

Paper 252-31

FREQ n' MEANSMarge Scerbo, CHPDM/UMBC
Mic Lajiness, Eli Lilly and Company**Abstract**

Procs **FREQ** and **MEANS** have been part of SAS® for over 30 years and are probably some of the most used and loved procedures. They provide useful information directly and as input to other routines and procedures and are easy to use, so people run them daily without thinking much about them. These procedures can facilitate the construction of many useful statistics and data views that are not readily evident from the documentation. This paper will discuss different forms of the procedures and provide you with ideas on where they may be most useful in your organization and should expand your horizons with respect to these two procedures.

Introduction

Proc FREQ and **Proc MEANS** are found in the Base SAS product. They have many uses, both statistical and descriptive. To start, it would be useful to review the descriptions of these procedures found in the SAS Procedures Guide:

- The frequency (**FREQ**) procedure produces one-way to n -way frequency and cross-tabulation tables. For two-way tables, **Proc FREQ** computes tests and measures of association. For n -way tables, **Proc FREQ** does stratified analysis, computing statistics within, as well as across, strata. Frequencies can also be output to a SAS dataset.
- The **MEANS** procedure produces simple univariate descriptive statistics for numeric variables. You can use the **OUTPUT** statement to request that **MEANS** output statistics to a SAS dataset.

As with many procedures, there are basic requirements:

- In both cases, these procedures require a dataset to analyze. The dataset can be specifically named in the **data =** option or can default to the last created dataset. Best practices tell us to always name the dataset.
- **Proc MEANS** requires at least one numeric variable, while **Proc FREQ** has no such limitation.

Let's look at what happens when no variables are specified. These first several examples will use a dataset called *colleges*. Here are the essentials of this dataset. It has 1,302 observations and 8 variables. These variables are:

Variable Name	Type	Length	Description
name	Char	45	College Name
state	Char	4	State
sponsor	Num	8	State School indicator
verbsat	Num	8	Verbal SAT Score
mathsat	Num	8	Math SAT Score
act	Num	8	ACT Score
intuit	Num	8	In State Tuition
outtuit	Num	8	Out of State Tuition

So let's first run these procedures without any specifications beyond the dataset name as in:

```
proc freq data = colleges;
run;
proc mean data = colleges;
run;
```

The **FREQ** procedure will produce as many one-way tables as there are variables in the dataset. Each table will include a count (frequency) of each unique value for each variable. The code above produces more than 80 pages of output since all values of all variables, regardless of type, will be included. Here is a small portion of the frequency listing for the variable *state*. This is the default output that would be created for all variables:

The FREQ Procedure				
state	Frequency	Percent	Cumulative Frequency	Cumulative Percent
AK	4	0.31	4	0.31
AL	25	1.92	29	2.23

AR	17	1.31	46	3.53
AZ	5	0.38	51	3.92
CA	70	5.38	121	9.29
CO	16	1.23	137	10.52
CT	19	1.46	156	11.98
DC	9	0.69	165	12.67
DE	5	0.38	170	13.06
FL	30	2.30	200	15.36
GA	36	2.76	236	18.13
HI	5	0.38	241	18.51
IA	29	2.23	270	20.74
ID	6	0.46	276	21.20
IL	49	3.76	325	24.96
IN	42	3.23	367	28.19
KS	20	1.54	387	29.72
KY	24	1.84	411	31.57
LA	20	1.54	431	33.10
MA	56	4.30	487	37.40
MD	23	1.77	510	39.17

The columns displayed are the values for *state*, a count of the Frequency, the Percent of this value (count of variable value/total non-missing observations), the Cumulative Frequency, and the Cumulative Percent.

The **MEANS** procedure produces description statistics for each numeric variable, and so there will be no output for the variables *name* or *state*. The default output or listing of this procedure looks like:

The MEANS Procedure

Variable	Label	N	Mean	Std Dev	Minimum	Maximum

sponsor		1302	1.6390169	0.4804702	1.0000000	2.0000000
verbsat	Verbal SAT	777	461.2239382	58.2984129	280.0000000	665.0000000
mathsat	Math SAT	777	506.8378378	67.8224407	320.0000000	750.0000000
act		714	22.1204482	2.5798990	11.0000000	31.0000000
intuit		1272	7897.27	5348.16	480.0000000	25750.00
outtuit		1280	9283.25	4170.02	1044.00	25750.00

There are a variety of ways that one can use **Proc FREQ** and **Proc MEANS** to summarize and display useful information. First, this paper will discuss **Proc FREQ** followed with a section dedicated to **Proc MEANS**. We will conclude with a section of how one might use the two procedures together to obtain descriptive information not directly attainable otherwise.

Proc FREQ

In addition to using the basic syntax shown above (`proc freq data = colleges;`), there are a number of options and statements associated with the procedure.

```
proc freq <options> ;
  by <variables> ;
  exact <statistic-options> <| computation-options> ;
  output <OUT= dataset> options ;
  tables <requests> <|options> ;
  test <options> ;
  weight <variable> ;
```

These options are described as follows:

BY	calculates separate frequency or crosstabulation tables for each BY group
EXACT	requests exact tests for specified statistics
OUTPUT	creates an output dataset that contains specified statistics
TABLES	specifies frequency or crosstabulation tables and requests tests and measures of association
TEST	requests asymptotic tests for measures of association and agreement

WEIGHT identifies a variable with values that weight each observation

Of these statements, **tables** is probably the most used and most useful. It is used to obtain frequency and cross-tabulation tables. Typically, the **tables** statement is used in conjunction with classification variables. For example, in the *colleges* dataset, one can generate a frequency table of *state* by *sponsor* with the following code.

```
proc freq data = colleges;
    tables state * sponsor;
run;
```

Which gives the following results (partial):

Table of state by sponsor

state	sponsor		Total
Frequency			
Percent			
Row Pct			
Col Pct	1	2	
AK	3	1	4
	0.23	0.08	0.31
	75.00	25.00	
	0.64	0.12	
AL	13	12	25
	1.00	0.92	1.92
	52.00	48.00	
	2.77	1.44	
AR	9	8	17
	0.69	0.61	1.31
	52.94	47.06	
	1.91	0.96	
Total	470	832	1302
	36.10	63.90	100.00

(Continued)

The **tables** statement can be particularly useful in seeing how the distribution of one variable changes in relation to another. What you typically do **NOT** want to do is to use the **tables** statement with continuous or integer numeric variables (for example, **tables** *intuit* * *mathsat*), because strata will be generated for every unique value. In the *colleges* example, you would produce literally hundreds of categories for each value of in-state tuition and math SAT score. What you **DO** want to do is transform the numeric variables into categorical variables. **Proc FORMAT** is a useful way to accomplish this as can be seen in the following example code.

```
proc format;
    value tuit_fmt      0-4999      = '<$5K'
                      5000-9999    = '$5K-$10K'
                      10000-high   = '>$10K';
    value sat_fmt      0-499        = '<500'
                      500-599      = '500-600'
                      600-high     = '>600';
run;
proc freq data = colleges;
    tables intuit * mathsat ;
    format intuit  tuit_fmt. ;
    format mathsat sat_fmt. ;
run;
```

which produces the output below:

Table of intuit by mathsat

intuit	mathsat(Math SAT)			
Frequency	<500	500-600	>600	Total
Percent				
Row Pct				
Col Pct				
<\$5K	137	96	12	245
	18.10	12.68	1.59	32.36
	55.92	39.18	4.90	
	36.73	30.09	18.46	
\$5K-\$10K	135	46	2	183
	17.83	6.08	0.26	24.17
	73.77	25.14	1.09	
	36.19	14.42	3.08	
>\$10K	101	177	51	329
	13.34	23.38	6.74	43.46
	30.70	53.80	15.50	
	27.08	55.49	78.46	
Total	373	319	65	757
	49.27	42.14	8.59	100.00

Frequency Missing = 545

Often, one does not need all the row, column and cell statistics normally produced by the **FREQ** procedure, and it is simple to suppress them using the **norow**, **nocol**, and **nopercent** options as can be seen below.

```
proc freq data = colleges;
  tables intuit * mathsat /norow nocol nopercent;
  format intuit tuit_fmt. ;
  format mathsat sat_fmt. ;
run;
```

which produces the table:

Table of intuit by mathsat

intuit	mathsat(Math SAT)			
Frequency	<500	500-600	>600	Total
<\$5K	137	96	12	245
\$5K-\$10K	135	46	2	183
>\$10K	101	177	51	329
Total	373	319	65	757

Frequency Missing = 545

Sometimes, it is desirable to expand the variable descriptions to make the table more presentable. One can do this with **label** statements as shown below.

```

proc freq data = colleges;
  tables intuit * mathsat ;
  format intuit  tuit_fmt. ;
  format mathsat  sat_fmt. ;
  label intuit = 'In-state Tuition';
  label mathsat = 'Math SAT score';
run;

```

which will produce the results:

Table of intuit by mathsat

intuit(In-state Tuition)		mathsat(Math SAT score)			Total
Frequency	<500	500-600	>600		
<\$5K	137	96	12	245	
\$5K-\$10K	135	46	2	183	
>\$10K	101	177	51	329	
Total	373	319	65	757	

Frequency Missing = 545

As one can see in the above tables, the frequency missing (545) is reported. This is the number of missing values encountered in formulating the cross-tabulation table. One can include these counts in the table using the **missing** option as in the following.

```

proc freq data = colleges;
  tables intuit * mathsat /norow nocol nopercnt missing;
  format intuit  tuit_fmt. ;
  format mathsat  sat_fmt. ;
  label intuit = 'In-state Tuition';
  label mathsat = 'Math SAT score';
run;

```

which produces the table:

Table of intuit by mathsat

intuit(In-state Tuition)		mathsat(Math SAT:			Total
Frequency	.	<500	500-600	>600	
.	10	7	7	6	30
<\$5K	239	137	96	12	484
\$5K-\$10K	147	135	46	2	330
>\$10K	129	101	177	51	458
Total	525	380	326	71	1302

One can also produce *n*-way tables in a natural way using the **tables** option as shown below.

```
proc freq data = colleges;
    tables state * sponsor * intuit * mathsat
        /norow nocol nopercnt missing;
run;
```

This will produce individual tables of *intuit*mathsat* for each combination of *state* and *sponsor* (not shown).

By adding the **by** statement to the frequency procedure, the report will be paged by each value of the by variable. Remember when using the **by** statement, the dataset must be presorted by the same variable.

```
proc sort data = colleges out = college1;
    by sponsor;
run;
proc freq data = college1;
    by sponsor;
    tables state;
run;
```

produces the tables shown below (each page is incomplete):

```
----- sponsor=1 -----
```

state	Frequency	Percent	Cumulative Frequency	Cumulative Percent
AK	3	0.64	3	0.64
AL	13	2.77	16	3.40
AR	9	1.91	25	5.32
AZ	3	0.64	28	5.96
CA	28	5.96	56	11.91
CO	12	2.55	68	14.47

```
----- sponsor=2 -----
```

state	Frequency	Percent	Cumulative Frequency	Cumulative Percent
AK	1	0.12	1	0.12
AL	12	1.44	13	1.56
AR	8	0.96	21	2.52
AZ	2	0.24	23	2.76
CA	42	5.05	65	7.81
CO	4	0.48	69	8.29

OUT= Datasets

Often one is not interested in obtaining a frequency table report but simply wants the output data for further processing. Thus, it can be very useful to store the results from the SAS procedure in a SAS dataset. To do so within the context of our example, we need to make two changes. The first is to add the **noprint** option to the **Proc FREQ** statement. This tells the system suppress print of the frequency table. **Out** = names the dataset to be created. These are shown below along with the **PRINT** procedure to display the results.

```
proc format;
    value tuit_fmt      0-4999      = '<$5K'
                      5000-9999   = '$5K-$10K'
                      10000-high  = '>$10K';
    value sat_fmt      0-499       = '<500'
                      500-599     = '500-600'
                      600-high    = '>600';
run;
```

```

proc freq data = colleges noprint;
    tables intuit * mathsat / norow nocol nopercnt
        out = outfreq;
    format intuit  tuit_fmt. ;
    format mathsat  sat_fmt. ;
    label intuit = 'In-state Tuition';
    label mathsat = 'Math SAT';
run;
proc print data = outfreq;
run;

```

This gives the results below.

Obs	intuit	mathsat	COUNT	PERCENT
1	.	.	10	.
2	.	<500	7	.
3	.	500-600	7	.
4	.	>600	6	.
5	<\$5K	.	239	.
6	<\$5K	<500	137	18.0978
7	<\$5K	500-600	96	12.6816
8	<\$5K	>600	12	1.5852
9	\$5K-\$10K	.	147	.
10	\$5K-\$10K	<500	135	17.8336
11	\$5K-\$10K	500-600	46	6.0766
12	\$5K-\$10K	>600	2	0.2642
13	>\$10K	.	129	.
14	>\$10K	<500	101	13.3421
15	>\$10K	500-600	177	23.3818
16	>\$10K	>600	51	6.7371

Later in this paper we will show how one can process the **out** = dataset with the **MEANS** procedure to generate some useful summarizations.

List and Order Options

Although producing frequency tables on variables with numerous values is not often very valuable, there are times that it is necessary to identify those values that occur most often. The frequency procedure provides options to simplify this process.

The **order** = option orders the values of the frequency and crosstabulation table variables according to the specified order, where:

data	orders values according to their order in the input dataset
formatted	orders values by their formatted values
freq	orders values by descending frequency count
internal	orders values by their unformatted values

For example, a SAS dataset of physician claims (*physician04*) for a year period has over 5 million observations. You are requested to identify the top 10 procedures in that period. To accomplish this, the **order** = **freq** option outputs the variables in descending count order. Additionally, the **out** = option on the **tables** statement creates an output dataset and the **noprint** option suppresses printing. Using the **obs** = option in the **Proc PRINT** statement will print out the first 10 observations. So the following code:

```

proc freq data = physician04 order = freq;
    tables procedures/noprint out = proccount;
run;
proc print data = proccount (obs = 10);
    var procedure count;
run;

```

produces:

OBS	PROCEDURE	COUNT
1	99213	699446
2	99214	216552
3	99212	168751
4	90471	121009
5	99283	113605
6	99391	95349
7	90472	84431
8	90472	81397
9	W9993	77603
10	90700	65448

If the procedure were run without any parameters, there would have been 129 pages of output, with the first 58 containing counts of greater than 10 and the rest of the pages with counts of less than 10.

The **list** option is also very useful in cases where an n -way table is produced but only the frequency counts and percents are required.

```
proc freq data = colleges noprint;
    tables sponsor * state/ list out = newlist;
run;
proc print data = newlist;
run;
```

which produces the following output (partial):

Obs	sponsor	state	count	percent
1	1	AK	3	0.23041
2	1	AL	13	0.99846
3	1	AR	9	0.69124
4	1	AZ	3	0.23041
5	1	CA	28	2.15054
6	1	CO	12	0.92166
8	1	DC	1	0.07680
9	1	DE	1	0.07680
10	1	FL	9	0.69124

Counting Missing and non-Missing Values

It is often useful to find out the number of missing and non-missing data in your data set. Towards this end one can use **Proc FORMAT** and a little trick (thanks to Ron Cody for this!) as can be seen in the following example. Note: this can also be applied to character variables as well substituting \$misscnt for misscnt and `_character_` for `_numeric_`.

```
proc format;
    value misscnt ' ' = 'Missing'
                other = 'Nonmissing';
run;
proc freq data = colleges;
    tables _numeric_ / missing nocum nopercnt;
    format _numeric_ misscnt.;
run;
```

Which gives the output below:

```

                                The FREQ Procedure

                                sponsor    Frequency
                                -----
                                Nonmissing    1302
```



```

          verbsat   Frequency
-----
Missing           525
Nonmissing        777

          mathsat   Frequency
-----
Missing           525
Nonmissing        777

          act       Frequency
-----
Missing           588
Nonmissing        714

          intuit    Frequency
-----
Missing           30
Nonmissing       1272

          outtuit   Frequency
-----
Missing           22
Nonmissing       1280

```

Another Little Trick

In the code copied below, there are two sections of code terminated by **RUN;** statements. Segregating sections in this way allows one to selectively run independent sections of code facilitating the debugging of the entire program. The way to this technique is to highlight the section of code to be run in the editor and then submit the code (using F3 for example). Thus, the **Proc FREQ** statement (bolded) is run after F3 is pressed but NOT the **Proc PRINT** statement.

```

proc freq data = colleges noprint;
    tables sponsor * state/ list noprint out = newlist;
run;
proc print data = newlist;
run;

```

This trick is very useful when writing and testing moderate to large SAS programs.

Proc FREQ Statistics

There are a variety of statistics available in **Proc FREQ** that can be useful in analyzing the relationship among variables. For two-way tables, **Proc FREQ** computes tests and measures of association. For *n*-way tables it performs stratified analysis, computing statistics within, and across strata. One should be aware, however that as versions change, new statistics and methods are added. So, it is useful to check the online help documentation and the SAS website for information and examples on new statistics. For example, http://support.sas.com/techsup/sample/sample_library.html contains a variety of sample programs that utilize some of the newest options.

Some of the statistics available through the **FREQ** procedure include:

- Chisq** provides chi-square tests of independence of each stratum and computes measures of association. The tests include Pearson chi-square, likelihood ratio chi-square, and the Mantel-Haenszel Chi-square. The measures include the phi coefficient, the continuity coefficient, Cramer's V and Fishers exact test for 2x2 tables.
- Cmh** computes the Cochran-Mantel-Haenszel statistics which test for association between the row and column variable.
- Exact** performs Fisher's exact test for tables that are larger than 2x2. For 2x2 tables the CHISQ option provides the exact test.
- Agree** performs tests for agreement between two different classifications in a two-way table. This statistic can be used with the TEST option to test hypotheses. Specifically it computes McNemar's test for 2 x 2 tables. McNemar's test is appropriate when you are analyzing data from matched pairs of subjects with a dichotomous (yes-no) response. It tests the null hypothesis of marginal homogeneity.

For the purpose of this beginning tutorial, we will focus on only some of the more common statistics. For example, by adding the **chisq** option, one can obtain a test of independence between in-state (*intuit*) tuition and math SAT scores (*mathsat*). A significant Chi Square ($p < .001$, for example) indicates that there is a strong dependence between in-state tuition and math SAT scores. The code to do this appears below.

```
proc freq data = colleges;
    tables intuit * mathsat /norow nocol nopercnt chisq;
run;
```

This produces the table seen previously in addition to the following statistics.

Statistics for Table of intuit by mathsat

Statistic	DF	Value	Prob
Chi-Square	4	105.3677	<.0001
Likelihood Ratio Chi-Square	4	111.5585	<.0001
Mantel-Haenszel Chi-Square	1	52.3315	<.0001
Phi Coefficient		0.3731	
Contingency Coefficient		0.3495	
Cramer's V		0.2638	

Effective Sample Size = 757

Frequency Missing = 545

WARNING: 42% of the data are missing.

Changes between V7 and V8 in the FREQ procedure

For those of you that are still using V7, there are a few changes that have occurred in V8. The **FREQ** procedure now includes a **test** statement that provides asymptotic tests for selected measures of association and measures of agreement. A new **binomial** option in the **tables** statement computes the binomial proportion for one-way tables. You can compute the confidence bounds of a one-way table and request the test that the proportion equals a specified value as well as produce exact confidence bounds for the binomial proportion and an exact p -value for the binomial proportion test. You can now request that Fleiss-Cohen scores be used to compute the weighted kappa coefficient with the AGREE (WT=FC) option. The **scorout** option requests that the row and column scores used for computing statistics such as Cochran-Mantel-Haenszel statistics and Pearson correlation be displayed. The **pchi** option in the **Exact** statement computes the exact chi-square goodness-of-fit test for one-way tables as well as the exact Pearson chi-square test for two-way tables. The **maxtime=** option in the **Exact** statement specifies the maximum time that **Proc FREQ** uses to compute an exact p -value. The **Exact** statement also includes the MC option for computing Monte Carlo estimates of exact p -values.

Changes and Enhancements in V9 for the FREQ procedure

In the **Proc FREQ** statement, the new **nlevels** option displays a table that shows the number of levels for each variable that is named in the **tables** statement(s). The new **zeros** option in the **weight** statement enables you to include observations that have 0 weight values. The frequency and crosstabulation tables will display any levels that correspond to observations that have 0 weights. **Proc FREQ** includes levels that have 0 weights in the chi-square goodness-of-fit test for one-way tables, in the binomial computations for one-way tables, and in the computation of kappa statistics for two-way tables.

The following new options are available in the **tables** statement:

CONTENTS=	enables you to specify the text for the HTML contents file links to crosstabulation tables.
BDT	enables you to request Tarone's adjustment in the Breslow-Day test for homogeneity of odds ratios when you use the CMH option to compute the Breslow-Day test for stratified 2x2 tables.
NOWARN	suppresses the log warning message that indicates that the asymptotic chi-square test might not be valid when more than 20% of the table cells have expected frequencies that are less than 5.
CROSSLIST	displays crosstabulation tables in ODS column format. This option creates a table that has a table definition that you can customize by using the TEMPLATE procedure.

Additionally, **Proc FREQ** now produces exact confidence limits for the common odds ratio and related tests.

Proc MEANS

Earlier we presented the default reporting of this procedure. There are, of course, additional statements and options available.

```
proc means <option(s)> <statistic-keyword(s)>;
  by <descending> variable(s);
  class variable(s) <option(s)>;
  freq variable;
  id variable(s);
  output <out = dataset> <output-specification(s)>;
  types request(s);
  var variable(s);
  ways list;
  weight variable;
```

BY	calculate separate statistics for each BY group
CLASS	identify variables whose values define subgroups for the analysis
FREQ	identify a variable whose values represent the frequency of each observation
ID	include additional identification variables in the output dataset
OUTPUT	create an output dataset that contains specified statistics and identification variables
TYPES	identify specific combinations of class variables to use to subdivide data
VAR	identify the analysis variables and their order in the results
WAYS	specify the number of ways to make unique combinations of class variables
WEIGHT	identify a variable whose values weight each observation in the statistical calculations

We will start by adding requests for statistics in the output. as shown earlier, by default the **MEANS** procedure reports **N** (the number of non-missing observations), **Mean**, **Std Dev** (Standard Deviation), **Minimum** and **Maximum**. We can request additional statistics. When statistics options are added, you must include those default requests if desired. Again, only numeric variables can be identified in the **var** statement.

```
proc means data = colleges range sum var mean nmiss median;
  var intuit;
run;
```

which produces the following output:

The MEANS Procedure					
Analysis Variable: intuit					
Range	Sum	Variance	Mean	Miss	Median
25270.00	10045333.00	28602843.48	7897.27	30	8050.00

If we add a **class** statement, the results will be grouped, although the dataset need not be sorted:

```
proc means data = colleges range sum var mean nmiss median;
  var intuit;
  class sponsor;
run;
```

resulting in:

Analysis Variable: intuit							
sponsor	N Obs	Range	Sum	Variance	Mean	N Miss	Median
1	470	13840.00	987093.00	1085181.41	2208.26	23	2055.00
2	832	24496.00	9058240.00	16461885.69	10979.68	7	10535.00

Note that there are only two values, 1 and 2, for the variable *sponsor*. A separate set of statistics is calculated for each value of each class variable.

OUT = Datasets

Proc MEANS is incredibly powerful in its creation of an output dataset. It is important to note here that **Proc SUMMARY** mirrors the capabilities of **Proc MEANS** in the creation of this output dataset, but **Proc SUMMARY** cannot create a printed report directly the way that **Proc MEANS** can.

It is best to start with a simple example. Using the *colleges* dataset, we will first create an output dataset containing some descriptive statistics:

```
proc means data = colleges noprint;
    var intuit;
    class sponsor;
    output out = test1 n = cnt sum = total mean = avg;
run;
proc print data = test1;
run;
```

This code is requesting the procedure produce an output dataset (**out =**) called *test1*, calculating the frequency (**n =**), sum (**sum =**) and means (**means =**) for the variable (**var**) *intuit* subgrouped by the variable *sponsor*. The output dataset contains these values:

Obs	sponsor	_TYPE_	_FREQ_	cnt	total	avg
1	.	0	1302	1272	10045333	7897.27
2	1	1	470	447	987093	2208.26
3	2	1	832	825	9058240	10979.68

In the above example we named each output statistic. If we requested the same analysis on multiple variables and would prefer not to name each new output variable, we would use the **autoname** option on the **output** statement to allow SAS to name each new field:

```
proc means data = colleges noprint;
    var intuit outtuit;
    class sponsor;
    output out = test2 sum = n = /autoname;
run;
proc print data = test2;
run;
```

The new fields are named the *variablename_statistic*:

Obs	sponsor	_TYPE_	_FREQ_	intuit_ Sum	outtuit_ Sum	intuit_N	outtuit_ N
1	.	0	1302	10045333	11882559	1272	1280
2	1	1	470	987093	2800861	447	455
3	2	1	832	9058240	9081698	825	825

The first column to discuss is **_TYPE_**, which is an automatic SAS variable. In this case, we requested only one class variable *sponsor*, and therefore there are two values, 0 and 1, in this field. When **_TYPE_ = 0**, it indicates the total statistics for the field, while **_TYPE_ = 1** indicates the different subgroups of the variable *sponsor*.

Similarly, **_FREQ_** is another automatic SAS variable that counts the number of observations in the current subgroup, whether they are missing or not. There are 1,302 observations in the dataset, and 470 observations when *sponsor* = 1 and 832 when *sponsor* = 2. This dataset also contains the three fields that were requested in the output statement (**n = intuit_sum outtuit_sum n = intuit_n outtuit_n**).

The next step is to request multiple class variables, for example *sponsor* and *state*. When we look at the output dataset, we will see four values of the `_TYPE_` variable, 0 to 3. Here is the code:

```
proc means data = colleges noprint;
  var intuit;
  class sponsor state;
  output out = test1 n = cnt sum = total mean = avg;
run;
```

and here is a sample of the output:

Obs	sponsor	state	_TYPE_	_FREQ_	cnt	total	avg
1	.		0	1302	1272	10045333	7897.27
2	.	AK	1	4	4	12786	3196.50
3	.	AL	1	25	25	109870	4394.80
4	.	AR	1	17	17	62164	3656.71
53	1		2	470	447	987093	2208.26
54	2		2	832	825	9058240	10979.68
55	1	AK	3	3	3	5226	1742.00
56	1	AL	3	13	13	25386	1952.77
57	1	AR	3	9	9	15166	1685.11

The dataset *test1* contains a total of 155 observations that can be identified as such:

<code>_TYPE_</code>	Description	Number of Observations
0 (nway)	total for dataset	1
1	one record per state (including DC)	51
2	one record per sponsor value	2
3	full crossing (sponsor by state)	<u>101</u>
Total number of observations		155

Note that the code requests two class variables, *sponsor* and *state*. The first class variable (`_TYPE_ = 1`) analyzed is *state* that is the right most field. Then the second class variable *sponsor* (`_TYPE_ = 2`) is analyzed, followed by the crossing (`_TYPE_ = 3`) of the two fields.

TYPES Statement

If there were circumstances where only certain crossing should be processed, the **types** statement can be used to identify exactly what is required. Assume we want to process only for `_TYPE_ = 1` and 3, where we want statistics on the individual *states* and on the crossing of *sponsor* and *state*.

```
proc means data = colleges noprint;
  var intuit;
  class sponsor state;
  types state sponsor*state;
  output out = test1 n = cnt sum = total mean = avg;
run;
```

The resulting dataset would contain 152 observations. There would be no total observation (`_TYPE_ = 0`) or totals for the values of *sponsor* (`_TYPE_ = 2`). Although the size of this dataset would not cause any processing problems, imagine the output for a very large dataset with many class variables. This statement would add efficiency to the program.

If the output was to contain only the crossing of all class variables (`_TYPE_ = 3`), using the **nway** option will limit the output to only the observations with the highest `_TYPE_` value. The code to produce a dataset with those 101 observations would look like:

```
proc means data = colleges nway noprint;
  var intuit;
  class sponsor state;
  output out = test1 n = cnt sum = total mean = avg;
```

```
run;
proc print data = test1;
run;
```

which would produce the following listing (partial):

Obs	sponsor	state	_TYPE_	_FREQ_	cnt	total	avg
1	1	AK	3	3	3	5226	1742.00
2	1	AL	3	13	13	25386	1952.77
3	1	AR	3	9	9	15166	1685.11
4	1	AZ	3	3	3	5484	1828.00
5	1	CA	3	28	18	46593	2588.50
52	2	AK	3	1	1	7560	7560.00
53	2	AL	3	12	12	84484	7040.33
54	2	AR	3	8	8	46998	5874.75
55	2	AZ	3	2	2	16670	8335.00
56	2	CA	3	42	42	578695	13778.45

Chartype Option

If you would prefer to identify the different levels of `_TYPE_` as binary values, you can use the **chartype** option. The **chartype** option specifies that the `_TYPE_` variable in the output data set is a character representation of the binary value of `_TYPE_`. The length of the variable equals the number of class variables.

```
proc means data = soclass.colleges noprint chartype;
  var intuit outtuit;
  output out = test1 sum = min = /autoname;
  types sponsor act sponsor*act state;
  class sponsor act state;
run;
proc print data = test1;
run;
```

This would produce the following output:

Obs	sponsor	act	state	_TYPE_	_FREQ_	intuit_ Sum	outtuit_ Sum	intuit_ Min	outtuit_ Min
1	.	.	AK	001	3	11044	18012	1742	5226
2	.	.	AL	001	18	77569	100407	1697	1740
3	.	.	AR	001	10	42056	52320	1392	3216
4	.	.	AZ	001	5	22154	30850	1828	6746
51	.	11		010	1	3750	3750	3750	3750
52	.	16		010	7	25979	29945	1392	3216
53	.	17		010	16	53614	85576	672	3009
54	.	18		010	18	80848	107309	764	3150
68	1	.		100	257	513791	1485679	608	1672
69	2	.		100	457	4783635	4794552	1640	1780
70	1	16		110	2	3556	7522	1392	3216
71	1	17		110	11	17635	49597	672	3009
81	1	27		110	4	11614	36885	784	4536
82	1	28		110	1
83	2	11		110	1	3750	3750	3750	3750
84	2	16		110	5	22423	22423	3605	3605

The easiest way to read the `_TYPE_` variable is by simply lining up the class statement with the binary characterization:

```
types sponsor act sponsor*act state;
class sponsor act state;

state as the rightmost field is first      _TYPE_ = 001
act is next                               _TYPE_ = 010
sponsor is next                           _TYPE_ = 100
sponsor by act is last                    _TYPE_ = 110
```

t-tests

It is often desired to perform simple t-tests evaluating the difference between two means. In this *colleges* example, one might be interested to see if the mathsat score is significantly different than 510. One can use **Proc MEANS** to do this.

Using the **t** and **prt** options of **Proc MEANS**, one can get the student's t and associated probability value. To see if mathsat is significantly different than 510, one can use the following code:

```
data college2;
    set colleges;
    mathsat_vs_510 = mathsat - 510;
run;
proc means data = college2 n mean stderr t prt;
    var mathsat mathsat_vs_510;
run;
```

Which produces the output below:

The MEANS Procedure						
Variable	Label	N	Mean	Std Error	t Value	Pr > t
mathsat	Math SAT	777	506.8378378	2.4331165	208.31	<.0001
mathsat_vs_510		777	-3.1621622	2.4331165	-1.30	0.1941

Since the **t** and **prt** options in **Proc MEANS** test the hypothesis that the population mean is zero, one needs to subtract 510 from the actual mathsat score to obtain the correct test. In the above test, the reported probability value indicates that the observed mean mathsat score is not significantly different than zero at the $p=.05$ level.

As can be seen in the above output, the values reported often include more decimal places than is typically desired and in a report form that is not very compact. To adjust this there are two useful options, **maxdec** and **fw** that affect the number of decimal places reported (**maxdec**) and the field widths (**fw**) used in the report. If we use these options, as in the code below,

```
data college2;
    set colleges;
    mathsat_vs_510 = mathsat - 510;
run;
proc means data = college2 n mean stderr t prt maxdec = 1 fw = 5;
    var mathsat mathsat_vs_510;
run;
```

we obtain a more compact and readable report.

The MEANS Procedure						
Variable	Label	N	Mean	Std Error	t Value	Pr > t
mathsat	Math SAT	777	506.8	2.4	208.31	<.0001
mathsat_vs_510		777	-3.2	2.4	-1.30	0.1941

In addition to t-tests of this nature, one can also obtain tests of paired observations, the so-called paired-t test. For example, one can test the hypothesis that the math SAT score is the same as the verbal SAT score. When there is a natural pairing of data, one can use **Proc MEANS** to test this hypothesis. If there is NOT a natural pairing, one needs to use alternate procedures such as **Proc TTEST**. To test this hypothesis, one can use the following code:

```
data college2;
    set colleges;
    mathsat_verbal = mathsat - verbsat;
run;
proc means data = college2 n mean stderr t prt maxdec = 1 fw = 5;
    var mathsat verbsat mathsat_verbsat;
run;
```

Which produces the output below:

The MEANS Procedure						
Variable	Label	N	Mean	Std Error	t Value	Pr > t
mathsat	Math SAT	777	506.8	2.4	208.31	<.0001
verbsat	Verbal SAT	777	461.2	2.1	220.53	<.0001
math_verbal		777	45.6	1.0	46.55	<.0001

Thus, we find that the math and verbal SAT scores differ significantly.

Changes and Enhancements in V8 for the MEANS procedure

A feature new to SAS version 8 is the **multilabel** option in **Proc FORMAT**. This option allows one to define two formats that apply to one variable. The multiple labels can be used in some procs, including **Proc TABULATE**, **Proc MEANS** and **Proc SUMMARY**. Note that the **mlf** option needs to be used on the **class** statement in the **Proc MEANS** to use the multiple labels.

Changes and Enhancements in V9 for the MEANS procedure

One of the great features of Version 9 is the support of multithreading in several procedures including the **MEANS** procedure. One can use the new **threads|nothreads** option to enable or prevent the activation of multi-threaded processing.

When you format class variables by using user-defined formats that are created with the **multilabel** and **notsorted** options, specifying the three options **mlf**, **preloadfmt**, and **order=data** in a **class** statement now orders the procedure output according to the label order in the format definition.

Using FREQ and MEANS Together

It is sometimes the case that one needs to access the row and column totals from the classification table produced by **Proc FREQ**. These can then be used to summarize values or compute statistics that can be displayed or saved in a SAS dataset.

For example, the following code combines some of the code previously discussed to create the marginal totals mentioned above. (Numbers on left match code to output).

```
proc format;
    value tuit_fmt
        0-4999          = '<$5K'
        5000-9999      = '$5K-$10K'
        10000-high     = '>$10K';
    value sat_fmt
        0-499          = '<500'
        500-599        = '5-600'
        600-high       = '>600';
run;

#1  proc freq data = colleges;
    tables intuit * mathsat /norow nocol nopercnt out = outfreq ;
    format intuit  tuit_fmt. ;
```



```
        format mathsat  sat_fmt. ;
        label intuit   = 'In-state Tuition';
        label mathsat  = 'Math SAT';
run;

#2  title 'OUTFREQ dataset with counts for INTUIT * MATHSAT';
    proc print data = outfreq;
run;

    proc sort data = outfreq;
        by intuit;
run;
    proc means data = outfreq noprint ;
        output out = out1 sum = marginal1;
        by intuit;
        var count;
        where mathsat ne .;
run;

#3  title 'Marginal totals for INTUIT';
    proc print data = out1;
run;

    proc sort data = outfreq;
        by mathsat;
run;
    proc means data = outfreq noprint;
        output out = out2 sum = marginal2;
        by mathsat;
        var count;
        where intuit ne .;
run;

#4  title 'Marginal totals for MATHSAT';
    proc print data = out2;
run;

    proc means data = outfreq noprint;
        output out = out3 sum = total;
        var count;
        where intuit ne . and mathsat ne .;
run;

#5  title 'Total Count';
    proc print data = out3;
run;
```

This results in the output shown below.

#1 Table of intuit by mathsat

intuit(In-state Tuition)	mathsat(Math SAT)			
Frequency	<500	5-600	>600	Total
<\$5K	137	96	12	245
\$5K-\$10K	135	46	2	183
>\$10K	101	177	51	329
Total	373	319	65	757

Frequency Missing = 545

#2 OUTFREQ dataset with counts for INTUIT * MATHSAT

Obs	intuit	mathsat	COUNT	PERCENT
1	.	.	10	.
2	.	<500	7	.
3	.	5-600	7	.
4	.	>600	6	.
5	<\$5K	.	239	.
6	<\$5K	<500	137	18.0978
7	<\$5K	5-600	96	12.6816
8	<\$5K	>600	12	1.5852
9	\$5K-\$10K	.	147	.
10	\$5K-\$10K	<500	135	17.8336
11	\$5K-\$10K	5-600	46	6.0766
12	\$5K-\$10K	>600	2	0.2642
13	>\$10K	.	129	.
14	>\$10K	<500	101	13.3421
15	>\$10K	5-600	177	23.3818
16	>\$10K	>600	51	6.7371

#3 Marginal totals for INTUIT

Obs	intuit	_TYPE_	_FREQ_	marginal1
1	.	0	3	20
2	<\$5K	0	3	245
3	\$5K-\$10K	0	3	183
4	>\$10K	0	3	329

#4 Marginal totals for MATHSAT

Obs	mathsat	_TYPE_	_FREQ_	marginal2
1	.	0	3	515
2	<500	0	3	373
3	5-600	0	3	319
4	>600	0	3	65

#5 Total Count

Obs	_TYPE_	_FREQ_	total
1	0	9	757

One can then use these marginal totals to compute a variety of statistics not normally computed by SAS Software. It should be pointed out that in the code above, one needed to specifically exclude missing values using the **where** clause (e.g. `where intuit ne .;`). If this statement was not included, the marginal totals would have included counts corresponding to the missing values, which are typically not desired.

In Conclusion

This introduction to **FREQ** and **MEANS** has illustrated some of the capabilities of these two procedures and demonstrates why they are some of the most frequently used SAS procedures. They can be used to directly generate output or to create SAS datasets for further processing. While the output one can obtain from these procedures is very readable, one can customize the output by the use of **Proc FORMAT** and other options as described above.

It is also possible to obtain row and column marginal totals for crosstabulation tables by combining the power of **Proc FREQ** and **Proc MEANS**. From these various summarizations, a variety of statistics can be derived. The best way to learn how to use these procedures and to discover new ways they meet your needs is by simply working with them. Of course, one should always check the LOG for warnings and errors, review datasets created, and study the results for reasonableness.

It is interesting to note that while the authors each have over 20 years of experience in using these procedures, we both discovered features that we either did not know existed or had forgotten. This points out an important truism. SAS software is always improving and changing, as are our skills and needs. Thus, we recommend one develop the habit of periodically re-reading documentation on commonly used procedures and commands, attending beginning tutorials at SAS-related conferences, and browsing the SAS technical support website. There is a vast amount of information and example code available that will not only be very informative but will make us more effective SAS programmers.

Contact Information

For more information contact:

Marge Scerbo

CHPDM/UMBC
1000 Hilltop Circle
Sondheim Hall, Room 309
Baltimore, MD 21250
Email: scerbo@chpdm.umbc.edu

Mic Lajiness

Eli Lilly and Company
Corporate Center Dr
DC: 1523
Indianapolis, IN 46285
Email: LajinessMS@Lilly.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.