The Utter Simplicity of the TABULATE Procedure

Dan Bruns, Tennessee Valley Authority, Chattanooga, TN

Before TABULATE

So many times when you first learn some new software product, you simply do what you are told or can find out about it. I learned base SAS software the same way at a university from a friend. It was for some math classes and I learned to do some simple inputting and frequency tables and summaries using PROC FREQ and SUMMARY. It wasn't until years later I discovered the power of PROC TABULATE.

If all you ever need to do is some simple cross-tabulations or simply print a list of summaries, PROC FREQ or MEANS or SUMMARY will do the job. But more often, you need to be able to control what goes in the cross-tabulation cell and how it is printed; PROC TABULATE gives you this control and versatility.

In this tutorial you will see how, with very little coding, you can produce some simple or very complex output. The output from a CONTENTS procedure below is just so you know a little about the data set we will be working with.

The Old Way

The FREQ procedure is very simple and easy to use to get either a one-way or two-way table by coding TABLE statements as below.

```
PROC FREQ DATA=CLASS;
CLASS ORG DATE;
TABLES ORG*DATE;
```

The output below shows that basically all you get is a count of observations and percentages.

<table>
<thead>
<tr>
<th>TABLE OF ORG BY DATE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ORG(Day)</strong></td>
</tr>
<tr>
<td>Frequency</td>
</tr>
<tr>
<td>------------</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

The MEANS procedure is also very simple to use and gives you more information about your data, i.e., MEAN and SUM, if requested as below.

```
PROC MEANS DATA=CLASS N NMISS MEAN SUM;
CLASS LOG;
VAR SCORE;
```

But the output is only produced as a simple list as below.

<table>
<thead>
<tr>
<th>Analytical Variable: SCORE Final Exam Score</th>
</tr>
</thead>
<tbody>
<tr>
<td>LOC</td>
</tr>
<tr>
<td>A</td>
</tr>
<tr>
<td>B</td>
</tr>
<tr>
<td>C</td>
</tr>
</tbody>
</table>

A Better Way

The TABULATE procedure is a marriage of the FREQ and MEANS procedures to produce an even more powerful and flexible procedure. TABULATE makes use of the CLASS and VAR statements of PROC FREQ and the TABLE statement of PROC MEANS. The CLASS and VAR statements haven't changed, they still define which variables are used for classification (or categorization) and analysis (or statistics). The TABLE statement has been greatly enhanced to allow you to specify not only what you want each cell to be but how to format it and label it. It has also been expanded to allow you to group and/or concatenate more than one table. Let's look at some simple examples.

```
PROC TABULATE DATA=CLASS;
CLASS ORG LOC DATE;
VAR SCORE;
TABLE ORG, DATE;
```

The PROC statement simply identifies the data set and other options to control output features. The CLASS and VAR statements simply define the variables used for classification and analysis. The TABLE statement specifies the table expressions used to define the pages, rows, and columns to be produced.
The key to using TABULATE are the table expressions which consist of a combination of operands and operators much like an arithmetic expression:

```
TABLE page-expression, row-expression, column-expression;
```

- Operators (discussed later)
  - for nesting
  - space for concatenation
- Operands
  - class variables (from CLASS stmt) or ALL
  - analysis variables (from VAR stmt)
  - () for grouping
  - statistics keywords
    - (ie N, NMISS, MEAN, STD, MIN, MAX, SUM)
  - title specification (= 'text')
  - format specification (F=w.d)

When only two expressions are coded (like ORG, DATE), they are the row-expression and the column-expression. What if only ONE expression is coded; will it be the row or column expression?

The three tables below show a very simple yet powerful use of TABULATE. Although our data set only contains a few observations, it could have contained half a million and the code would not change.

The above output shows that when CLASS variables are used, TABULATE will give you a count(N) by default.

```
TABLE ORG, SCORE;
```

The above output shows that when VAR variables are used, TABULATE will give you a total(SUM) of that variable by default.

```
TABLE LOC, ORG, DATE;
```

The above outputs show the results of using a page-expression.

**Formatting**

The above examples show a very simple use of tables, and if all you needed was the numbers to plug in some report for your boss, this would suffice. But I have found that sometimes these tables go beyond your eyes or you need to make the cells smaller (or larger). As you can see, the default format for the cells is 12.2, but you can control the size of each cell by attaching a format specification to the variable in the table expression.

```
TABLE ORG, DATE*(N*F=6.0);
```

You could also specify FORMAT=6.0 as an option on the PROC statement, but that changes the default for all cells. By using the FORMAT=(or :=) in the table expression you can control the format of individual rows or columns based on which variable you attach it to.

**Nesting**

Nesting is a very common use of TABULATE and is produced by coding an asterisk (*) between the variables you want to break-down even further for more detailed information. So the expression DATE*LOC will produce a row/column for each date (4 in our example) broken-down even further by each location within that date. Notice that even though the table is in sorted order, we did NOT have to do a PROC SORT prior to this step—the CLASS statement takes care of this for us. In the following example of nesting columns, parentheses are used to aid in clarification. Later they are also used to group expressions just like in arithmetic.
TABLE ORG, DATE, LOC*(N*F = 3.0);

Notice how TABULATE split the "Location" label because it was too long to fit over its nested column. TABULATE makes no attempt to hyphenate it correctly; it simply puts as much as it can on the first line, inserts a hyphen, and continues on the next line. The best way to control the splitting is to use labels with blanks where you want it split since it will split at a blank, the same way the PRINT procedure does.

Now by simply moving the nesting to the row expression you have changed the whole look of the table.

TABLE ORG*DATE, LOC*(N*F = 3.0);

WOW!! What happened? The look of the table completely changed by simply moving a variable from nesting in the columns to nesting in the rows.

You'll also notice that TABULATE uses the same amount of space to label the rows, regardless of the levels of nesting. The above output is alright in this case, but could get very narrow with another level of nesting. The default amount of space is 1/4 of the linesize (LS= option), but can be specified with the RTS= table option as below. The RTS space will be divided equally among the levels of nesting. Can you guess what RTS stands for?

TABLE ORG*DATE, LOC*(N*F = 3.0) / RTS = 20;

As with most base SAS software statements used in procedures (i.e. PLOT and CHART), you can follow the table request with a slash followed by table options. We'll see more of these options later.

Concatenation

Concatenation is like putting tables side-by-side or stacked on top of each other. By simply using a blank between the variables you get tables for each variable concatenated from left-to-right or top-to-bottom.

The above output shows the results of concatenating in the row expression; basically stacked tables.

TABLE ORG, (LOC DATE)*(N*F = 3.0);

The above output shows the results of concatenating in the column expression; basically side-by-side tables.
TABLE ORG, (LOC*N*F=2.0 DATE*N*F=6.0);

This output shows the results of concatenating in the column expression and specifying which statistic(N) and how to format it(F=) for each column.

Statistics

TABULATE gives you the same statistics as the MEANS and SUMMARY procedures, plus two new ones: PCTN and PCTSUM. With the flexibility of TABULATE you can now specify what statistic you want where and how. One requirement is you can only specify one statistic for a given variable. The way to look at nesting a group is like the distributive law of mathematics. Whatever is outside the parenthesis is distributed to each item inside the parenthesis. Thus you could code the above expression as:

TABLE ORG, SCORE*(N MEAN MAX PCTN), F-S.T;

Here you see how to nest (or attach) a format to a group. The same distributive principle applies as earlier. The PCTN statistic you've seen in the last couple examples is a percentage of a certain number of observations. By default it computes the percentage based on the total number of observations. This can be controlled with a denominator specification and is WAY beyond the scope of this tutorial. Look for a sequel in the near future.

TABLE ORG*(N MEAN MAX PCTN), SCORE;

Here again you can completely change the look of the table by simply moving the grouping to the row expression. Notice the statistics are not of ORG but rather specify in which dimension to put them.

Here are a few more examples and their totally different looking outputs by simply changing where and how the variables are coded.

TABLE ORG, LOC*SCORE*(N MEAN)*F=5.1;

Here you see a column for the count(N) and mean of SCORE for each location.

TABLE ORG*LOC, SCORE*(N MEAN MAX PCTN)*F=5.1;

This is the mean of the scores for each organization. Remember the default statistic is SUM if not specified.

This is the mean of the scores for each organization. Remember the default statistic is SUM if not specified.

TABLE ORG, SCORE*(N MEAN MAX PCTN);

Notice here that by grouping with parenthesis you can get more than one statistic for a given variable. The way to look at nesting a group is like the distributive law of mathematics. Whatever is outside the parenthesis is distributed to each item inside the parenthesis. Thus you could code the above expression as:

TABLE ORG, SCORE*(N MEAN MAX PCTN);

But that seems a little much.

TABLE ORG, SCORE*(N MEAN MAX PCTN)*F=5.1;
Here are the same numbers from the previous output. Location was simply moved from the column expression to the row expression.

**TABLE ORG LOC, SCORE*(N MEAN PCTN)*F=5.1;**

Here are two tables in one: the N, MEAN, MAX, and PCTN statistics in the column expression allows you to use the row expression to see statistics in the column expression. That means it does not add up the means from the tables and then divide by the number of observations. That means it does not compute true statistics (i.e. MEAN above). That means it does not use statistic's name instead of a variable name. That means it does not specify what to put in the upper-left corner box of the table.

**TABLE ORG ALL, (LOC ALL)*SCORE*(N MEAN PCTN)*F=5.1;**

The above example has another table option specified (BOX=) that specifies what to put in the upper-left corner box of the table.

In the following example we added the MISSING and NOSEPS options to the PROC statement to have TABULATE treat missing values as a valid category (which it does not do by default) and remove the separation lines between the rows. I also specified some table options: BOX=SCORE to label the upper-left box with the SCORE variable's label; and MISSTEXT='None' to label missing values in the tables with the text 'None' instead of the standard period.

**PROC TABULATE DATA=CLASS MISSING NOSEPS;**

**CLASS ORG LOC DATE;**

**/ TITLE & LABELS */**

**TABLE ORG ALL= Totals --**, (LOC ALL=Row Totals) *(SCORE*MEAN)*F=5.1;**

**/ BOX=SCORE ROW=FLOAT MISSTEXT='None';**

Notice that since the MEAN label was blank and the ROW=FLOAT was specified, that no space was wasted for it.

Now as one final farewell to labeling, a table that doesn't look like a table.

**PROC TABULATE DATA=CLASS MISSING NOSEPS FORMCHAR=’ ;**

**CLASS ORG LOC DATE;**

**/ VAR SCORE;**

**TABLE ORG ALL= Totals --**, (LOC ALL=Row Totals) *(SCORE=MEAN) F=6.1;**

**/ BOX=SCORE ROW=FLOAT MISSTEXT='None';**

**367**
By simply adding the FORMCHAR = option to the PROC statement and specifying 16 blanks, you remove all the lines from around the table. If you have access to a laser printer you can also use characters that form "solid" lines around your table.

Subtotalling

Subtotalling is probably not a beginning tutorial topic, but here is an example anyway.

```
TABLE ORG='Loc Subtotal'; ALL='Org Total';
SCORE='Average Final Exam Score'*MEAN='F'=6.1
/ RTS=25 BOX=SCORE ROW=FLOAT MISSTEXT='None';
```

The only real trick is the nesting of ALL in the row expression.

In Summary

You have seen several reasons to use PROC TABULATE over other methods:

- displays statistics in hierarchical tables
- provides a concise and powerful control language
- provides a greater degree of flexibility and complexity in classification hierarchies than MEANS, FREQ, or SUMMARY
- provides a very flexible mechanism for titling and formatting

The base SAS statements required are:

- PROC TABULATE DATA=sasdataset options;
  options: MISSING NOSEPS FORMAT=
  ORDER = FORMCHAR=
- CLASS variables;
  classification variables: character or numeric
- VAR variables;
  analysis variables: numeric only
- KEYLABEL keyword = 'text';
- TABLE page-expression,
  row-expression,
  column-expression
  / options;
- Others used: LABEL, FORMAT, BY, TITLEn

The key to using TABULATE is the table expression, which consists of a combination of operators and operands:

- Operators
  * for nesting
  space for concatenation
- Operands
  class variables(from CLASS stmt) and ALL
  analysis variables(from VAR stmt)
- () for grouping
  statistics keywords
  title specification(= 'text')
  format specification(F=w.d)
- Standard statistics names from MEANS or SUMMARY
  N, NMISS, MEAN, STD, MIN, MAX, SUM, etc.
  Plus two new ones: PCTN and PCTSUM

Acknowledgements


SAS is a registered trademark of the SAS Institute, Inc., Cary, NC

Author

If you have any questions or comments, please write or call:
Dan Bruns
Tennessee Valley Authority
1101 Market Street (MP 28)
Chattanooga, TN 37402
(615) 751-6430 FAX: (615) 751-3163
Internet: tvawdpb.deb.office@mhs.attmail.com