Paper P02-04

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

Imelda C. Go, South Carolina Department of Education, Columbia, SC

ABSTRACT

Two data sets are often matched by using the MERGE and BY statements in a DATA step. If not all records between the two data sets are matched successfully, there may be a need to continue matching those records that did not match using a different set of BY variables. If the goal is to match as many records as possible between the two original data sets, then this process may have to be repeated several times using many sets of BY variables. This paper goes over cautions regarding the matching process described and discusses using macros for more manageable SAS code.

Although SAS programmers in the education field are accustomed to producing student longitudinal data involving test scores for various evaluation and accountability purposes, President George Bush's *No Child Left Behind* (NCLB) legislation has raised the premium for longitudinal data. Because of the high stakes involved in determining *Adequate Yearly Progress* (AYP) among educational agencies (state, district, school), the programmer is charged with the task of producing as many matches as possible.

Critical to any high-stakes programming situation is an understanding of data quality issues, sources of error, and how SAS processes data. Knowing how things can go wrong during the matching process is essential to anticipating problems that need to be addressed during longitudinal data collection. Being unaware of how problems can arise could potentially allow such problems to occur undetected and eventually invalidate the data.

There are also various SAS programming solutions that produce the same results. PROC SQL, mentioned briefly in the paper, is another powerful tool that can be used to match data.

Some of the things that need to be considered when matching data include:

- **Determining the quality of the data**
  Data quality affects the data processing and the programming required to produce the longitudinal data.

- **Determining the matching algorithm**
  Can the data be matched on social security number (SSN) alone? The answer will depend on how reliable the SSN data are and what the project's tolerance for error is. What other sets of variables can the data be matched on? Should the leftover data stay unmatched? Should they be matched using other criteria or even manually inspected to see if more matches can be found? For certain studies, matched case data may be hard to find so a manual inspection may be worth the trouble.

- **Determining the project's tolerance for error**
  The tolerance for error is usually determined by the consequences of an incorrect match. The SSN may have incorrect digits and may accidentally match another SSN. If the SSN is expected to be problematic, consider matching with the SSN and other variables. If the tolerance for error is on the high end, perhaps less stringent or even fuzzy matching methods can be used.

A REVIEW OF DATA STEP MERGING

There is more than one way to match data in SAS. With whatever tool is used, it is important to know how the tool behaves and what its limitations are.

Do Not Use the DATA Step for Many-to-Many Matching

The DATA step can handle one-to-one, one-to-many, and many-to-one matching but not many-to-many matching. For true many-to-many matches, the result should be a cross product. For example, if there are two records that match from one contributing data set to two records from the other, the resulting data set should have $2 \times 2 = 4$ records.

The following example illustrates this important point. Consider two data sets, `data1` and `data2`.

data1 data set

GENDER	NAME1
Female	Linda
Female	Marcy

data2 data set

GENDER	NAME2
Female	May
Female	Gloria

When the data1 records are merged by gender with the data2 records, the ideal result would be 2 × 2 = 4 records.

GENDER	NAME1	NAME2
Female	Linda	May
Female	Linda	Gloria
Female	Marcy	May
Female	Marcy	Gloria

Using the following statements, only two records will result and all possible combinations are **not** given.

```
proc sort data=data1; by gender;
proc sort data=data2; by gender;
data combo; merge data1 data2;
            by gender;
```

Obs	gender	name1	name2
1	Female	Linda	May
2	Female	Marcy	Gloria

If Many-to-Many Matches are a Must

What should be done if the result **must** have all possible combinations? This is possible using the following PROC SQL statement. When large data sets are being matched, the resulting cross product could be a very large data set that requires a lot of system resources.

```
proc sql;
  select data1.gender, name1, name2
  from data1, data2
  where data1.gender=data2.gender;
```

gender	name1	name2
Female	Linda	May
Female	Linda	Gloria
Female	Marcy	May
Female	Marcy	Gloria

Watch Out for the Details

Consider two data sets, time1 and time2. In the examples, SS stands for scaled score.

time1 data set

GRADE	LAST	FIRST	SS	LUNCH	SSN
4	Garbo	Greta	434	R	111111111
3	Davis	Betty	380	R	222222222
2	Taylor	Liz	245	R	333333333
9	Kidman	Nicole	333	R	444444444

time2 data set

GRADE	LAST	FIRST	SS	LUNCH	SSN
5	Garbo	Greta	533	F	111111111
4	Davis	Betty	493	F	222222222
3	Taylor	Liz	399	F	333333333
8	Loren	Sophia	723	F	555555555

When using a DATA step to merge time1 and time2 by ssn, the two data sets need to be sorted by ssn (or the BY-variables). When contributing data sets have variables with the same name, the variables need to be renamed in order to prevent the values of one data set from overwriting the values of the other data set during the merge.

The results shown below are problematic because the variables with the same name were not renamed or dropped from either data set.

```
proc sort data=time1; by ssn;
proc sort data=time2; by ssn;

data test; merge time1 time2; by ssn;
```

OBS	GRADE	LAST	FIRST	SS	LUNCH	SSN
1	5	Garbo	Greta	533	F	111111111
2	4	Davis	Betty	493	F	222222222
3	3	Taylor	Liz	399	F	333333333
4	9	Kidman	Nicole	333	R	444444444
5	8	Loren	Sophia	723	F	555555555

The following data step shows some variables being dropped and renamed prior to merging. The resulting data set has correct values.

```
data test;
merge time1 (drop=grade lunch rename=(ss=ss1))
    time2 (drop=grade lunch rename=(ss=ss2));
by ssn;
```

OBS	LAST	FIRST	SS1	SSN	SS2
1	Garbo	Greta	434	111111111	533
2	Davis	Betty	380	222222222	493
3	Taylor	Liz	245	333333333	399
4	Kidman	Nicole	333	444444444	.
5	Loren	Sophia	.	555555555	723

What happens if the BY statement is *accidentally* omitted from the previous example? No error message will appear in the log because it is valid SAS syntax and SAS does merges without BY statements. The records are merged in the order in which they occur in the data set and without regard to any other criteria. The resulting data are invalid and shown below. In the fourth record, Sophia Loren's (`ssn` = 555555555) information is listed with Nicole Kidman's (`ssn` = 444444444) `ss1` value.

```
data test;
merge time1 (drop=grade lunch rename=(ss=ss1))
      time2 (drop=grade lunch rename=(ss=ss2));
```

OBS	LAST	FIRST	SS1	SSN	SS2
1	Garbo	Greta	434	111111111	533
2	Davis	Betty	380	222222222	493
3	Taylor	Liz	245	333333333	399
4	Loren	Sophia	333	555555555	723

The DATA step can handle one-to-many and many-to-one matches properly. If there is 1 record from the first source data set and there are 2 records from the second source data set for the same `ssn`, then the resulting data set will have 2 records. If the ultimate goal is to have only one record per `ssn`, then make sure there is only one record per `ssn` prior to the merge, or consolidate the multiple records into one record after the merge. If multiple records per `ssn` are invalid at any time, then that needs to be addressed during data validation.

Source of Resulting Records

Is there a way to determine which data set contributed to the resulting merged record? Use the IN= data set option. The `in1` variable is a 0/1 indicator of whether the `time1` data set contributed to the current observation (`in1=0` if it did not and `in1=1` if it did). The `in2` variable is the 0/1 indicator with respect to the `time2` data set.

```
data test;
merge time1 (in=in1 drop=grade lunch
             rename=(ss=ss1))
      time2 (in=in2 drop=grade lunch
             rename=(ss=ss2)); by ssn;
      if in1 and in2 then source='both ';
         else if in1 then source='time1';
         else if in2 then source='time2';
```

OBS	LAST	FIRST	SS1	SSN	SS2	SOURCE
1	Garbo	Greta	434	111111111	533	both
2	Davis	Betty	380	222222222	493	both
3	Taylor	Liz	245	333333333	399	both
4	Kidman	Nicole	333	444444444	.	time1
5	Loren	Sophia	.	555555555	723	time2

MATCHING ON KEY VARIABLES

Matching can be done on one or more key variables. When only one variable is used, a nonmatch implies the values on that one variable were not the same. If two variables are used for the match, a nonmatch implies the values differed on one or both variables. As more variables are used, the matching criteria become more stringent. Matches must be equal on more variables and differences between those variables are more likely to occur than if fewer variables were used.

When a unique identifier, such as the SSN, is used to match, make sure the SSN data type is the same. It is either numeric or character across all data sets that need to be matched on SSN.

Would there even be a situation where the SSN should be a character type? The answer depends on the nature of the data. If one or more digits of the SSN are missing and a space is used to indicate the missing digit(s), the character type would preserve the information. If the SSN of '251112222' is bubbled on a machine-scannable form and some of the bubble marks were not scanned properly, the data file might have '25?12222' where ? is a digit that did not scan properly. If the data type is numeric, '251 12 22' (4^{th} and 7^{th} digits are missing) and '25?12222' are invalid numeric values and are set to missing.

When data are matched on character variables (e.g., last name, first name), make sure the character values are as consistent as possible. Reducing inconsistencies in character values can help maximize the number of successful matches. However, people can also change their names over time.

Comparing strings is case-sensitive. That is, `'Edward'` is not the same as `'EDWARD'`. Typing is subject to human error. For example, `'EDWARD'` may have been typed as `'EDWard'`. One solution is to use the UPCASE or LOWCASE functions.

sample statement	value of test
test=upcase('Edward');	'EDWARD'
test=lowcase('EDWard');	'edward'

`'Mary Ann'` might be entered as `'Mary  Ann'` where there are two spaces between `Mary` and `Ann`. Use the COMPBL function to convert two or more consecutive blanks into one blank.

test=compbl(name);

value of name	value of test
'Mary Ann'	'Mary Ann'
'Mary Ann'	'Mary Ann'

`'Mary Ann'` might be entered as `' Mary Ann'` where there is a leading space before `Mary.` Use the LEFT function to left align a character expression.

test=left(name);

value of name	value of test
' Mary Ann'	'Mary Ann'
' Mary Ann'	'Mary Ann'

SAS ignores trailing spaces when character expressions are compared (e.g., 'Mary Ann ' is equivalent to 'Mary Ann'). If there are any unwanted spaces in the first name, use the COMPRESS function to remove specified characters from a string. If no characters for removal are specified, the function removes spaces by default.

```
test=compress(name);
```

value of name	value of test
'Edw ard'	'Edward'
' Edward'	'Edward'

Is it possible to have a first name with two words? Whatever the answer is, consistency is the key word when matching. If all the spaces are removed from all the variables used to match, then the only issue is whether the original name value needs to be retained. If there is any such concern, keep two name fields. The first one would be the original name, the second one would be the edited name used for matching.

Other characters may also need to be removed. List the characters to be removed in single quotes.

```
test=compress(name,'?-');
```

value of name	value of test
'Kidman-Cruise?'	'KidmanCruise'

There may be an attempt to type accent marks into the value of a name. For example, *André* might be typed in as Andre' and this is really a data entry standard issue. When the typist encounters an accent mark, should they attempt to type the accent mark? That particular accent mark lends itself well to an apostrophe. There are of course other foreign accent marks (e.g., ö) that would not lend themselves well to plain text characters.

What if the apostrophe itself needs to be eliminated? Use the following syntax.

```
test=compress(name,'''');
```

value of name	value of test
Andre'	Andre

Should nonletter characters be eliminated? The answer depends on the situation. An actual example encountered is the accidental or intentional typing of nonletter characters in names on student databases. Student names are sent to a testing company that creates bar-coded labels for test documents. In the process of creating the labels, the testing company eliminates all nonletter characters from the student names. Not all students will have documents with bar-coded labels during testing. Such students indicate their names by bubbling a machine-scannable form. The forms often only provide bubbles for letter characters in name fields. If data on the student database need to be matched to data received from the testing company later, original names need to be matched with the names from the testing company. The first set of names has nonletter characters while the second set of names has no such characters. The number of matches would not be maximized if the nonletter characters are not edited out of the original names prior to matching.

There are many data sources and data entry standards. Errors may also occur during the import and export of data between various people and computer systems. Files can get corrupted. A programmer might accidentally leave out the last digit of the SSN being written out to a raw data file, etc.

PROGRAMMING EXAMPLE

The programming example uses the following data sets. The variable names were assigned to facilitate the discussion of the examples. The variables in data set A all begin with A and those in data set B all begin with B.

Description	Data set A Variables	Data set B Variables
School	Aschool	Bschool
Student ID	Astudentid	Bstudentid
Name	Aname	Bname
Grade	Agrade	Bgrade
Date of Birth	Adob	Bdob

Using data sets A and B, the goal is to match as many records as possible between the two data sets. Another constraint is the matching that needs to be done can only involve one-to-one matches. That is, one-to-many, many-to-one, and many-to-many matches are invalid. Hence, additional SAS statements were written (see steps 1.3 and 2.3 below) to exclude records that will produce a many-to-many situation from the data sets being merged. However, the excluded records in step 1.3 are later considered for stage 2 matching (step 2.1).

Two stages of matching will be used. The first one involves matching by student ID and name while the second one involves matching by school, name, and date of birth. The next two pages explain the steps in the process. The example assumes that the source data sets A and B have been validated and the variables have been examined to maximize matching on key variables.

THE PROCESS OUTLINED

The original input data sets are data sets A and B. To differentiate between the various data sets that contain records from these original data sets, the initial input data sets will be called ainput1 and binput1 for stage 1. Data sets ainput1 and binput1 are the same as data sets A and B respectively in step 1.1.

Stage 1:

Matching the data by student ID and name

1.1

For both input data sets, create the common student ID and name variables to be used in the BY statement used with the MERGE statement.

```
*1.1;
data ainput1;
  set a;
  studentid=astudentid; name=aname;
data binput1;
  set b;
  studentid=bstudentid; name=bname;
```

1.2

Sort data sets ainput1 and binput1 by student ID and name.

```
*1.2;
proc sort data=ainput1; by studentid name;
proc sort data=binput1; by studentid name;
```

1.3

Using a DATA step, remove records from data sets ainput1 and binput1 that have duplicated student ID and name combinations and reserve them for stage 2 of the matching. The following data sets result:

ainput1: This is the new ainput1 data set, which now only has records with unduplicated student ID and name combinations within the data set.
binput1: This is the new binput1 data set, which now only has records with unduplicated student ID and name combinations within the data set.
adup1: These were the records removed from the original data set ainput1 in this step. Since these will be used in stage 2, drop the student ID and name variables.
bdup1: These were the records removed from the data set binput1 in this step. Since these will be used in stage 2, drop the student ID and name variables.

```
*1.3;
data ainput1 adup1 (drop=studentid name);
  set ainput1; by studentid name;
  if first.name and last.name
     then output ainput1;
     else output adup1;
data binput1 bdup1 (drop=studentid name);
  set binput1; by studentid name;
  if first.name and last.name
     then output binput1;
     else output bdup1;
```

1.4

Match the data by student ID and name and generate three data sets. Make sure that the only variables in data sets A and B are the original data set A1 and B1 variables respectively.

M1: These are the successful matches.
A1: These are the records from data set A that did not match.
B1: These are the records from data set B that did not match.

The student ID and name variables can be dropped from the M1 data set. The variable recordsource is set to 1 to indicate the stage the successfully matched records came from.

```
*1.4;
data m1 (drop=studentid name)
     a1 (keep= aschool
         astudentid aname agrade adob)
     b1 (keep= bschool
         bstudentid bname bgrade bdob);
  merge ainput1 (in=ina) binput1 (in=inb);
  by studentid name;
  if ina and inb
    then do; recordsource=1; output m1; end;
    else if ina then output a1;
    else if inb then output b1;
```

Stage 2:

Matching by school, name, and date of birth the data not matched in Stage 1

2.1

Construct the input data sets. These will be called data sets `ainput2` and `binput2` for stage 2. These two data sets contain records that are not in `M1` and that have not been matched in stage 1.

`ainput2`: These are records from `adup1` and `A1`.
`binput2`: These are records from `bdup1` and `B1`.

For both input data sets, create the common school, name, and date of birth variables to be used in the BY statement used with the MERGE statement.

```
*2.1;
data ainput2;
  set a1 adup1;
  school=aschool; name=aname; dob=adob;
data binput2;
  set b1 bdup1;
  school=bschool; name=bname; dob=bdob;
```

2.2

Sort `ainput2` and `binput2` by school, name, and date of birth.

```
*2.2;
proc sort data=ainput2; by school name dob;
proc sort data=binput2; by school name dob;
```

2.3

Using a DATA step, remove records from data sets `ainput2` and `binput2` that have duplicated school, name, and date of birth combinations and reserve them for later use should there be a need to continue matching after stage 2. The following data sets result:

`ainput2`: This is the new `ainput2` data set, which now only has records with unduplicated school, name, and date of birth combinations within the data set.
`binput2`: This is the new `binput2` data set, which now only has records with unduplicated school, name, and date of birth combinations within the data set.
`adup2`: These were the records removed from the original data set `ainput2` in this step. Since these might be used for further matching beyond stage 2, drop the school, name, and date of birth variables.
`bdup2`: These were the records removed from the original data set `binput2` in this step. Since these might be used for further matching beyond stage 2, drop the school, name, and date of birth variables.

```
*2.3;
data ainput2 adup2 (drop=school name dob);
  set ainput2; by school name dob;
  if first.dob and last.dob
     then output ainput2;
     else output adup2;
data binput2 bdup2 (drop=school name dob);
  set binput2; by school name dob;
  if first.dob and last.dob
     then output binput2;
     else output bdup2;
```

2.4

Match the data by student ID and name and generate three data sets. Make sure that the only variables in data sets `A2` and `B2` are the original data set `A` and `B` variables respectively.

`M2`: These are the successful matches.
`A2`: These are the records from data set `A` that did not match.
`B2`: These are the records from data set `B` that did not match.

The school, name, and date of birth variables can be dropped from the `M2` data set. The variable `recordsource` is set to 2 to indicate the stage the successfully matched records came from.

```
*2.4;
data m2 (drop=school name dob)
     a2 (keep= aschool
         astudentid aname agrade adob)
     b2 (keep= bschool
         bstudentid bname bgrade bdob);
  merge ainput2 (in=ina) binput2 (in=inb);
  by school name dob;
  if ina and inb
    then do; recordsource=2; output m2; end;
    else if ina then output a2;
    else if inb then output b2;
```

If the matching ends after two stages, then the successfully matched records are the records in data sets M1 and M2.

```
data finalmatchesafter2stages;
  set m1 m2;
```

If there is going to be another matching stage that uses a different set of BY variables than the preceding two stages, then the input data sets can be named ainput3 and binput3 for stage 3. These two data sets contain records that are not in M1 and that are not in M2, and contain records that have not been matched in the previous stages.

ainput3: These are records from a2 and adup2.
binput3: These are records from b2 and bdup2.

The process can be easily extended using the steps outlined above. Because of the potentially large number of data sets involved, it is best to use an intuitive naming convention that the programmer can easily follow in order to trace the results of the program. The following sample data sets can be used to trace the programming statements.

A data set

Obs	aschool	astudentid	aname	agrade	adob
1	ABC School	12	Jane	2	19950101
2	ABC School	13	Jim	2	19950101
3	ABC School	14	Jane	2	19950101
4	ABC School	15	Jim	2	19950101
5	ABC School	19	Jane	2	19950101
6	ABC School	19	Jane	2	19950101

B data set

Obs	bschool	bstudentid	bname	bgrade	bdob
1	ABC School	12	Jane	2	19950101
2	ABC School	13	Jim	2	19950101
3	ABC School	16	Jane	2	19950101
4	ABC School	17	Jim	2	19950101
5	ABC School	19	Jane	2	19950101
6	ABC School	19	Jane	2	19950101

ainput1 data set after step 1.1

Obs	aschool	astudentid	aname	agrade	adob	studentid	name
1	ABC School	12	Jane	2	19950101	12	Jane
2	ABC School	13	Jim	2	19950101	13	Jim
3	ABC School	14	Jane	2	19950101	14	Jane
4	ABC School	15	Jim	2	19950101	15	Jim
5	ABC School	19	Jane	2	19950101	19	Jane
6	ABC School	19	Jane	2	19950101	19	Jane

binput1 data set after step 1.1

Obs	bschool	bstudentid	bname	bgrade	bdob	studentid	name
1	ABC School	12	Jane	2	19950101	12	Jane
2	ABC School	13	Jim	2	19950101	13	Jim
3	ABC School	16	Jane	2	19950101	16	Jane
4	ABC School	17	Jim	2	19950101	17	Jim
5	ABC School	19	Jane	2	19950101	19	Jane
6	ABC School	19	Jane	2	19950101	19	Jane

ainput1 data set after step 1.3

Obs	aschool	astudentid	aname	agrade	adob	studentid	name
1	ABC School	12	Jane	2	19950101	12	Jane
2	ABC School	13	Jim	2	19950101	13	Jim
3	ABC School	14	Jane	2	19950101	14	Jane
4	ABC School	15	Jim	2	19950101	15	Jim

binput1 data set after step 1.3

Obs	bschool	bstudentid	bname	bgrade	bdob	studentid	name
1	ABC School	12	Jane	2	19950101	12	Jane
2	ABC School	13	Jim	2	19950101	13	Jim
3	ABC School	16	Jane	2	19950101	16	Jane
4	ABC School	17	Jim	2	19950101	17	Jim

adup1 data set after step 1.3

Obs	aschool	astudentid	aname	agrade	adob
1	ABC School	19	Jane	2	19950101
2	ABC School	19	Jane	2	19950101

bdup1 data set after step 1.3

Obs	bschool	bstudentid	bname	bgrade	bdob
1	ABC School	19	Jane	2	19950101
2	ABC School	19	Jane	2	19950101

a1 data set after step 1.4

Obs	aschool	astudentid	aname	agrade	adob
1	ABC School	14	Jane	2	19950101
2	ABC School	15	Jim	2	19950101

b1 data set after step 1.4

Obs	bschool	bstudentid	bname	bgrade	bdob
1	**ABC School**	**16**	**Jane**	**2**	**19950101**
2	**ABC School**	**17**	**Jim**	**2**	**19950101**

ainput2 data set data set after step 2.1

Obs	aschool	astudentid	aname	agrade	adob	school	name	dob
1	ABC School	14	Jane	2	19950101	ABC School	Jane	19950101
2	ABC School	15	Jim	2	19950101	ABC School	Jim	19950101
3	ABC School	19	Jane	2	19950101	ABC School	Jane	19950101
4	ABC School	19	Jane	2	19950101	ABC School	Jane	19950101

binput2 data set data set after step 2.1

Obs	bschool	bstudentid	bname	bgrade	bdob	school	name	dob
1	ABC School	16	Jane	2	19950101	ABC School	Jane	19950101
2	ABC School	17	Jim	2	19950101	ABC School	Jim	19950101
3	ABC School	19	Jane	2	19950101	ABC School	Jane	19950101
4	ABC School	19	Jane	2	19950101	ABC School	Jane	19950101

ainput2 data set data set after step 2.3

Obs	aschool	astudentid	aname	agrade	adob	school	name	dob
1	ABC School	15	Jim	2	19950101	ABC School	Jim	19950101

binput2 data set after step 2.3

Obs	bschool	bstudentid	bname	bgrade	bdob	school	name	dob
1	ABC School	17	Jim	2	19950101	ABC School	Jim	19950101

adup2 data set after step 2.3

Obs	aschool	astudentid	aname	agrade	adob
1	ABC School	14	Jane	2	19950101
2	ABC School	19	Jane	2	19950101
3	ABC School	19	Jane	2	19950101

bdup2 data set after step 2.3

Obs	bschool	bstudentid	bname	bgrade	bdob
1	ABC School	16	Jane	2	19950101
2	ABC School	19	Jane	2	19950101
3	ABC School	19	Jane	2	19950101

m1 data set

Obs	aschool	astudentid	aname	agrade	adob	bschool	bstudentid	bname	bgrade	bdob	recordsource
1	ABC School	12	Jane	2	19950101	ABC School	12	Jane	2	19950101	1
2	ABC School	13	Jim	2	19950101	ABC School	13	Jim	2	19950101	1

m2 data set

Obs	aschool	astudentid	aname	agrade	adob	bschool	bstudentid	bname	bgrade	bdob	recordsource
1	ABC School	15	Jim	2	19950101	ABC School	17	Jim	2	19950101	2

FIRST. AND LAST. VARIABLES

Steps 1.3 and 2.3 use the FIRST. and LAST. variables. These are automatic variables generated by SAS when a SET and a BY statement are used in a DATA step. Both types of variables are created for each variable listed in the BY statement. Suppose there is only one BY variable in the BY statement. The FIRST. variable is 1 for the *first* record in the BY group and 0 for all other records in the BY group. The LAST. variable is 1 for the *last* observation in the BY group and 0 for all other records in the BY group. A BY group involving one BY variable is a set of records with the same value for that BY variable. The example, below, has five BY groups for `studentid`: 12, 13, 14, 15, 19. In general, a *BY group for a set of BY variables* is a group of records with the same values for each variable in that set of BY variables.

The same definition for creating FIRST. and LAST. variables is used for all the variables listed in a BY statement. However, the definition for the n^{th} BY variable applies to each BY group involving all preceding (1^{st} till $n\text{-}1^{th}$ BY variables according to the order they are listed in the BY statement). For example, in step 1.3 two BY variables were used: `studentid` and `name`. The `first.studentid` and `last.studentid` values, shown below, has exactly the same values had the BY statement only involved `studentid`. The example has five BY groups for `studentid`: 12, 13, 14, 15, 19. The `first.name` and `last.name` variables are determined by applying the definition to each of the five `studentid` BY groups.

STUDENTID	NAME	FIRST.SUTDENTID	LAST.STUDENTID	FIRST.NAME	LAST.NAME
12	Jane	1	1	1	1
13	Jim	1	1	1	1
14	Jane	1	1	1	1
15	Jim	1	1	1	1
19	Jane	1	0	1	0
19	Jane	0	1	0	1

Suppose there were three variables in the BY statement: `studentid`, `name`, and `dob`. The FIRST. and LAST. variables for `studentid` and `name` would be the same as shown above. To get the pair of values for `dob`, the definition would be applied to each `studentid` and `name` BY group and there are also five of these in the example above.

USING MACROS

The steps in stages 1 and 2 have a noticeable pattern and can be generalized using macros. SAS macros can be used to facilitate coding for several stages of matching. The following statements can be used to replace the previous code.

```
%macro merging(stage,byvars,lastbyvar);

%let avars= aschool astudentid aname agrade adob;
%let bvars= bschool bstudentid bname bgrade bdob;
```

```
%*stage = stage number;
%*byvars = variables for BY statement;
%*lastbyvar = last variable in byvars;

*1.2 and 2.2;
proc sort data=ainput&stage; by &byvars;
proc sort data=binput&stage; by &byvars;

*1.3 and 2.3;
data ainput&stage adup&stage (drop=&byvars);
  set ainput&stage; by &byvars;
  if first.&lastbyvar and last.&lastbyvar
     then output ainput&stage;  else output adup&stage;

data binput&stage bdup&stage (drop=&byvars);
  set binput&stage; by &byvars;
  if first.&lastbyvar and last.&lastbyvar
     then output binput&stage;  else output bdup&stage;

*1.4 and 2.4;
data m&stage (drop=&byvars)
     a&stage (keep=&avars)
     b&stage (keep=&bvars);
  merge ainput&stage (in=ina) binput&stage (in=inb);
  by &byvars;
  if ina and inb then do; recordsource=&stage; output m&stage; end;
     else if ina then output a&stage;
     else if inb then output b&stage;
run;
%mend merging;

data ainput1; set a; studentid=astudentid; name=aname;
data binput1; set b; studentid=bstudentid; name=bname;

%MERGING(STAGE=1,  BYVARS=studentid name,  LASTBYVAR=name);

data ainput2; set a1 adup1;  school=aschool; name=aname; dob=adob;
data binput2; set B1 bdup1;  school=bschool; name=bname; dob=bdob;

%MERGING(STAGE=2,  BYVARS=school name dob,  LASTBYVAR=dob);

data finalmatchesafter2stages;
  set m1 m2;
```

If there is a need to go beyond two stages, the previous macro can be used to extend the programming easily. Even more aspects of the program can be rewritten using macros. However, it is good to maintain a balance between the use of macros and the readability of programs. Sometimes the time saved by using macros is nullified when a program has little or no documentation, is difficult to read, or cannot be easily modified especially by someone with fewer SAS macro skills than the original programmer.

What happens to those records that fail to match after all the stages? Depending on how high the stakes are, a person can visually inspect the records that failed to match to determine if there are any more possible matches. Such inspections often provide insights into the nature of the data. These insights can sometimes be incorporated into the programming to improve the matching process. For example, if two data sets failed to produce much fewer than expected matches, a visual inspection showed that one data set's last name field is only 10 characters long instead of the 14 characters in the other data set. If last name data with the same length cannot be obtained, then adjustments in the programming need to be made.

CONCLUSION

An understanding of data quality issues, sources of error, how SAS processes data, and how things can go wrong during the matching process are 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 eventually invalidate the data.

REFERENCES

SAS Institute Inc., *SAS® Language Reference, Version 8*, Cary, NC: SAS Institute Inc., 1999. 1256 pp.
SAS Institute Inc., *SAS OnLineDoc®, Version 8*, Cary, NC: SAS Institute Inc., 1999.

TRADEMARK NOTICE