Making Sense of Proc Tabulate

Jonas V. Bilenas, JP Morgan Chase, New York, NY

ABSTRACT

The TABULATE procedure in SAS provides a flexible platform to generate tabular reports. Many beginning SAS programmers have a difficult time understanding the syntax of PROC TABULATE and as a result tend to avoid using the procedure. This tutorial will explain the syntax of PROC TABULATE for the beginning programmer and offer a framework for generating a report using PROC TABULATE. The data used in this paper represents simulated consumer credit card usage data and the code was developed using version 6.12 of SAS.

INTRODUCTION

PROC TABULATE is based on a table generation code developed by the U.S. Department of Labor. The syntax does not resemble other SAS PROCS and will appear to be cryptic to the beginning SAS programmer. To make matters worse, looking at sample TABULATE code provides no clue for the novice TABULATE user. Sample code often looks like long mathematical equations that make no sense to the individual learning TABULATE. Frustration often leads to dismissal of the procedure.

The TABULATE procedure is so flexible that there is no single correct way of generating tabular reports. The method I propose in this paper is the way I have learned TABULATE and still use it to generate custom tabular reports. I teach this method to all new users of TABULATE and find it helps them grasp the procedure. Here are my 7 steps to generate tabular reports using PROC TABULATE:

1. **Don't panic**. Relax and take a few meditative breaths.
2. **Design the report on paper.** The design of the TABULATE report should be first specified on paper. The report design that is developed on paper will guide the code genereation.
3. **Generate the initial test code.** Return to the computer and start coding the design that you have formulated on paper. There are a few rules of TABULATE syntax that will generate the report and we will review these here.
4. **Test, retest and verify using a small sample.** Test the initial code using a small number of observations. Use a random sample or use an OBS= option to test the syntax. TABULATE is tricky and you may have to run a few versions of your code and possibly include some DATA preparation. Verify that the results make sense. Look at your data and report! Don't assume the report is correct just because you have no syntax errors. At this step, don't be too concerned with report appearance. Verify that the results are correct.
5. **Clean up the appearance of the report.** Once the code has generated a report that makes sense, clean up the output of the report using a few tricks we will review here. Also consider adding additional summaries and or statistics to make the report even more useful. This may be difficult to do if the report has already been specified, but you can recommend changes to the specifications.
6. **Run code with OBS=MAX.**
7. **Sit back, smile and be proud of your report.** Your manager may be confused at how you were able to produce the report without sorting the data or without any spreadsheet crunching. He/She may be confused at your code, but then you can take him/her through the 7 steps of TABULATE harmony.

STEP 1 – DON'T PANIC

SAS code generation can be frustrating to beginning SAS users. To begin a TABULATE code generation, get into a relaxed and calm mode. The design and code may take a few iterations, but that should be expected with TABULATE. Table generation and formatting will take a bit of code work and re-work. It is to be expected that the first TABULATE code generated is not the final version.

STEP 2 – DESIGN THE REPORT ON PAPER

The structure of the report must first be specified in order to generate the TABULATE code that will produce the report. Designing the report on paper will help you identify which variables to use, how variables are defined in TABULATE and what summary information is required.

As an illustration for this paper, take a simple example from an account management experiment in consumer credit. A treatment is applied to existing accounts (test group) with a hold out sample (control group) set aside to compare against. The design sample also includes a factor for credit score levels (low risk, medium risk, high risk) that are recorded at time of experimentation.

After the test is completed and results are available, management is looking for some quick reports to read results. The first report specified is a simple report that could be generated using PROC FREQ, but we will code it up in TABULATE since management still has not finalized report design. Using PROC TABULATE will allow us to modify the code to accommodate further design specifications. The initial design on paper look as follows:

Treatment Group	Total Accounts	Delinquent Accounts
Test		
Control		
All		

Variables we have in our SAS DATASET are:

BAD:	1 if account is delinquent, 0 if not.
TEST:	1 if in test group, 0 if in control
SCORE:	Integer credit score with values specifying risk level: 370-600 = high risk 601-700 = medium risk 701-870 = low risk

To proceed to the next step, we have to familiarize ourselves with some syntax rules of PROC TABULATE.

STEP 3 – GENERATE THE INITIAL TEST CODE

There are few syntax rules that one has to get familiar with in order to use PROC TABULATE. The first step is to identify if variables are classified as "CLASS" variables or "VAR" variables.

CLASS variables are those that are categorical and/or have levels that you wish to generate summaries for each level. CLASS variables can be either numeric or character.

VAR variables identify analysis variables that you wish to report statistics for. A variable cannot be listed as both a CLASS and a VAR variable in the TABULATE procedure. A

VAR statement is not required in the TABULATE procedure.

Table dimensions are specified in the TABLE statement of PROC TABULATE. Table dimensions are separated by commas. This is different from PROC FREQ where an asterisk is used to separate table dimensions.

If the TABLE statement does not include any commas, the resulting report will be a single row with multiple column output. A TABLE statement with one comma produces a row by column report with the row dimension specified first. The maximum number of commas is 3, producing a page by row by column report.

The restriction on the number of commas does not limit the number of CLASS variables you can include in your report. Further crossing of CLASS variables can be introduced with an asterisk. The asterisk is also used to specify VAR variables, statistical summaries and/or format of output.

Statistical summaries are similar to the ones available in PROC MEANS. These include sums, means, number of records, etc. that one sees in other PROCS. Version 6.12 does not include percentile statistics, but these are available in Version 8.

The TABULATE procedure also includes a PCTN and a PCTSUM statistical summary. Syntax may be difficult to grasp for these summaries providing more reason to highlight the importance of STEP 4 (test and retest). PCTN calculates the percent of a frequency and PCTSUM calculates the percent of a SUM. These require denominator definitions that are specified between brackets <>.

So now we have to worry about CLASS and VAR variables on top of considering where to use commas, asterisks and brackets. And, to make it even more confusing we can even include an ‘=’ within the body of the TABLE statement to specify labels to be printed. Wow, no wonder TABULATE code can look cryptic. After much practice, these will become second nature. In the meantime, follow these guidelines:

- **CLASS:** Used for categorical variables or variables for which you want to see summaries for each value of the variable. Default statistic applied to a class variable if none provided is N (number of observations). A PCTN statistic can also be specified to produce frequency percents.
- **VAR:** Used to specify analysis variables that you want generate statistics for each level of CLASS variables. The default statistic if none specifies is SUM.
- **,:** Used for dimension splits; page, row, column. Dimension splits do not have to be limited to CLASS variables. You can, for example, have a CLASS dimension as your row dimension and a VAR variable as your column dimension.
- ***:** Use to specify any of the following:
  - Another CLASS variable Split
  - A VAR variable
  - A statistic
  - A Format
- **<>:** Specify the denominator CLASS dimensions for PCTN or to specify denominator VAR variable when using the PCTSUM statistic.
- **=** For labels specification that will improve the readability of the report.

Let’s try some code using our hypothetical example. Both variables can be specified as CLASS variables. However, we may want to report some additional statistics on our BAD variable such as a bad rate, or maybe even a ‘good to bad’ odds ratio. Initial code is included here:

```
proc tabulate data=nesug01.data
     formchar=’           ‘;
  class test;
  var bad;
    table test
          ,
          n
          bad;
run;
```

Note that the table statement can be listed on one line, but listing variables and dimensions on separate lines can aide in debugging of code. Output form this code and modifications are reviewed in Step 4 (note: the formchar=’ ‘ (11 spaces) options removes boxes around table cells making it readable for those without SAS MONOSPACE fonts).

STEP 4 – TEST, RETEST AND VERIFY USING A SMALL SAMPLE

This step often requires a number of iterations and tests until the table generates results that we expect. It may take a number of runs to get a table that has all the statistics that one requires. During this step we can make changes in the table output and we may wish to redraw each modification on paper to get a clear picture of how to proceed in code generation.

Output form the run is:

```
                                           BAD

                              N            SUM

TEST

0                       1545.00         369.00

1                      14129.00        2323.00
```

Not bad, but we would like row totals on the report as well. PROC TABULATE has an ALL keyword that we can provide totals for all levels of the CLASS variable. The code is modified along with the output:

```
proc tabulate data=nesug01.data
      formchar=’           ‘;
  class test;
  var bad;
    table test all
          ,
          n
          bad;
run;
```

```
                                           BAD

                              N            SUM

TEST

0                       1545.00         369.00

1                      14129.00        2323.00

ALL                    15674.00        2692.00
```

Looking better, but can we add a row percent for account totals and a bad rate statistic (bad total/N)?

The PCTN statistic generates the required row percent. This statistic also requires a denominator definition that is placed inside the <>. Setting up this definition can be complicated and may require a number of iterations to get the correct output. The values entered inside the <> are class variables that make up the dimension that one wishes to see a percentage. The row dimension is required to see row percents and column dimensions are required to see a column percents. Sounds confusing so keep testing until the numbers look correct. If you expect row percents, verify that the numbers reported are indeed row percents.

Use the PCTSUM statistic to get the bad rates (MEAN can also generate bad rates as well for this example but I have always used the PCTSUM calculation). We need to specify the VAR variable that identifies the denominator. This may require some DATA step preparation if the denominator variable is not included in the DATA. Bad Rate is defined as:

(sum of bads)/(sum of all accounts).

The code required for these changes are:

```
data stuff;
  set nesug01.data;
  noa=1;

proc tabulate data=stuff noseps
      formchar=’           ‘;
  class test;
  var bad noa;
    table test all
          ,
          n
          pctn<test all>
          bad
          bad*pctsum<noa>
          /rts=10;
run;
```

This code also includes a NOSEPS option. This option prevents separate lines (or line breaks) between levels of row CLASS variables. I also included an RTS= option for the TABLE statement. TABULATE reserves a quarter of the page for the row dimension labels. The RTS=10 specifies that only 10 spaces are to be used to generate the width of the row dimension (including vertical separation lines). Output is listed:

```


                                             BAD          BAD

              N           PCTN               SUM          PCTSUM

 TEST
 0             1545.00          9.86          369.00
23.88
 1            14129.00         90.14         2323.00
16.44
 ALL          15674.00        100.00         2692.00
17.17
```

STEP 5 – CLEAN UP THE APPEARANCE OF THE REPORT

After the report has been validated, we can clean up the report presentation. There are a number of tricks to make the report look professional and they are described with examples. Things we would like to change in the above report are:

1. More descriptive labels that will make the report easier to read.
2. Specific formats for the numeric output.
3. Change the levels of the CLASS variable to read “TEST” and “CONTROL” rather than numeric values of 0 and 1.
4. List the “Test” level before the “Control” level.

These changes are easily added to the code. Continue to work with small sample datasets to minimize run time and computer costs. The code and output is listed followed by an explanation of the methods used to format the output:

```
proc format;
  value test 0 = 'Control'
             1 = ' Test'
  ;
  picture p7r (round) 0-100 = '009.00%'
  ;
  picture p6r (round) 0-100 = '09.99%'
  ;

proc tabulate data=stuff noseps
              formchar=’           ‘
              order=formated;
  class test;
  var bad noa;
  format test test.;
    table test='Test/Cntl Group'
          all='Total'
          ,
          n='Number of Accounts'*f=comma8.
          pctn<test all>='Row Percent'*f=p7r.
          bad='Number of Delinquent
               Accounts'*sum=' '*f=comma10.
          bad='Bad  Rate '*
               pctsum<noa>=' '*f=p6r.
          /rts=17;
run;
```

```
                   Number
                     of      Row   Number of   Bad
                  Accounts Percent Delinquent  Rate

Test/Cntl Group
Test                14,129  90.14%      2,323 16.44%
Control              1,545   9.86%        369 23.88%
Total               15,674 100.00%      2,692 17.17%
```

The output looks much neater and is ready to be placed in production. The coding tricks used are summarized:

1. Use PROC FORMAT to assign values to CLASS variables if required. Some of the format labels can include spaces so that results can be ordered by FORMAT labels. Specify ORDER=FORMATED option in TABULATE to order output by FORMATED labels as opposed to ordered by data values.
2. Set up a PICTURE FORMAT to print TABULATE PCTN and PCTSUM’s with trailing % signs. Using a percent format within TABULATE will not work since the values are not outputted as proportions.
3. Specify specific formats for statistics within the TABLE statement using *F=format_name.
4. Specify formats for CLASS variables with a FORMAT assignment statement in PROC TABULATE.
5. Assign labels after variables and statistics in the TABLE statement with the use of = specification. If you do not want a default label printed (i.e., SUM, PCTSUM) us a null label (= ‘ ‘).
6. Adjust label spaces and format widths to generate a report that is presentable. Tabulate will split labels depending on width of format. If format is too short, the words contained in your labels may split before the word end. Keep testing with small datasets by adjusting format widths and TABLE labels until the desired output is obtained.

Adding additional CLASS variables can be added as page breaks or cross breaks within other CLASS variables. Here is some code and output for different possibilities we have available when we add another CLASS variable. For this example we have the credit score that we can set up levels for by use of PROC FORMAT and specify the variable as a CLASS variable in the TABLE statement of PROC

TABULATE. Use of PROC FORMAT is more desirable than creating levels in a DATA step since it is faster, requires no additional DATA generation or modification, and is less likely to result in assignment errors. The first example runs ‘risk’ level as a page dimension. Output is shown after the code.

```
proc format;
  value test 0 = 'Control'
             1 = ' Test'
  ;
  value score 370-600  = '  High '
              601-700  = ' Medium'
              701-high = 'Low'
  ;
  picture p7r (round) 0-100 = '009.00%'
  ;
  picture p6r (round) 0-100 = '09.99%'
  ;

proc tabulate data=stuff noseps order=formatted
     formchar=’           ‘;
  class test score;
  var bad noa;
  format test test. score score.;
    table score='Risk = '
          all='All Risk Levels'
          ,
          test='Test/Cntl Group'
          all='Total'
          ,
          n='Number of Accounts'*f=comma8.
          pctn<test all>='Row Percent'*f=p7r.
          bad='Number of Delinquent Accounts'
               *sum=' '*f=comma10.
          bad='Bad  Rate '*pctsum<noa>=' '*f=p6r.
          /box=_page_ rts=17 misstext=' ' condense;
run;
```

OUTPUT:

```
Risk = High      Number          Number of
                    of      Row   Delinquent  Bad
                 Accounts Percent  Accounts   Rate

 Test/Cntl Group
 Test               8,462  90.35%      1,727 20.41%
 Control              904   9.65%        253 27.99%
 Total              9,366 100.00%      1,980 21.14%


 Risk = Medium    Number          Number of
                    of      Row   Delinquent  Bad
                 Accounts Percent  Accounts   Rate

 Test/Cntl Group
 Test               1,436  90.37%        170 11.84%
 Control              153   9.63%         33 21.57%
 Total              1,589 100.00%        203 12.78%


 Risk = Low       Number          Number of
                    of      Row   Delinquent  Bad
                 Accounts Percent  Accounts   Rate

 Test/Cntl Group
 Test               4,231  89.66%        426 10.07%
 Control              488  10.34%         83 17.01%
 Total              4,719 100.00%        509 10.79%


 All Risk Levels  Number          Number of
                    of      Row   Delinquent  Bad
                 Accounts Percent  Accounts   Rate

 Test/Cntl Group
 Test              14,129  90.14%      2,323 16.44%
 Control            1,545   9.86%        369 23.88%
 Total             15,674 100.00%      2,692 17.17%
```

For the output generated, a number of TABLE options were also introduced. The TABLE OPTIONS included are:

1. **BOX=_PAGE_** prints out the value of the page dimension in the upper left hand corner box of the table.
2. **MISSTEXT=** sets missing text to the text specified in the label. In this case the null label was specified so that if the report results in a missing value, a blank is printed rather than SAS MISSING values.
3. **CONDENSE** option causes multiple tables to be fit on one page if space permitting. Without the option, each page dimension prints out a separate report.

The ‘risk’ CLASS variable can be specified as a row dimension and then cross each level of ‘risk’ by the levels of the ‘test’ CLASS variable. The TABULATE code that generates this table design is provided along with output.

```
proc tabulate data=stuff noseps order=formatted
     formchar='           ';
 class test score;
 var bad noa;
 format test test. score score.;
  table (score='Risk'
        all='All Risk Levels'
        )
        *
        (test='Group '
        all=' '
        )
        ,
        n='Number of Accounts'*f=comma8.
        pctn<test all>='Row Percent'*f=p7r.
        bad='Number of Delinquent Accounts'
             *sum=' '*f=comma10.
        bad='Bad  Rate '*pctsum<noa>=' '*f=p6r.
        /box=_page_ rts=21 misstext=' ' condense;
run;
```

```
                   Number          Number of
                     of      Row   Delinquent  Bad
                  Accounts Percent  Accounts   Rate

Risk      Group
High      Test       8,462  90.35%      1,727 20.41%
          Control      904   9.65%        253 27.99%
                     9,366 100.00%      1,980 21.14%
Medium    Group
          Test       1,436  90.37%        170 11.84%
          Control      153   9.63%         33 21.57%
                     1,589 100.00%        203 12.78%
Low       Group
          Test       4,231  89.66%        426 10.07%
          Control      488  10.34%         83 17.01%
                     4,719 100.00%        509 10.79%
All Risk  Group
Levels    Test      14,129  90.14%      2,323 16.44%
          Control    1,545   9.86%        369 23.88%
                    15,674 100.00%      2,692 17.17%
```

For the above table, we needed to place parentheses around SCORE ALL and TEST ALL specifications since we wanted to see each level of TEST ALL for each level of SCORE and ALL.

For more complicated designs, remember to always design the table on paper first. This will ease the coding up of complicated reports and give you more confidence in designing reports with PROC TABULATE.

Let us look at one more table design and another DATA step trick. A number of statisticians look at odds ratios rather then percentages. Version 6.12 does not have a ratio statistic. Here is code that fools TABULATE to provide an

odds ratio in the output. The definition of the odds ratio here is:
total good accounts/total bad accounts
For this example the definition of good is an account that has not become delinquent over the test evaluation period.

The code and output follow (note that FORMAT code is not included here since it is assumed the FORMATS have been created):

```
data sushi;
  set stuff;
  if bad=0 then good=1/100;
  else good=0;

proc tabulate data=sushi noseps order=formated
     formchar='           ';
  class test score;
  var bad noa good;
  format test test. score score.;
    table (score='RISK LEVEL'
          all='Total'
          )
          ,
          (test='GROUP '
          all='Total'
          )
          *
          good=' '*pctsum<bad>=' '*f=8.2
          /box='Good/Bad Odds' rts=21 misstext=' ';
run;
```

OUTPUT:

```
Good/Bad Odds             GROUP

                      Test   Control   Total

RISK LEVEL
High                    3.90     2.57     3.73
Medium                  7.45     3.64     6.83
Low                     8.93     4.88     8.27
Total                   5.08     3.19     4.82
```

The trick above was to divide the numerator in the PCTSUM calculation by 100 in the DATA step. Since TABULATE always multiplies the result of a PCTSUM by 100, the resulting output is a ratio. Now if I can only figure out how to report the log of odds, I will become a TABULATE master.

Note that the 'test' CLASS variable is now in the column dimension. We put parentheses around TEST and ALL since we wanted the PCTSUM summary for all levels of TEST as well as ALL. The parentheses around SCORE and ALL are not required here, but they don't impact the output.

STEP 6 - RUN CODE WITH OBS=MAX

Run production code on full datasets. Always test TABULATE code on smaller datasets. This is more efficient and time saving especially for large datasets.

STEP 7 - SIT BACK, SMILE AND BE PROUD OF YOUR REPORT

Once your TABULATE report is generated, feel confident that you have begun your trip to understanding the mystery of PROC TABULATE. Your trip will be a long one, grasshopper, but it will be a full of enlightenment as you become a TABULATE Master.

CONCLUSION

The TABULATE procedure is a complex PROC that generates tabular reports. There is no single correct way to generate reports in TABULATE. This paper provided a framework that will help you understand the workings of TABULATE and will provide a starting framework on how to generate table reports using PROC TABULATE.

Often reports can be generated without additional DATA steps. The use of PROC FORMAT can assist in labeling output. DATA steps may be required to generate additional variables for TABULATE output and/or modify variables to generate statistics required in your output.

Version 8+ of SAS has many more options and statistical reporting capabilities. These include ODS, OUTPUT of SAS DATASETS, and percentile statistics.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Jonas V. Bilenas
JP Morgan Chase
Decision Sciences
55 Water Street, 1618
New York NY 10041
Email: Jonas.Bilenas@Chase.com