

Introduction to Using PROC TABULATE

Thomas J. Winn, Jr.
Office of the Comptroller of Public Accounts
State of Texas

PROC TABULATE is used to build tabular summary reports containing descriptive statistical information, including hierarchical relationships among variables. PROC TABULATE is the SAS® System's implementation of TPL (Table Producing Language), which was developed at the U.S. Bureau of Labor Statistics during the 1970's, for generating tabular reports of descriptive statistics involving employment data. PROC TABULATE is more powerful for producing tabulations than PROC FREQ, and it is a more flexible statistical report writer than PROC MEANS.

Here are some examples of reports generated by PROC TABULATE:

The screenshot displays a SAS window with a report titled "Highway Accident Data" and a program editor showing the SAS code used to generate it. The report is a table with columns for Driver injury status, Seatbelt Usage (no, yes), Speed of car prior to impact (0-29, 30-49, 50+), and a total column. The data shows that for "not" injured drivers, there are 2 counts in each speed category, totaling 12. For "serious" injured drivers, there are also 2 counts in each speed category, totaling 12. The overall total for all categories is 24.

	Seatbelt Usage						total	
	no			yes				
	Speed of car prior to impact			Speed of car prior to impact				
	0-29	30-49	50+	0-29	30-49	50+		
Driver injury status								
not	2	2	2	2	2	2	12	
serious	2	2	2	2	2	2	12	
total	4	4	4	4	4	4	24	

```
proc tabulate data=ssuser.highway format=5.0;
class seatbelt speed status;
var count;
keylabel n = ' ';
all = 'total';
table status all, seatbelt*speed all;
title 'Highway Accident Data';
run;
```

SAS

OUTPUT (Untitled)

			Asking price	
			N	MEAN
Style of home	Number of bath-rooms	Number of bedrooms		
CONDO	1.5	2	1.00	80050.00
	2	2	1.00	110700.00
	2.5	3	1.00	79350.00
		4	1.00	127150.00
RANCH	1	1	1.00	34550.00
		2	1.00	64000.00
	3	3	2.00	87875.00
SPLIT	1	1	1.00	65850.00
	1.5	3	1.00	73650.00
	3	4	1.00	84450.00
TWO-ST-DRY	1	2	2.00	82550.00
	2.5	4	1.00	102950.00
	3	4	1.00	107250.00

NOTE: Aborted

SAS

Start SAS Microsoft PowerPoint - [ab.] 10:48 AM

SAS

OUTPUT (Untitled)

The SAS System 10:25 Wednesday, December 31, 1997

		Sales of first variety of oranges	Sales of second variety of oranges
		SUM	SUM
Day of week	STORE		
1	1	11.32	0.00
	2	7.69	7.12
	3	4.26	6.03
	4	5.62	3.81
	5	7.80	7.00
	6	11.35	4.96
2	1	12.92	0.00
	2	0.00	1.00
	3	0.00	2.09
	4	14.99	7.28
	5	5.23	3.69
	6	9.47	11.73

SAS

Start SAS Microsoft PowerPoint - [ab.] 10:48 AM

Getting Started With PROC TABULATE

The first thing the would-be PROC TABULATE programmer needs to do, before beginning to write any SAS code, is to decide what the final report should look like. What is the intended design for the pages, rows, and columns of the report?

Basic Syntax for PROC TABULATE

```
PROC TABULATE <option-list>;  
  CLASS class-variable-list;  
  VAR analysis-variable-list;  
  TABLE <<page-expression,> row-expression,>  
        column-expression </ table-option-list>;  
  BY <NOTSORTED> <DESCENDING> variable-1  
    < ... <DESCENDING> variable-n>;  
  FORMAT variable-list-1 format-1  
        < ... variable-list-n format-n>;  
  FREQ variable;  
  KEYLABEL keyword-1='description-1'  
        < ... keyword-n='description-n'>;  
  LABEL variable-1='label-1' <...variable-n='label-n'>;  
  WEIGHT variable;  
  TITLE 'text';
```

Here are some options which are frequently used in PROC TABULATE statements:

```
DATA=SAS-dataset-name  
FORMAT=formatname  
MISSING  
NOSEPS
```

The DATA= option specifies the SAS dataset to be used.

The FORMAT option specifies a default format for each table cell. The default format is overridden by any format specified in a subsequent TABLE statement. The MISSING option requests that missing values be regarded as valid levels for classification variables. Unless the MISSING option is specified, observations with missing values for class variables will not be included in the analysis. The NOSEPS option removes the interior vertical lines from the printed report.

It is a requirement that every PROC TABULATE step must include a PROC TABULATE statement, either a CLASS statement or a VAR statement, or both, and a TABLE statement. Each variable appearing in a TABLE statement must be mentioned in either the CLASS statement or the VAR statement. Variable-names may not be mentioned in both the CLASS and the VAR statements.

Steps for Coding PROC TABULATE

The simplest way to approach coding for PROC TABULATE is in terms of the following five steps:

- Specification of classification variables,
- Specification of analysis variables,
- Definition of dimensions of table,
- Identification of desired statistics, and
- Labeling and formatting the table.

Classification variables are used to identify categorical groups on which calculations are performed. They are the variables which make up the rows and columns of the table. They may be either character or numeric. If numeric, generally only a small number of distinct values are permitted.

Analysis variables are numeric variables which are used to compute statistics which are reported in the body of the table.

Step 1 -- Specification of Classification Variables

The CLASS statement is used to specify any categorical variables which will be used for grouping purposes in the analysis. In most cases, the CLASS statement is required.

```
PROC TABULATE DATA=SASUSER.FITNESS;
  CLASS GROUP; . . .
```

Step 2 -- Specification of Analysis Variables

The VAR statement is used to list any variables which will be used for computing the statistics which are to appear in the body of the table. Frequency counts can be computed without an analysis variable. However, under most conditions, the VAR statement is required.

```
PROC TABULATE DATA=SASUSER.FITNESS;
  CLASS GROUP;
  VAR RUNTIME; . . .
```

Step 3 -- Definition of the Table's Dimensions

The TABLE statement is used to define both the arrangement of the rows and columns of the table, as well as the requests for any summary statistics. It is the tricky aspect of using PROC TABULATE. So let's consider this complex statement one piece at a time.

In a TABLE statement, the comma is a very important symbol, because it separates the dimensions of the table. If two commas were specified, then the table would have three dimensions, and the order would be pages, rows, and columns. If only one comma was specified, then the table would have two dimensions, and the order would be rows, columns. No comma would be interpreted to mean that the table's only dimension would be the column dimension. The table would only have one row.

So, TABLE statements look like the following:

TABLE page-expression, row-expression, column-expression . . . ;
TABLE row-expression, column-expression . . . ; or
TABLE column-expression . . . ;

In this context, an expression can consist of variables, statistics, operators, format specifications, and label assignments. Since our immediate concern is only to define the table's dimensions, we are pleased to discover that only three operators are needed to specify the page, row and column headings which identify the structure of a table.

An asterisk (*) can be used to cross the classification variables; that is, to arrange them in a *nested* manner, according to the order listed (top, middle, and lower). A blank space is used to *concatenate* two classification variables (which will appear in the table: top-to-bottom for row headings, left-to-right for column headings). Parentheses () are used to group the elements of an expression, and to associate an adjacent operator with each concatenated element inside the parentheses.

Here are some simple examples:

TABLE var1, var2;

The preceding would generate a two-dimensional table in which the row dimension would be the values of var1, and the column dimension would be the values of var2.

TABLE var1, var2 var3;

This would result in a two-dimensional table in which the row dimension would be the values of var1, and the column dimension would be comprised of values resulting from the side-by-side concatenation of var2 and var3.

TABLE var1, var2*var3;

This statement would generate a two-dimensional table in which the row dimension would be the values of var1, and the column dimension would be a hierarchical arrangement of the values of var2 and var3, with the values of var2 comprising the top columns, and for each of these values as columns, the values of var3 as the lower columns.

TABLE var1, var2 * (var3 var4) var5 ;

The preceding would generate a two-dimensional table in which the row dimension would be the values of var1, and the column dimension would be two side-by-side components, the first of which would be a hierarchical nesting of the values of var2 over a side-by-side arrangement of the values of var3 and var4, and the second component would be the values of var5.

TABLE var1, (var2 var3) * var4 ;

This would result in a two-dimensional table in which the row dimension would be the values of var1, and the column dimension would be two side-by-side

hierarchical components, the first of which would be the values of var2 over the values of var4, and the second would be the values of var3 over the values of var4.

```
TABLE var1 var2, var3 * var4 ;
```

This code would generate a two-dimensional table in which the row dimension would be the concatenated values of var1 and var2, and the column dimension would be a hierarchical nesting of the values of var3 over the values of var4.

Step 4 -- Identification of Desired Statistics

We have been considering incomplete TABLE statements (because up to this point we have only focused on the dimensions of the tables). The TABLE statement also specifies which summary statistics should be produced, and pertaining to which analysis variables. Each statistic is identified by a keyword.

N = the number of observations, the frequency count,

MIN = the smallest value,

MAX = the largest value,

MEAN = the arithmetic mean, or the average,

STD = the standard deviation,

VAR = the variance,

SUM = the sum of the values,

PCTN = the percentage that one frequency is of another frequency,

PCTSUM = the percentage that one sum is of another sum,

... (and other descriptive statistics).

Whenever you cross a variable with a keyword for a statistic, you are identifying the statistic to be applied to that variable (which tells PROC TABULATE what type of calculation to perform). You can cross class variables only with the N or PCTN statistics. By default, if the TABLE statement does not include an analysis variable or a statistic, then PROC TABULATE automatically crosses the N statistic with the indicated class variables. Analysis variables can be crossed with any statistic. By default, if the TABLE statement includes an analysis variable but without crossing it with any statistic, PROC TABULATE automatically crosses it with SUM.

Here are some examples:

```
PROC TABULATE DATA=MYLIB.EMPLOYEE;  
  CLASS JOBCATG RACE SEX;  
  TABLE JOBCATG, RACE*SEX;  
  TABLE JOBCATG*PCTN, RACE*SEX;  
  TABLE JOBCATG, RACE*SEX*PCTN;  
  TABLE JOBCATG PCTN, RACE*SEX;
```

The first TABLE statement would generate a hierarchical breakdown of frequency counts in the data set, according to values of JOBCATG (the rows) and the nested values of RACE and SEX (the columns).

The second and third TABLE statements would generate a hierarchical breakdown of percentages represented in each cell, according to values of JOBCATG and the nested values of RACE and SEX.

The fourth TABLE statement would generate a hierarchical breakdown of frequency counts represented in each cell, according to values of JOBCATG and the nested values of RACE and SEX. It would include an additional row which would represent the percentage of the total population of the data set included in each column.

Here are a few more examples:

```
PROC TABULATE DATA=MYLIB.EMPLOYEE;  
  CLASS JOBCATG RACE SEX;  
  VAR SALARY;  
  TABLE JOBCATG, RACE*SEX*SALARY;  
  TABLE JOBCATG, RACE*SEX*SALARY*PCTSUM;  
  TABLE JOBCATG, RACE*SEX*SALARY*MEAN;
```

The first TABLE statement would generate a hierarchical breakdown of the sum of the salary amounts represented in each cell, according to values of JOBCATG and the nested values of RACE and SEX.

The second TABLE statement would generate a hierarchical breakdown of the percentage of the total of salary amounts represented in each cell, according to values of JOBCATG and the nested values of RACE and SEX.

The third TABLE statement would generate a hierarchical breakdown of the average salary represented in each cell, according to values of JOBCATG and the nested values of RACE and SEX.

PROC TABULATE has a universal class variable, ALL, which can be used to generate totals for any specified class variable. Just concatenate the keyword ALL into the row or column expression of a TABLE statement.

```
PROC TABULATE DATA=MYLIB.EMPLOYEE;  
  CLASS JOBCATG RACE SEX;  
  TABLE JOBCATG ALL , RACE*SEX ALL;  
  TABLE JOBCATG ALL , (RACE ALL)*SEX;
```

[Question: What would the difference be between the tables produced by these two TABLE statements?]

Now, using PROC TABULATE to compute percentages is a bit problematic. PCTN and PCTSUM are the keywords which specify, respectively, the percentage of the value in a particular cell to the value in another cell, or to the

total of a group of cells. Whenever someone performs percentage calculations, it is essential to be very clear concerning which number is to be used as the *denominator* in the computation.

PROC TABULATE allows a programmer to identify the specific denominator to be used in computing percentages.

By default, if a denominator is not explicitly defined in a TABLE statement which includes percentages, then PROC TABULATE uses the total of the N-frequency-counts for the entire data set as the denominator in a PCTN calculation, and the total of all of the SUM cells for the denominator in a PCTSUM calculation – in each case, the percentage is computed based upon *all* of the observations in the data set. If that is what a programmer intends, then specifying PCTN or PCTSUM in a TABLE statement isn't much different than requesting any other summary statistic. In that case, you wouldn't need to define the denominator, because the default denominator is implicitly assumed (see the examples, above).

However, whenever it is necessary to use a particular total as the denominator in a PROC TABULATE percentage calculation, then a *denominator definition* must be specified in the TABLE statement. To specify how the percentage is to be calculated, the “less-than” (<) and “greater-than” (>) inequality signs are used as brackets to enclose an expression which tells PROC TABULATE how to identify the appropriate denominator. Now, this probably doesn't work the way you might think that it should, so pay close attention to the examples, below.

Here are a couple of examples:

```
PROC TABULATE DATA=MYLIB.EMPLOYEE;  
  CLASS JOBCATG RACE SEX;  
  VAR SALARY;  
  TABLE JOBCATG, RACE*(N PCTN<RACE>);  
  TABLE JOBCATG, RACE*SALARY*(SUM PCTSUM<JOBCATG>);
```

The first TABLE statement would generate a breakdown of the counts and frequency percentages for each combination of values of JOBCATG and RACE, where the percentages are calculated in a row-wise manner (that is, using the total count for each value of JOBCATG, across all values of RACE, as the denominator for that particular row).

The second TABLE statement would generate breakdowns of the total salary amounts, and the associated percentages, for each combination of values of JOBCATG and RACE, where the percentages are calculated in a column-wise manner (using the SALARY subtotal for each value of RACE, across all values of JOBCATG, as the denominator for that column).

Observe that, to obtain percentages by row, we use the column-expression in the “denominator definition”; and to obtain percentages by column, we use the row-expression in the “denominator definition”.

Following are a couple of examples which illustrate the use of the universal class variable, ALL, in a denominator definition for percentage calculations. Notice that whenever row- or column-percentages are to be produced for a column- or row-expression which include ALL, then ALL also must be included in the denominator definition.

```
PROC TABULATE DATA=MYLIB.EMPLOYEE;
  CLASS JOBCATG RACE SEX;
  VAR SALARY;
  TABLE JOBCATG, (RACE ALL)*(N PCTN<RACE ALL>);
  TABLE JOBCATG ALL,
    RACE*SALARY*(SUM PCTSUM<JOBCATG ALL>);
```

Step 5 -- Labeling and Formatting the Table

Now that we know how to define the basic structure of the tables we will generate, we would like to be able to make the tables more self-explanatory; that is, easier to read and interpret.

As in many other SAS procedures, you can use a LABEL statement to replace variable names with more descriptive headings for your class variables. There also is a way to specify temporary labels in a TABLE statement. Similarly, TITLE and FOOTNOTE statements also can be used to enhance the tabular reports generated by PROC TABULATE. To assign labels to procedure-generated statistics and the universal class variable, we use the KEYLABEL statement.

```
KEYLABEL N = 'Count'
        ALL = 'Total'
        PCTN = 'Percent';
```

As in other SAS procedures, formats can be used to substitute labels for values of the classification variables. Formats also can be used to combine many values of the classification variables into a much smaller number of values to be printed in the report. We create custom formats by using PROC FORMAT, and we invoke those formats in PROC TABULATE either through a FORMAT statement, or by crossing F=format-name. in the TABLE statement with the particular variable.

The default for displaying cells with missing numeric values is a period. You can change the way missing values are displayed by using the MISSTEXT= option to define up to twenty characters of text which will print in the table cells whenever a particular combination of class variable values is not found in the input data set.

```
PROC TABULATE DATA=MYLIB.EMPLOYEE;
  CLASS JOBCATG RACE SEX;
```

TABLE JOBCATG, RACE*SEX / MISSTEXT='NONE';

Example

```
PROC FORMAT $CTGFMT 'A' = 'Administrative Staff'
                   'C' = 'Clerical Personnel'
                   'E' = 'Executives'
                   'M' = 'Managers'
                   'S' = 'Sales Staff'
                   'T' = 'Technicians';
SALFMT LOW-<1000 = 'Less than $1,000'
        1000-<2000 = '$1,000 - $1,999'
        2000-<3000 = '$2,000 - $2,999'
        3000-<4000 = '$3,000 - $3,999'
        4000-HIGH = '4,000 or more';
```

```
PROC TABULATE DATA=MYLIB.EMPLOYEE MISSING FORMAT=9.1;
  CLASS JOBCATG RACE SEX SALARY;
  FORMAT JOBCATG $CTGFMT. SALARY SALFMT.;
  LABEL JOBCATG = 'Job Category';
  KEYLABEL PCTN = 'Percent' ALL = 'Total';
  TABLE JOBCATG*SALARY ALL, (RACE*SEX ALL)*PCTN
          / MISSTEXT='0';
  TITLE 'Tabular Summary of Employee Information';
```

Conclusion

PROC TABULATE is a very useful and very powerful procedure for constructing tabular reports containing descriptive statistical information, including hierarchical relationships among variables. It is well-worth the necessary investment of time and effort for learning the intricacies and subtleties of its syntax. Concentrating on the five steps makes it much easier to learn how to code PROC TABULATE. We have only just "scratched the surface" of this wonderful procedure! But now you know enough to continue learning about it on your own. Happy tabulating!

References

- Jeffery M. Abolafia & Stephen M. Noga, "The TABULATE Procedure: One Step Beyond the Final Chapter",
SESUG '97 Proceedings (1997), pp. 293-300; and
Proceedings of the Twenty-Third Annual SAS Users Group International Conference (1998), pp. 839-844.
- Dan Bruns, "The Utter Simplicity of the TABULATE Procedure",
Proceedings of the Sixteenth Annual SAS Users Group International Conference (1991), pp. 365-371; and
Proceedings of the Twentieth Annual SAS Users Group International Conference (1995), pp. 363-368; and
Proceedings of the Seventh Annual South-Central SAS Users' Conference (1997), pp. 54-60.

- Dan Bruns, "The Utter 'Simplicity?' of the TABULATE Procedure", Proceedings of the Seventeenth Annual SAS Users Group International Conference (1992), pp. 216-220.
- Dan Bruns, "Advanced Features of PROC TABULATE -- or -- The Utter Simplicity of the TABULATE Procedure - The Sequel", Proceedings of the Twenty-First Annual SAS Users Group International Conference (1996), pp. 242-247.
- Dan Bruns, "The Utter 'Simplicity?' of the TABULATE Procedure -- The Final Chapter",
Proceedings of the Twenty-Second Annual SAS Users Group International Conference (1997), pp. 251-256 ; and
Proceedings of the Seventh Annual South-Central SAS Users' Conference (1997), pp. 61-66.
- Michele M. Burlew, "A Stepwise Approach to Using PROC TABULATE", Chapter 5 of Reporting From the Field (SAS Institute Inc., 1994), pp. 67-88.
- SAS Institute Inc., "The TABULATE Procedure", Chapter 37 of SAS Procedures Guide, Version 6, Third Edition (1990).
- SAS Institute Inc., SAS Guide to TABULATE Processing, Second Edition (1990).
- SAS Institute Inc., "Summary Reports -- PROC TABULATE", Chapter 5 of SAS Views: SAS Report Writing (1987).
- Bob Virgile, "The Right Approach to Learning PROC TABULATE", SESUG '97 Proceedings (1997), pp. 189-195.

Author Information

Tom Winn
Fiscal Management Support
Texas Comptroller of Public Accounts
L.B.J. State Office Building
Austin, TX 78774

Telephone: 512 / 463-4907
E-mail: tom.winn@cpa.state.tx.us

SAS is a registered trademark of SAS Institute Inc. in the USA and other countries.
® indicates USA registration