Paper CC88

Having a Mean Headache on Summary Data? Using SAS® to Compute Actual Data from Summary Data

William “Gui” Zupko II, Federal Law Enforcement Training Centers

ABSTRACT

SAS programmers might not get to choose how their data is formatted. It is very easy to take raw data and provide descriptive statistics on data that has not been modified. Unfortunately, sometimes raw data is unavailable and only the summary data can be provided. One of the trickiest problems occurs with this summary data, as SAS has difficulty breaking data from one line into many. Since many of the functions SAS would perform need the actual data, such as proc means, performing descriptive statistics on summary data can at best provide misleading data or, at worst, completely incorrect data. This paper describes how to create data sets from summary data into simulated raw data sets, which allow accurate descriptive statistics on each single variable. The following SAS code will be utilized and/or discussed: proc means, DATA step, do loops, and the output statement.

INTRODUCTION

As SAS programmers, it must seem that we have a rabid hunger for data. Not only do SAS programmers want as much data as possible, but also want it formatted in a certain way, provided in specific files, and, if possible, get it with a warm chocolate chip cookie and milk. While we may want specific data in definitive ways, sometimes we do not always get the data that we want in the easiest format to use. One of the most difficult forms of data to receive are summary or descriptive statistics, since this masks the data in such a way that we are unable to further analyze specific data for additional information.

In order to illustrate this point, here is a simple questionnaire with three questions. This questionnaire illustrates a five-point Likert scale, with five being the highest rating and 1 being the lowest rating.

Table 1. Summary Statistics

Rating	Question1	Question2	Question3
5	50	5	17
4	10	30	17
3	18	20	19
2	7	22	17
1	2	10	17

Looking at this data, we can make some general assumptions. For example, question1 has significantly more people who answered 5 than the other questions. However, question2 and question3 are murkier. What would be really nice to sort this data out would be a weighted average and/or the variance of these questions and how they relate to one another.

FINDING THE MEAN

Our first impulse is to multiply the rating by question1, question2, and question3. This will give us numbers that we can then run a proc means on in order to get mean, variance, and standard deviation. Unfortunately, in this case, the numbers are not only indecipherable, they are also wrong. When we run proc means on the resulting numbers, we get the following result:

Figure 1. Proc Means on Incorrect Summary Statistics

Variable	Mean	Variance	Std Dev
wrong1	72.0000000	10324.00	101.6070864
wrong2	51.8000000	1811.20	42.5581955
wrong3	52.2000000	729.7000000	27.0129599

So, we have some statistics, but what do they mean (no pun intended)? In this case, it looks like question1 has a higher mean, but significantly higher variance than the other two questions. Question2 has the lowest mean and a

larger variance than question3, which would make question2 our clear least favorite. However, what does a mean of 72 even stand for in this case, where our rating falls between one and five?

As figure 1 illustrates, trying to merge summary statistics with some standard provides fallacious results. The numbers are garbled and do not relate to what we are trying to determine.

RECREATING THE ACTUAL DATA

What we need are the actual, raw data, which is being obscured by the summary statistics. With SAS, we can recreate the data to some degree in order to run some statistics on it. This procedure does have some warnings that need to be attached to it:

- Statistics can only be run on one specific variable. We are unable to see relationships between questions (if they answered 5 on question1, they were likely to answer 4 on question2)
- This procedure looks at the question overall, not at specific individuals and how they responded.

So, just because our SAS program has row 13 saying specific information, that does not mean respondent 13 provided that information. We can only look at each question separately.

THE PROBLEM

SAS likes to look at code line by line, so it can get a little tricky to get SAS to add rows. However, it is possible to do this by using the output statement. Usually, the output statement is used to declare which data set a specific row needs to go into. However, by using the output statement alone, SAS is able to create an exact duplicate of the line in processing and output that.

Using our summary data from before, we will use the following SAS code to break out question1, question2, and question3 into the actual data. The following example only shows question1 being broken into a new data set called ratings_1:

```
data ratings_1(keep=rating_1 number1 counter1);
    set ratings ;
           rating_1=rating;
           do until (counter1=0);
           counter1+1;
           output;
                  if counter1=number1 then counter1=0;
           end;
run;
```

Let us take a closer look at what this code is doing. The rating variable is essentially renamed into rating_1, because we want to see each actual iteration of someone putting that specific rating in. Then, a do loop starts looping around, specifically continuing to run the code until the counter1 variable is 0. The fun part of this do loop is that counter1 has not even been created yet. With the do loop running, it is time to break apart our summary statistics.

The counter1 variable is created, adding one for each iteration. Now, here comes the output statement. The output statement says to print the line currently in processing memory to the data set, which it does. The do loop continues outputting the same line into the data set with only one change, the counter1 variable increasing by one each time. When the counter1 variable is equal to the number of respondents who answered this question, the do loop ends, which then allows the next line of code to be parsed.

So, if 50 respondents answered with a five rating to question1, then the counter variable will go to 50 before it is reset to 0. When it is reset, the do loop ends with the fulfillment of the condition (counter variable is 0) and SAS then starts reading the next line, where the rating is four instead of five. SAS will continue this process until it has reached the end of all rows and finished out the do loop.

By continuing this process, instead of having just five rows in our data set, we will instead have the sum of question1 number of rows in our data set. In the case of question1, that would be 87. Since question1 has a large number of rating 5, I chose to show question2 for illustrative purposes on how this would look.

Figure 2. Question2 Data Set

	number2	rating_2	counter2
1	5	5	1
2	5	5	2
3	5	5	3
4	5	5	4
5	5	5	5
6	30	4	1
7	30	4	2
8	30	4	3
9	30	4	4

Here, we can see that there were five people who gave question2 a rating of 5. Instead of having one row with a five, however, we can see that there are now five rows with a five, at which point it starts counting to 30 with a rating of 4.

CHECK YOUR RESULTS

There are several ways to check the results to make sure that the data is correct. Whenever manipulating data like this, it is always a good idea to do some checks to make sure that everything is ok. A quick proc freq is one of the easiest ways, providing summary data that can be checked with the original summary data.

At times, if the code is not tight enough, duplicates will be formed in the data set. While it is possible to simply make the code tighter to prevent duplication, another way to clear those duplicates out is with the nodups function in proc sort. It is recommended to put a proc sort nodups code into your code regardless of individual checks, in order to have that protection from spurious results distorting analysis.

BEGIN THE ANALYSIS

Now that we have actual data that has been checked and verified with original summary data, we can try out the analysis again. Using proc means as before, we get the following result from our actual data.

Figure 3. Proc Means on Actual Data

Variable	Mean	Variance	Std Dev
rating_1	4.1379310	1.3063352	1.1429502
rating_2	2.9770115	1.3017910	1.1409605
rating_3	3.0000000	1.9767442	1.4059673

Unlike our results before, we can immediately see not only the statistics that we wanted, but their relevance to the point at hand. Just like before, we can see that the first question has the highest average rating. This time, however, we can see that it averages in the 4 range. Question2 and question3 actually have similar means, but the variance is much smaller on question2, meaning there is more agreement on this question than on question3. Unlike the earlier analysis, we are able to get verifiable results that allow us to make determinations on the data, allowing data discovery that had been hidden before.

CONCLUSION

It can be painful for SAS programmers to receive data that does not allow for easy analysis. However, using the output statement, we can recreate actual data from summary data in order to run accurate descriptive statistics on that data. Unlike many other statistical programs, SAS allows the creation of extra observations which allows us to approximate the data we need. While this methodology does not work on variable to variable comparisons, like modeling or regressions, it provides a way to allow accurate statistical descriptives on individual variables.

ACKNOWLEDGMENTS

I would like to thank FLETC and SAS for the opportunity to write this paper.

This paper is released to inform interested parties of (ongoing) research and to encourage discussion of work in progress. Any views expressed on statistical, methodological, technical, or operational issues are those of the author and not necessarily those of FLETC.

CONTACT INFORMATION

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

William Zupko II
U.S. Department of Homeland Security, Federal Law Enforcement Training Centers
William.ZupkoII@fletc.dhs.gov