Paper 76-27

Fast and Easy Ways to Check Your Work: Using PROC MEANS to Confirm Continuous Variable Categorizations

Rick M. Mitchell, Westat, Rockville, MD

ABSTRACT

The purpose of this paper is to present a process where a SAS® programmer may utilize PROC MEANS to check one's work after taking a continuous variable and creating a new variable with discrete categories. While variable coding may often appear to be simple enough not to warrant extensive checking of a program, the integration of such an approach into standard checking procedures can not only help ease the mind of the SAS programmer, but also help provide proof to the requestor that a task has been done correctly. This process is also beneficial in identifying potential outliers of data and in serving as a test to confirm just how good a set of specifications are for a given variable. By taking advantage of basic procedures such as PROC MEANS, one can avoid making the same variable categorization mistakes that have tripped up even the best of programmers.

INTRODUCTION

Because of tight deadlines or just plain laziness, programmers have been known to skip one of the most basic, but important, steps in the analysis process – checking one's work. This step is essential if one is to truly trust the presentation and interpretation of results. Generating clear and understandable SAS output as part of the checking process can give the programmer greater confidence, and the output can be used as an excellent source of proof to the requestor as to the task's correctness. So, when the requestor asks, "How do I really know that your work is correct?" you can deliver concrete proof and say "Hey look, I did it right!" While there are endless methods of checking one's work where each method may be best suited for selected scenarios, a process is presented in this paper to show how PROC MEANS may be utilized to confirm that continuous variables have been categorized appropriately (Mitchell, 2001). The author incorporates this method of checking into his daily work routine and encourages others to do the same.

Why is this approach "Fast and Easy?"

Basically, this PROC MEANS approach to checking discrete categories allows one to easily view the minimum and maximum values of the original analysis variable to confirm that all values have been correctly grouped into their designated ranges.

DISCUSSION OF CODE AND OUTPUT

Let us take an example of a task in which a programmer is provided with a continuous variable called MILE_WK, which represents the "Number of Miles Run Per Week" for members of a local running club. The programmer is requested to group the data into categories so that the data may be read and interpreted more easily as the club attempts to show the breakdown of mileage for selected levels of their members. A new discrete variable is created that is supposed to contain 3 categories representing different ranges of running mileage including low (0-20), medium (21-40), and high (>40). This categorical variable called MILE_CAT (Mileage, Categorical) is coded as values of 1, 2, and 3 respectively, where a format is also created in order to display the appropriate ranges for any reports that may be needed (see code below in Figure 1).

```
proc format;
  value milecatf
    . ='Missing'
    1='0-20'
    2='21-40'
    3='>40';
run;

data testdata;
  set in.rundata;

  * Create a categorized weekly mileage variable;

  if (mile_wk le 20) then mile_cat=1;
  else if (21 le mile_wk le 40) then mile_cat=2;
  else if (mile_wk > 40) then mile_cat=3;

  format mile_cat milecatf.;
run;
```

Figure 1 – Code Before Checking

After the new variable has been coded, it would be appropriate to do some type of check to confirm that the ranges are what they are expected to be. The approach provided in this paper utilizes PROC MEANS. Figure 2 on the following page shows that the original continuous variable, MILE_WK, is used as the analysis variable (VAR) and the new categorical variable, MILE_CAT, is used as the CLASS variable.

```
proc means data=testdata nmiss n min max maxdec=1
        missing;
    class mile_cat;
    var mile_wk;
```

Figure 2 – Code For PROC MEANS Approach

While it may seem to the programmer that this type of check is unnecessary as one might think that it is clear from the programming code that everything "looks good," PROC MEANS will provide us with a better summary of how our algorithms work on any given dataset. Below in Figure 3, one may see the output that is generated by PROC MEANS prior to the programmer doing any checking of the code.

Team Mileage - Confirmation of Variable Categorizations
Analysis Variable: mile_wk

mile_cat	Obs	N Miss	N	Minimum	Maximum
0-20	15	1	14	**-50.0**	17.0
21-40	46	0	46	23.0	40.0
>40	36	0	36	40.2	**9999.0**

Figure 3– Output Before Checking

Note that in this example the missing values are accidentally included in the 1st category as well as negative values (e.g. –50). We can also see that high outliers (e.g. 9999) are included in the 3rd category. While the programmer may have followed the requestor's specifications, it is apparent that there are values that were not considered to be a part of the analysis, or perhaps these special values were not even included in datasets when the algorithms were first written and therefore never addressed. If the specifications did take these types of values into account, then the programmer needs to go back to the drawing board to make the categorization algorithms more flexible.

Below in Figure 4, one can see the modifications that were necessary to accommodate both missing values and outliers. Note that upper and lower values have been added to all ranges in the algorithm and that a 4th category has been created to accommodate outliers.

```
proc format;
  value milecatf
    . ='Missing'
    1='0-20'
    2='21-40'
    3='>40'
    4='Outliers';
run;

data testdata;  set in.rundata;

  * Create a categorized weekly mileage variable;

  if (0 le mile_wk le 20) then mile_cat=1;
  else if (21 le mile_wk le 40) then mile_cat=2;
  else if (41 le mile_wk le 100) then mile_cat=3;
  else if (mile_wk > .Z) then mile_cat=4;
  format mile_cat milecatf.;
```

Figure 4 – Code After Checking

Now, looking at Figure 5 below, one sees that all values fit nicely into each of the categories as one eyes down the minimum and maximum values for each category range.

Team Mileage - Confirmation of Variable Categorizations
Analysis Variable: mile_wk

mile_cat	Obs	N Miss	N	Minimum	Maximum
Missing	1	1	0	.	.
0-20	13	0	13	0.0	17.0
21-40	46	0	46	23.0	40.0
>40	35	0	35	40.2	74.0
Outliers	2	0	2	-50	9999

Figure 5 – Output After Checking

CONCLUSIONS

Categorizing continuous variables can be a tricky process which should not be taken lightly, and programmers should take full advantage of the many tools available in SAS that can be used to check one's work. While some tasks may appear so simple that checking isn't necessary, one should always try to incorporate some type of checking mechanism into their daily work routine. Finding the optimal medium between generating output quickly and producing a high quality product can be difficult for programmers of all levels. By utilizing PROC MEANS to confirm that continuous variables have been categorized correctly, one will be well on the way to becoming a better and more trustworthy programmer.

REFERENCES

- Mitchell, R.M. (2001). Fast and Easy Ways to Annoy a Statistician: The Sharing and Presentation of Data Between a SAS Programmer and a Statistician (Paper 243-26). *Proceedings of the 26th Annual SAS Users Group International (SUGI) Conference*.

ACKNOWLEDGEMENTS

CONTACT INFORMATION

Rick M. Mitchell
Westat
1650 Research Boulevard
WB 496
Rockville, MD 20850
(301) 251-4386 (voice)
(301) 738-8379 (fax)
RickMitchell@Westat.com