item found on the line beginning with the word "R-SQUARE" would be written as the following line of code:

```
if line='R-SQUARE' then  
  rsquare = scan(line,2,' ');
```

Further logic controls the retaining of variables and the outputting of observations to the output datasets.

(optional) Step 3: Merging datasets containing information of the same "type". For example, in

reading PROC GLM output, we encounter both Type I and Type III sums-of-squares and tests. Each kind of data is temporarily output to a dataset with one observation per term in the model. Then these datasets are merged to yield a single output dataset containing all of the term-level statistical results. A separate output dataset would contain the model-level statistical results.

Table 1:
Output datasets produced by RD macros.

Macro	Output data	Contents
RDFREQ	RDOUT	One obs per 2x2 table
RDGLM	RDOUT_M	Model-level statistics
	RDOUT_V	Term-level statistics
RDREG	RDOUT_M	Model-level statistics
	RDOUT_S	Step-level statistics
	RDOUT_SV	Variable-within step statistics
	RDOUT_V	Final or only model variable-level statistics
RDLOGIST	RDOUT_M	Model-level statistics
	RDOUT_S	Step-level statistics
	RDOUT_SV	Variable-within step statistics
	RDOUT_V	Final model variable-level statistics
RDPHREG	RDOUT_M	Model-level statistics
	RDOUT_V	Variable-level statistics

In addition to what is shown in Table 1 above, all output datasets would contain any BY-variables specified as a parameter of the macro.

Types of Regression Output

Focusing on output from the regression-type procedures in SAS, we find that the statistical

output from procedures in this class falls into one or more distinct types or levels.

1. Model-level statistics include the name of the dependent variable, the number of observations used in the analysis, and the information in the analysis of variance table.
2. Step-level statistics (produced by PROC REG, LOGISTIC or PHREG run stepwise) include the step number, what action was taken in that step to enter or remove a variable or group of variables from the model.
3. Step-variable-level statistics include the parameter estimates of variables included in the model at a given step.
4. Variable-level statistics. Normally these are the result pertaining to the independent variables in a model. If a stepwise procedure is used, these also include the step-variable-level statistics from the final model.

This list does not cover all possible statistical output. For example, PROC GLM may include least-squares means and other items generated by the numerous options available on this procedure. The features of the macros have been developed on an as-needed basis. Also, many users have no need for step-level or step-variable-level statistics and can simply ignore the output datasets containing these results.

LEVEL 2: ARCHIVING RESULTS

In several data analysis projects, as a way to manage the volume of different models that are run, we have stored results of all statistical procedures in a "models archive". The archive is simply a dataset containing the statistical results from all the models tried during an analysis. Each observation on this dataset contains a set of identifying variables that make sense for the given analysis, plus a variable called "Model", containing a unique identifier for each different selection of independent variables or methodology. The other variables are project-specific. In one recent project the other key identifying variables were: Dependent Variable, Transformation, Period, and Subset. When new models are run, the results are read into a dataset which is concatenated onto the existing

Analysis and Modeling

archive dataset using a SET statement (not PROC APPEND). If a model is rerun with a correction we want to replace the earlier version of the results. In this case any observations already in the archive having the same values of the identifying variables are deleted.

Since the macros which read in regression output store model-level and variable-level statistics in separate datasets, these datasets are merged into one prior to storing in the archive. (There is no need to combine them if you are willing to organize the archive as separate model-level and variable-level archive datasets.) Additional identifying information, such as the date-time when the model was run, also are stored with the results.

Example: Storing results in an archive.

The following code reads in the output from PROC REG, merges the model-level dataset onto the variable-level dataset, and stores it in a main archive dataset.

```
*****
** EXECUTE REG AND READ RESULTS
*****;

* Direct output to a file;
proc printto print='regfile.prn';
run;

proc reg data=datal;
  by region;
  model a=b c;
run;

proc printto;
run;

* Macro call to read in results;
%RDREG(procfile="regfile.prn",
      by=region,
      data=datal,
      rdout_m=mdata,
      rdout_v=vdata);

*****
** STORE RESULTS IN ARCHIVE
*****;
* Merge model-level results with
variable-level results, and add
some new variables;
data results;
  merge mdata vdata;
  by region;
```

```
whenrun=datetime();
length model $ 8 moddesc $ 40;
model='BASIC004';
moddesc='Basic with B and C';
run;
```

```
proc sort data=results;
  by depvar region model;
run;
```

```
* Add new results to archive,
deleting any earlier versions
with the same key identifier
variables.;
data perm.archive;
  set results(in=innew)
      perm.archive(in=inold);
  by depvar region model;
  if first.model then
    hasnew=innew;
  retain hasnew;
  if hasnew and inold then
    delete;
run;
```

Our standard way of using the above code is to enclose it in a macro and make parameters out of the items that tend to vary frequently, such as the list of independent variables, the model name, the model description, and (in order to permanently save the SAS procedure output) the name of the file written to by PROC PRINTTO. An analysis program consists of a series of executions of this macro, with the code commented out after they have executed. At the end of the project, the analysis program serves as documentation.

One advantage of archiving results is that a wide range of reports can be generated easily. For example, statistics about a particular variable of interest can be selected and listed, showing how they changed when various models were tried, or different subsets of the data were analyzed.

Other advantages include (1) the summarization of multiple bootstrap replications of an analysis, and (2) statistical comparison of models. These topics are covered in the following sections.

LEVEL 3: COMPARING AND SUMMARIZING

A. BOOTSTRAPPING

Bootstrapping is a technique where an original sample is used to generate numerous "bootstrap" samples. Each bootstrap sample contains the same number of observations (N) as the original, but is made up of N random picks of an observation from the original sample. A particular original observation might get picked 0, 1, 2, 3, etc. times to be included in a given bootstrap sample.

To "bootstrap" an analysis, both the original sample and each bootstrap sample is subjected to the same analysis. From each sample, the analysis will produce a slight difference in the statistical results, for example in the estimate of the regression coefficients for the independent variables in a regression model. A "bootstrap estimate" of a regression coefficient is calculated by taking the mean of the coefficients obtained from each of the bootstrap samples. Clearly this is a case where results of a statistical analysis need to be stored for further analysis.

The technique we have used for bootstrapping starts with a macro which calculates the bootstrap weighting of each observation in the original dataset. (A description of the BOOTSTRP macro is beyond the scope of this paper, but can be obtained from the authors.) If we were creating 100 bootstrap samples, these weightings are stored in variables BOOT1-BOOT100 as additional variables on the dataset. To do an analysis on bootstrap sample number 3, for example, we would add the statement FREQ BOOT3; to the procedure. Then each observation would be either skipped, if BOOT3=0, or counted 1, 2, 3, etc. times.

The following program calculates bootstrap estimates of the parameters in a regression model, and also calculates the percentage of the samples in which each independent variable had a significant p-value:

```
* This macro adds the variables
boot1-boot100 to the dataset;
%BOOTSTRP(in=data1,out=data2,
repeat=100,prefix=BOOT);
```

```
<< At this point you would probably
want to save data2 permanently >>
```

```
* This view feeds out the data one
bootstrap sample at a time. This is
more efficient than 100 separate
executions of PROC REG;
data samples / view=samples;
do sample=1 to 100;
  do p=1 to nob;
    set data2 pointer=p nob=nob;
    array boot_ (*) boot1-boot100;
    boot=boot_(sample);
    drop boot1-boot100;
    output;
  end;
end;
run;
```

```
* This directs the procedure output
to a file "myfile". It will have
the output from 100 repeated
executions of PROC REG;
proc printto print="myfile";
run;
```

```
proc reg data=samples;
  by sample;
  freq boot;
  model income = age educ sex;
run;
```

```
* Restores the default print file;
proc printto;
run;
```

```
* This macro reads the regression
output. Since we are only
interested in the variable-level
file, we can ignore the model-level
information;
%RDREG(procfile="myfile",
  by=sample,
  data=samples,
  rdout_v=vdata);
```

```
* Calculate a new variable SIGNIF
which will be summarized later to
show what percent of the time each
independent variable was
significant at less than .05;
data results;
  set rdout_v(keep=sample variable
    parmest pvalue);
  * signif will be 0 or 100;
  signif= (pvalue<=.05)*100;
run;
```

```
<< continued >>
```

Analysis and Modeling

```
* Calculate "bootstrap estimates"
of the regression parameter for
each independent variable. Mean of
signif can be interpreted as a
percent.;
```

```
proc summary data=results;
  class variable;
  var parmest signif;
  output out=summs mean=meanest
         pctsig;
run ;
```

```
* print the summary statistics;
proc print data=summs;
  var variable meanest pctsig;
run;
```

LEVEL 3 (continued): COMPARING AND SUMMARIZING

B: MODEL COMPARISON

Saving and comparing results from different models enables you to conduct tests of the significance of groups of variables. The method we have used for this purpose is the likelihood ratio chi-square test, based on the Log Likelihood statistic produced by several procedures, including LOGISTIC and PHREG.

Why is it important to be able to test the significance of groups of variables? Because often the preferred way to represent independent variables in an analysis is as a set of dummy variables (zero-one). For example, a variable "Education" might be represented by a set of 5 dummy variables (no pun intended), one for each level of education completed. By doing this, we avoid having to be concerned with the actual shape of the distribution of the effects of education levels.

The disadvantage is that some regression procedures give you no means of testing for the significance of an entire group of dummy variables. PROC REG gives a test of significance of a group of dummy variables when you use the GROUP option in a stepwise or backwards regression, and PROC GLM would allow you to model Education as a class variable. However, both PROC LOGISTIC and PROC PHREG lack any feature to test the significance of groups of dummy variables.

The saving grace is that both of these procedures print the "-2 Log Likelihood" statistic for the

model as a whole. A test of significance of the difference between two models can be constructed by subtracting the -2 Log Likelihood of the two models to get a log likelihood ratio statistic. This can be assumed to have a Chi-square distribution with degrees of freedom equal to the difference between the degrees of freedom of the two models. The code to perform this test is found at the end of the following example.

Example 1: Testing the significance of groups of dummy variables using PROC LOGISTIC.

The following program shows how one can test the significance of adding a group of dummy variables representing "Education" to a model attempting to predict whether people consider themselves "happy". When a set of dummy variables covers all possible cases, one variable must be omitted from a regression model to avoid over-specifying the model. In this example we omitted the "high-school" dummy variable.

```
*****
** 1. Run the basic model
*****;
* Capture the procedure output;
proc printto print="myfile";
run;

proc logistic data=mydata;
  model happy= <basic list>;
run;

* This macro reads the regression
output into a model-level file,
ignoring the variable-level
results;
%RDLOGIST(procfile="myfile",
          rdout_m=rdout_ml);

*****
** 2. Run the changed model
*****;
* Repeat the above except that the
set of dummy variables representing
Education are added to the model;

* Capture the procedure output;
proc printto print="myfile";
run;

* NOTE THE CHANGED MODEL;
proc logistic data=mydata;
  model happy= <basic list> lesshs
              somecoll collgrad postcoll;
run;
```

* This macro reads the regression output into a different model-level file, ignoring the variable-level results;

```
%RDLOGIST(procfile="myfile",
           rdout_m=rdout_m2);
```

```
*****
** 3. Compare using likelihood
** ratio chi-square test.
*****;
```

* Merge the results and perform a statistical test of significance. Note that the RDLOGIST macro stores the "-2 Log Likelihood" statistic in variable M2LL COV, and the degrees of freedom due to the covariates in LL_DF;

```
data test;
merge
rdout_m1
(rename=( M2LL_COV=l11 LL_DF=df1))
rdout_m2
(rename=(M2LL_COV=l12 LL_DF=df2));
* by-statement if needed;
diff11 = abs(l12-l11);
diffdf = abs(df2-df1);
chipval=1-probchi(diff11,diffdf);
run;
```

Key point: Proc Logistic writes out the "-2 Log Likelihood". If it had only written the "Log Likelihood" it would have been necessary to multiply it by 2 before calculating the test shown above. Also note that the LL_DF statistic is not the "model degrees of freedom" since it does not include the one degree of freedom due to the intercept term, but this does not matter since we are subtracting to get the change in the degrees of freedom, which is correct.

Example 2: Testing for interaction terms.

Only PROC GLM has a feature for testing interaction terms directly. With other procedures you have to employ a do-it-yourself methodology. Sometimes you may be interested in the interaction between two yes-no variables. That is a special case in which the interaction is itself a dummy variable, calculated as true if both basic variables are true. This case is handled by adding the interaction variable to the model and letting the SAS proc give you a test of its significance. The more problematic case is where either of the basic variables has more than two levels. Then you must calculate several dummy variables to represent the interaction, and to test the significance of the interaction (as

a whole) you employ the same technique as in Example 1 above.

The following program shows how one can test the significance of the interaction between a variable "Married" and the list of education dummy variables used earlier.

```
data mydata2;
set mydata;
* calculate interaction vars;
int1 = (married * lesshs);
int2 = (married * hs);
int3 = (married * somecoll);
int4 = (married * collgrad);
int5 = (married * postcoll);
run;
```

* Now compare a model with and without the interaction terms, using the macros and methods shown in Example 1 above. To avoid over-specifying the model, omit one interaction dummy variable, such as INT2;

The above technique was used with PROC PHREG (Proportional Hazards Regression) which was implemented with time-varying covariates as predictors of survival. Procedure output was stored in an archive, and a separate program was executed to compare specific model pairs, using the likelihood ratio test described above, testing the significance of interactions and groups of dummy variables.

LEVEL 4: STEPWISE MODELING

The "stepwise" algorithm in regression model-building comes in three varieties: Forward, Backward, and Stepwise. We refer to all three as "stepwise regression" methods. The Forward method starts with no independent variables (or a short list of variables said to be "forced into" the model), and at each step it adds whichever independent variable will add the most explanatory power to the model. Backward elimination works in the opposite way. The "Stepwise" method is Forward combined with Backward. It starts the way the Forward method would, and adds variables. The difference is that after adding a variable, it removes any variable in the model which fails to meet the criterion for staying in the model. This can happen occasionally when adding a variable which is correlated with variables already in the model.

Analysis and Modeling

All that is required to run any regression procedure "stepwise" is to imbed the comparison of two models into a framework of forward or backward stepping, either adding or removing independent variables from the model.

One feature of PROC REG is of particular interest -- the GROUPS= option on the MODEL statement. This instructs the procedure to enter or remove variables in groups, rather than singly.

Stepwise modeling with GROUPS

We implemented the stepwise algorithm in two versions. Macro STEPWISE is able to handle groups of variables, but for a limited range of procedures -- those printing the Log Likelihood statistic. Macro STEPVAR does the stepwise algorithm one variable at a time, and can be applied to the full range of regression procedures listed below.

Table 2: Stepwise features of regression procedures.

Regression procedure	Does it have step-wise?	If yes, with Groups ?	Does it print Log Likelihood?
CALIS	++		
CATMOD	++		
GENMOD	+	+	Yes
GLM	++		
LIFEREG	+	+	Yes
LOGISTIC	Yes	+	Yes
MIXED	++		
NLIN	++		
ORTHOREG	++		
PHREG	Yes	+	Yes
PROBIT	+	+	Yes
REG	Yes	Yes	
RSREG	++		
TRANSREG	++		

+ These cells can be enabled by STEPWISE
++ These cells can be enabled by STEPVAR

Currently STEPWISE is implemented for procs PHREG and LOGISTIC. The STEPVAR macro is implemented for GLM.

Table 3: STEPWISE macro parameters:

PROC	Name of procedure to be run stepwise. Currently accepts only LOGISTIC and PHREG.
DATA	Name of input dataset
DEPVAR	Dependent variable name
INDVARS	List of independent variables. Groups of variables are separated by a vertical bar.
GROUPS	List of group names, also separated by vertical bars.
METHOD	F, B or S for Forward, Backward or Stepwise selection method
INCLUDE	=n. If given, the first n groups are forced into all models regardless of significance.
SLENTER	=.05. The p-value required to enter a group in the model.
SLSTAY	=.05. The p-value required to not be removed from the model.
M_OPTS	Options to go on Model statement. A selection method (if available) should NOT be requested here.
P_OPTS	Options to go on Proc statement.

Example: The STEPWISE macro used with Proc Logistic using groups of variables and backward stepwise elimination.

Using the macro in this situation is an advantage because PROC LOGISTIC does not have a GROUPS= option, as PROC REG does. In the following code, variables "poor" and "hs" are grouped as "Demog".

```
%STEPWISE(
PROC=logistic,
DATA=mydata,
DEPVAR=outcome,
INDVARS=age|sex|treated|poor hs
|hungry,
GROUPS=Age|Sex|Treatment|Demog|
Hunger,
METHOD=B,
INCLUDE=1);
```

Discussion: You would run this macro if you want to use groups of variables. Otherwise you would use a straight PROC LOGISTIC step with SELECTION=B. The grouping of two variables (poor and hs) may seem like minor reason in this example, but our analysis projects have often involved numerous groups of variables, in particular groups of dummy variables representing the levels of several class variables.

Output: Printed output from the above example consisted of the following:

Step 0: All groups entered

```
<PROC LOGISTIC output from a
model with all variables in the
model >
```

Step 1:

```
To remove "Sex"      pvalue=0.1532
To remove "Treated"  pvalue=0.2792
To remove "Demog"    pvalue=0.7178
To remove "Hungry"   pvalue=0.5245
REMOVED group Demog
consisting of: poor hs
```

```
<PROC LOGISTIC output from a
model with Age, Sex, Treated, and
Hungry in the model >
```

Step 2:

```
To remove "Sex "     pvalue=0.1649
To remove "Treated"  pvalue=0.3447
To remove "Hungry"   pvalue=0.5808
REMOVED group Hungry
consisting of: hungry
```

```
<PROC LOGISTIC output from a
model with Age, Sex and Treated
in the model >
```

Step 3:

```
To remove "Sex "     pvalue=0.0184
To remove "Treated"  pvalue=0.3447
REMOVED group Treated
consisting of: treated
```

```
<PROC LOGISTIC output from a
model with Age and Sex in the
model >
```

No further groups qualified to be added or removed.

Computational efficiency: The STEPWISE macro carries out a large number of model comparisons. If N represents the number of groups not always included (controlled by the INCLUDE parameter), then the procedure is executed up to $N*(N+1)/2$ times. In actual practice it will be less because the stepwise process halts when no further changes to the model pass the significance criteria. If on the average half of the groups are significant, we still would require $N*(N+1)/4$ executions. Using the "Stepwise" method requires another doubling of the number of executions, compared to the "Forward" or "Backward" methods.

Necessary details: The input dataset must be consistent with regard to the presence of missing values among the independent variables. This insures that every model run will be based on the same set of observations.

Validation of p-values: Proc Logistic produces p-values using its own test of the significance of a single variable in the model. When a group consists of a single variable, the p-value which the macro produces by using the likelihood ratio test comparing models with and without the variable is close to but not exactly equal to the procedure-based p-value for the variable. The procedure's p-values were not be used because some groups have multiple variables, and it seemed preferable to use the same statistical test to compare the effect of groups regardless of their size.

As we shall discuss in the next section, a wider variety of procedures can be run in a stepwise manner if we abandon the feature of testing groups of variables.

The STEPVAR Macro

Clearly it is possible to give the stepwise variable selection feature to additional procedures, including GLM, if we do not try to implement the GROUPS feature. This is what the STEPVAR macro does.

To understand how simple this is, imagine that you wish to perform backwards stepwise regression using, say, PROC GLM with a simple linear model. When a given model has been executed, you would look at the (Type III)

Analysis and Modeling

significance of the different terms in the model, remove the term with the least significance, and rerun the model. Continuing in this fashion, you would arrive at a model with only significant terms.

The STEPVAR macro combines this with a forward stepping procedure. In this case the procedure is run several times per step, each time adding a single variable to the model. Then the algorithm picks the variable which has the most significant p-value when added to the current model.

The STEPVAR macro has the same parameters as the STEPWISE macro, with the exception of the GROUPS parameter. The output has the same sequence and format.

CONCLUSION

The topics presented in this paper may be of use on several levels. First, macros are presented which enable you to read in and manipulate SAS procedure output. These have been used with the set of options that are favorites of the authors, but we also wish to enhance these macros to handle the widest possible variety of printing options.

Secondly, we strongly recommend archiving statistical results to maintain order within the often confusing complexity of a typical data analysis project. In addition, it saves trees.

Finally, the technique of model comparison, and the STEPWISE macro presented here have made life easier in the projects in which we have used groups of dummy variables as independent variables. And the STEPVAR macro promises to speed up the search for reduced models using GLM or any other regression procedure.

ACKNOWLEDGMENTS

We wish to thank Richard Brand, a statistician at UCSF, for statistical consulting, and also Reiling Lee and other programmers at the UCSF Health Economics Research Group for their pioneering work on earlier versions of the bootstrapping macros.

REFERENCES

- Efron, B., and Tibshirani, R.J. (1993). An introduction to the Bootstrap. New York: Chapman & Hall.
- Hur, K., Oprian, C.A., Henderson, W.G., Thakkar, B., Urbanski, S. A SAS macro for validating a logistic model with split sample and bootstrap methods. Proceedings of the Twenty-first Annual SAS Users Group International Conference, p. 953-956.
- Thompson, P.A. (1996) A tutorial on bootstrapping in the SAS System. Proceedings of the Twenty-first Annual SAS Users Group International Conference, p. 1295-1304.

CONTACT INFORMATION

We look forward to collaborating with other SAS users who will adapt these macros to their own liking and share the revisions. To receive copies of the macros discussed in this paper, please email your inquiry to:

Ted Clay
Clay Software and Statistics
168 Meade St.
Ashland, OR 97520
1-541-482-6435 (phone and fax)
clay@mind.net

Linda Remy, Ph.D.
Family Health Outcomes Project
University of California, San Francisco
3333 California St.#365
San Francisco, CA 94118
1-415-435-3075 (phone), -3071 (fax)
lremy@itsa.ucsf.edu