

Overcoming the Challenges of Longitudinal Data Collection

Imelda C. Go, Lexington County School District One, Lexington, SC

ABSTRACT

This paper discusses problems encountered when data are collected longitudinally. Longitudinal data collection implies data collected over time are to be matched on key variables (e.g., social security number, last name, first name). Data for the key variables are not always consistent due to a variety of reasons (e.g., human error, name changes). This lack of consistency creates problems during the matching process. The paper also discusses matching data using the DATA step versus PROC SQL.

INTRODUCTION

Using SAS to produce longitudinally matched data involves understanding several aspects of how SAS processes data. There are also various SAS programming solutions that produce the same results. The paper does not go into great detail regarding all these aspects. However, it provides an overview of things that need to be considered in producing longitudinal data. Most examples are actual situations encountered with school test data.

Some of the things that need to be considered include:

- Determining the quality of the data

The 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)? 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 that did not match 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 already 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.

The rest of the paper is divided into four parts:

- Data quality
- Matching on key variables
- DATA step
- PROC SQL

DATA QUALITY

Input data can come from a variety of sources and it is not always possible to control how they are created and their quality. High quality data naturally require less troubleshooting and programming. Some strategies for preventing data processing problems are:

- Establish data entry standards. For example, how should name suffixes (e.g., Jr., II, III) be entered as data? Should there be another field specifically for name suffixes? Consistent data facilitates programming.
- Prevent invalid values from being entered as data. For example, use lookup tables that restrict values of variables to valid values only.
- Validate the data prior to processing. The quality of the data can be gauged by validating the data. For example, does the data set contain what it is expected to contain? If it is supposed to have data for 5-year old students, but it contains data for 10-year old students, there is something obviously wrong.
- Plan the data's structure to be compatible with the uses of the data. For data used in applications that rely heavily on street address data (e.g., transportation routing, mapping, geographic information systems), it might be useful to decompose a street address into a number of variables (street number, street name, street type, and apartment number) instead of using one variable to represent it.
- Know the data. Be aware of the sources of data, sources of error, and the quality of the data. Poor data quality often interferes with data processing and is a major cause of processing delays. It not only creates problems during processing, but it casts doubt on the results.

MATCHING ON KEY VARIABLES

Matching can be done on one or more key variables. When only one variable is used, a non-match implies the values on that one variable were not the same. If two variables are used for the match, a non-match 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 scan 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' (4th and 7th 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 COMB1 function to convert two or more consecutive blanks into one blank.

test=comb1(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., ð) 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 non-letter characters be eliminated? The answer depends on the situation. An actual example encountered is the accidental or intentional typing of non-letter 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 non-letter 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 scan form. Scan 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 non-letter characters while the second set of names has no such characters. The number of matches would not be maximized if the non-letter 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.

POSTERS

DATA STEP

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

Using a `DATA` step to merge `time1` and `time2` by `ssn` has disadvantages. When data are merged this way, 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 `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 result should have $2 \times 2 = 4$ records in the merged data set.

The merge results shown below are problematic because of the many-to-many situation and 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;
```

```
proc print;
```

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 be given because it is valid SAS syntax and SAS does merges without `BY` statements. The records are merged in the order in which they occur on the data set and without regard to any other criteria. The resulting data are invalid and shown below.

```
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.

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

PROC SQL

Consider another two data sets, *time1* and *time2* below.

time1 data set

GRADE	LAST	FIRST	SS	LUNCH	SSN
4	Garbo	Greta	434	R	111111111
2	Taylor	Liz	245	R	333333333
2	Taylor	Liza	232	R	333333333
9	Kidman	Nicole	333	R	444444444

time2 data set

GRADE	LAST	FIRST	SS	LUNCH	SSN
5	Garbo	Greta	533	F	111111111
3	Taylor	Liz	399	F	333333333
3	Monroe	Marilyn	387	F	333333333

The following PROC SQL example will produce the correct results when many-to-many matches are involved. PROC SQL would produce all the possible matches while the DATA step would not. PROC SQL is a powerful procedure and can also perform, under one procedure, what could be done using a combination of DATA step statements, and the PRINT, SORT, and SUMMARY procedures.

```
proc sql;
  select time1.ss as ss1,
         time2.ss as ss2,
         time1.last as last1,
         time2.last as last2,
         time1.first as first1,
         time2.first as first2
  from time1,time2
  where time1.ssn=time2.ssn
  order by ss1, ss2;
```

SS1	SS2	LAST1	LAST2	FIRST1	FIRST2
232	387	Taylor	Monroe	Liza	Marilyn
232	399	Taylor	Taylor	Liza	Liz
245	387	Taylor	Monroe	Liz	Marilyn
245	399	Taylor	Taylor	Liz	Liz
434	533	Garbo	Garbo	Greta	Greta

Suppose there is a need to limit the results to those matches that show a greater than 150-point gain in scaled score points from *time 1* to *time 2*. The WHERE statement is expanded to include the criterion *time2.ss-time1.ss>150*. This difference is also computed as *ssdiff* and included in the query by adding *time2.ss-time1.ss as sssdiff* to the SELECT statement.

```
proc sql;
  select time1.ss as ss1,
         time2.ss as ss2,
         time2.ss-time1.ss as sssdiff,
         time1.last as last1,
         time2.last as last2,
         time1.first as first1,
         time2.first as first2
  from time1,time2
  where time1.ssn=time2.ssn and
         time2.ss-time1.ss>150;
```

SS1	SS2	SSDIFF	LAST1	LAST2	FIRST1	FIRST2
245	399	154	Taylor	Taylor	Liz	Liz
232	399	167	Taylor	Taylor	Liza	Liz
232	387	155	Taylor	Monroe	Liza	Marilyn

The data above were matched on *ssn*. Examining names for records with the same *ssn* value from both data sets show differences (Liza Taylor vs. Liz Taylor, and Liza Taylor vs. Marilyn Monroe). If the *ssn* values are correct and unique, then perhaps the name values are incorrect, or legal name changes were involved, or aliases were used. If the *ssn* values were incorrect, then the example illustrates a disadvantage of using only one variable to match data. If the *ssn* values are not considered very reliable, using more variables than *ssn* alone to verify an individual's identity can prevent incorrect matches.

Regarding the plausibility of two individuals with practically the same name having almost the same SSN, examination of large databases with student biographical data revealed identical twins with the same sex, the same birth date, almost identical names (e.g., Sierra & Tierra), and almost the same SSN (e.g., consecutive numbers -- 222-22-2212 and 222-22-2213).

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 during longitudinal data collection. 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

SAS is a registered trademark or trademark of the SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Imelda C. Go (icgo@juno.com)
 Lexington County School District One
 Lexington, SC