Paper 261-31

Using SAS® ODS to extract and merge statistics from multiple SAS procedures into a single summary report, a detailed methodology.

Stuart Long, Westat, Durham, NC
Jeff Abolafia, Rho Inc., Chapel Hill, NC
Lawrence Park, Murdoch University, Perth, Australia

Abstract

SAS® procedures often provide more statistical output than is needed for a given analysis. When an analysis involves iterative processing on numerous groups of variables using multiple procedures, the number of output pages to be reviewed may be quite massive.

By using SAS ODS output datasets, one can extract only the desired statistics from a procedure. The ODS output datasets from numerous procedures can be merged together and used by the programmer to generate concise reports summarizing the results from the different SAS procedures..

This presentation provides a detailed blueprint for 1) generating ODS output datasets from multiple procedures, 2) retaining specific statistics in these datasets, and 3) merging them together to produce one summary dataset. This is an advanced tutorial intended for SAS programmers knowledgeable with BASE/SAS and the SAS Macro facility.

Introduction

A typical statistical analysis using SAS involves the execution of multiple procedures on the same sets of variables. These procedures often generate many more statistics than are desired for review by a client. In addition to large amounts of unwanted statistics, different SAS procedures can generate the same statistics, resulting in redundant output. In the end, the programmer must combine and summarize such results in a report for the client. The SAS Output Delivery System (ODS) provides the programmer with a convenient facility to capture subsets of output generated from a SAS procedure and store them in SAS datasets Prior to ODS, only a subset of output could be stored in SAS datasets for later use. Capturing additional output involved using PROC PRINTTO to route output to an external file and then using INFILE/INPUT statements to extract the desired statistics from the external file. This process was both cumbersome and error prone.

ODS Output datasets from multiple procedures can be merged to create a single summary dataset that contains all of the desired statistics. This dataset can form the basis for creating a report for the client.

The first part of this paper presents the reader with a blueprint for creating ODS Output datasets from SAS procedures, trimming them down to contain only the variables of interest, and then merging these datasets into a summary dataset with statistics from multiple procedure calls. The second section of the paper demonstrates how to automate this process to repeat a particular set of analyses on a list of variables.

The examples presented in this paper are derived from analyses of the Agricultural Health Study by the National Cancer Institute, the National Institutes of Environmental Health Sciences, and the Environmental Protection Agency. Here, we will quantify the association between a 3-level independent variable *daysmix* (average days per year mixed pesticides) with a dichotomous response variable *asthma* (diagnosed with asthma). All statistics included in this paper have been altered for presentation purposes and do not represent actual statistical associations from the study.

PART 1: Coding for a Single Analysis

Creating ODS Output datasets

By default, the output from a SAS procedure is stored in internal data objects that are then printed in full to the SAS listing destination. The first step in routing this output to a dataset is to identify the names of the objects generated by the procedure.

The ODS TRACE statement prints information about the generated objects to the SAS Log. To execute the ODS TRACE statement, a procedure must be embedded within the ODS TRACE ON and ODS TRACE OFF statements as shown below. Alternatively, for all SAS procedures except some of those in BASE SAS, the names of the ODS objects can be found in the procedure documentation section Details – ODS Table Names.

```
ODS TRACE ON;

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

ODS TRACE OFF;
```

Figure 1: SAS LOG

```
1  ODS TRACE ON;
2
3  PROC FREQ DATA=mydata;
4    TABLES DaysMix * asthma  ;
5
6  RUN;

Output Added:
-------------
Name:       CrossTabFreqs
Label:      Cross-Tabular Freq Table
Data Name:
Path:       Freq.DaysMix_by_asthma.CrossTabFreqs
-------------
NOTE:There were 22395 observations read from mydata
NOTE: PROCEDURE FREQ used:
      real time           0.07 seconds
      cpu time            0.07 seconds

7  ODS TRACE OFF;
```

The above code requests that the names of objects created by PROC FREQ be added to the SAS LOG, as shown in Figure 1. In this example, the name of the object containing the statistics is CrossTabFreqs.

The second step required to route the output to a dataset is to assign an output dataset to contain all the statistics that are found in the object. In the code below, we use the ODS OUTPUT statement to store the data from CrossTabFreqs in the dataset *_CTAB*. During execution of the PROC FREQ, output is written to the dataset *_CTAB*, as well as to the SAS Listing. The ODS OUTPUT CLOSE statement is used to turn off the ODS OUTPUT destination. Figure 2 shows the output that is sent to the SAS Listing, and Figure 3 contains the information in the dataset *_CTAB*.

```
ODS OUTPUT CrossTabFreqs = _ctab;

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

ODS OUTPUT CLOSE;

PROC PRINT DATA=_ctab UNIFORM; RUN;
```

Trimming the dataset *_CTAB*

In Figures 2 and 3, the subset of the statistics we want to keep, the cell counts and column percentages, are highlighted. Note in Figure 3 that the observations of interest are those where the value of the variable *_type_* is equal to "11". The code on the next page shows how to reduce the dataset *_CTAB* to three observations containing only the data of interest.

First, we subset the dataset, keeping only the observations where *_type_* = "11". Second, we use the RETAIN and OUTPUT statements to further reduce the dataset to three observations that correspond to the three levels of the variable *daysmix*. In the third part of this process, descriptive information, such as variable names, labels, formats and values, are generated; these are used later to facilitate the reporting process. During the Data Step, we create the descriptive variable *var2_value*, which will also act as a key variable for merging this dataset with additional summary datasets created by other procedures.

Figure 2: PROC FREQ Listing Output

The FREQ Procedure

Table of DaysMix by asthma

DaysMix(Average days per year mixed pesticides)
asthma (Ever Diagnosed with asthma)

Frequency Percent Row Pct Col Pct	No	Yes	Total
None	6658 38.15 95.06 39.84	346 1.98 4.94 46.95	7004 40.14
1-90 Days	6358 36.44 95.81 38.04	278 1.59 4.19 37.72	6636 38.03
> 90 Days	3697 21.19 97.03 22.12	113 0.65 2.97 15.33	3810 21.83
Total	16713 95.78	737 4.22	17450 100.00

Frequency Missing = 4945

Figure 3: PROC PRINT Listing output of dataset *_CTAB*

Obs	Table	DaysMix	asthma	_type_	_TABLE_	Frequency	Row Percent	Col Percent	Missing
1	DaysMix by_asthma	None	No	11	1	6658	38.155	95.0600	39.8373	.
2	DaysMix by_asthma	None	Yes	11	1	346	1.983	4.9400	46.9471	.
3	DaysMix by_asthma	None	.	10	1	7004	40.138	.	.	.
4	DaysMix by_asthma	1-90 Days	No	11	1	6358	36.436	95.8107	38.0422	.
5	DaysMix by_asthma	1-90 Days	Yes	11	1	278	1.593	4.1893	37.7205	.
6	DaysMix by_asthma	1-90 Days	.	10	1	6636	38.029	.	.	.
7	DaysMix by_asthma	> 90 Days	No	11	1	3697	21.186	97.0341	22.1205	.
8	DaysMix by_asthma	> 90 Days	Yes	11	1	113	0.648	2.9659	15.3324	.
9	DaysMix by_asthma	> 90 Days	.	10	1	3810	21.834	.	.	.
10	DaysMix by_asthma	.	No	01	1	16713	95.777	.	.	.
11	DaysMix by_asthma	.	Yes	01	1	737	4.223	.	.	.
12	DaysMix by_asthma	.	.	00	1	17450	100.000	.	.	4945

```
DATA __freqs( KEEP = var1_name var1_value1 var1_fmt1 var1_value2 var1_fmt2 var1_label
                     var2_name var2_value var2_fmt var2_label
                     n_col1 p_col1 n_col2 p_col2 );
   LENGTH var1_name var2_name $20
          var1_label var1_fmt1 var1_fmt2 var2_label var2_fmt $40;
   SET _ctab (WHERE = (_TYPE_="11"));
     BY DaysMix asthma ;
   RETAIN var1_value1 var1_fmt1 n_col1 p_col1;
   IF FIRST.DaysMix THEN DO;
     n_col1       = frequency;
     p_col1       = colpercent;
     var1_value1  = asthma;
     var1_fmt1    = VVALUE(asthma);
   END; ELSE
   IF LAST.DaysMix THEN DO;
     n_col2         = frequency;
     p_col2         = colpercent;
     var1_value2    = asthma;
     var1_fmt2      = VVALUE(asthma);
     var1_name      = "asthma";
     var1_label     = VLABEL(asthma);
     var2_value     = DaysMix;
     var2_fmt       = VVALUE(DaysMix);
     var2_name      = "DaysMix";
     var2_label     = VLABEL(DaysMix);
     OUTPUT;
END; RUN;
```

Figure 4 shows the PROC PRINT listing of this dataset. The original statistics of interest are high-lighted in this table. The variables containing the labels, *var1_label* and *var2_label*, have been omitted from this listing.

Figure 4: PROC PRINT Listing for the trimmed dataset *_FREQS*

Obs	var2	var1	var1_ value1	var1_ fmt1	var1_ value2	var1_ fmt2	var2_ value	var2_ fmt	n_col1	p_col1	n_col2	p_col2
1	DaysMix	asthma	0	No	1	Yes	1	None	6658	39.8373	346	46.9471
2	DaysMix	asthma	0	No	1	Yes	2	1-90 Days	6358	38.0422	278	37.7205
3	DaysMix	asthma	0	No	1	Yes	3	> 90 Days	3697	22.1205	113	15.3324

Summary of statistics from a second procedure: PROC LOGISTIC

In addition to the frequency counts and column percentages obtained from PROC FREQ, our analysis also requires information derived from logistic regression models, specifically odds ratios, 95% confidence intervals, and p-values for the independent variable in the models. Again, ODS TRACE is used to find the names of the objects used by PROC LOGISTIC. Figure 5 is an extract of the SAS LOG showing the 10 objects generated by PROC LOGISTIC.

```
ODS TRACE ON;

PROC LOGISTIC DATA=mydata DESCENDING;
  CLASS DaysMix (PARAM=REF REF="None");
  MODEL asthma = DaysMix;  RUN;

ODS TRACE OFF;
```

The data we want come from the objects ParameterEstimates and OddsRatios. Again, we use ODS Output datasets to save the results of interest.

```
ODS OUTPUT ParameterEstimates =_pe
           OddsRatios         =_or;

PROC LOGISTIC DATA=mydata DESCENDING;
  CLASS DaysMix(PARAM=REF REF="1");
  MODEL asthma=DaysMix;
  FORMAT DaysMix ;  RUN;

ODS OUTPUT CLOSE;
```

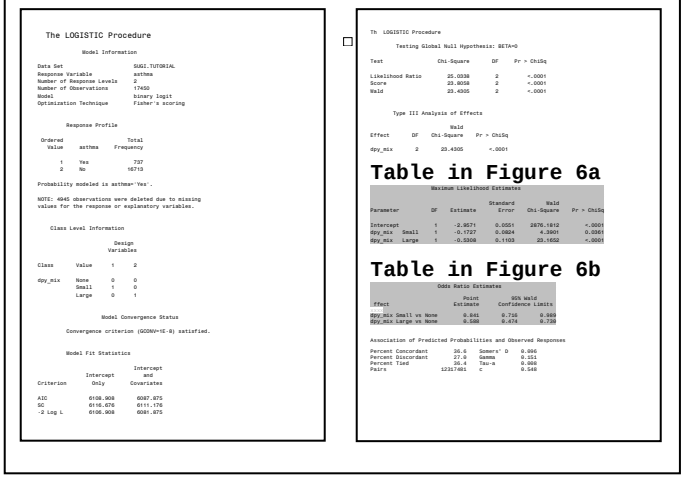
FIGURE 5: Partial Log for PROC LOGISTIC

```
Output Added:
-------------
Name:       ModelInfo
Name:       ResponseProfile
Name:       ClassLevelInfo
Name:       ConvergenceStatus
Name:       FitStatistics
Name:       GlobalTests
Name:       TypeIII
Name:       ParameterEstimates
Name:       OddsRatios
```

Figures 6, 6a and 6b show these objects as printed to the SAS Listing. Figures 7a and 7b display a PROC PRINT of the datasets *_PE* and *_OR*, respectively. PROC LOGISTIC creates character variables that contain text strings for the formatted values of numeric variables. These format text strings can often be cumbersome to work with. For this reason, we have removed formatting from our independent variable in the PROC LOGISTIC since the numeric values will be easier to work with.

The next task is to merge the p-value statistics from the dataset *_PE* with the estimates and confidence limits from dataset *_OR*. This is tricky because appropriate key variables for a matched merge are not present on these datasets. There are alternatives to a matched merge. First, we could make assumptions about the order of the variable levels on the datasets and use an unmatched merge (i.e., without a BY statement), but this is an unorthodox and extremely dangerous method. Second, the statistics found in the OddsRatios object are easily calculated from the statistics in the ParameterEstimates object. However, since the focus of this paper is the use of ODS Output datasets for working with procedural output, we will use a third option and create the needed key variables on which to merge these datasets. The variables *classval0* on *_PE* and *effect* on *_OR* can

Figure 6: PROC LOGISTIC Full Listing



The LOGISTIC Procedure

Model Information

Data Set	SUGI.TUTORIAL
Response Variable	asthma
Number of Response Levels	2
Number of Observations	17450
Model	binary logit
Optimization Technique	Fisher's scoring

Response Profile

Ordered Value	asthma	Total Frequency
1	Yes	737
2	No	16713

Probability modeled is asthma='Yes'.

NOTE: 4945 observations were deleted due to missing values for the response or explanatory variables.

Class Level Information

Class	Value	Design Variables
dpy_mix	None	0 0
	Small	1 0
	Large	0 1

Model Convergence Status

Convergence criterion (GCONV=1E-8) satisfied.

Model Fit Statistics

Criterion	Intercept Only	Intercept and Covariates
AIC	6108.908	6087.875
SC	6116.076	6111.176
-2 Log L	6106.908	6081.875

Th LOGISTIC Procedure

Testing Global Null Hypothesis: BETA=0

Test	Chi-Square	DF	Pr > ChiSq
Likelihood Ratio	25.0338	2	<.0001
Score	23.9058	2	<.0001
Wald	23.4305	2	<.0001

Type III Analysis of Effects

Effect	DF	Wald Chi-Square	Pr > ChiSq
dpy_mix	2	23.4305	<.0001

Table in Figure 6a

Maximum Likelihood Estimates

Parameter	DF	Estimate	Standard Error	Wald Chi-Square	Pr > ChiSq
Intercept	1	-2.9571	0.0551	2876.1812	<.0001
dpy_mix Small	1	-0.1727	0.0824	4.3901	0.0361
dpy_mix Large	1	-0.5308	0.1103	23.1652	<.0001

Table in Figure 6b

Odds Ratio Estimates

Effect	Point Estimate	95% Wald Confidence Limits	
dpy_mix Small vs None	0.841	0.716	0.989
dpy_mix Large vs None	0.588	0.474	0.730

Association of Predicted Probabilities and Observed Responses

Percent Concordant	36.6	Somers' D	0.096
Percent Discordant	27.9	Gamma	0.151
Percent Tied	35.4	Tau-a	0.068
Pairs	12317481	c	0.548

Figure 6a: ParameterEstimates Object

Analysis of Maximum Likelihood Estimates

Parameter		DF	Estimate	Standard Error	Chi-Square	Wald Pr>ChiSq
Intercept		1	-2.9571	0.0551	2876.1812	<.0001
DaysMix	2	1	-0.1727	0.0824	4.3901	0.0361
DaysMix	3	1	-0.5308	0.1103	23.1652	<.0001

Figure 6b: OddsRatios Object

Odds Ratio Estimates

Effect	Point Estimate	95% Wald Confidence Limits	
DaysMix 2 vs 1	0.841	0.716	0.989
DaysMix 3 vs 1	0.588	0.474	0.730

Figure 7a: PROC PRINT of dataset *_PE*

variable	class val0	DF	Estimate	StdErr	WaldChiSq	Prob ChiSq
Intercept		1	-2.9571	0.0551	2876.1812	<.0001
DaysMix	2	1	-0.1727	0.0824	4.3901	0.0361
DaysMix	3	1	-0.5308	0.1103	23.1652	<.0001

Figure 7b: PROC PRINT of dataset *_OR*

effect	Odds RatioEst	LowerCL	UpperCL
DaysMix 2 vs 1	0.841	0.716	0.989
DaysMix 3 vs 1	0.588	0.474	0.730

be used to create these key variables. Once these key variables are created, we can sort *_PE* and *_OR* and merge them to combine all the desired statistics on a single dataset. The code fragment below shows how to create the key variables for each dataset and then perform the merge.

```
DATA _pe (KEEP = var2 var2_value probchisq);
  LENGTH var2_value 8;
  SET _pe (WHERE = (variable="DaysMix"));
  var2_value = input(classval0,8.);
  RENAME variable = var2;  RUN;

DATA _or (KEEP = var2_value OddsRatioEst LowerCL UpperCL);
  LENGTH var2_value 8;
  SET _or (WHERE = ( UPCASE(SCAN(EFFECT,1)) = "DAYSMIX" ));
  var2_value = INPUT(SCAN(EFFECT,2),8.);  RUN;

PROC SORT DATA=_pe;
  BY var2_value;  RUN;

PROC SORT DATA=_or;
  BY var2_value;  RUN;

DATA __crude_log_stats;
  MERGE _pe _or;
    BY var2_value;  RUN;
```

Figure 8 shows the dataset *_CRUDE_LOG_STATS*.

In the next code fragment, the datasets *_FREQS* and *_CRUDE_LOG_STATS* are merged using the key variable *var2_value* to create the dataset *_SUMMARY*.

```
DATA __summary;
  MERGE __freqs __crude_log_stats;
    BY var2_value;  RUN;
```

Figure 9 shows the PROC PRINT listing for the *__SUMMARY* dataset, which contains the combined statistics from the PROC FREQ and PROC LOGISTIC ODS Output datasets. The Variables containing labels, *var1_label* and *var2_label*, have been omitted from this listing.

Figure 8: PROC PRINT of Dataset *_CRUDE_LOG_STATS*

var2_ value	var2	Prob ChiSq	Odds RatioEst	LowerCL	UpperCL
2	DaysMix	0.0361	0.841	0.716	0.989
3	DaysMix	<.0001	0.588	0.474	0.730

Figure 9: PROC PRINT of the combined dataset *__SUMMARY*

v a r 1	v a r 2	V a r 1 _ v a l u e	V a r 1 _ F m t	v a r 2 _ v a l u e	V a r 2 _ F m t	var2 _ value	n _ c o l 1	p _ c o l 1	n _ c o l 2	p _ c o l 2	P r o b S q	O d d s R a t i o	L o w e r C L	U p p e r C L
asthma	DaysMix	0	No	1	Yes	1	None	6658	39.8373	346	46.9471	.	.	.	.
asthma	DaysMix	0	No	1	Yes	2	1-90 days	6358	38.0422	278	37.7205	0.0361	0.841	0.716	0.989
asthma	DaysMix	0	No	1	Yes	3	> 90 days	3697	22.1205	113	15.3324	<.0001	0.588	0.474	0.730

Summary statistics from a third procedure: a multivariable logistic model using PROC LOGISTIC

Thus far, we have combined statistics from a PROC FREQ and an unadjusted PROC LOGISTIC. The model in the previous PROC LOGISTIC contained only a single predictor variable, *daysmix*, so the statistics from this model are crude estimates. The next step in an analysis might be to identify a set of base co-variables that are associated with the outcome variable and thus might be confounders of the association between the independent exposure of interest and the outcome. These base co-variables can then be used to adjust the exposure estimates for the independent variable. In this section, we run an multivariable model, extract from it the adjusted exposure statistics, and merge them with the dataset of crude statistics created earlier.

```
ODS OUTPUT parameterestimates = _pe
           OddsRatios         =_or;

PROC LOGISTIC DATA=mydata DESCENDING;
  CLASS bmi_group smoke DaysMix (PARAM=REF REF="1" );
  MODEL asthma = age bmi_group smoke DaysMix;
  FORMAT DaysMix;    RUN;

ODS OUTPUT CLOSE;
```

Figure 10: PROC PRINT listing of ParameterEstimates(*_PE*) dataset

Obs	Variable	classval0	DF	Estimate	StdErr	WaldChiSq	Prob ChiSq
1	Intercept		1	-3.7743	0.1652	521.7912	<.0001
2	age		1	0.7102	0.0716	98.6602	<.0001
3	bmi_group	Normal	1	-0.2371	0.0927	6.5505	0.0105
4	bmi_group	Obese	1	0.5243	0.0923	32.2980	<.0001
5	smoke	Past Smoker	1	0.000765	0.1429	0.0000	0.9957
6	smoke	Current Smoker	1	-0.1046	0.1959	0.2850	0.5934
7	DaysMix	2	1	0.0527	0.1390	0.1434	0.7049
8	DaysMix	3	1	-0.1335	0.2166	0.3797	0.5378

Figure 11: PROC PRINT listing of OddsRatios(*_OR*) dataset

Obs	effect		Odds RatioEst	LowerCL	UpperCL
1	age		2.034	1.768	2.340
2	bmi_group	Normal vs Thin	0.951	0.652	1.388
3	bmi_group	Obese vs Thin	0.444	0.305	0.647
4	smoke	Past Smoker vs Never Smoked	1.001	0.756	1.324
5	smoke	Current Smoker vs Never Smoked	0.901	0.614	1.322
6	DaysMix	2 vs 1	0.949	0.722	1.246
7	DaysMix	3 vs 1	1.143	0.747	1.747

The ODS Output datasets created from the adjusted model contain estimates for all of the variables in the model (i.e., the independent exposure of interest and the base co-variables). Figures 10 and 11 show the PROC PRINT listing for the data in these objects as they appear in ODS Output datasets. In these figures, the observations and statistics we want are highlighted.

To create the summary dataset of adjusted PROC LOGISTIC statistics, we modify the code used to create the *_CRUDE_LOG_STATS* dataset for the crude PROC LOGISTIC model. Once again, we will select only the observations from the ParemeterEstimates dataset, *_PE*, where the variable, *variable*, contains the name of our modeled independent variable, *daysmix*. As for the crude summary statistics, the variable *classval0* contains the unformatted numeric values for our modeled independent variable, *daysmix*. In the OddsRatio dataset, *_OR*, we must identify the rows that contain the statistics that we will summarize. To do this, we keep the observations where the name of the independent variable is the first word of the variable *effect*. As shown earlier, we need to create our key variable, *var2_value* for merging. We also create variable names for the adjusted statistics that are distinct from those used in the crude statistics.

```
DATA _pe (KEEP = var2 var2_value probchisq
          RENAME = (probchisq = Adjusted_probchisq));
  LENGTH var2_value 8;
  SET _pe (WHERE = (variable="DaysMix"));
  var2_value = input(classval0,8.);
  RENAME variable = var2;  RUN;

DATA _or (KEEP   = var2_value OddsRatioEst LowerCL UpperCL
          RENAME = (OddsRatioEst = Adjusted_OddsRatioEst
                    LowerCL      = Adjusted_LowerCL
                    UpperCL      = Adjusted_UpperCL));
  LENGTH var2_value 8;
  SET _or (WHERE =( UPCASE(SCAN(EFFECT,1))="DAYSMIX"));
  var2_value = INPUT(SCAN(EFFECT,2),8.); RUN;

PROC SORT DATA=_pe;
  BY var2_value;  RUN;

PROC SORT DATA=_or;
  BY var2_value;  RUN;

DATA __adjusted_log_stats;
  MERGE _pe _or;
    BY var2_value;  RUN;
```

The dataset containing the adjusted statistics for the independent variable *daysmix* is shown in Figure 12.

Figure 12: PROC PRINT of *_ADJUSTED_LOG_STATS*

var2_value	var2	Adjusted_ ProbChiSq	Adjusted_ OddsRatio	Adjusted_ LowerCL	Adjusted_ UpperCL
2	DaysMix	0.7049	0.949	0.722	1.246
3	DaysMix	0.5378	1.143	0.747	1.747

Combining three datasets of summary statistics to create the final summary dataset

We now have three datasets of summary statistics derived from three separate procedures. These three datasets have a common key variable to merge on, *var2_value*. The contents of the SAS dataset *__SUMMARY* are shown in Figure 13.

```
DATA __summary;
  MERGE __freqs
        __crude_log_stats
        __adjusted_log_stats;
    BY var2_value;  RUN;
```

Figure 13: Contents of finalized __*SUMMARY* dataset

The CONTENTS Procedure

-----Variables Ordered by Position-----

#	Variable	Type	Len	Pos	Format	Label
1	var1_label	Char	40	165		Label of Response Variable
2	var1_value1	Char	40			Value for Level 1 of Response variable
3	var1_fmt1	CHAR	40			Format value for Level 1 of Response variable
4	var1_value2	Char	40			Value for Level 2 of Response variable
5	var1_fmt2	CHAR	40			Format value for Level 2 of Response variable
6	var1_name	Char	32	80		Name of Response Variable
7	var2_label	Char	40	123		Label of Independent Variable
8	var2_value	Num	8	72		Value of Independent Variable
9	var2_fmt	CHAR	40			Format value for the Independent variable
10	var2_name	Char	6	207		Name of Independent Variable
11	n_col1	Num	8	32		PROC FREQ Frequency of Column 1
12	n_col2	Num	8	40		PROC FREQ Frequency of Column 2
13	p_col1	Num	8	48		PROC FREQ Percent of Column 1
14	p_col2	Num	8	56		PROC FREQ Percent of Column 2
15	crude_p_value	Num	8	64	PVALUE6.4	PROC LOGISTIC Crude p-value
16	crude_OddsRatio	Num	8	8	ODDSR8.3	PROC LOGISTIC Crude Odds Ratio Estimate
17	crude_LowerCL	Num	8	16	ODDSR8.3	PROC LOGISTIC Crude Lower 95% Limit
18	crude_UpperCL	Num	8	24	ODDSR8.3	PROC LOGISTIC Crude Upper 95% Limit
19	adjusted_p_value	Num	8	0	PVALUE6.4	PROC LOGISTIC Adjusted p-value
20	adjusted_OddsRatio	Num	8	8	ODDSR8.3	PROC LOGISTIC Adjusted Odds Ratio Estimate
21	adjusted_LowerCL	Num	8	16	ODDSR8.3	PROC LOGISTIC Adjusted Lower 95% Limit
22	adjusted_UpperCL	Num	8	24	ODDSR8.3	PROC LOGISTIC Adjusted Upper 95% Limit

Figure 14 contains an example of a report created using the summary dataset in Figure 13. In this report, the source procedure for the various statistics are noted by the following letters above the appropriate columns:

(a) PROC FREQ
(b) Crude PROC LOGISTIC
(c) Adjusted PROC LOGISTIC

Figure 14: Example summary report created from the above __*SUMMARY* dataset

Variable	Ever Diagnosed with asthma								
	Yes		No		P-Value		OR	95% CI	
Label/Format	N	%	N	%	Crude	Adjusted	Adjusted		
	(a)	(a)	(a)	(a)	(b)	(c)	(c)	(c)	(c)
Average days per year mixed pesticides									
None	346	46.9	6658	39.8	.	.			
1-90 days	278	37.7	6358	38.0	0.0361	0.7049	0.949	0.722	1.246
> 90 days	113	15.3	3697	22.1	<.0001	0.5378	1.143	0.747	1.747

If we repeat the steps shown so far in this paper for another variable of interest, *farmsize*, we can use PROC APPEND to concatenate the summary datasets for both variables and use the resulting dataset to prepare a composite summary report.

```
PROC APPEND BASE = __summary DATA= farmsize FORCE;  RUN;
```

Figure 15 shows a summary report for the two independent variables *daysmix* and *farmsize*.

Figure 15: Example of a composite summary report

	Ever Diagnosed with asthma							
Variable	Yes		No		P-Value		OR	95% CL
Label/Format	N	%	N	%	Crude	Adjusted	Adjusted	
Average days per year mixed pesticides								
None	346	46.9	6658	39.8	.	.		
1-90 days	278	37.7	6358	38.0	0.0361	0.7049	0.949	0.722 1.246
> 90 days	113	15.3	3697	22.1	<.0001	0.5378	1.143	0.747 1.747
Size of Farm								
None	51	17.9	1262	21.7	.	.	.	. .
< 50 acres	47	16.5	917	15.7	0.2507	0.2305	1.287	0.852 1.943
51-500 acres	90	31.6	1539	26.5	0.0393	0.0477	1.435	1.004 2.052
501-1000 acres	50	17.6	1005	17.3	0.3071	0.2669	1.258	0.839 1.887
> 1000 acres	46	16.1	1083	18.6	0.8107	0.6627	1.096	0.725 1.657

The default SAS listing for the three procedures on these two variables is 10 pages long. Using SAS ODS as outlined above, we have produced the concise summary report shown in Figure 15.

Part 2: Automating the Analysis Using Macros

Designing the automation of this process

The analysis in Part 1 was written specifically for the analysis of a single variable. Now we will write a set of general code that can be used to analyze numerous variables. The SAS Macro facility can be used 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 and then pass processing on to the next macro. Overworking a macro by including multiple tasks may make it cumbersome to review, debug and edit. Below is the list of the main tasks needed to automate the above analysis to handle numerous main independent variables. Here our goal is to create a single composite dataset containing the summary information derived from all of the independent variables.

1) Validate the user supplied information.
2) Identify base co-variables as class (categorical) or continuous variables.
3) Create an analysis dataset.
4) Produce a dataset of counts and percentages.
5) Produce a dataset of crude Chi Square probabilities and Odds Ratios.
6) Produce a dataset of adjusted Chi Square probabilities and Odds Ratios.
7) Merge these three datasets into a composite summary dataset.
8) Create a supervising macro to delegate these tasks through invocations of other macro modules.

Coding a validation module – macro VALIDATE()

This will be the largest module since exhaustive checks must be performed on the data supplied by the user to the supervising macro. When a complex macro such as this fails during compilation or execution, it is quite often due to omitted or misspecified 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) Does the response variable contain 2, and only 2, unique levels.
7) Is the independent variable of interest specified.
8) Does the independent variable exist within the user supplied dataset.
9) Is the independent variable numeric.
10) Is the referent level for the independent variable specified.
11) Is the referent level a valid value of the independent variable.
12) Do each of the base co-variables exist within the user supplied dataset.
13) Are each of the base model co-variables numeric.
14) Is the final output dataset specified.

If any of these 14 checks fail, an error message will be written to the log to identify the item that failed in the call of the supervising macro. The macro proccessing will subsequently be halted. Appendix 1 contains the code for the **VALIDATE()** macro module.

Specifying the base co-variables for the adjusted model – macro SPLIT()

Our example task includes running an adjusted PROC LOGISTIC with a pre-determined set of base co-variables. For the purposes of this paper, we require that the main independent variable be a categorical variable. The base co-variables included in the adjusted model can be either categorical data or continuous. In the logistic 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, we attach an identifying character as a prefix to the name of the continuous co-variables in the call to the supervising macro. Here, we use the "@" symbol as the control character. In the invocation to the supervising macro, we will supply the list of co-variables in the parameter *__basevs*:

```
__basevs = @age bmi_group smoke
```

The variable *age* is a continuous variable and the rest are categorical. We use a macro, **SPLIT(),** to parse this string of co-variables and save them in two macro variables named *__continuous_vars* and *__class_vars*. Each variable name with the prefix of "@" will be in *__continuous_vars*, and the variable names without the "@" prefix will be placed in *__class_vars*.

The macro **SPLIT()** takes one parameter, the list of base co-variables to include in the adjusted PROC LOGISTIC. The following line of code shows a call of this macro.

```
%split(__vars = &__basevs)
```

This invokes the following macro:

```
%MACRO split(__vars=);
```

The macro variables identifying the continuous and class variables will be used by several other macros and procedures, so they must be defined as global.

```
%GLOBAL __continuous_vars __class_vars;
```

These macro variables must be initialized to a null string so that they exist even if there are no class or continuous variables.

```
%LET __continuous_vars=;
%LET __class_vars=;
```

Loop over the variable list and separate them into continuous and class variables:

```
    %IF &__vars NE %THEN %DO;
        %LET __count=1 ;
        %DO %UNTIL (%QSCAN(&__vars,&__count)= ) ;
            %LET __var = %QSCAN(&__vars,&__count);
            %IF %SUBSTR(&__var,1,1)=@ %THEN %LET __continuous_vars=&__continuous_vars &__var;
            %ELSE %LET __class_vars =&__class_vars &__var ;
           %LET __count=%EVAL(&__count+1);
        %END ;
        %LET __continuous_vars = %SYSFUNC(COMPRESS(&__continuous_vars,@)) ;
    %END;

%MEND split;
```

The control characters "@" are removed from the list of continuous variables in the macro variable *__continuous_vars* with the code:

```
        %LET __continuous_vars = %SYSFUNC(COMPRESS(&__continuous_vars,@)) ;
```

After execution of the above macro, the variable *__class_vars* will now contain the string

```
"bmi_group smoke"
```

while the macro variable *__continuous_vars* will contain

```
"age"
```

These macro variables will be used in the adjusted PROC LOGISTIC. Both of these strings will be inserted into the MODEL statement, while only the string identified by the *__class_vars* macro variable will be written to the CLASS statement.

Creating an analysis dataset – macro MAKESET()

It is common in most 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; it is impossible to determine if differences in the estimates are due to adjustment for the co-variables or the differences in the observations included in the models. For this reason, the macro **MAKESET()** will create an analysis dataset containing only records with complete data for the variables included in the adjusted logistic regression model. The five parameters passed to this macro are:

1) the dataset to be analyzed
2) the variable to appear as the column variable in the PROC FREQ and the response variable in the logistic regressions
3) the variable to appear as the row variable in the PROC FREQ and the independent variable in the logistic regressions
4) the categorical variables for the adjusted logistic regression
5) the continuous variables for the adjusted logistic regression

```
%MACRO makeset(__dset            = ,
               __var1            = ,
               __var2            = ,
               __class_vars      = ,
               __continuous_vars =   ) ;

   DATA __analysis_set ;
     SET &__dset(KEEP = &__var1 &__var2 &__class_vars &__Continuous_vars);
     IF NMISS(OF _numeric_) = 0 THEN OUTPUT;  RUN;

%MEND makeset;
```

The code above keeps only those variables required for the analysis and removes any observations that contain missing data.

Summarizing counts and percents – macro CTAB()

The next macro, **CTAB()**, contains the execution of the PROC FREQ, which is used to obtain the summary counts and percentages. This macro module requires three parameters:

1) the analysis dataset (created in **MAKESET()**)
2) the column (response) variable
3) the row (independent exposure) variable

```
%MACRO ctab(__dset    = ,
            __col_var = ,
            __row_var =
           );
```

To reduce the output to the SAS listing, we optionally suppress output to it before running the PROC FREQ. If a record of the output is desired, PROC PRINTTO can be used instead of the ODS LISTING CLOSE statement.

```
ODS LISTING CLOSE;
```

The first part of this main task is to generate the CrossTabFreqs object and save it to an ODS Output dataset. At this point in the process, we will also trim the output dataset to contain only the needed variables, using the KEEP = statement.

```
ODS OUTPUT CrossTabFreqs = __ctab
                           (KEEP = &__col_var &__row_var _TYPE_ FREQUENCY COLPERCENT);
PROC FREQ DATA=&__dset;
  TABLES &__row_var * &__col_var;  RUN;

ODS OUTPUT CLOSE;
```

The SAS Listing destination is reset as the default output destination.

```
ODS LISTING;
```

Next we repeat the three main steps shown in Part 1 to trim the *__CTAB* dataset and prepare it for merging with other summary datasets. First, we remove unwanted observations from the *_CTAB* dataset with the WHERE option in the SET statement. Second, we combine the two remaining observations for each level of the row variable into one observation with the RETAIN and OUTPUT statements. Finally, we add descriptive information for the row and column variables.

```
DATA __freqs ( KEEP = var1_name var1_label var1_value1 var1_fmt1 var1_value2 var1_fmt2
                      var2_name var2_label var2_value var2_fmt
                      n_col1 p_col1 n_col2 p_col2 ) ;
  LENGTH var1_name var2_name  $20
         var1_label var1_fmt1 var1_fmt2 var2_label var2_fmt $40;
  SET __ctab (WHERE = (_type_="11"));
    BY &__var2 &__var1;
  RETAIN var1_value1 var1_fmt1 n_col1 p_col1;
  IF FIRST.&__var2 THEN DO;
    n_col1       = frequency;
    p_col1       = colpercent;
    var1_value1  = &__var1;
    var1_fmt1    = VVALUE(&__var1);
```

```
      END; ELSE
       IF LAST.&__var2 THEN DO;
         n_col2         = frequency;
         p_col2         = colpercent;
         var1_value2    = &__var1;
         var1_fmt2      = VVALUE(&__var1);
         var1_name      = "&__var1";
         var1_label     = VLABEL(&__var1);
         var2_value     = &__var2;
         var2_fmt       = VVALUE(&__var2);
         var2_name      = "&__var2";
         var2_label     = VLABEL(&__var2);
         OUTPUT;
       END; RUN;
%MEND ctab;
```

Producing crude and adjusted statistics – macro LOGIT()

Two logistic regression models are run to produce our summary statistics. The first model generates the crude association between the response, *&__var1*, and the independent exposure, *&__var2*. The second model gives the association between the response and independent variables, adjusted for the user supplied base co-variables. These models will be very similar so we can use separate calls to the same macro for each one of them. The following seven parameters will be passed during the invocation of this macro:

1) A parameter identifying the PROC LOGISTIC as crude or adjusted
2) the analysis dataset
3) the response variable
4) the independent exposure
5) the referent level of the independent exposure variable
6) the list of base co-variables to appear in the CLASS statement
7) the list of additional base co-variables to appear in the MODEL statement

```
%MACRO logit(__test            = ,
             __dset            = ,
             __response        = ,
             __independent     = ,
             __ref             = ,
             __class_vars      = ,
             __continuous_vars =
            );
```

Again, we close the SAS listing.

```
ODS LISTING CLOSE;
```

As we saw in Figures 6a and 6b, PROC LOGISTIC generates two objects, ParameterEstimates and OddsRatios, that contain the information we want. The ODS OUTPUT statements below save this information, as well as keep only the needed information while renaming the variables to identify them as crude or adjusted statistics.

```
ODS OUTPUT ParameterEstimates = __pe (KEEP = variable classval0 probchisq estimate StdErr
                                      RENAME = (ProbChiSq = &__test._p_value) )
           OddsRatios         = __or (KEEP = EFFECT OddsRatioEst LowerCL UpperCL
                                      RENAME = (OddsRatioEst = &__test._OddsRatio
                                                LowerCL      = &__test._LowerCL
                                                UpperCL      = &__test._UpperCL ) );
```

In the call to PROC LOGISTIC below, the independent variable is included in the CLASS statement with the referent level set to the user supplied value of *&__ref*. The base co-variables in the class statement (identified by *&__class_vars*) will be a null string in the event that this is a crude model or if all the co-variables are continuous. This allows us to include the *&__class_vars* macro variable in the CLASS statement during each execution of the **LOGIT()** macro, whether the model is crude or adjusted. As stated in Part 1, formatting is removed from the *&__independent* variable.

```
PROC LOGISTIC DATA = &__dset DESCENDING;
  CLASS &__class_vars &__independent (PARAM=REF REF="&__ref")   ;
  MODEL &__response = &__class_vars &__continuous_vars &__independent ;
  FORMAT &__independent;   RUN;
```

Next, close the ODS OUTPUT destination and reopen the SAS Listing destination.

```
ODS OUTPUT CLOSE;
ODS LISTING;
```

Now we can automate the process of trimming and combining the datasets output from the PROC LOGISTIC.

```
    DATA __pe ( KEEP = var2_value &__test._p_value );
      SET __pe (WHERE = (UPCASE(variable)="&__independent"));
      var2_value = input(classval0,8.);  RUN;

    DATA __or (KEEP = var2_value &__test._OddsRatio &__test._LowerCL &__test._UpperCL);
      LENGTH var2_value 8;
      SET __or (WHERE =( UPCASE(SCAN(EFFECT,1))= "&__independent"));
      var2_value = SCAN(EFFECT,2);    RUN;

    PROC SORT DATA=__or;
      BY var2_value;  RUN;

    PROC SORT DATA=__pe;
      BY var2_value;  RUN;

    DATA __&__test._log_stats ;
      MERGE __pe __or ;
        BY var2_value;  RUN;

  %MEND logit;
```

The above process will produce datasets identical to those found in Figures 8 and 12. Each is ready for the final merging with the *__FREQS* dataset using the key variable *var2_value*.

Merging three ODS OUTPUT datasets to create one summary dataset – macro SUMMARYSET()

The next macro, **SUMMARYSET()**, merges the three datasets of summary statistics on the key variable *var2_value*. This macro takes one parameter which is the name of the final dataset that accumulates the results for all of the analyses.

```
  %MACRO summaryset(__base=);
    PROC SORT DATA=__freqs; by var2_value; RUN;
    PROC SORT DATA=__crude_log_stats; by var2_value; RUN;
    PROC SORT DATA=__adjusted_log_stats; by var2_value; RUN;
    DATA __summary;
      MERGE __freqs __crude_log_stats __adjusted_log_stats;
      BY var2_value;   RUN;
```

Once the *__SUMMARY* dataset has been created, it is appended to the final composite dataset, *&__base*.

```
    PROC APPEND BASE=&__base DATA=__summary FORCE;  RUN;
```

Last, the datasets created and named by this program must be removed to prevent potential contamination of future runs by the program:

```
    PROC DATASETS NOLIST;
      DELETE __analysis_set
             __ctab
             __freqs
             __pe
             __or
             __crude_log_stats
             __adjusted_log_stats
             __summary;
    RUN;
    QUIT;
  %MEND summaryset;
```

A supervising macro to invoke each of the macro modules – macro SUMMARY()

We now have six macros to automate the major tasks for the analysis. Since these modules must be executed in proper order, we create a supervising macro that will pass the correct parameters to each module in order. In this example, the **SUMMARY()** macro acts as the supervising 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 column/response variable
3) The row/independent variable
4) The referent level of the independent variable
5) The base co-variables to appear in the CLASS and MODEL statements
6) The name of the composite dataset which will contain the accumulated statistics

```
%MACRO summary(__dset   = ,
               __var1   = ,
               __var2   = ,
               __ref    = ,
               __basevs = ,
               __out    = );
```

To insure proper processing, force the names of the row and column variables to caps.

```
%LET __var1 = %UPCASE(&__var1);
%LET __var2 = %UPCASE(&__var2);
```

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

1) The dataset to be analyzed
2) The column/response variable
3) The row/independent variable
4) The list of co-variables
5) The output dataset

```
%validate(__dset = &__dset,
          __var1 = &__var1,
          __var2 = &__var2
          __vars = &__basevs,
          __out  = &__out  );
```

The second macro, **SPLIT(),** separates the base co-variables into class and continuous variables. These class and continuous base co-variables will be used later in the **MAKESET()** and **LOGIT()** macro modules.

```
%split(__vars=&__basevs);
```

The third macro called is **MAKESET()**, which creates the analysis dataset:

```
%makeset(__dset           = &__dset ,
         __var1           = &__var1 ,
         __var2           = &__var2 ,
         __class_vars     = &__class_vars,
         __continuous_vars = &__continuous_vars );
```

Fourth is the **CTAB()** macro, which creates our first summary dataset, *__FREQS*. This macro takes three parameters:

```
%ctab(__dset     = __analysis_set ,
      __col_var  = &__var1 ,
      __row_var  = &__var2 );
```

Our fifth and sixth macro invocations both call the **LOGIT()** macro. The first invocation of the **LOGIT()** macro module runs the crude logistic regression and creates our second summary dataset, *__CRUDE_LOG_STATS*. The *__class_vars* and *__continuous_vars* parameters do not appear in the first invocation of the **LOGIT()** macro module since they are null values.

```
%logit(__test           = Crude,
       __dset           = __analysis_set,
       __response       = &__var1,
       __independent    = &__var2,
       __ref            = &__ref);
```

The second invocation of the **LOGIT()** macro runs the multiple logistic regression and creates the dataset, __ADJUSTED_LOG_STATS. In this invocation of the **LOGIT()** macro, the *__class_vars* and *__continuous_vars* parameters are used to pass the names of the co-variables for this model.

```
%logit(__test           = Adjusted,
       __dset           = __analysis_set,
       __response       = &__var1,
       __independent    = &__var2,
       __ref            = &__ref,
       __class_vars     = &__class_vars,
       __continuous_vars = &__continuous_vars );
```

The seventh macro called is **SUMMARYSET()**. This module assembles the final dataset of summary statistics. First, it merges the three datasets into one summary dataset containing all of the statistics generated to this point. Second, it appends this dataset to all other summary datasets created in previous invocations of the **SUMMARY()** macro. By doing this, associations for multiple sets of variables can be summarized in one summary report:

```
    %summaryset(__base = &__out);

  %MEND summary;
```

Two calls to **SUMMARY()** produce the dataset used to create the results shown in Figure 15 .

```
%summary(__dset   = mydata,
         __var1   = asthma,
         __var2   = DaysMix,
         __ref    = 1,
         __basevs = @age bmi_group smoke,
         __out    = log_summary);

%summary(__dset   = mydata,
         __var1   = asthma,
         __var2   = farmsize,
         __ref    = 1,
         __basevs = @age bmi_group smoke,
         __out    = log_summary);
```

Appendix 1 contains the code for all of the macro modules discussed in Part 2 of this paper.

Appendices 2 and 3 contain an example of code that automates the iteration of this process. The supervising macro, **SUMMARY()**, has been adjusted to allow multiple independent variables to be specified by the user. The **VALIDATE()** macro module must be adjusted to validate multiple independent variables, as well as multiple referent levels in the example in Appendix 3. The supervising macro will add the summary statistics for each iteration to the composite dataset specified by *&__out*. An option has been added for the user to request that a summary report be printed using the final composite summary dataset. The code for the **PRINTREPORT()** macro module is not included with this paper since numerous coding options exist for facilitating the creation of this report.

Appendix 2 shows how this macro could be coded if the referent levels for all of the independent variables are identical (e.g. __ref =1). The **SUMMARY()** macro iterates all macro modules, except the **VALIDATE()** and **SPLIT()** macros, for each independent variable that appears in the *__var2_list* parameter. The following call to the **SUMMARY()** supervising macro as seen in Appendix 2, will generate the report found in Figure 15.

```
%summary(__dset      = mydata,
         __var1      = asthma,
         __var2_list = daysmix farmsize,
         __ref       = 1,
         __basevs    = @age bmi_group smoke,
         __out       = log_summary,
         __report    = YES);
```

Appendix 3 shows a slight modification of the iteration process in Appendix 2. In this version of the supervising macro, the *__ref* parameter is eliminated. The user will supply a list of pairs of items through the *__var2_list* parameter. These pairs represent the independent variables and their corresponding referent levels, separated by an "=" sign. The two items of each pair are split before the iteration process occurs in the %DO %UNTIL statement. This will provide a method of assigning the appropriate referent level to each independent variable, if the referent levels vary. The following call to the **SUMMARY()** supervising macro as seen in Appendix 3, will generate the report found in Figure 15.

```
%summary(__dset      = mydata,
         __var1      = asthma,
         __var2_list = daysmix=1 farmsize=1,
         __basevs    = @age bmi_group smoke,
         __out       = log_summary,
         __report    = YES);
```

Iteration can include multiple datasets, response variables and sets of base co-variables. There are many methods that a programmer can use to code this process for iteration. Programmers should use their own programming preferences. Additional methods have been published by the authors for automating macro iteration[1].

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 summarization that encompasses the entirety of procedural output. The methods presented in Part 2 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.

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.*

CONTACT INFORMATION

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

APPENDIX 1: The Code for the Macro Modules

```
%MACRO validate(__dset = ,
                __var1 = ,
                __var2 = ,
                __ref  = ,
                __vars = ,
                __out  = );
  %GLOBAL _echeck ;
  %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;

    %* check that response variable has 2 levels *;
    proc sql noprint ;
       select count(distinct(&__var1)) into :rscount
       from &__dset
       where &__var1 is not missing ;;
    quit ;
    %if &rscount ne 2 %then %do ;
      %let _echeck=1 ;
      %put Summary-> THE RESPONSE VARIABLE &__var1 DOES NOT HAVE 2 LEVELS Execution
terminating. ;
    %end;

    %* check that response variable values are 0 and 1 *;
    %else %do ;
      proc sql noprint ;
         select distinct(&__var1) format=best.  into :rsvalues separated by ' '
        from &__dset
        where &__var1 is not missing ;
     quit ;
     %if %scan(&rsvalues,1) ne 0 or %scan(&rsvalues,2) ne 1 %then
       %put WARNING: Values for response variable &__var1 are not 0/1  ;
    %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 __REF macro variable specified *;
   %if &__ref. = %str( ) %then %do;
      %let _echeck=1 ;
      %put Summary-> THE __REF PARAMETER was not specified. Execution terminating. ;
   %end;

   %* check that value for __REF is valid *;
   %else %if &_indflag = 0 %then %do;
      proc sql noprint ;
         select distinct(&__var2) format=best. into :rfvalues separated by ' '
        from &__dset
        where &__var2 is not missing ;
      quit ;
      %if %sysfunc(indexw(&rfvalues.,&__ref.)) = 0 %then %do;
         %let _echeck=1 ;
         %put Summary-> THE VALUE SPECIFIED FOR THE __REF PARAMETER [ &__ref ] is not valid.
Execution terminating. ;
         %put Summary-> Valid values are &rfvalues ;
      %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 split(__vars=);

  %GLOBAL __continuous_vars __class_vars;

  %LET __continuous_vars=;
  %LET __class_vars=;
  %IF &__vars NE %THEN %DO;
      %LET __count=1 ;
      %DO %UNTIL (%QSCAN(&__vars,&__count)= ) ;
          %LET __var = %QSCAN(&__vars,&__count);
          %IF %SUBSTR(&__var,1,1)=@ %THEN %LET __continuous_vars=&__continuous_vars &__var ;
          %ELSE %LET __class_vars =&__class_vars &__var ;
         %LET __count = %EVAL(&__count+1);
      %END ;
      %LET __continuous_vars = %SYSFUNC(COMPRESS(&__continuous_vars,@)) ;
  %END;
%MEND split;

%MACRO makeset(__dset           = ,
               __var1           = ,
               __var2           = ,
               __class_vars     = ,
               __continuous_vars =   ) ;

   DATA __analysis_set ;
     SET &__dset(KEEP = &__var1 &__var2 &__class_vars &__Continuous_vars);
     IF NMISS(OF _numeric_) = 0 ; RUN;
%MEND makeset;

%MACRO ctab(__dset     = ,
            __col_var  = ,
            __row_var  = );

  ODS LISTING CLOSE;
```

```
  ODS OUTPUT CrossTabFreqs = __ctab
                             (KEEP = &__col_var &__row_var _TYPE_ FREQUENCY COLPERCENT);

  PROC FREQ DATA=&__dset;
    TABLES &__row_var * &__col_var; RUN;

  ODS OUTPUT CLOSE;

  ODS LISTING;

  DATA __freqs ( KEEP = var1_name var1_label var1_value1 var1_fmt1 var1_value2 var1_fmt2
                        var2_name var2_label var2_value var2_fmt
                        n_col1 p_col1 n_col2 p_col2 ) ;
      LENGTH var1_name var2_name  $20
             var1_label var1_fmt1 var1_fmt2 var2_label var2_fmt $40;
      SET __ctab (WHERE = (_type_="11"));
        BY &__var2 &__var1;
      RETAIN var1_value1 var1_fmt1 n_col1 p_col1;
      IF FIRST. &__var2 THEN DO;
        n_col1       = frequency;
        p_col1       = colpercent;
        var1_value1  = &__var1;
        var1_fmt1    = VVALUE(&__var1);
     END; ELSE
      IF LAST. &__var2 THEN DO;
        n_col2         = frequency;
        p_col2         = colpercent;
        var1_value2    = &__var1;
        var1_fmt2      = VVALUE(&__var1);
        var1_name      = "&__var1";
        var1_label     = VLABEL(&__var1);
        var2_value     = &__var2;
        var2_fmt       = VVALUE(&__var2);
        var2_name      = "&__var2";
        var2_label     = VLABEL(&__var2);
        OUTPUT;
      END;
      LABEL var1_name     = "Name of Response Variable"
            var1_label    = "Label of Response Variable"
            var1_value1   = "Value for Level 1 of Response variable"
            var1_fmt1     = "Formatted value for Level 1 of Response variable"
            var1_value2   = "Value for Level 2 of Response variable"
            var1_fmt2     = "Formatted value for Level 2 of Response variable"
            var2_name     = "Name of Independent Variable"
            var2_label    = "Label of Independent Variable"
            var2_value    = "Value of Independent Variable"
            var2_fmt      = "Formatted value for Independent variable"
            n_col1        = "PROC FREQ Frequency of Column 1"
            n_col2        = "PROC FREQ Frequency of Column 2"
            p_col1        = "PROC FREQ Percent of Column 1"
            p_col2        = "PROC FREQ Percent of Column 2"; RUN;
%MEND ctab;


%MACRO logit(__test            = ,
             __dset            = ,
             __response        = ,
             __independent     = ,
             __ref             = ,
             __class_vars      = ,
             __continuous_vars =
            );

  ODS LISTING CLOSE;

  ODS OUTPUT ParameterEstimates = __pe (KEEP = variable classval0 probchisq estimate StdErr
                                        RENAME = (ProbChiSq = &__test._p_value) )
             OddsRatios         = __or (KEEP = EFFECT OddsRatioEst LowerCL UpperCL
                                        RENAME = ( OddsRatioEst = &__test._OddsRatio
                                                   LowerCL      = &__test._LowerCL
                                                   UpperCL      = &__test._UpperCL ) );
```

```
  PROC LOGISTIC DATA = &__dset DESCENDING;
    CLASS &__class_vars &__independent (PARAM=REF REF="&__ref")   ;
    MODEL &__response = &__class_vars &__continuous_vars &__independent ;
    FORMAT &__independent;  RUN;

  ODS OUTPUT CLOSE;
  ODS LISTING;

  DATA __pe ( KEEP = var2_value &__test._p_value );
    SET __pe (WHERE = (UPCASE(variable)="&__independent"));
    var2_value          = input(classval0,8.); RUN;

  DATA __or (KEEP = var2_value &__test._OddsRatio &__test._LowerCL &__test._UpperCL);
    LENGTH var2_value 8;;
    SET __or (WHERE =( UPCASE(SCAN(EFFECT,1))= "&__independent"));
    var2_value = input((SCAN(EFFECT,2)),2.);   RUN;

  PROC SORT DATA=__or;
    BY var2_value; RUN;

  PROC SORT DATA=__pe;
    BY var2_value; RUN;

  DATA &__test._log_stats ;
    %IF &__test = Adjusted %THEN %DO;
      LENGTH class_vars continuous_vars $80;
    %END;
    MERGE __pe __or ;
      BY var2_value;
    %IF &__test = Adjusted %THEN %DO;
      class_vars      = “&__class_vars”;
      continuous_vars = “&__continuous_vars”;
    %END;
    LABEL  &__test._p_value   = "&__test PROC LOGISTIC Chi-Square probability"
           &__test._OddsRatio = "&__test PROC LOGISTIC Odds Ratio Estimate(OR)"
           &__test._LowerCL   = "&__test PROC LOGISTIC Lower 95% Confidence Limit for OR"
           &__test._UpperCL   = "&__test PROC LOGISTIC Upper 95% Confidence Limit for OR"
    %IF &__test = Adjusted %THEN %DO;
           class_vars         = “Co-variables appearing in the CLASS STATEMENT”
           continuous_vars    = “Co-variables not appearing in the CLASS STATEMENT”
    %END;
    ;
  RUN;
%MEND logit;

%MACRO summaryset(__base=);
  DATA __summary;
    MERGE __freqs crude_log_stats adjusted_log_stats;
      BY var2_value;   RUN;

  PROC APPEND BASE=&__base DATA=__summary FORCE;  RUN;

  PROC DATASETS NOLIST;
    DELETE __analysis_set
           __ctab
           __freqs
           __pe
           __crude_log_stats
           __adjusted_log_stats
           __summary;
  RUN;QUIT;
%MEND summaryset;

%MACRO summary(__dset   = ,
               __var1   = ,
               __var2   = ,
               __ref    = ,
               __basevs = ,
               __out    = );
```

```
    %LET __var1 = %UPCASE(&__var1);
    %LET __var2 = %UPCASE(&__var2);

    %validate(__dset = &__dset ,
              __var1 = &__var1 ,
              __var2 = &__var2 ,
              __ref  = &__ref  ,
              __vars = &__basevs,
              __out  = &__out  )

    %IF &_echeck=1 %THEN %GOTO nodata;

    %split(__vars=&__basevs)

    %makeset(__dset            = &__dset ,
             __var1            = &__var1 ,
             __var2            = &__var2 ,
             __class_vars      = &__class_vars,
             __continuous_vars = &__continuous_vars )

    %ctab(__dset    = __analysis_set,
          __col_var = &__var1 ,
          __row_var = &__var2 )

    %logit(__test            = Crude,
           __dset            = __analysis_set,
           __response        = &__var1,
           __independent     = &__var2,
           __ref             = &__ref )

    %logit(__test            = Adjusted,
           __dset            = __analysis_set,
           __response        = &__var1,
           __independent     = &__var2,
           __ref             = &__ref,
           __class_vars      = &__class_vars,
           __continuous_vars = &__continuous_vars )

    %summaryset(__base = &__out)

    %nodata:

  %MEND summary;
```

APPENDIX 2: Iteration for multiple independent variables

```
%MACRO summary(__dset      = ,
               __var1      = ,
               __var2_list = ,
               __ref       = ,
               __basevs    = ,
               __out       = ,
               __report    = );

  %LET __var1 = %UPCASE(&__var1);
  %LET __var2 = %UPCASE(&__var2);

  %IF &__echeck=1 %THEN %GOTO nodata;

  %split(__vars=&__basevs)

  %LET i_count =1 ;
  %DO %UNTIL (%SCAN(&__var2_list,&i_count)= ) ;
    %LET __var2=%SCAN(&__var2_list,&i_count);

    %makeset(<same as in Appendix 1>)
    %ctab(<same as in Appendix 1>)
    %logit(<same as in Appendix 1>)
    %logit(<same as in Appendix 1>)
```

```
    %summaryset(<same as in Appendix 1>)
    %LET i_count=%EVAL(&i_count+1);

  %END;

  %IF %UPCASE(&__report) = YES %THEN %PrintReport(__dset=&__out);

  %nodata:

%MEND summary;
```

APPENDIX 3: Iteration for multiple independent variables with corresponding referent levels.

```
%MACRO summary(__dset      = ,
               __var1      = ,
               __var2_list = ,
               __basevs    = ,
               __out       = ,
               __report    = );

  %LET __var1 = %UPCASE(&__var1);
  %LET __var2 = %UPCASE(&__var2);

  %validate(__dset = &__dset ,
            __var1 = &__var1 ,
            __var2 = &__var2_list ,
            __vars = &__basevs  )

  %IF &__echeck=1 %THEN %GOTO nodata;

  %split(__vars=&__basevs)

  %LET i_count =1 ;
  %DO %UNTIL (%SCAN(&__var2_list,&i_count)= ) ;
    %LET __var2=%SCAN(%SCAN(&__var2_list,&i_count),1,=);
    %LET __ref=%SCAN(%SCAN(&__var2_list,&i_count),2,=);

    %makeset(<same as in Appendix 1>)
    %ctab(<same as in Appendix 1>)
    %logit(<same as in Appendix 1>)
    %logit(<same as in Appendix 1>)
    %summaryset(<same as in Appendix 1>)
    %LET i_count=%EVAL(&i_count+1);

  %END;

  %IF %UPCASE(&__report) = YES %THEN %PrintReport(__dset=&__out);

  %nodata:

%MEND summary;
```