Paper S3

Fit Your Data into the FREQ Procedure and More

Dachao Liu, Northwestern University, Chicago, IL

ABSTRACT

Normally, the FREQ procedure with the WEIGHT statement can take a data set with at least three variables and four observations in each run to produce a two-way table. But sometimes the data set is so formed that each time we have to use the FREQ procedure with the WEIGHT statement to pick up only two observations and three or four variables from that data set. And that data set can have hundreds of thousands of observations. Because of this, we have to make use of the index variables in the TITLE statement to track which block the data in question is referred to. This paper will discuss how to use proper data steps and the FREQ procedure with the WEIGHT statement to run that kind of data set. This technique is so useful that it may be developed into a NEW statement like the WEIGHT statement itself to be used in the FREQ procedure. This paper is intended for those familiar with the data step like array processing and prior experience with SQL to create a macro variable.

INTRODUCTION

SAS users are very familiar with the FREQ procedure with the WEIGHT statement. It's the one of the easiest procedures to use next to the PRINT procedure when the data set is in shape. But often time, the data set is not in shape. We have to fit the data into the FREQ procedure with the WEIGHT statement. If the data set has the index information, we can make use of the index variables to identify which block of data we are using.

DISCUSSION

The FREQ procedure produces one-way to *n*-way frequency and crosstabulation (contingency) tables. For *n*-way frequency and crosstabulation (contingency) tables, You can see my paper in MUSUG 2006 proceedings, where specifically a 10-way frequency and crosstabulation (contingency) table was discussed. This paper only deals with a two-way crosstabulation table. For a two-way table, the FREQ procedure provides an easy way for testing for association of variables in the table.

Normally we have the data like this

VAR1	VAR2	COUNT
a	c	n1
a	d	n2
b	c	n3
b	d	n4

The variable COUNT contains the number corresponding to each value combination of VAR1 and VAR2. Specifically we have the data named `school` like this

GROUP	SEX	COUNT
T	M	30
T	F	15
C	M	65
C	F	38

We can easily use the FREQ procedure with the WEIGHT statement to do the work.

```
proc freq data=school order=data;
weight COUNT;
tables GROUP*SEX / chisq;
run;
```

Since the input data are in cell count form, the WEIGHT statement is required. The WEIGHT statement specifies a numeric variable with a value that represents the frequency of the observation. It is most commonly used to input cell count data. The WEIGHT statement names the variable COUNT, which provides the frequency of each value combination of GROUP and GENDER.

But sometimes we might have data sets that we can not use the FREQ procedure with the WEIGHT statement directly. Here is what the data might look like

GROUP	M	F
T	30	15
C	65	38

Instead of four observations, we have two observations here. In order to use the FREQ procedure with the WEIGHT statement, we have to reshape the data.

This is the program to do the work

```
data one;
input M F;
cards;
30  15
65  38
;
data many;
   set one;
    array mf[*] M F;
  do I = 1 to dim(MF);
        COUNT = mf[I];
          output;
      end;
keep  I COUNT;
rename  I=SEX;
run;

data two;
input GROUP $ @@;
cards;
T T C C
;
proc format;
value SEX
1='M'
2='F';
run;

data three;
merge two many;
format SEX SEX.;
run;
```

Notice that merging without a BY statement is done by observation.

After the data steps, we use the FREQ procedure with the WEIGHT statement.

```
proc freq data=three; weight COUNT;
table GROUP*SEX/ chisq;
run;
```

USING ARRAYS TO RESTRUCTURE A SAS DATA SET

Here an array is used. Arrays in SAS are often used with do loops, to carry out repeated actions on a sequence of variables. An array has name and subscript. It only exists in the data step. It begins with the array statement to define an array: a set of elements that can be processed as a group. The elements of the array can be referenced by the array name and subscript and must be either all numeric or all character, in other words, they must be the same data type. The array name is not a variable. The variables referenced by the array are called elements. Once an array is defined, the array name and an index reference the elements of the array. The value of a subscript will determine the size of the array. When the asterisk is used in place of a subscript, the DIM function can determine the size of the array.

The use of arrays simplifies the data processing, which helps us to deal with repetitive data with a minimum of coding.

With the help of arrays, the data reshaping is simple, because it only has one block of data. How about we have many, many blocks of data with more variables? This paper will further discuss more complicated situation like this.

Suppose we have the data set SURVEY, part of which is like this:

```
disagree agree  totad  question source prepost;
 103       197    300     3        1       2
 354       484    838     3        1       1
  10        22     32     3        2       2
  78       102    180     3        2       1
   4         5      9     3        3       2
   0         0      0     3        3       1
  19        38     57     3        4       2
  56        69    125     3        4       1
  12        32     44     3        5       2
  75        86    161     3        5       1
   1         3      4     3        6       2
   0         0      0     3        6       1
...
```

This data SURVEY can have hundreds of thousands of observations. But first we deal with first two rows

```
103   197   300   3     1     2
354   484   838   3     1     1
```

The values for the count variable

```
103   197
354   484
```

We also need values for the index variables

```
3     1
3     1
```

to go to the title statement to distinguish one block from the other. Using the method described above, we can reshape the data like this:

```
PREPOST      AGR      COUNT      QUESTION        SOURCE

 POST      DISAGRE     103     03:Question 3    1:OVERALL
 POST      AGREE       197     03:Question 3    1:OVERALL
 PRE       DISAGRE     354     03:Question 3    1:OVERALL
 PRE       AGREE       484     03:Question 3    1:OVERALL
```

Then we use

```
proc freq data=three; weight COUNT;
title1 'Data Three';
table PREPOST*AGR/ exact;
title2 "question=&QUESTION  souce=&SOURCE";
run;
```

We will get this output

```
                 Data Three
        question=03:Question 3   souce=1:OVERALL

                 The FREQ procedure

              Table of PREPOST by AGR

          PREPOST     AGR

          Frequency|
          Percent  |
          Row Pct  |
          Col Pct  |DISAGRE |AGREE    |  Total
          ---------+--------+--------+
          POST     |    103 |    197 |    300
                   |   9.05 |  17.31 |  26.36
                   |  34.33 |  65.67 |
                   |  22.54 |  28.93 |
          ---------+--------+--------+
          PRE      |    354 |    484 |    838
                   |  31.11 |  42.53 |  73.64
                   |  42.24 |  57.76 |
                   |  77.46 |  71.07 |
          ---------+--------+--------+
          Total         457      681     1138
                      40.16    59.84   100.00


          Statistics for Table of PREPOST by AGR

 Statistic                     DF       Value      Prob
 ------------------------------------------------------
 Chi-Square                     1      5.7519    0.0165
 Likelihood Ratio Chi-Square    1      5.8234    0.0158
 Continuity Adj. Chi-Square     1      5.4274    0.0198
 Mantel-Haenszel Chi-Square     1      5.7468    0.0165
 Phi Coefficient                      -0.0711
 Contingency Coefficient               0.0709
 Cramer's V                           -0.0711


                Fisher's Exact Test
           ----------------------------------
           Cell (1,1) Frequency (F)       103
           Left-sided Pr <= F          0.0096
           Right-sided Pr >= F         0.9934

           Table Probability (P)       0.0030
           Two-sided Pr <= P           0.0165

                 Sample Size = 1138
```

Please notice that the values of the index variables were passed through macro variables created in the following procedure

```
proc sql noprint;
   select QUESTION, SOURCE
      into :QUESTION, :SOURCE
      from three;
quit;
```

USING MACRO VARIABLES TO IDENTIFY EXTRA INFORMATION

Here macro variables help us to display extra information in the data set. There are many ways of creating macro variables, one of them being like %let x=1 shown below. In this situation, however, PROC SQL is the most convenient way to create macro variables, because we can revoke them and display their values in the title or a footnote if we like.

The INTO clause on the SELECT statement creates and updates macro variables from the column values in the first row of a table taken by PROC SQL during the execution of PROC SQL. The macro variables are stored in the global macro symbol table because PROC SQL here is not submitted from within a macro program. The macro variable names are preceded with colons (:), not ampersands(&). Macro variables are explicitly named on the INTO clause. Their values can be seen in the SAS log by issuing the %PUT statement. Output results are suppressed with the NOPRINT option here.

After dealing with the first two rows, we can deal with the third and fourth rows and nth and nth +1 rows in the same way by using statement if _n_ in (&x,&y) then output. The whole program is as follows:

```
%let x=1;
%let y=2;

data SURVEY0(rename=(disagree=D agree=A totad=T  question=Q source=S prepost=P));
set SURVEY; if _n_ in (&x,&y) then output; run;

data SURVEY1;
set SURVEY0;
keep D A;
run;
data SURVEY2;
set SURVEY0;
keep Q S;
run;

data SURVEY3;
set SURVEY2 SURVEY2;
run;

data many;
set SURVEY1;
    array da[*] D A;
  do I = 1 to dim(da);
        COUNT = da[I];
          output;
      end;
keep I COUNT;
rename I=AGR;
run;

data two;
input PREPOST $ @@;
cards;
POST POST PRE PRE
;
proc format;
value AGR
1='DISAGRE'
2='AGREE';
```

```
value question
1='01:Question 1'
2='02:Question 2'
3='03:Question 3';

value source
1='1:OVERALL'
2='2:NURSE'
3='3:ORT'
4='4:SURG RES'
5='5:SURG ATT'
6='6:CRNA';
run;

data three;
merge two many SURVEY3;
format AGR AGR. Q question. S source.;
run;

proc sql noprint;
   select Q, S
      into :Q, :S
      from three;
quit;

proc freq data=three; weight COUNT;
title1 'Data Three';
table PREPOST*AGR/exact;
title2 " question=&Q  souce=&S";
run;
```

This program can be put into a macro. You can delve into it if you are interested.

CONCLUSION

When a data set is not in shape for PROC FREQ procedure with the WEIGHT statement, we can reshape it , often using arrays, to fit it into PROC FREQ procedure which takes a data set with at least three variables and four observations in each run to produce a two-way table. And we can also use extra information in the data set, if any, to indicate which block of data we are referring to, by using macro variables created in PROC SQL procedure. Since the WEIGHT statement empowers PROC FREQ procedure to deal with a data set with at least three variables and four observations, another NEW statement may be developed to empower PROC FREQ procedure to deal with a data set with two observations and three or four variables by using the technique discussed in this paper.

REFERENCES

Dachao Liu, 2006 *How to Use Summary Statistics as Raw Data to Do Basic Statistical Analysis* presented at the MWSUG 2006 Conference (October 2006) and published in Proceedings of the MWSUG 2006 Conference.

CONTACT INFORMATION

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

Dachao Liu
Northwestern University
Suite 1102
680 N Lake Shore Dr.
Chicago, IL 60611
Phone (312)503-2809
Email: dachao-liu@northwestern.edu