

The Right Approach to Learning PROC TABULATE

Bob Virgile
Robert Virgile Associates, Inc.

Overview

PROC TABULATE produces tables as illustrated by the samples in Figures 1 and 2. These tables incorporate some special features, features which no other SAS® procedure can produce. The first feature is that the rows (or columns) may be composed of a number of variables. The variables may be concatenated (as with STATE and CUSTOMER in Figure 1) or nested (as with COMPANY and JOB in Figure 2). The other unique feature of PROC TABULATE tables is that the numbers in the table are not necessarily frequency counts. The numbers can represent statistics (means, sums, etc.) based on the values of numeric variables.

The Approach

To create a table, the programmer must perform five basic functions:

1. Specify which variables will be used as classification variables to define the rows and columns of the table.
2. Specify which variables will be used as analysis variables to compute the statistics in the body of the table.
3. Define the structure of the rows and columns.
4. Request particular statistics based on the analysis variables.
5. Provide additional formatting and labeling instructions.

PROC TABULATE tables may include a page dimension in addition to a row and column dimension. This paper does not address the issue of the page dimension.

The primary reason PROC TABULATE can become complex has to do with the formatting and labeling instructions. These instructions can appear in a variety of places throughout the procedure.

Standard statements such as LABEL, FORMAT, and TITLE are permissible. A special KEYLABEL statement also exists. And essential formatting instructions can also be specified on both the PROC statement itself and the TABLE statement which defines the row and column dimensions. Other complex topics exist, such as the computation of percentage statistics. However, the main reason PROC TABULATE appears to be complex is the diverse locations where labeling and formatting instructions may be specified.

The complexity can be reduced by thinking (and learning) in terms of the five objectives outlined above. The alternative is to focus on individual statements, their syntax, and their usage. This alternative approach has two drawbacks. First, it becomes more difficult to place the individual statements in perspective, to figure out when they should or must be used. Second, it becomes more difficult to tell when a program is complete. All statements may possess good syntax, but will they combine to produce the complete instructions that will generate the desired table? To avoid these drawbacks, learn PROC TABULATE in terms of:

1. Specifying classification variables,
2. Specifying analysis variables,
3. Defining table dimensions,
4. Requesting statistics, and
5. Formatting and labeling the table.

The sample tables we build will omit some of the fancier PROC TABULATE features. This paper will not tackle the more complex statistics based on percentages, nor will it discuss the global classification value ALL. And the examples will define row and column dimensions only, not the page dimension. These and other topics are well documented in the standard reference materials listed at the end of this paper.

PROC TABULATE takes on the usual form for SAS procedures: the PROC statement comes first, while PROC statement options and subsequent

Figure 1

Sample PROC TABULATE Output

TABLE STATE CUSTOMER,
COST * SUM;

	COST
	SUM
STATE	
MA	6705.86
NY	2336.64
CUSTOMER	
02847AD1422G12	800.00
12847AD1422XC2	1625.98
22847AD1422YC2	499.90
32847AD142JZ12	990.00
42847AD1422A72	775.00
52847AD1422B82	1346.64
62847AD1422C42	885.00
72847AD1422DB2	799.99
82847AD1422E22	720.00
92847AD1422FC2	599.99

Note: All output has been retyped. Insignificant differences may exist between the actual output and these diagrams.

Figure 2

Sample PROC TABULATE Output

```

PROC TABULATE DATA=SALARIES;
CLASS COMPANY JOB;
VAR SALARY;
TABLE COMPANY * JOB,
       SALARY * (MIN MEAN MAX);

```

		SALARY		
		MIN	MEAN	MAX
COMPANY	JOB			
ABC CORP	MANAGER	960.00	1070.71	1275.00
	PROGRAMMER	520.00	713.57	832.00
	SECRETARY	260.00	329.29	400.00
XYZ CORP	MANAGER	920.00	979.86	1056.00
	PROGRAMMER	520.00	713.43	900.00
	SECRETARY	358.00	437.14	550.00

		WEEKLY SALARY		
		MINIMUM	AVERAGE	MAXIMUM
COMPANY	JOB			
ABC CORP	MANAGER	\$960.00	\$1,070.71	\$1,275.00
	PROGRAMMER	\$520.00	\$713.57	\$832.00
	SECRETARY	\$260.00	\$329.29	\$400.00
XYZ CORP	MANAGER	\$920.00	\$979.86	\$1,056.00
	PROGRAMMER	\$520.00	\$713.43	\$900.00
	SECRETARY	\$358.00	\$437.14	\$550.00

Note: All output has been retyped. Insignificant differences may exist between the actual output and these diagrams.

statements clarify what the procedure is supposed to do. So the overall form is:

```
PROC TABULATE DATA=IN.DATA;  
... further TABULATE statements that  
    accomplish the five objectives  
    listed above
```

Accomplishing the Five Objectives

The first responsibility is to specify the classification variables, which are the variables that define the rows and columns of the table. In Figure 1, STATE and CUSTOMER are classification variables. In Figure 2, COMPANY and JOB are classification variables. This step simply requires a CLASS statement:

```
PROC TABULATE DATA=IN.DATA;  
CLASS ... list of classification  
        variables;  
... further statements that  
    accomplish the remaining four  
    objectives
```

Classification variables may be either character or numeric. The values of these variables generate rows and columns, so if a variable takes on many different values, the table becomes correspondingly large. Therefore, be careful about using a numeric variable as a classification variable. The CLASS statement is required.

One quirk applies to classification variables. If any classification variable has a missing value, that observation is omitted from the table and from all computations. This holds true for all variables mentioned in the CLASS statement, even for variables which do not get used in a row or column of the table. As with similar SAS procedures, the MISSING option (not illustrated in this paper) changes the handling of missing values, allowing them to be treated as a valid value.

The second objective is to define the analysis variables, which are the variables used to compute the statistics reported in the body of the table. In Figure 1, COST is the analysis variable. In Figure 2, SALARY is the analysis variable. Again, this is a simple task. The VAR statement simply lists any analysis variables:

```
PROC TABULATE DATA=IN.DATA;  
CLASS ... list of classification  
        variables;  
VAR ... list of analysis variables;  
... further TABULATE statements that  
    accomplish the remaining three  
    objectives
```

Analysis variables must be numeric. Under most conditions, the VAR statement is required. Occasionally, a table may request statistics that can be computed without an analysis variable, such as frequency counts. In those cases, the VAR statement may be omitted.

The next two programming objectives are defining the table dimensions and requesting summary statistics. The complexities begin here, because one statement, the TABLE statement, must accomplish both tasks. The program becomes:

```
PROC TABULATE DATA=IN.DATA;  
CLASS ... list of classification  
        variables;  
VAR ... list of analysis variables;  
TABLE ... arrangement of rows and  
        columns, and requests for  
        summary statistics;  
... further TABULATE statements for  
    labeling and formatting
```

In a TABLE statement, the comma is a key symbol: it separates the dimensions of the table. If two commas were specified (meaning three dimensions), the order would be PAGES, ROWS, and COLUMNS. If one comma were specified (meaning two dimensions), the order would be ROWS, COLUMNS. With no commas, the only dimension would be the column dimension. All the examples in this paper specify two dimensions, so the overall format will be:

```
TABLE row dimension,  
      column dimension;
```

where the comma indicates the end of the description of the row dimension. The following example produces column headings for two classification variables, COLUMN1 and COLUMN2:

```
TABLE row dimension,  
      COLUMN1 COLUMN2;
```

Figure 3

Potential Column Headings

COLUMN1		COLUMN2		
Value 1	Value 2	Value 1	Value 2	Value 3

(a)

TOPCOLS							
TOPCOLS Value 1				TOPCOLS Value 2			
MIDCOLS				MIDCOLS			
Value 1		Value 2		Value 1		Value 2	
BOTTOM		BOTTOM		BOTTOM		BOTTOM	
Value1	Value2	Value 1	Val1	Val3	Val1	Val3	Val4

(b)

LEFTCOL				RIGHTCOL	
LEFTCOL Value 1		LEFTCOL Value 2			
BOTTOM1	BOTTOM2	BOTTOM1	BOTTOM2	Value 1	Value 2
Val1	Val2	Val1	Val2	Val1	Val3

(c)

		VAR3					
		VAR3 Value 1			VAR3 Value 2		
		VAR4			VAR4		
		Val1	Val2	Val3	Val1	Val4	Val5
VAR1	Value 1						
	Value 2						
	Value 3						
	Value 4						
	Value 5						
VAR2	Value 1						
	Value 2						

(d)

Figure 3a depicts these column headings, using Value 1, Value 2, etc. to represent the possible data values for the COLUMN1 and COLUMN2 variables. PROC TABULATE generates as many columns as are needed, based on all values of variables in the column dimension of the table. When the TABLE statement simply lists classification variables, with a space between them, the variables are concatenated (side by side for column headings, or top to bottom for row headings). An asterisk can be used to nest the classification variables. Figure 3b nests three classification variables (TOPCOLS, MIDCOLS, and BOTTOM) using:

```
TABLE row dimension,  
      TOPCOLS * MIDCOLS * BOTTOM;
```

The space and the asterisk, along with the parentheses to alter the order of evaluation, are the only tools for structuring row and column headings. No other tools exist. The default order for the headings is in sorted order according to the actual data values. (To a small extent, this order can be controlled using PROC statement options. See the reference materials for details on these options.) Parentheses might be used as follows:

```
TABLE row dimension,  
      LEFTCOL * (BOTTOM1 BOTTOM2) RIGHTCOL;
```

in order to produce Figure 3c column headings. You can write more complex looking descriptions, such as:

```
(VAR1 * (VAR2 VAR3)) * VAR4 *  
(VAR5 VAR6)
```

However, this is not the recommended practice. For one thing, the order of evaluation becomes very tricky with the extensive use of parentheses. The actual order of the headings may be different from what you expect. In addition, the same column headings can usually be obtained by writing a clearer (although longer) TABLE statement using fewer parentheses.

The entire TABLE statement will define both the rows and the columns. For example:

```
TABLE VAR1 VAR2,  
      VAR3 * VAR4;
```

asks for row headings with VAR1 followed by VAR2, and column headings with VAR4 nested within VAR3. The table would look something like Figure 3d.

Even this TABLE statement is incomplete. It must specify which summary statistics should be produced, and which numeric (analysis) variables those statistics will be based on. Each statistic possesses a key word, such as SUM or MEAN, which identifies that statistic. If the table uses more than one analysis variable, they must all appear in the same dimension. Similarly, if more than one summary statistic is to be reported, all statistics must appear in the same dimension. It is possible, however, for analysis variables to appear in one dimension, while statistics appear in another dimension.

The final programming objective is to format the output. Formatting becomes complex because many statements have an effect in this regard.

TITLE statements print at the top of the output.

FORMAT statements translate the values of classification variables from the original to a formatted value. Although the technique is not illustrated here, the FORMAT statement may group values of the classification variables, combining many rows or columns into one. This process is very similar to the way a FORMAT statement would group many values as one in PROC FREQ or PROC SUMMARY. See the reference manuals cited at the end of this paper for details. Note that the FORMAT statement does not control the format in which summary statistics are printed. That function must be controlled by the TABLE statement (usually) or by PROC statement options (to set up a default format).

LABEL statements set up default labels for classification and analysis variables. This statement allows variable labels to replace variable names in the row and column headings.

KEYLABEL statements set up default labels for summary statistics. This statement allows the label to replace the summary statistic's name (MEAN, SUM, etc.) in row and column headings.

The **PROC** statement can set up default formats for all statistics, and can control the characters used to construct the box (horizontal and vertical lines, corners, intersections) that surrounds the output.

The **TABLE** statement in practice controls most of the formatting. It overrides the default formats and labels set up by the **FORMAT**, **LABEL**, and **KEYLABEL** statements. It permits text strings to replace a dot to indicate missing values for

statistics. It can specify text to appear in the box in the upper left hand corner of the table. It can control the amount of space used to print row headings.

This paper can only begin to show the wide variety of methods that may format the output of PROC TABULATE. Consider Figure 2, for example, where the top and bottom tables represent the same data in a different format. The following statements could have produced the formatted version:

```
PROC TABULATE DATA=IN.DATA;
CLASS COMPANY JOB;
VAR SALARY;
TABLE COMPANY * JOB,
      SALARY * (MIN *F=DOLLAR10.2
                MEAN*F=DOLLAR10.2
                MAX *F=DOLLAR10.2)
      / RTS=24;
LABEL SALARY = 'WEEKLY SALARY';
KEYLABEL MIN='MINIMUM'
      MAX='MAXIMUM'
      MEAN='AVERAGE';
```

The bold type marks the additions needed to transform output from the top of Figure 2 to the bottom of Figure 2.

The rows: nest JOB within COMPANY.

The columns: for SALARY, calculate three statistics.

For each statistic, print it in the DOLLAR10.2 format.

Print labels instead of the names of the statistics.

The RTS (row title space) option prints the row titles in 24 columns, including the vertical bars before and after the titles themselves.

For additional details on all PROC TABULATE features, see the standard reference manuals (or later versions if and when they are printed):

SAS Guide to TABULATE Processing, Second Edition, Cary, NC: SAS Institute Inc., 1990.

SAS Procedures Guide, Version 6, Third Edition, Cary, NC: SAS Institute Inc., 1990.

As you read these books, however, fit what you read into the structure that this paper has presented. You can learn the syntax and uses of each statement. But try to relate each statement to the functions it will help you accomplish; each

statement plays a role in accomplishing the five objectives listed at the beginning of this paper. Concentrating on these objectives will make PROC TABULATE much easier to learn.

Feel free to write or call the author:

Bob Virgile
Robert Virgile Associates, Inc.
3 Rock Street
Woburn, MA 01801

(781) 938-0307

SAS is a registered trademark of SAS Institute Inc.