Paper 114-2008

Stupid Human Tricks with PROC SURVEYSELECT®

David L. Cassell, Design Pathways, Corvallis OR

ABSTRACT

Surprisingly, PROC SURVEYSELECT has features which make it useful for some basic, non-statistical, data management tasks. We will cover three simple tasks which may otherwise require coding and/or macro programming.. unless PROC SURVEYSELECT resides in your bag of tricks. We show how to: create a data set by stacking a starting file over and over; sort a data set in a complex order (a 'serpentine' sort); and split a data set into two pieces randomly.

INTRODUCTION

SAS® tools like PROC SURVEYSELECT typically have a large number of features, to permit a wide range of applications. This often means that a given procedure has functionality that we wouldn't expect. In the case of PROC SURVEYSELECT, we can use it for some non-sampling kinds of database tasks that we occasionally need.

In this paper, we will look at three simple uses of PROC SURVEYSELECT that are NOT statistical, even though PROC SURVEYSELECT is in the SAS/STAT® module. These uses are not so common that they would be a natural part of some Base SAS procedures, but they do turn up when you least expect them.

TRICK 1: STACKING THE SAME DATA SET OVER AND OVER

Sometimes we need to stack one data set atop another, i.e. append one to the end of the other. The DATA step can do this, as can PROC SQL. PROC APPEND is designed to stack data sets like this. However, sometimes we have to append a lot of data sets into a single file. And, once in a while, we need to stack N copies of a data set into a new data set.

Some people choose to do this using a big macro loop that uses PROC APPEND over and over. The code typically looks like this:

```
%macro stacker( input=, reps= , out= );

  %do i = 1 %to &REPS ;

    %if &I = 1 %then %do;
      data &OUT ;
        set &INPUT ;
        run;
      %end;

    %else %do;
      proc append base=&OUT data=&INPUT;
        run;
      %end;
    %end;

%mend;

%stacker(input=YourData, reps=20, out=YourOut )
```

Okay, this is more complex than needed, since most people do not realize that PROC APPEND will perform just fine when the BASE= data set does not exist. But the loop will take longer than a single DATA step:

```
data out;
  set YourData YourData YourData YourData YourData YourData YourData
      YourData YourData YourData YourData YourData YourData YourData
      YourData YourData YourData YourData YourData YourData;
  run;
```

And this 'wallpaper code' can be replaced by a simple macro variable, so the code would look like this:

```
data _null_;
  call symput( 'indata', repeat( 'YourData ', &REPS – 1) );
  run;

data YourOut;
  set &INDATA ;
  run;
```

But this is still more than we need. PROC SURVEYSELECT has two subtle features that we can exploit. First, when we ask for a 100% sample, using a method that will only select any given record one time (at most), the procedure is smart enough to realize that we have selected the entire data set with no alterations. So it just returns the original data set, in order. The default sampling method will do this for us. And second, the procedure has a way of automatically generating N samples in a row, stacking them in the output data set. So the following code does exactly what the above programs do, only more simply:

```
proc surveyselect data=YourData out=YourOut samprate=1 rep=&REPS ;
 run;
```

The SAMPRATE=1 (or SAMPRATE=100) option asks for 100% samples each time (so we get the exact data set back), and the REP= option just tells how many times to stack the input data set. Simpler than you expected, isn't it?

TRICK 2: THE SERPENTINE SORT

Every once in a while, we have to sort a data set in an odd way. And once in a blue moon, we need to use what is called a 'serpentine' sort. This means that we sort by several variables, the first in ascending order, the second in descending order (i.e., sorting from highest to lowest, instead of the usual order), then ascending order, then descending order, and so on. This can be done in PROC SORT if you want. Suppose we have sort keys AGE GENDER LATITUDE LONGITUDE INCOME and MED_EXPENSES that we need to use in a serpentine sort. We can use the DESCENDING keyword on every other variable to do this in PROC SORT:

```
proc sort data=YourIn out=YourOut;
  by age descending gender latitude descending longitude income descending
     med_expenses;
  run;
```

This sorts first by AGE, then it sorts by GENDER in descending order. The third sort-key, LATITUDE, is then sorted in ascending order, and LONGITUDE is sorted in descending order. And so on, and so on…

Clearly, this can be constructed using a macro so that you do not have to figure out where the DESCENDING keywords go. But, if the list of sort variables is already in a macro, say, &SORTLIST , then we would have to decompose that macro variable and construct a new macro variable that has the word DESCENDING in all the right places.

*** shall we write that macro? %do loop with %scan()

If we use the power of PROC SURVEYSELECT, we do not have to worry about writing that macro. PROC SURVEYSELECt will let you perform a type of sampling called 'control sampling' which lets you put your data in some order by the 'control' variables, and then select your sample in a sequential order as you step through the previously-ordered data. One way in which PROC SURVEYSELECT is designed to arrange the data by the control variables is the serpentine sort. This means that there's an easier way:

```
proc surveyselect data=YourIn outsort=YourOut  /* the sorted output*/
        out=junk n=1;                          /* JUNK is not needed */
  control &SORTLIST ;
  run;
```

Here, when we use the CONTROL statement, serpentine sorting is the default! The output file is created by the OUTSORT option. The OUT= option creates a data set that we will throw away, so we used N=1 to create a one-record data set that will not take up much space.

As an aside, if we do this in PROC SURVEYSELECT and leave off the OUTSORT= option, the sorted data set replaces our input data set, so this will work like PROC SORT without the OUT= option.

TRICK 3: SPLITTING A DATA SET IN TWO

It is more common to need to split a data set in two, than it is to need either of the above tricks. In particular, programmers are often asked to split a data set into two pieces, one holding P percent of the records and the other holding the remainder, so that someone else can analyze or model with the first set, and then test or validate with the second.

(As an aside, I don't recommend this approach, despite its popularity. I recommend using cross-validation instead, since it is more powerful, even though it will require more computing time. If you decide that you want to use cross-validation instead of the approach we cover here, there is SAS code already written to do cross-validation. There is a section on the subject in my SGF 2007 paper "Don't Be Loopy: Re-sampling and Simulation the SAS® Way", which is available on the SAS website.)

Rather than splitting the data into two separate data sets, I advocate creating a variable that tells whether each record is in group 1 or group 2, then using a WHERE clause to separate out the subsets for modeling and validating.

A common way I have seen this done is as follows:

```
data split;
  retain seed 384747;
  set YourData;
  if ranuni(seed) < .7 then selected = 1;
                       else selected = 0;
  run;

/* now run whatever proc is relevant for the project goals and the data */
proc model data=split(where=(selected=1));
  . . . .
  run;

/* and use the above results for the validation steps */
data validat;
  set split(where=(selected=0));
  . . . .
  run;
```

Note that the WHERE clauses in the later steps let us choose which segment of the data set SPLIT to work with at each step.

Unfortunately, this does not split the data into groups of 70% and 30%. It splits the data into groups which are somewhere near 70% and 30%, but not exactly where the user specified. This is due to the natural randomness inherent in a random variable, or in a pseudo-random number generator like the RANUNI() function. We can fix that problem with PROC SURVEYSELECT, as well as simplifying the code.

SAS 9.1 added the OUTALL keyword to PROC SURVEYSELECT. The OUTALL option keeps all the records in the output data set, and marks the records chosen for the sample by creating a variable SELECTED and setting it equal to one for the selected records… Oh wait, that's exactly what we wanted. So our first data step above can be fixed like this:

```
proc surveyselect data=YourData out=split samprate=.7 outall;
  run;
```

Note that the code asks for a sampling rate of 70%, as before. It does not specify a random seed, as the data step did. It doesn't have to: the procedure will choose one, and tell you in the output which seed it used. The OUTALL keyword tells the procedure to create the variable SELECTED and populate it with 0 and 1 values, as the data step before did. But this does a real 70/30 split, instead of an approximate one. That is because PROC SURVEYSELECT uses a more sophisticated selection method under the hood, where we do not have to worry about it.

CONCLUSIONS

It is always important to have a 'bag of tools' when programming. A rich environment like SAS gives you the opportunity to have a bag of tools the size of a Winnebago. A procedure like PROC SURVEYSELECT may not seem to have much relevance to standard non-statistical areas like database management and data manipulation, but (not too surprisingly) the complexity of SAS procedures gives us the opportunity to turn them into new and ingenious tools for our toolbag.

REFERENCES

"Don't Be Loopy: Re-sampling and Simulation the SAS® Way", David L. Cassell, Design Pathways, Corvallis Oregon; ******

SAS OnlineDoc® 9.1.3, Copyright © 2002-2005, SAS Institute Inc., Cary, NC, USA; All rights reserved. Produced in the United States of America.

ACKNOWLEDGMENTS

CONTACT INFORMATION

The author welcomes questions and comments. He can be reached at his private consulting company, Design Pathways:

David L. Cassell
Design Pathways
3115 NW Norwood Pl.
Corvallis, OR 97330
DavidLCassell@msn.com

541-754-1304