Paper 1762-2015

Did Your Join Work Properly?

Emmy Pahmer, inVentiv Health Clinical

ABSTRACT

We have many options for performing merges or joins these days, each with various advantages and disadvantages. Depending on how you perform your joins, different checks can help you verify whether the join was successful. In this presentation, we look at some sample data, use different methods, and see what kinds of tests can be done to ensure that the results are correct. If the join is performed with PROC SQL and two criteria are fulfilled (the number of observations in the primary data set has not changed [presuming a one-to-one or many-to-one situation], and a variable that should be populated is not missing), then the merge was successful.

INTRODUCTION

In my experience, many-to-one joins are the most common, so this is the type that we'll be taking into consideration. The objective is to see if the join was successful so we'll go through a few simple steps to check the join. The checks can be added to your code so that they are done automatically every time the program is run.

PRIMARY DATA

We'll use this small sample as our primary data, and we want to assign a VISIT to each record. Note that there are 12 observations.

	subject	period	hour
1	101	1	0
2	101	1	1
3	101	1	72
4	101	2	0
5	101	2	1
6	101	2	96
7	102	1	0
8	102	1	1
9	102	1	72
10	102	2	0
11	102	2	1
12	102	2	96

SECONDARY DATA

Here's what our secondary data looks like.

	period	hour	visit
1	1	0	Period 1 Pre-Dose
2	1	1	Period 1 1H Post-Dose
3	1	72	Period 1 72H Post-Dose
4	1	96	Period 1 Follow-Up
5	2	0	Period 2 Pre-Dose
6	2	1	Period 2 1H Post-Dose
7	2	72	Period 2 72H Post-Dose
8	2	96	Follow-Up

By matching the values of PERIOD and HOUR, we will assign VISIT.

STEP 1: COUNT THE NUMBER OF OBSERVATIONS OF PRIMARY DATA PRIOR TO JOIN

```
proc sql;
   select nobs into :nobs1
   from dictionary.tables
   where libname = 'WORK' and memname = 'P1';
quit;
```

The result is 12 observations. The macro variable NOBS1 has a value of 12.

STEP 2: DO THE JOIN/MERGE

```
proc sql;
      create table combined1 as
      select a.*, b.visit
      from p1 as a left join s1 as b
      on a.period = b.period and
         a.hour = b.hour;
quit;
```

STEP 3: COUNT AGAIN AND COMPARE

1. DOES THE PRIMARY DATA CONTAIN THE SAME NUMBER OF OBSERVATIONS?

Count the number of observations again, as above.

```
proc sql;
   select nobs into :nobs2
   from dictionary.tables
   where libname = 'WORK' and memname = 'COMBINED1';
quit;
```

The result is 12 observations. The macro variable NOBS2 has a value of 12. We'll compare the number before and after and write a note to the log.

```
%macro compare_nobs2;
%if &nobs1 ne &nobs2 %then %put %str(W)ARNING: number of observations is not
the same. Before: %cmpres(&nobs1).  After: %cmpres(&nobs2).;
%else %put NOBS1 and NOBS2 are the same (%cmpres(&nobs1.)) --> OK. ;
%mend;
%compare_nobs2;
```

LOG:

```
NOBS1 and NOBS2 are the same (12) --> OK.
```

2. IS VISIT POPULATED FOR ALL RECORDS?

Get unique cases, where VISIT is missing, based on the merge variables.

```
proc freq data = combined1 (where = (missing(visit))) noprint;
   table period * hour / out = not_assigned;
run;
```

If there are any, write a warning message in the log:

```
data _null_;
   set not_assigned;
   put 'W' 'ARNING: VISIT not assigned. ' period= hour=;
run;
```

In this case all records have been assigned a value for VISIT and there is no warning message written to the log:

```
NOTE: There were 0 observations read from the data set WORK.NOT_ASSIGNED.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

This is the desired result, no warnings for either check.

EXAMPLE WITH A DUPLICATE IN THE SECONDARY DATA

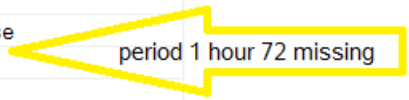
	period	hour	visit
1	1	0	Period 1 Pre-Dose
2	1	1	Period 1 1H Post-Dose
3	1	72	Period 1 72H Post-Dose
4	1	96	Period 1 Follow-Up
5	2	0	Period 2 Pre-Dose
6	2	1	Period 2 1H Post-Dose
7	2	72	Period 2 72H Post-Dose
8	2	96	Period 2 96H Post-Dose
9	2	96	Follow-Up

Result of checking the number of observations:

```
WARNING: number of observations is not the same. Before: 12.  After: 14.
```

EXAMPLE WITH A VALUE MISSING IN THE SECONDARY DATA

	period	hour	visit
1	1	0	Period 1 Pre-Dose
2	1	1	Period 1 1H Post-Dose
3	1	96	Period 1 Follow-Up
4	2	0	Period 2 Pre-Dose
5	2	1	Period 2 1H Post-Dose
6	2	72	Period 2 72H Post-Dose
7	2	96	Follow-Up



Result of checking for missing values:

```
WARNING: VISIT not assigned. period=1 hour=72
```

MACRO-IZED

To run these checks without changing the code:

1. **CHECK 1**

MACRO CODE:

```
** MACRO TO COMPARE NUMBER OF OBSERVATIONS IN 2 DATASETS, GIVE WARNING
MESSAGE IF THEY ARE NOT THE SAME **;
%macro compare_nobs(before = , after=);
%if &&&before ne &&&after %then
   %put %str(W)ARNING: number of observations is not the same. Before:
%cmpres(&&&before).  After: %cmpres(&&&after).;
%else %put &before and &after are the same (%cmpres(&&&before.)) --> OK. ;
      %let &before = ;
      %let &after = ;
%mend;
```

MACRO CALL:

Macro variables NOBS1 and NOBS2 have already been created with PROC SQL as above.

```
%compare_nobs(before = nobs1, after= nobs2);
```

2. **CHECK 2**

MACRO CODE:

```
** MACRO TO CHECK WHERE SECONDARY DATA DID NOT POPULATE AFTER THE MERGE.
PRINT ONE LINE FOR EACH UNIQUE CASE **;
%macro print_unmerged(ds= , reqd_var= , by_vars=);

proc freq data = &ds (where = (missing(&reqd_var))) noprint;
      table &by_vars / out = not_assigned;
```

```
run;

data _null_;
      set not_assigned (drop = count percent);
      put 'W' "ARNING: &reqd_var not assigned. " (_all_)(=);
run;

%mend;
```

MACRO CALL:

```
%print_unmerged(ds= combined1,     /* dataset to check */
                reqd_var= visit,   /* the variable that should be populated */
                by_vars= %str(period * hour)); /* by variables, asterisk
                                                  between them */
```

CONCLUSION

With two simple checks, you can see if your merge/join worked properly. If it did not, a warning message is sent to the log so no extra checking is required. If the merge was performed with a MERGE statement or with PROC SQL, these checks will work on one-to-one or many-to-one joins.

ACKNOWLEDGMENTS

Thanks to Bruce Gilsen for reviewing the paper.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Emmy Pahmer
inVentiv Health Clinical
emmy.pahmer@inventivhealth.com