SESUG Paper 140-2018

Matching SAS® Data Sets with Hash Objects: If at First You Don't Succeed, Match, Match Again

Imelda C. Go, Questar Assessment, Inc.

ABSTRACT

Two data sets can be matched using a number of techniques available in SAS. If the goal is to match as many records as possible between two data sets, then the matching process may have to be repeated several times between the two data sets by using different matching criteria. Although programmers new to hash objects will need to learn a few things before they can feel confident about using the technique, the resulting ease of use/maintenance of the matching code is well worth the effort. (This is the hash-object version of two prior papers on the same topic: the 2009 SESUG paper focused on PROC SQL and the 2004 SESUG paper focused on the DATA step.)

This paper does not provide details of hash object basics, but papers by Paul Dorfman and other authors on the topic are highly recommended. In general, this paper is for programmers who are aware of the pros and cons of the different matching techniques and the things you have to be cautious about when matching (e.g., do the data sets to be matched contain variables with exactly the same name that could be overwritten accidentally?). Although it is not difficult to get started with matching in SAS, it does take some experience to match correctly and without undesirable outcomes using the various techniques available in SAS.

In the example below, there is no multiplicative effect where the number of records increases as a result of matching. An increase in the number of records is fine if that does not conflict with the programmer's intentions. With the use of hash objects, the multiplicative problems/complications produced by one-to-many, many-to-one, and many-to-many matching are bypassed. When hash objects are used for matching, SAS will match the first record in the hash object that provides a match. It is your responsibility to make sure that this is the desired effect. If not, you will have to work on or prepare the data that produces the hash object so that you produce the desired matching results.

There are computer programming tasks that require that the number of records remain the same. For example, there might be a set of student test scores that need to be matched to demographic variables from student records. We do not want Jane Doe's test record that appeared once to appear 10 times after matching because Jane was enrolled in 10 different schools. (This unwanted duplication of records is often due to one-to-many, many-to-one, and many-to-many situations using the DATA step or PROC SQL to match records.) This is the scenario that will be presented in the programming example below.

PROGRAMMING EXAMPLE

The programming example will attempt to match two data sets, which are:

- A data set named `testscores` that contains tests scores. This data set has `ctr` as its primary key or unique record identifier.
- A data set named `students` that contains demographic information for students. This data set has `ctrf` as its primary key or unique record identifier.

Step 1:

- Determine all the variables needed for the matching criteria.
- Clean up the data for the variables to match on. For example, make them as consistent as possible—remove spaces, capitalize all the letters, etc. This reduces unnecessary variation in the data values, which helps increase the number of matches (e.g., "John" does not match to "JOHN").
- All of these variables must be present in both data sets. The variable names need to be identical. For example, you might want to match on the first three letters of the first name and the first two letters of the last name. Define these substrings in the data set prior to matching.

Step 2:

- Identify the variables to be used per matching round. In general, the more variables used to match on, the more stringent the criterion is and the stringency of the criterion can decrease for each subsequent round. That is, the most stringent criterion can be used for the first round and subsequent rounds would be less stringent.

Therefore, let us suppose that you have prepared the two data sets you want to match and that you also decided on the rounds listed below as your matching strategy. The variables used for matching involve the school ID (`schoolid`); and the student's state ID (`stateid`), last name (`lname`), first name (`fname`), and date of birth (`dob`).

Each of the two data sets will contain all the variables listed in the table below. (Note for Variables Below: Variable names that begin with `m` mean they are the **m**odified version of the original value [e.g., `mlname` is the modified version of variable `lname`--all values have consistent case, all unnecessary spaces removed, etc.]. Variable names that end in a number (`n`) mean the original variable is truncated to the first `n` characters. For example, `mlname3` contains the first 3 characters of the `mlname` variable's value. Variables names that begin with `sw` mean the first and last names were interchanged since that is a common data error with student names. At this point, it is not really that important what the variables below represent. What is important is that you decide which variables you will use when you match your data and what your matching strategy is.)

Round	Variables Used to Match On
1	`schoolid stateid mlname mfname dob`
2	`schoolid stateid mlname mfname`
3	`schoolid mlname mfname dob`
4	`schoolid stateid mlname3 mfname1 dob`
5	`schoolid stateid mlname3 mfname1`
6	`schoolid mlname3 mfname2 dob`
7	`schoolid stateid swmlname swmfname`
8	`schoolid swmlname swmfname dob`
9	`stateid mlname mfname mmname dob`
10	`stateid mlname mfname dob`
11	`stateid mlname dob`
12	`stateid mfname dob`
13	`stateid mlname3 mfname2 dob`
14	`stateid swmlname swmfname dob`
15	`stateid mlname mfname`
16	`stateid swmlname swmfname`
17	`schoolid mlname mfname mmname`
18	`schoolid mlname mfname`

Because of so many matching rounds, a macro will be written to accommodate the rounds. There will be two macro parameters: the matching round (`n`) and the matching variables (`matchvar`). The macro is described as follows:

SAS CODE	EXPLANATION
`%macro round (n,matchvar);`	There are 2 macro parameters. N is for the matching round number. `Matchvar` is for the variables used for matching.
`%let m=%eval(&n-1);`	Macro variable `m` is just macro variable `n` minus one.

`data _null_;`
`length str $150.;`
`str="&matchvar";`
`strlength=lengthn(str);`
`cstrlength=lengthn(compress(str));`
`numvars=strlength-cstrlength+1;`
`str="'"||tranwrd(str," ","','")||"'";`
`str=substr(str,1,strlength+numvars*2);`
`lastvar=scan("&matchvar",numvars);`
 	
`call symput ('str',str);`
`call symput ('lastvar',lastvar);`
`run;`
	This DATA _NULL_ step assigns to macro variable `str` the macro variable `matchvar` reformatted with punctuation for use in the `definekey` method of the hash object. For example, if `matchvar` is the value:
	a b c
	then the final result for the variable `str` is:
	"a","b","c"
	
	and the result for the variable `lastvar` is:
	c
	
	The `lastvar` variable is the last variable listed in the `str` variable value.

SAS CODE	EXPLANATION
proc sort data=students (keep=&matchvar ctrf) out=subset; by &matchvar; run;	PROC SORT will sort on the matchvar matching variables. Only the matchvar variables and the ctrf variable will be in the data set subset. Because the goal is to match the ctrf value to the matching testscores record, removing all other variables is a way to save on system resources since only the matching variables and ctrf are needed for the hash object.

data keys dupkeys;
  set subset; by &matchvar;
  if first.&lastvar and last.&lastvar then output keys; else output dupkeys;
run;
	This DATA step divides data set subset into two. The first data set keys contains all records in subset that have no duplicates on the matchvar variables. None of those with duplicates appear in data set keys; duplicates will all appear in data set dupkeys.
	
	In this example, a conscious decision was made to exclude any record from students that has duplicate values on the matching variables.
	
	Preparing the data set for creating the hash object is critical to the success of the match. This is where the programmer has to know what he/she is doing and know how the hash object behaves during matching so that the outcome of the match is the desired correct result. After all, there are a number of ways to resolve duplicates. Depending on how you prepare the data, the duplicates may be included or excluded in the matching process.

data matches;
length ctrf $30.;
if _n_=1 then do;
 declare hash round (dataset:'keys');
 round.definekey(&str);
 round.definedata('ctrf');
 round.definedone();
 call missing (ctrf);
 end;
	These lines define the hash object named round.
	It will take the values of the hash object from the dataset called keys. For this hash object, the key consists of the matchvar variables expressed in the str macro variable. The data that the hash object provides is ctrf.
	
	You can add more variables to this declaration, but for now the priority is to find the ctrf value for the record in data set students that matches to a particular record in data set testscores.
	
	The LENGTH statement gives ctrf a length of 30. The CALL MISSING statement sets the variable ctrf to missing. Since it is a character variable in this case, it is set to blank.

%if &n>1 %then
  %do;
  set matches;
  if min (of rc1-rc&m) ne 0 then rc&n=round.find();
  %end;
%else %if &n=1 %then
  %do;
  set testscores;
  rc1=round.find();
  %end;
	The rest of the macro specifies the data sets to be processed and the matching performed using the hash object.
	
	During the first round (&n=1) of matching (round 1), the match begins with data set testscores.
	
	The return code for round 1 is variable rc1. Whenever the return code value is 0, there is a match.
	
	For subsequent rounds (&n>1 for rounds 2, 3, etc.), the processing will start with data set matches. The condition min(of rc1-rc&m) checks if any of the prior rounds produced a return code of 0. Note that the minimum for the 1st through the mth (n-1th) return code is computed. A return code of 0 means there was a match. If there was a match found then the matching will not be attempted for that round.

if rc&n=0 then match=&n; run; %mend round;	The last statement in the macro simply assigns the round to the match variable if the return code for corresponding return code variable (rc&n) is 0.

The macro will be invoked as many times as the number of rounds. The following show the 18 macro calls.

```
%round(n=1,matchvar=schoolid stateid mlname mfname dob);
%round(n=2,matchvar=schoolid stateid mlname mfname);
%round(n=3,matchvar=schoolid mlname mfname dob);
%round(n=4,matchvar=schoolid stateid mlname3 mfname1 dob);
%round(n=5,matchvar=schoolid stateid mlname3 mfname1);
%round(n=6,matchvar=schoolid mlname3 mfname2 dob);
%round(n=7,matchvar=schoolid stateid swmlname swmfname);
%round(n=8,matchvar=schoolid swmlname swmfname dob);
%round(n=9,matchvar=stateid mlname mfname mmname dob);
%round(n=10,matchvar=stateid mlname mfname dob);
%round(n=11,matchvar=stateid mlname dob);
%round(n=12,matchvar=stateid mfname dob);
%round(n=13,matchvar=stateid mlname3 mfname2 dob);
%round(n=14,matchvar=stateid swmlname swmfname dob);
%round(n=15,matchvar=stateid mlname mfname);
%round(n=16,matchvar=stateid swmlname swmfname);
%round(n=17,matchvar=schoolid mlname mfname mmname);
%round(n=18,matchvar=schoolid mlname mfname);
run;
```

After the macro is invoked 18 times for the 18 rounds of matching, we can check to see how many successful matches were made.

```
proc freq data=matches;
tables
rc1*rc2*rc3*rc4*rc5*rc6*rc7*rc8*rc9*rc10*rc11*rc12*rc13*rc14*rc15*rc16*rc17*rc18*match/
list missing;
where ctrf>'';
run;
```

rc1	rc2	rc3	rc4	rc5	rc6	rc7	rc8	rc9	rc10	rc11	rc12	rc13	rc14	rc15	rc16	rc17	rc18	match	Frequency	Percent	Cumulative Frequency	Cumulative Percent
0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	1	3475	98.05	3475	98.05
160038	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	2	5	0.14	3480	98.19
160038	160038	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	.	3	17	0.48	3497	98.67
160038	160038	160038	0	.	.	.	.	.	.	.	.	.	.	.	.	.	.	4	8	0.23	3505	98.90
160038	160038	160038	160038	160038	160038	0	.	.	.	.	.	.	.	.	.	.	.	7	2	0.06	3507	98.96
160038	160038	160038	160038	160038	160038	160038	0	.	.	.	.	.	.	.	.	.	.	8	1	0.03	3508	98.98
160038	160038	160038	160038	160038	160038	160038	160038	0	.	.	.	.	.	.	.	.	.	9	34	0.96	3542	99.94
160038	160038	160038	160038	160038	160038	160038	160038	160038	0	.	.	.	.	.	.	.	.	10	1	0.03	3543	99.97
160038	160038	160038	160038	160038	160038	160038	160038	160038	160038	160038	160038	160038	160038	160038	160038	160038	0	18	1	0.03	3544	100.00

The above shows us that 3,475 matches were made during round 1, 5 matches for round 2, etc. We are essentially adding the variable `ctrf` to the data set `testscores`. To reduce the resources needed to find the matches, the most important thing is to find the value of `ctrf` that produces the match. After all the rounds are done, the variables of interest in data set `students` can be be added to the `matches` data set since we have the variable `ctrf` that identifies which `students` record matches to the `testscores` record.

CONCLUSION

SAS provides a number of ways to match data. An understanding of how SAS processes data and how things can go wrong during the matching process is essential to anticipating problems that need to be addressed when matching data. Being unaware of how problems can occur could potentially allow such problems to occur undetected, and thus endanger the integrity of the matched data.

REFERENCES

Go, Imelda. 2004. *Matching SAS® Data Sets: If at First You Don't Succeed, Match, Match Again.* Proceedings CD of the Twelfth Annual Conference of the SouthEast SAS Users Group, Nashville, TN, USA, 12.

Go, Imelda. 2009. *Matching SAS® Data Sets with PROC SQL: If at First You Don't Succeed, Match, Match Again.* Proceedings CD of the 17th Annual Conference of the Southeast SAS Users Group, Birmingham, AL, USA, 17.

CONTACT INFORMATION

Imelda C. Go
igo@questarai.com
Working remotely from Columbia, SC

TRADEMARK NOTICE