PROC TABULATE Applications
Andrew A. Norton, ORI Inc.

PROC TABULATE is exceptionally flexible and powerful. Its syntax allows the specification of an endless variety of tables. The structure of the table is further specified by the input data itself -- the table becomes larger or smaller as needed, automatically splitting into multiple pages. The result is nicely formatted and a variety of labeling options is provided.

In addition to the formatting of tables, PROC TABULATE also computes the statistical results reported in the tables. The statistics supported are the same as those supported by PROC SUMMARY.

In some applications, however, PROC TABULATE cannot compute all of the statistics that we would like to present in a PROC TABULATE table. This paper will describe techniques for using PROC TABULATE to format and report statistical results obtained from other sources: other procedures, DATA steps, other statistical packages and programs, published data, and so on. In effect, the statistical computation phase of PROC TABULATE is bypassed. The reporting facilities of the procedure are thereby available for the presentation of data from any desired source. This capability substantially enlarges the realm of situations in which the flexible and powerful table formatting capabilities of PROC TABULATE can be used.

Potential Applications

The techniques described in this paper can be used to incorporate statistics such as medians and other percentiles into PROC TABULATE tables. They can also be used when PROC TABULATE supports the statistic, but not the desired weighting method. For example, variances estimated from stratified samples may be computed with a SAS program, then included in a table.

Sometimes table designs include certain cells which are derived from other cells. The most common example of this is percentages, where the denominator is computed by totaling across a group of cells. Percentages are directly supported by PROC TABULATE, but the techniques of this paper allow additional flexibility.

Another group of applications are those in which PROC TABULATE could do the job in a traditional manner, but it would be more expensive. Suppose a large dataset is to be summarized, and both a SAS dataset and PROC TABULATE table are desired as end products. PROC SUMMARY and PROC TABULATE could both be run independently, but this would entail two passes of the large dataset. A much more economical solution would be to use PROC TABULATE to format and print the dataset created by PROC SUMMARY. The second procedure would only need to process the small summarized dataset. In fact, PROC SUMMARY is more efficient than PROC TABULATE at reducing large amounts of data to summary statistics, so this method can be more efficient even when the PROC SUMMARY dataset is not of interest in itself.

Basic Method

All variables used in the TABLE statement defining a PROC TABULATE table must be declared in either the CLASS or VAR statements. Variables declared in the VAR statement are called analysis variables; variables declared in the CLASS statement are called class or classification variables.

Class variables are used to define the cells of the table. Unless the option MISSING is specified, all observations with missing values on any class variable will be excluded from the table. All other observations in the input dataset are classified into cells of the table according to their class variable values. The syntax of the TABLE statement specifies exactly one analysis variable, statistic, and format for each cell. In each cell, the statistic is computed for the analysis variable and printed using the specified format. Observations with missing values for the analysis variable do not affect any statistic except NMISS.

The examples in this paper use a small dataset of fictitious data. The SAS dataset INCOMES includes three variables. REGION has three values: 'EAST', 'MIDWEST', and 'WEST'. PARTY has the values 'DEM' and 'REP'. These character variables will be used as the class variables. In other applications, some of the class variables may be numeric, and the methods described would require slight modifications to the code.

The analysis variable used in the examples is the continuous numeric variable INCOME. Analysis variables must be numeric.
We will be presenting medians and ranges computed on the analysis variable INCOME. Neither of these statistics is supported by PROC TABULATE. The values will be computed using PROC UNIVARIATE with a BY statement, and then reported using PROC TABULATE.

Suppose we want to produce a table of median INCOME by PARTY and REGION. The key is the simple principle that the SUM or MEAN of a single value is that value itself. First we construct a dataset with one observation per cell of the final table, specifying each combination of values of the class variables PARTY and REGION, and including a new analysis variable MEDN_INC whose value is the median of INCOME for that cell. We then compute the SUM of MEDN_INC over the single observation in that cell, and the desired value will be printed in the table.

Unless otherwise directed, PROC TABULATE will label the table to indicate that we have calculated the SUM of the analysis variable MEDN_INC. We can relabel the table to give the appearance of having calculated the MEDIAN of INCOME by using the statistic and variable relabeling capabilities of the TABLE statement.

As long as the table involves only a simple nesting without any concatenations, constructing the input dataset is a simple task. It should have one observation for each cell of the final table. Observations are not needed for cells defined by combinations of class variable values that do not appear in the data.

Example 1

SAS code:

```sas
DATA INCOMES;
  INFILE RAWIN;
  INPUT PARTY $ REGION $ INCOME;
PROC SORT DATA=INCOMES OUT=SORTED;
  BY PARTY REGION;
PROC UNIVARIATE DATA=SORTED NOPRINT;
  BY PARTY REGION;
VAR INCOME;
OUTPUT OUT = TABINPUT
  MEDIAN = MEDN_INC;
PROC TABULATE DATA=TABINPUT F=9.0;
  CLASS PARTY REGION;
  VAR MEDN_INC;
  TABLE PARTY,
    REGION * MEDN_INC='INCOME' *
    SUM='MEDIAN' /* RTS=8; */
```

The Concatenation Problem

The situation becomes more complex when the table contains subtables concatenated together. Concatenations may include class variable nestings, analysis variables, statistics, and the ALL operator. In effect, concatenations direct SAS to construct multiple subtables, each computed using every observation on the dataset (except observations with missing class variables, unless option MISSING is specified.) Each subtable computes one statistic on one analysis variable within a single nesting of class variables. Additional statistic or analysis variable requests generate additional subtables, as does the ALL operator used for summary cells. The subtable structure may not always be physically obvious (because rows or columns of different subtables may be interleaved), but it is always logically present.

When PROC TABULATE is used in the traditional manner, concatenations are not a problem. There are usually many observations for each cell of each subtable. In each subtable, the observations are grouped into cells according to the class variable nesting, which often differs from subtable to subtable. The statistics are computed directly from the raw data.
The methods of this paper, however, require that the SUM statistic be computed on a single observation in each cell of the final table. If different subtables group the data in different ways, it is likely that there will not be perfect correspondence between the cells of one subtable and the cells of another. Since every observation is used in every subtable, in some cells there will be more than one observation. The values from the constructed analysis variable are summed together, and this sum is printed in the table rather than the desired result.

**Concatenations Using Multiple Analysis Variables**

The solution is based on the fact that even if some analysis variables are missing, those observations may still be used in calculations requiring only analysis variables which are not missing. If an analysis variable is missing, that observation will not affect any statistics computed on that variable (except NMISS). To do a concatenation, we construct a dataset with a separate set of observations for each subtable, using a different analysis variable for the contents of each subtable. By setting the analysis variable for a subtable to missing on observations not intended for that subtable, we can control which observations are used in which subtable. Each observation will be used in only one subtable, and will have only one non-missing analysis variable. Only one observation within each cell of the subtable will have a non-missing value for the analysis variable corresponding to that subtable.

When different class variable nestings are used in the different subtables, special treatment is required for class variables not used in every subtable. Even though an observation is intended for a subtable whose nesting does not use a particular class variable, the value of that class variable can still affect the expansion of other nestings using that variable. We therefore must avoid introducing any new values. It does not matter what value is assigned to unused class variables as long as it is an actual value for that class variable in the raw data. It is convenient to obtain a value from the first observation of a subtable which uses the class variable in its nesting. The examples below demonstrate the use of the MERGE statement to obtain such "fill" values.

This method allows complete flexibility in concatenation within a single dimension. Because analysis variables cannot appear in more than one dimension, this method cannot be used to concatenate in more than one direction at once. The other dimensions may only include a single nesting each without any concatenations. It is often useful to simulate concatenations within such nestings as described in the next section.

**Example 2**

The row definition of this table contains REGION concatenated with PARTY, using the multiple analysis variable method.

The macro %COMPUTE creates two datasets one for the REGION section of the table and one for the PARTY section of the table.

On the first pass through the DATA step that creates TABINPUT, the variables REGFILL and PRTYFILL are loaded with values for REGION and PARTY obtained from observations for subtables that actually use those variables. These fill variables are then used to provide actual data values for variables undefined in the dataset currently being read in the SET statement.

The variables VALUE_1 and VALUE_2 contain the values for the REGION and PARTY sections of the table. The SET statement in the DATA step for TABINPUT automatically sets each analysis variable to missing for all subtables except the one to which it applies.

In PROC TABULATE, note that each class nesting in the concatenation is nested with its own analysis variable. The SUM statistic has been relabeled as 'MEDIAN'. The analysis variables VALUE_1 and VALUE_2 have been relabeled as 'INCOME'. The analysis variable labels cannot be suppressed because they are in the row dimension.

**SAS code:**

```sas
%MACRO COMPUTE(DATA=_LAST_,OUT=_DATA_,
BYGROUP=,ANALYSIS=,STAT=);
PROC SORT DATA=&DATA OUT=_SORTED;
BY &BYGROUP;
RUN;
PROC UNIVARIATE DATA=_SORTED NOPRINT;
BY &BYGROUP;
VAR &ANALYSIS;
OUTPUT OUT=&OUT &STAT=VALUE;
RUN;
%MEND COMPUTE;
```
Concatenations in Additional Dimensions

Concatenations in additional dimensions may be simulated by using nestings which include constructed variables. For example, nesting a character variable having the values 'MEDIAN' and 'RANGE' with other class variables will give the appearance of a concatenation of statistics.

A variation of the same idea is to add values to an existing variable. For example, the system missing value '.' can be used to represent the ALL operator, using PROC FORMAT to relabel the values as desired. PROC SUMMARY output datasets are generated in this form and are therefore especially well suited for printing with PROC TABULATE.

This method is more limited than the multiple analysis variable method because there can be no actual concatenations. Simulated concatenations are constrained by the nesting process. The same set of class variables will be nested in each simulated subtable. Simulated concatenations are useful because they are simple to implement and can be used to give the appearance of multiple dimensions of concatenation.

Example 3

This example illustrates the simulated concatenation of two statistics. In this case, multiple analysis variables could have been used instead of the simulation because there are no true concatenations in the other dimensions.

The analysis variable VALUE has been relabeled as 'INCOME'. The labeling of the SUM statistic and the STAT class variable has been suppressed.

SAS code:

```sas
%COMPUTE (DATA=INCOMES, OUT=_STAT_1,
   BYGROUP=REGION,
   ANALYSIS=INCOME, STAT=MEDIAN)

%COMPUTE (DATA=INCOMES, OUT=_STAT_2,
   BYGROUP=REGION,
   ANALYSIS=INCOME, STAT=RANGE)

DATA TABINPUT; SET _STAT_1 (IN=IN1)
   _STAT_2;
   IF IN1 THEN STAT = 'MEDIAN';
   ELSE STAT = 'RANGE';
```

Constructed dataset:

<table>
<thead>
<tr>
<th>REGION</th>
<th>PARTY</th>
<th>VALUE 1</th>
<th>VALUE 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAST</td>
<td>DEM</td>
<td>42500</td>
<td></td>
</tr>
<tr>
<td>MIDWEST</td>
<td>DEM</td>
<td>34000</td>
<td></td>
</tr>
<tr>
<td>WEST</td>
<td>DEM</td>
<td>37500</td>
<td></td>
</tr>
<tr>
<td>EAST</td>
<td>REP</td>
<td>30000</td>
<td>38000</td>
</tr>
</tbody>
</table>

PROC TABULATE table:

```
--------------------------------------
<table>
<thead>
<tr>
<th>MEDIAN</th>
</tr>
</thead>
<tbody>
<tr>
<td>REGION</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>EAST</td>
</tr>
<tr>
<td>MIDWEST</td>
</tr>
<tr>
<td>WEST</td>
</tr>
<tr>
<td>EAST</td>
</tr>
</tbody>
</table>
```
PROC TABULATE DATA=TABINPUT FORMAT=9.0;
CLASS REGION PARTY STAT;
VAR VALUE;
TABLE
  REGION * VALUE= 'INCOME',
  STAT='MEDIAN' * RANGE='RANGE',
/RTS=18;

PROC TABULATE table:

<table>
<thead>
<tr>
<th>REGION</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>EAST</td>
<td>INCOME</td>
<td>42500</td>
</tr>
<tr>
<td></td>
<td>RANGE</td>
<td>130000</td>
</tr>
<tr>
<td>MIDWEST</td>
<td>INCOME</td>
<td>34000</td>
</tr>
<tr>
<td></td>
<td>RANGE</td>
<td>88000</td>
</tr>
<tr>
<td>WEST</td>
<td>INCOME</td>
<td>37500</td>
</tr>
<tr>
<td></td>
<td>RANGE</td>
<td>195000</td>
</tr>
</tbody>
</table>

Example 4

This example demonstrates the use of both methods of concatenation in combination. The row definition contains PARTY concatenated with REGION, using the multiple analysis variable method. The column definition simulates a concatenation of the statistics MEDIAN and RANGE, both computed on the variable INCOME.

The macro %TWO_STAT creates two datasets, one for the PARTY section of the table and one for the REGION section of the table. Each dataset contains separate observations for the MEDIAN cells and the RANGE cells, distinguished by the value of the STAT variable.

Fill values are used for the class variables PARTY and REGION. The STAT class variable does not need a FILL variable because it is used in the nesting for every subtable.

Note that the DATA step that creates the dataset TABINPUT is identical to that used in Example 2. The use of simulated concatenations does not require a change in the code constructing the true concatenation.

The labeling of the STAT class variable and the SUM statistic has been suppressed. The analysis variables have been relabeled as 'INCOME'.

SAS code:

%MACRO TWO_STAT (DATA=_LAST_, OUT=_DATA_, BYGROUP=&BYGROUP, ANALYSIS=&ANALYSIS);
%COMPUTE (DATA=&DATA, OUT=_STAT_1, BYGROUP=&BYGROUP, ANALYSIS=&ANALYSIS, STAT=MEDIAN);
%COMPUTE (DATA=&DATA, OUT=_STAT_2, BYGROUP=&BYGROUP, ANALYSIS=&ANALYSIS, STAT=RANGE);
DATA &_OUT;
  SET _STAT_1 (IN=IN1)
  _STAT_2 (IN=IN2);
  IF IN1 THEN STAT='MEDIAN';
  ELSE STAT='RANGE';
RUN;
%MEND TWO_STAT;

%TWO_STAT (DATA=INCOMES, OUT=SUBTAB_1, BYGROUP=REGION, ANALYSIS=INCOME);
%TWO_STAT (DATA=INCOMES, OUT=SUBTAB_2, BYGROUP=PARTY, ANALYSIS=INCOME);

DATA TABINPUT (DROP=PRTFILL REGFILL);
ARRAY _CLASSVR $ 7; ARRAY _FILLVR $ 7;
RETAIN PRTFILL REGFILL;
IF _N_ EQ 1 THEN
  DO / * SET FILL VARIABLES */;
    MERGE
      SUBTAB_1 (KEEP=REGION)
      SUBTAB_2 (KEEP=PARTY);
    DO OVER _CLASSVR;
      _FILLVR = _CLASSVR;
    END;
  END /* SET FILL VARIABLES */;
SET
  SUBTAB_1 (RENAME=(VALUE=VALUE_1))
  SUBTAB_2 (RENAME=(VALUE=VALUE_2));
* FILL MISSING CLASS VARIABLES;
  DO OVER _CLASSVR;
    IF _CLASSVR EQ ' ' THEN
      _CLASSVR = _FILLVR;
  END;
PROC TABULATE DATA=TABINPUT FORMAT=9.0;
CLASS REGION PARTY STAT;
VAR VALUE_1 VALUE_2;
TABLE
  REGION * VALUE_1= 'INCOME',
  PARTY * VALUE_2= 'INCOME',
  STAT='MEDIAN' * SUM='SUM',
/RTS=18;

Input dataset TABINPUT:

<table>
<thead>
<tr>
<th>REGION</th>
<th>PARTY</th>
<th>STAT</th>
<th>VALUE_1</th>
<th>VALUE_2</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAST</td>
<td>DEM</td>
<td>MEDIAN</td>
<td>42500</td>
<td>.</td>
</tr>
<tr>
<td>MIDWEST</td>
<td>DEM</td>
<td>MEDIAN</td>
<td>34000</td>
<td>.</td>
</tr>
<tr>
<td>WEST</td>
<td>DEM</td>
<td>MEDIAN</td>
<td>37500</td>
<td>.</td>
</tr>
<tr>
<td>EAST</td>
<td>DEM</td>
<td>RANGE</td>
<td>130000</td>
<td>.</td>
</tr>
<tr>
<td>MIDWEST</td>
<td>DEM</td>
<td>RANGE</td>
<td>88000</td>
<td>.</td>
</tr>
<tr>
<td>WEST</td>
<td>DEM</td>
<td>RANGE</td>
<td>195000</td>
<td>.</td>
</tr>
<tr>
<td>EAST</td>
<td>REP</td>
<td>MEDIAN</td>
<td>30000</td>
<td>.</td>
</tr>
<tr>
<td>EAST</td>
<td>REP</td>
<td>RANGE</td>
<td>38000</td>
<td>.</td>
</tr>
<tr>
<td>EAST</td>
<td>DEM</td>
<td>RANGE</td>
<td>185000</td>
<td>.</td>
</tr>
<tr>
<td>EAST</td>
<td>REP</td>
<td>RANGE</td>
<td>140000</td>
<td>.</td>
</tr>
</tbody>
</table>
PROC TABULATE table:

<table>
<thead>
<tr>
<th>REGION</th>
<th>MEDIAN</th>
<th>RANGE</th>
</tr>
</thead>
<tbody>
<tr>
<td>EAST</td>
<td>42500</td>
<td>130000</td>
</tr>
<tr>
<td>MIDWEST</td>
<td>34000</td>
<td>88000</td>
</tr>
<tr>
<td>WEST</td>
<td>37500</td>
<td>195000</td>
</tr>
<tr>
<td>PARTY</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DEM</td>
<td>30000</td>
<td>185000</td>
</tr>
<tr>
<td>REP</td>
<td>38000</td>
<td>140000</td>
</tr>
</tbody>
</table>

The `ORDER=DATA` option

When using the methods of this paper (or using PROC TABULATE in general), the `ORDER=DATA` option can often be useful. This option instructs SAS to order values of class variables in the same sequence they are encountered in the data. Special sort order observations (with all analysis variables set to missing) are often useful at the beginning of the dataset to define the order. All class variables other than the one being sequenced must be set to the first value (in the desired sort order) for that variable. The following example illustrates the use of sort order observations to order the marginals in a cross tabulation of medians.

Example 5

This cross tabulation of medians includes simulated concatenations to provide summary columns and rows. These columns and rows are identified by using the value 'ALL' for the class variables REGION and PARTY.

The medians are computed in four different executions of PROC UNIVARIATE, one for each simulated subtable. Three of these executions are generated by calls of the macro `%COMPUTE`. This macro cannot be used for computing the median of all cases, because it requires the specification of a BY group. To do this computation, PROC UNIVARIATE is specified explicitly.

The code to find fill values and assign them to class variables is only needed for the sort order observations. For data observations, the value 'ALL' is assigned to class variables over which the statistic is being summarized.

SAS code:

```sas
%COMPUTE (DATA=INCOMES,OUT=TBL10, 
            BYGROUP=REGION, 
            ANALYSIS=INCOME,STAT=MEDIAN)
%COMPUTE (DATA=INCOMES,OUT=TBL01, 
            BYGROUP=PARTY, 
            ANALYSIS=INCOME,STAT=MEDIAN)
%COMPUTE (DATA=INCOMES,OUT=TBL11, 
            BYGROUP=REGION PARTY, 
            ANALYSIS=INCOME,STAT=MEDIAN)
PROC UNIVARIATE DATA=INCOMES NOPRINT; 
  VAR INCOME; 
  OUTPUT OUT=TBL00 
    MEDIAN=VALUE;
PROC SORT DATA=TBL10 OUT=SORTREG 
  (KEEP=REGION); 
PROC SORT DATA=TBLO1 OUT=SORTPTY 
  (KEEP=PARTY); 
DATA TABINPUT (DROP = REGFILL PRTYFILL); 
  ARRAY _CLASSVR $ 7 REGION PARTY; 
  ARRAY _FILLVR $ 7 REGFILL PRTYFILL; 
  RETAIN PRTYFILL REGFILL; 
  IF _N_ EQ 1 THEN 
    DO /* SET FILL VARIABLES */;
      MERGE 
        SORTREG (KEEP=REGION) 
        SORTPTY (KEEP=PARTY); 
      DO OVER _CLASSVR; 
        _FILLVR = _CLASSVR; 
      END; 
    END /* SET FILL VARIABLES */;
  SET 
    SORTREG 
    SORTPTY 
    TBL00 (IN=IN00) 
    TBL01 (IN=IN01) 
    TBL10 (IN=IN10) 
    TBL11 (IN=IN11); 
```
* FILL MISSING CLASS VARIABLES;

IF INOO OR INOI THEN REGION = 'ALL';
IF INOO OR INIO THEN PARTY = 'ALL';

DO OVER _CLASSVR;
  IF _CLASSVR EQ ' , THEN
    _CLASSVR = _FILLVR;
END;

PROC TABULATE DATA=TABINPUT FORMAT=9.0
ORDER=DATA;
CLASS REGION PARTY;
VAR VALUE;
TABLE
  REGION,
  PARTY * VALUE='INCOME' *
  SUM='MEDIAN'
/RTS=9;

Constructed dataset:

REGION PARTY VALUE

| MIDWEST DEM | . (from SORTREG) |
| WEST DEM | . (from SORTREG) |
| EAST DEM | . (from SORTREG) |
| MIDWEST DEM | . (from SORTPTY) |
| MIDWEST REP | . (from SORTPTY) |
| ALL ALL | 36500 (from TBL00) |
| ALL DEM | 30000 (from TBL01) |
| ALL REP | 38000 (from TBL01) |
| EAST ALL | 42500 (from TBL10) |
| MIDWEST ALL | 34000 (from TBL10) |
| WEST ALL | 37500 (from TBL10) |
| EAST DEM | 25000 (from TBL11) |
| EAST REP | 50000 (from TBL11) |
| MIDWEST DEM | 30000 (from TBL11) |
| MIDWEST REP | 38000 (from TBL11) |
| WEST DEM | 40000 (from TBL11) |
| WEST REP | 35000 (from TBL11) |

PROC TABULATE table:

<table>
<thead>
<tr>
<th>PARTY</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEM</td>
</tr>
<tr>
<td>INCOME</td>
</tr>
<tr>
<td>MEDIAN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>REGION</th>
</tr>
</thead>
<tbody>
<tr>
<td>MIDWEST</td>
</tr>
<tr>
<td>WEST</td>
</tr>
<tr>
<td>EAST</td>
</tr>
<tr>
<td>ALL</td>
</tr>
</tbody>
</table>

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

The author can be contacted at:

ORI, Inc.
122 C Street, NW
Suite 250
Washington, DC 20001
(202) 737-2666