Paper 7720-2016

Omitting Records with Invalid Default Values

Lily Yu, Statistics Collaborative Inc.

ABSTRACT

Many databases include default values that are set inappropriately. These default values may be scattered throughout the database. There may be no pattern to their occurrence. Records without valid information should be omitted from statistical analysis. Choosing the best method to omit records with invalid values can be difficult. Manual corrections are often prohibitively time-consuming. SAS® offers useful and efficient approaches: a combined DATA step with SQL or SORT procedures. This paper provides a step-by-step tutorial to accomplish this task.

INTRODUCTION

There are times when we need to delete records based on missing values in a dataset. For example, a top-level “Yes/No” question may be followed by a group of check boxes, to which a respondent may indicate multiple answers. When creating the database, programmers may choose to set a default value of “0” for each of the check boxes under a top level question. One should interpret these default values with caution.

Depending on whether the top-level question is answered “Yes” or “No”, or is missing, the values for the sub-group questions are treated as missing when the top-level question is missing, even though sub-group questions have values of “0” from the default setting. A similar scenario occurs with a “Yes/No” or “Checked/Not Checked” question where “No” or “Not Checked” is the default value if the question is not answered (the value actually should be missing).

Programmers should understand the difference between missing values and invalid values (that is, incorrectly set defaults) because it is important to handle these records differently. Clearly records should be omitted when their values are all missing for all the variables, but records may also need to be deleted even if we see values for some of the variables when the values are meaningless.

There are many ways to select and to delete records. It is much easier if the deletion is based on certain variables; for example, if we want to delete only the male with HIV population or delete women age <= 18 records from a demographic dataset, we could use a conditional IF statement:

```
if upcase(GENDER)=‘MALE’ and upcase(HIV)=‘POSITIVE’ then delete ;

if AGE <=18 and upcase(GENDER)=‘FEMALE’ then delete ;
```

In most instances, we use conditional statements based on given conditions to delete un-wanted records.

When we want to use the full set of the variables as the condition of a deletion, programming becomes more challenging, especially when dealing with a large database having hundreds of variables. These situations are hard to detect since not all of the values are missing. Some variables have values that should be treated as missing, since they provide no valid information, and can even dilute or mislead the results. Though they are not missing from what we see in the database, they should be considered as missing since the values were pre-set as default when the database was set up.

Database designers set up defaults to simplify the data collection process. Basically, the data process needs to be short in data entry time and the collected data needs to be as accurate as possible. The hope is that setting up default values will facilitate the data entry process. Another thing that is often done is to carry over the key variable values from screen to screen, or from page to page. For example, subject ID information can be carried over from visit to visit. For a database with multiple records at each of the visits, the visit number and date information can also be carried over.

This approach can be good, because those records would not only be entered faster but also for the sake of consistency and accuracy when moving from one record to the next. On the other hand though, it makes harder to detect erroneous records.

There are times that data entry people accidentally hit the next record button when there is no data to be entered for that record and they forget to delete or remove that data entry page later. These invalid records are kept along with the valid records leaving more work for the programmers to deal with later on. They must know how to sort out invalid records from valid records and how to delete the invalid data. We see more and more database set-ups like this as the developers rely on programmers to omit the records programmatically at the data cleaning and analysis phases.

For illustration purposes, an example partial CRF page is illustrated below (PE – Physical Examination):

Subject Name: ______________________

Visit: ________________

Date: ________________

Age: ______

Gender: □ 1. Male □ 2. Female

Is physical Examination done?: □ 1. Yes □ 0. No

If Yes, any significant abnormality:

Body System	Abnormality	
1. Immunologic/Allergy	□ 1. Yes	□ 0. No
2. HEENT	□ 1. Yes	□ 0. No
3. Respiratory	□ 1. Yes	□ 0. No
4. Cardiovascular	□ 1. Yes	□ 0. No
5. Gastrointestinal	□ 1. Yes	□ 0. No
6. Urogenital	□ 1. Yes	□ 0. No
7. Musculoskeletal	□ 1. Yes	□ 0. No
….	□ 1. Yes	□ 0. No
12. Other	□ 1. Yes	□ 0. No

Based on this CRF, the example dataset used for this paper is as below:

Obs	**SUBJ_ID**	**VISIT**	**AGE**	**GENDER**	**QQ**	**Q1**	**Q2**	**Q3**	**Q4**	**Q5**
1	Chris	1	20	M	1	1	1	1	0	0
2	Elaine	1	36	F	0	0	0	0	0	0
3	John	1	.	.		0	0	0	0	0
4	Mary	1		F	1	1	0	1	0	0
5	Mary	2	.		.	0	0	0	0	0
6	Paul	1	18	M	1	0	1	0	1	0

We have three tasks here:

1. Identify a record that needs to be omitted and populate it for all records.
2. Create a temporary dataset to be used as a comparison dataset.
3. Delete the identified records and create a new dataset with the invalid records omitted.

The following goes through these steps one by one.

```
** Create a PE dummy dataset ;
data PE ;
   infile datalines delimiter=',';
   input subj_id $ visit age gender $ QQ Q1 Q2 Q3 Q4 Q5 ;
   datalines;
      Chris,  1, 20, M,  1,  1,  1,  1,  0,  0
      Elaine, 1, 36, F,  0,  0,  0,  0,  0,  0
      John,   1,   ,  ,   ,  0,  0,  0,  0,  0
      Mary,   1,   , F,  1,  1,  0,  1,  0,  0
      Mary,   2,   ,  ,   ,  0,  0,  0,  0,  0
      Paul,   1, 18, M,  1,  0,  1,  0,  1,  0
run ;
```

1. IDENTIFY A RECORD THAT NEEDS TO BE OMITED AND POPULATE IT FOR ALL RECORDS:

```
** Get template from a blank record ;
data DUMMY ;
    set PE (where=(SUBJ_ID='John')) nobs=LAST_OBS ;
    do I=1 to LAST_OBS ;
          output ;
    end ;
    drop I SUBJ_ID VISIT ;
run ;
```

We are not only picking the record, but will also need to populate it within each of the records in the DUMMY dataset so that we can then compare the records one by one with the PE dataset. The NOBS option from the set statement is used to get the last record number so that it can be used as an end point for the DO loop. There is also a need to delete the unique variable information (John's SUBJ_ID and VISIT in this case) for the record used to populate the rest of the records so that they will be generic records when merging the dataset with the one we want to check, that is, PE. The dataset DUMMY is created after running the data step.

2. CREATE A TEMPORARY DATASET TO BE USED AS A COMPARISON DATASET:

Here we create a temporary dataset (TEMP) by merging the DUMMY dataset with the PE dataset but only bring in the key variables from PE (Subject ID and Visit, in this example, although it is not limited just by the two variables) to make the TEMP dataset a comparable and completed dataset. By saying comparable and completed, it means that It has the same number of subjects, the same number of records and the same number and names of the variables (of course, the key variables are part of the variables in the dataset) as the PE dataset. The only difference is the contents of the two datasets. The PE dataset has the original information (valid and invalid). The TEMP dataset contains all of the records in PE but those records are populated by an invalid record from PE.

```
data TEMP ;
   merge PE (keep=SUBJ_ID VISIT) DUMMY) ;
run ;
```

Merging two datasets with no BY statement is generally not preferable because information has a risk of being combined incorrectly. In this case, as you can see from the previous data steps that the TEMP dataset mimics the number of records and variables from the PE dataset but replaces the values of non-key variables with the ones in DUMMY. So it is not problematic when merging. It performs a One-to-One MERGE by combining the two datasets horizontally. Since there are no common variables in the two datasets, the BY statement does not apply. Observations are combined based upon their relative position in each of the datasets.

```
Proc print data=TEMP;
   title "TEMP dataset" ;
run ;
```

Now we have created a dataset with one missing record for each of the records in the PE dataset. We will use it in the next step when comparing the two datasets.

3. DELETE THE IDENTIFIED RECORDS AND CREATE A NEW DATASET WITH THEINVALID RECORDS OMITTED.

```
/** Approach 1 – Using SQL procedure ;  **/

proc sql ;
   create table PE_FINAL as
   select * from PE
   except
   select *
   from TEMP ;
quit ;

Proc print data=PE_FINAL ;
   title "Approach 1 – Using SQL procedure" ;
run ;
/** end ;  **/
```

The FINAL_PE output:

```
Approach 1 - Using SQL procedure

Obs    SUBJ_ID    VISIT    AGE    GENDER   QQ    Q1    Q2    Q3    Q4    Q5

 1     Chris        1       20      M       1     1     1     1     0     0
 2     Elaine       1       36      F       0     0     0     0     0     0
 3     Mary         1        .      F       1     1     0     1     0     0
 4     Paul         1       18      M       1     0     1     0     1     0
```

Using SQL, the deletion can be easily done. Here, we create a new dataset (FINAL_PE) without the missing/invalid records by comparing the PE dataset with the TEMP dataset. We then delete the records in PE if they exist in the TEMP dataset and keep the rest if they are not in the TEMP dataset. The EXCEPT statement in the SQL procedure is the key to accomplish this task.

```
/** Approach 2 – Using SORT procedure ;  **/
** Before using SORT procedure, combine the two datasets ;

data PE0 ;
   set PE TEMP ;
run ;
```

In order to use the SORT procedure as a way to delete the unwanted records, we need to combine the PE and TEMP datasets vertically. It means that you would see twice as many records in the combined dataset. A data step with a set statement can simply combine the two datasets together. Please note that the order of the two datasets matters. The TEMP dataset which contains the missing records must be placed as the second dataset in the SET statement. You will see why in a later SORT procedure.

After the merge, two SORT procedures are engaged.

1. The first SORT procedure provides the duplicate record dataset (PE1) and the unique record dataset (PE2):

```
proc sort data=PE0 out=PE1 nouniquerec uniqueout=PE2  ;
     by SUBJ_ID VISIT ;
run ;
```

As we can see, the option UNIQUEOUT must be paired with the NOUNIQUEREC option in order to get the unique dataset PE2. It means that both of them need to be selected in the Proc SORT statement. What would happen if the NOUNIQUEREC option is omitted?

A warning will be issued in the log:

```
WARNING: None of the following options have been specified: NODUPKEY,
NODUPREC, NOUNIQUEKEY, or NOUNIQUEREC.
          The DUPOUT= or UNIQUEOUT= data set will contain zero
          observations.
```

The first SORT procedure outputs PE2 with the paired option being selected:

```
Obs    SUBJ_ID    VISIT    AGE    GENDER    QQ     Q1     Q2     Q3     Q4     Q5

 1     Chris        1       20      M        1      1      1      1      0      0
 2     Chris        1        .               .      0      0      0      0      0
 3     Elaine       1       36      F       0      0      0      0      0      0
 4     Elaine       1        .               .      0      0      0      0      0
 5     Mary         1        .      F        1      1      0      1      0      0
 6     Mary         1        .                      0      0      0      0      0
 7     Paul         1       18      M        1      0      1      0      1      0
 8     Paul         1        .               .      0      0      0      0      0
```

By examining the data, we know this is still not what we wanted. We need to use the SORT procedure again in order to delete the invalid records.

2. The second SORT procedure to delete the missing/invalid records in PE2 :

```
proc sort data=PE2 out=PE_FINAL nodupkey ;
      by SUBJ_ID VISIT ;
run ;
/** end  **/ ;
```

Most of us should have used the NODUPKEY option. Using the NODUPKEY option, the first record of each of the specified key variables (SUBJ_ID and VISIT) will be kept if they have more than one SUBJ_ID and VISIT records in the dataset. In this particular case, the missing records will be deleted since they are all the second records for the key variables in the SORT procedure. Remember we put the TEMP dataset as the second dataset when setting with PE dataset before engaging the SORT procedures.

```
** The Final dataset without invalid records ;
```

Approach 2 – Output using the second SORT procedure:

```
Obs    SUBJ_ID    VISIT    AGE    GENDER    QQ    Q1    Q2    Q3    Q4    Q5

 1     Chris        1       22      M        1     1     1     1     0     0
 2     Elaine       1       36      F        0     0     0     0     0     0
 3     Mary         1        .      F        1     1     0     1     0     0
 4     Paul         1       18      M        1     0     1     0     1     0
```

CONCLUSION

As seen in the two approaches illustrated above, the first approach with the SQL procedure is a more favorable approach than the second one since it involves the use of fewer procedures and thus is more efficient. The same result can be achieved with multiple methods. This is the beauty of the many aspects provided by SAS. The example dataset shown above is a relatively small one. Greater efficiency (benefits) will be gained with larger datasets. The program described above has been used successfully for datasets with thousands of records and hundreds of variables.

REFERENCES

SAS Institute Inc. (2010), “SAS Procedures”, SAS Version 9 Online

ACKNOWLEDGMENTS

The author would like to extend her appreciation to Statistics Collaborative, Inc. for supporting her participation in the SAS Global Forum 2016 and to her colleagues for reviewing the paper (Janet Wittes, Janelle Rhorer, Lisa Weissfeld and Elizabeth Toretsky).

CONTACT INFORMATION

Your comments are valuable and encouraged. Please feel free to contact the author about the paper:

Lily Yu
Sr. Statistical Programmer
Statistics Collaborative Inc.
1625 Massachusetts Ave., NW. Suite 600
Washington, DC 20036
202-247-9996
Lily.Yu@statcollab.com
www.statcollab.com