

Paper PO–133

PROC MEANS for Disaggregating Statistics in SAS®: One Input Data Set and One Output Data Set with Everything You Need

Imelda C. Go, South Carolina Department of Education, Columbia, SC
Abbas S. Tavakoli, College of Nursing, University of South Carolina–Columbia

ABSTRACT

The need to calculate statistics for various groups or classifications is ever present. Calculating such statistics may involve different strategies with some being less efficient than others. A common approach by new SAS programmers who are not very familiar with PROC MEANS is to create a SAS data set for each group of interest and to execute PROC MEANS for each group. This strategy can be resource-intensive when large data sets are involved. It requires multiple PROC MEANS statements due to multiple input data sets and involves multiple output data sets (one per group of interest). In lieu of this, an economy of programming code can be achieved using a simple coding strategy in the DATA step to take advantage of PROC MEANS capabilities. Variables that indicate group membership (1 for group membership, blank for non-group membership) can be created for each group of interest in a master data set. The master data set with these blank/1 indicator variables can then be processed with PROC MEANS and its different statements (i.e., CLASS and TYPES) to produce one data set with all the statistics generated for each group of interest.

A programmer can calculate disaggregate statistics in a number of different ways. A common approach by new SAS programmers who are not very familiar with PROC MEANS is to create a SAS data set for each group of interest and to execute PROC MEANS for each group. This strategy can be resource-intensive when large data sets are involved. It requires multiple PROC MEANS statements due to multiple input data sets and involves multiple output data sets (one per group of interest).

```
data master;
input gender $ score;
cards;
M 30
F 100
;

data males;
set master (where=(gender='M'));

data females;
set master (where=(gender='F'));

proc means data=males;
output out=statsfemales;

proc means data=females;
output out=statsforfemales;
```

You could save a little code by using the WHERE statement with PROC MEANS. By doing this, you do not create a data set for each group of interest. However, that still leaves you with more than one data set of statistics.

```
proc means data=master;
output out=statsfemales;
where gender='M';

proc means data=master;
output out=statsforfemales;
where gender='F';
```

METHOD

The following discussion will show you how to create blank/1 indicator variables for any group of interest in your master data set. Using this single master data set as input, you can obtain a single data set with the disaggregate statistics according to these groups of interest from a single PROC MEANS statement.

Consider the following data set.

| Obs | gender | meals | rawscore | scalescore |
|-----|--------|-------|----------|------------|
| 1 | M | F | . | . |
| 2 | M | R | 5 | 40 |
| 3 | M | P | 10 | 100 |
| 4 | F | F | 5 | 40 |
| 5 | F | R | 10 | 100 |
| 6 | F | P | 5 | 40 |
| 7 | F | R | 10 | 100 |
| 8 | F | P | 5 | 40 |
| 9 | | R | 10 | 100 |
| 10 | M | | 5 | 40 |

The following needs to be determined for the raw and scale scores:

- Average
- Total number of observations (i.e., denominator of the average) with a score

According to the following groups:

1. Males (gender = M)
2. Females (gender = F)
3. Students receiving free meals (meals = F)
4. Students receiving reduced-price meals (meals = R)
5. Students paying full price for meals (meals = P)

These results can be produced by the following code. The TYPES statement specifies which combinations of the class variables are to be used for the calculations.

```
proc means data=example;
  class gender meals;
  var rawscore scalescore;
  types gender meals;
  output out=test n=nrs nss mean=mrs mss;
```

The MEANS Procedure

| meals | N Obs | Variable | N | Mean | Std Dev | Minimum | Maximum |
|-------|----------|------------|---|------------|------------|------------|-------------|
| F | 2 | rawscore | 1 | 5.0000000 | . | 5.0000000 | 5.0000000 |
| | | scalescore | 1 | 40.0000000 | . | 40.0000000 | 40.0000000 |
| P | 3 | rawscore | 3 | 6.6666667 | 2.8867513 | 5.0000000 | 10.0000000 |
| | | scalescore | 3 | 60.0000000 | 34.6410162 | 40.0000000 | 100.0000000 |
| R | 3 | rawscore | 3 | 8.3333333 | 2.8867513 | 5.0000000 | 10.0000000 |
| | | scalescore | 3 | 80.0000000 | 34.6410162 | 40.0000000 | 100.0000000 |

| gender | N Obs | Variable | N | Mean | Std Dev | Minimum | Maximum |
|--------|----------|------------|---|------------|------------|------------|-------------|
| F | 5 | rawscore | 5 | 7.0000000 | 2.7386128 | 5.0000000 | 10.0000000 |
| | | scalescore | 5 | 64.0000000 | 32.8633535 | 40.0000000 | 100.0000000 |
| M | 3 | rawscore | 2 | 7.5000000 | 3.5355339 | 5.0000000 | 10.0000000 |
| | | scalescore | 2 | 70.0000000 | 42.4264069 | 40.0000000 | 100.0000000 |

The corresponding output data set from PROC MEANS is shown below.

| Obs | gender | meals | _TYPE_ | _FREQ_ | nrs | nss | mrs | mss |
|-----|--------|-------|--------|--------|-----|-----|---------|-----|
| 1 | | F | 1 | 2 | 1 | 1 | 5.00000 | 40 |
| 2 | | P | 1 | 3 | 3 | 3 | 6.66667 | 60 |
| 3 | | R | 1 | 3 | 3 | 3 | 8.33333 | 80 |
| 4 | F | | 2 | 5 | 5 | 5 | 7.00000 | 64 |
| 5 | M | | 2 | 3 | 2 | 2 | 7.50000 | 70 |

The `_TYPE_` variable indicates which CLASS variables produced the data. Although it can be clearly seen from the output, as shown above, which levels of each CLASS variable were represented for each line in the data set, the CLASS variable configuration can be used to determine the `_TYPE_` value. The following example involves two CLASS variables, but the concept easily extends to however many CLASS variables there are.

1. Because there are two CLASS variables, think of a binary number that has two digits. The decimal equivalents of the binary numbers can be computed as shown.

| Binary Number | Decimal Equivalent |
|---------------|-----------------------|
| 00 | $0(2^1) + 0(2^0) = 0$ |
| 01 | $0(2^1) + 1(2^0) = 1$ |
| 10 | $1(2^1) + 0(2^0) = 2$ |
| 11 | $1(2^1) + 1(2^0) = 3$ |

2. Gender is the first CLASS variable listed. Use a 0/1 variable based on `gender` as the first digit in the binary number. The digit is:
 - 0 whenever `gender` was not used towards the calculations
 - 1 whenever `gender` was used towards the calculations
3. Meals is the second CLASS variable listed. Use a 0/1 variable based on `meals` as the second digit in the binary number. The digit is:
 - 0 whenever `meals` was not used towards the calculations
 - 1 whenever `meals` was used towards the calculations
4. Use the digits from steps #2 and #3 to as digits for the binary number. Since there is a one-to-one correspondence between binary numbers and their corresponding decimal equivalents, the value of the `_TYPE_` variable can only be produced by one binary number. The binary number, using the 1s and 0s as defined above, will indicate which CLASS variables went towards a specific record in the output data set.

The `_FREQ_` variable is automatically generated by SAS and shows the number of observations for each level of the CLASS variable, if there is only one CLASS variable involved. When there is more than one CLASS variable, then it will show the number for observations for the combination of levels in the CLASS variables. `_FREQ_`, `nrs`, and `nss` are not necessarily equal as shown in the example. `_FREQ_` indicates the number of observations per level or combination of levels. The `nrs` and `nss` values are the numbers of data points that were included in the calculation of `mrs` and `mss` respectively. When there are missing values for the raw and scale scores, the values of `nrs` and `nss` are less than the corresponding `_FREQ_` value. Had there been no missing values, `_FREQ_`, `nrs`, and `nss` would have been all equal to each other.

Suppose that a cross-tabulation between `gender` and `meals` is required and that missing values for `gender` and `meals` should be included in the cross-tabulation. Missing values can be included by using the MISSING option with the CLASS statement. The NOPRINT option suppresses the output normally generated by the procedure.

```
proc means nway data=example noprint;
  class gender meals/missing;
  var rawscore scalescore;
  output out=test n=nrs nss mean=mrs mss;
```

The SUM statement adds up the values of the specified variables and provides the total in the PROC PRINT output;

```
proc print data=test;
  sum _freq_ nrs nss;
```

| Obs | gender | meals | _TYPE_ | _FREQ_ | nrs | nss | mrs | mss |
|-----|--------|-------|--------|--------|-----|-----|-----|-----|
| 1 | | R | 3 | 1 | 1 | 1 | 10 | 100 |
| 2 | F | F | 3 | 1 | 1 | 1 | 5 | 40 |
| 3 | F | P | 3 | 2 | 2 | 2 | 5 | 40 |
| 4 | F | R | 3 | 2 | 2 | 2 | 10 | 100 |
| 5 | M | | 3 | 1 | 1 | 1 | 5 | 40 |
| 6 | M | F | 3 | 1 | 0 | 0 | . | . |
| 7 | M | P | 3 | 1 | 1 | 1 | 10 | 100 |
| 8 | M | R | 3 | 1 | 1 | 1 | 5 | 40 |
| | | | | ===== | === | === | | |
| | | | | 10 | 9 | 9 | | |

Let us suppose the same statistics need to be computed according to the following groups:

1. All students
2. Males
3. Females
4. Students receiving free or reduced-price meals
5. Students paying full price for meals
6. Students receiving free meals

Note that there is an overlap between categories 4 and 6.

The first step is to create counters for each variable. Counter1 defaults to 1 since this applies to all students. The remaining counters are each set to 1 if the criteria for group membership are satisfied. The advantage of using counters this way is that they can be easily coded for any group of interest no matter how many variables are used to make the determination for group membership (e.g., females receiving free meals, etc.). If it is just counts that are needed, the sum of the counters will provide the total number of records per group.

```
data example;
set example;
counter1=1;
select(gender);
when ('M') counter2=1;
when ('F') counter3=1;
otherwise;
end;
**F = free, R = reduced, P = full pay;
select(meals);
when ('F') do; counter4=1; counter6=1; end;
when ('R') counter4=1;
when ('P') counter5=1;
otherwise;
end;

proc means data=example sum maxdec=0;
var counter1-counter6;
```

The MEANS Procedure

| Variable | Sum |
|----------|-----|
| counter1 | 10 |
| counter2 | 4 |
| counter3 | 5 |
| counter4 | 6 |
| counter5 | 3 |
| counter6 | 2 |

There is often a need to compute various types of statistics and not just determine the number of members in groups of interest. PROC MEANS can be useful for this purpose. List all the counters in the CLASS and TYPES statements.

```
proc means data=example;
  class counter1-counter6/missing;
  var rawscore scalescore;
  types counter1 counter2 counter3 counter4 counter5 counter6;
  output out=test n=nrs nss mean=mrs mss;
```

| Obs | counter1 | counter2 | counter3 | counter4 | counter5 | counter6 | _TYPE_ | _FREQ_ | nrs | nss | mrs | mss |
|-----|----------|----------|----------|----------|----------|----------|--------|--------|-----|-----|---------|---------|
| 1 | . | . | . | . | . | . | 1 | 8 | 8 | 8 | 7.50000 | 70.0000 |
| 2 | . | . | . | . | . | 1 | 1 | 2 | 1 | 1 | 5.00000 | 40.0000 |
| 3 | . | . | . | . | . | . | 2 | 7 | 6 | 6 | 7.50000 | 70.0000 |
| 4 | . | . | . | . | 1 | . | 2 | 3 | 3 | 3 | 6.66667 | 60.0000 |
| 5 | . | . | . | . | . | . | 4 | 4 | 4 | 4 | 6.25000 | 55.0000 |
| 6 | . | . | . | 1 | . | . | 4 | 5 | 5 | 5 | 8.00000 | 76.0000 |
| 7 | . | . | . | . | . | . | 8 | 5 | 4 | 4 | 7.00000 | 70.0000 |
| 8 | . | . | 1 | . | . | . | 8 | 5 | 5 | 5 | 7.00000 | 64.0000 |
| 9 | . | . | . | . | . | . | 16 | 6 | 6 | 6 | 7.50000 | 70.0000 |
| 10 | . | 1 | . | . | . | . | 16 | 4 | 3 | 3 | 6.66667 | 60.0000 |
| 11 | 1 | . | . | . | . | . | 32 | 10 | 9 | 9 | 7.22222 | 66.6667 |

In the output data set, the only rows of interest are the ones with a counter value of 1. The following code will eliminate the rows that are not of interest by specifying a WHERE option in the OUTPUT statement. Note that the MISSING option was used with the CLASS statement.

```
proc means data=example;
  class counter1-counter6/missing;
  var rawscore scalescore;
  types counter1 counter2 counter3 counter4 counter5 counter6;
  output out=test
    (where=(sum(counter1,counter2,counter3,counter4,counter5,counter6)>0))
    n=nrs nss mean=mrs mss;
```

| Obs | counter1 | counter2 | counter3 | counter4 | counter5 | counter6 | _TYPE_ | _FREQ_ | nrs | nss | mrs | mss |
|-----|----------|----------|----------|----------|----------|----------|--------|--------|-----|-----|---------|---------|
| 1 | . | . | . | . | . | 1 | 1 | 2 | 1 | 1 | 5.00000 | 40.0000 |
| 2 | . | . | . | . | 1 | . | 2 | 3 | 3 | 3 | 6.66667 | 60.0000 |
| 3 | . | . | . | 1 | . | . | 4 | 6 | 5 | 5 | 8.00000 | 76.0000 |
| 4 | . | . | 1 | . | . | . | 8 | 5 | 5 | 5 | 7.00000 | 64.0000 |
| 5 | . | 1 | . | . | . | . | 16 | 4 | 3 | 3 | 6.66667 | 60.0000 |
| 6 | 1 | . | . | . | . | . | 32 | 10 | 9 | 9 | 7.22222 | 66.6667 |

Because of how PROC MEANS processes data, no observations will result in the data set without the MISSING option as shown in the SAS log below.

```
1565 proc means data=example;
1566 class counter1-counter6;
1567 var rawscore scalescore;
1568 types counter1 counter2 counter3 counter4 counter5 counter6;
1569 output out=test
1570       (where=(sum(counter1,counter2,counter3,counter4,counter5,counter6)>0))
1571       n=nrs nss mean=mrs mss;
1572 run;
```

WARNING: A class or frequency variable is missing on every observation.

NOTE: There were 10 observations read from the data set WORK.EXAMPLE.

NOTE: The data set WORK.TEST has 0 observations and 12 variables.

NOTE: PROCEDURE MEANS used (Total process time):

```
real time      0.01 seconds
cpu time       0.01 seconds
```

```
1573 proc print data=test;
1574 run;
```

NOTE: No observations in data set WORK.TEST.

NOTE: PROCEDURE PRINT used (Total process time):

```
real time      0.00 seconds
cpu time       0.00 seconds
```

Instead of using the TYPES statement shown above, using the “ways 1;” statement instead has the same effect. It tells SAS that only levels of single (1) class variables should be considered for the calculations. If the “ways 2;” statement was used, that means the 2-way combinations of all pairs of class variables should be used in the calculations.

```
proc means data=example;
  class counter1-counter6/missing;
  var rawscore scalescore;
  ways 1;
  output out=test
    (where=(sum(counter1,counter2,counter3,counter4,counter5,counter6)>0))
    n=nrs nss mean=mrs mss;
```

| Obs | counter1 | counter2 | counter3 | counter4 | counter5 | counter6 | _TYPE_ | _FREQ_ | nrs | nss | mrs | mss |
|-----|----------|----------|----------|----------|----------|----------|--------|--------|-----|-----|---------|---------|
| 1 | . | . | . | . | . | 1 | 1 | 2 | 1 | 1 | 5.00000 | 40.0000 |
| 2 | . | . | . | . | 1 | . | 2 | 3 | 3 | 3 | 6.66667 | 60.0000 |
| 3 | . | . | . | 1 | . | . | 4 | 6 | 5 | 5 | 8.00000 | 76.0000 |
| 4 | . | . | 1 | . | . | . | 8 | 5 | 5 | 5 | 7.00000 | 64.0000 |
| 5 | . | 1 | . | . | . | . | 16 | 4 | 3 | 3 | 6.66667 | 60.0000 |
| 6 | 1 | . | . | . | . | . | 32 | 10 | 9 | 9 | 7.22222 | 66.6667 |

This table shows the correspondence between the groups and the _TYPE_ values. Although the example is for six counters, the concept can easily be generalized to however many counters are involved. The nth counter will have a _TYPE_ value of $2^{(n-1)}$.

| Variable | Group Description | _TYPE_ Value | Binary Number |
|----------|--|--------------|---------------|
| counter1 | All students | $32 = 2^5$ | 100000 |
| counter2 | Males | $16 = 2^4$ | 010000 |
| counter3 | Females | $8 = 2^3$ | 001000 |
| counter4 | Students receiving free or reduced-price meals | $4 = 2^2$ | 000100 |
| counter5 | Students paying full price for meals | $2 = 2^1$ | 000010 |
| counter6 | Students receiving free meals | $1 = 2^0$ | 000001 |

In looking at the code, the SUM function appears to have many arguments and using

```
sum(of counter1-counter6)>0
```

instead of

```
sum(counter1,counter2,counter3,counter4,counter5,counter6)>0
```

is an idea. However, the log shows that there is a problem with this.

```
87  proc means data=example;
88  class counter1-counter6;
89  var rawscore scalescore;
90  ways 1;
91  output out=test (where=(sum(of counter1-counter6)>0)) n=nrs nss mean=mrs mss;
                                -----
                                22
                                76
ERROR: Syntax error while parsing WHERE clause.
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &, (, *, **, +, ', -, /, <, <=, <>, =, >, >=, ?,
AND, BETWEEN, CONTAINS, EQ, GE, GT, LE, LIKE, LT, NE, OR, ^=, |, ||, ~=,
ERROR 76-322: Syntax error, statement will be ignored.
```

This is a known issue as shown in the SAS Usage Note # 14554.

Usage Note 14554: Syntax error when using OF operator within a WHERE statement

[Details](#)
[About](#)
[Rate It](#)

When the OF keyword is used prior to a variable list in a function call within a WHERE statement such as:

```
where max(of abc1-abc5) > 33;
```

the following syntax error is generated:

```
ERROR: Syntax error while parsing WHERE clause.
ERROR 22-322: Syntax error, expecting one of the following: !, !!, &,
(,
    *, **, +, ', ' -, /, <, <=, <>, =, >, >=, ? , AND,
    BETWEEN, CONTAINS, EQ, GE, GT, LE, LIKE, LT, NE, OR,
    ^=, |, ||, ~=.
```

The syntax for WHERE statements is derived from SQL, and in some cases does not provide for certain features otherwise available in SAS, such as the OF keyword. To prevent the error, specify each of the variables rather than using OF and a variable list.

A macro can be useful when there is a large number of counters.

```
%macro example(n);
proc means data=example;
  class counter1-counter&n/missing;
  var rawscore scorescore;
  ways 1;
  output out=test (where=(sum(
    %do i=1 %to %eval(&n);
    counter&i
    %if &i < %eval(&n) %then %do;,%end;
    %end;
  )>0)) n=nrs nss mean=mrs mss;
%mend example;

%example(6)
```

When the program is run with the MPRINT system option in effect, the SAS code generated by the macro appears in the SAS log.

```
1827 options mprint;
1828
1829 %macro example(n);
1830 proc means data=example;
1831 class counter1-counter&n/missing;
1832 var rawscore scorescore;
1833 ways 1;
1834 output out=test (where=(sum(
1835   %do i=1 %to %eval(&n);
1836   counter&i
1837   %if &i < %eval(&n) %then %do;,%end;
1838   %end;
1839   )>0)) n=nrs nss mean=mrs mss;
1840 %mend example;
1841
1842 %example(6);
MPRINT(EXAMPLE):      proc means data=example;
MPRINT(EXAMPLE):      class counter1-counter6/missing;
MPRINT(EXAMPLE):      var rawscore scorescore;
MPRINT(EXAMPLE):      ways 1;
MPRINT(EXAMPLE):      output out=test (where=(sum( counter1, counter2, counter3, counter4, counter5, counter6 )>0)) n=nrs
nss mean=mrs mss;
1843
NOTE: There were 10 observations read from the data set WORK.EXAMPLE.
NOTE: The data set WORK.TEST has 6 observations and 12 variables.
NOTE: PROCEDURE MEANS used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

REFERENCES

SAS Institute Inc. 2009. *Base SAS® 9.2 Procedures Guide*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2010. *SAS® 9.2 Language Reference: Concepts, Second Edition*. Cary, NC: SAS Institute Inc.

SAS Institute Inc. 2011. *SAS® 9.2 Language Reference: Dictionary, Fourth Edition*. Cary, NC: SAS Institute Inc.

Usage Note 14554: Syntax error when using OF operator within a WHERE statement. Retrieved July 18, 2011, from the SAS Support Web site: <http://support.sas.com/kb/14/554.html>

TRADEMARK NOTICE

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.