Paper 164-31

Finding a Duplicate in a Haystack

Brett J. Peterson, Medtronic Inc., Minneapolis, MN

ABSTRACT

Duplicate observations are unwanted in many data sets. Nevertheless, they have a way of popping up unexpectedly and interfering with DATA step merges and compromising data integrity. I present multiple methods to find and eliminate erroneous duplicates using SAS®, including a macro. A proactive approach including a weekly production job that alerts clinical study team members of duplicates to be reconciled is also discussed. The examples shown use Base SAS® and the SAS® macro language, work for versions 8 and above, and may work for earlier versions. The programming presented is at a beginner's level except for the macro example.

TERMS

<u>Key variables</u> – A variable or combination of variables whose values are supposed to identify a unique observation in a data set, for example, PatientID, or CenterID and PatientID.
<u>Exact duplicates</u> – Two or more observations in a data set that have all variable values equal.
<u>Non-exact duplicates</u> – Two or more observations in a data set that have all key variable values equal, but have differences in non-key variable values.
<u>Clinical study team</u> – A group of people responsible for conducting a research study. Members specialize in designing and conducting the study, collecting, entering, cleaning, and analyzing data, and writing publications.

INTRODUCTION

Where there is data, there are unwanted duplicates. This is especially true when people are interfacing with data entry screens to manually enter data. My experience comes from large clinical research studies that utilize multiple SAS data sets. Data can be entered from paper forms by data entry personnel, through on-line entry forms, or loaded by other tools such as faxed data collection forms that are ultimately translated into data sets. There is room for duplicates to appear when using any of these data entry methods. In many studies, clinical study team personnel access and modify data sets using entry screens. Many times duplicates are inadvertently created during these situations, even when they accessed the data to fix another problem such as an erroneous date! Frequently duplicates cannot be simply eliminated by a computer program. They require human review instead. For example, a clinical study team member may access a patient's data to enter a date and mistakenly create a duplicate in which the new date is placed. A program blindly eliminating duplicates would not know whether to remove the observation with the date or the observation without the date. In this case someone would have to visually review both observations, make sure one observation has complete and accurate data, and delete the other observation. This paper will present four methods for finding duplicates in SAS data sets using SAS versions 6 and 8. The first three utilize various combinations of the SORT procedure, the FREQ procedure, and the DATA step, while the fourth is a SAS macro that allows greater flexibility for dealing with duplicates. Finally, a proactive approach to handling duplicates in a clinical study setting is presented.

EXAMPLES

Imagine we have been given a request to identify all duplicate observations in the data set FORM1 based on patient ID. Let's explore the different ways we can tackle this assignment.

DATA

Code Block 1 creates sample data sets FORM1 and CENTERS. The data set FORM1 lists patients at a clinic and indicates whether or not they have a cardiac medical device. The following information is also present for patients with a cardiac medical device: the device type, and, if the device is an implantable cardiac defibrillator (ICD), which features it has. There are two duplicates in the data set FORM1, one exact duplicate and one non-exact duplicate. The sample data set CENTERS contains gender data for patients who are uniquely identified by the combined values of the variables CenterID and PatientID. This data set will be used later.

Code Block 1. Sample data sets named FORM1 and CENTERS.

```
data Form1 ;
    input ID Implanted $ DeviceType $ History FluidStatus Shockless Total ;
    datalines ;
    1 Yes ICD   1 1 1 4
    1 Yes ICD   1 1 1 3 /* Non-exact duplicate */
    2 No  .     . . . .
    3 Yes IPG   . . . .
    4 Yes IPG   . . . .
    5 No  .     . . . .
    6 Yes ICD   1 0 0 1
    7 Yes ICD   1 1 1 3
    8 Yes IPG   . . . .
    8 Yes IPG   . . . .  /* Exact duplicate */
    9 No  .     . . . .
   10 Yes ICD   1 1 1 3
    ;
run ;


data Centers ;
    input CenterID PatientID Sex $ ;
    datalines ;
    1 1 M
    1 2 F
    1 3 M
    2 1 F
    2 2 M
    2 3 M
    2 3 F
    3 1 F
    3 2 F
    3 3 M
    ;
run ;
```

METHOD 1 – PROC SORT

The shortest method in terms of lines of code uses PROC SORT. This procedure has options that facilitate removal of duplicate observations. The option NODUP will keep one observation and remove all subsequent duplicates by comparing all variables in the input data set to the prior observation. The option NODUPKEY will keep one observation and remove all subsequent duplicates by comparing only the variables specified in the BY statement. It is important to name an output data set using the OUT= option, otherwise your input data set will be overwritten. Also note that the resulting data set will be sorted according to the BY variables specified. Table 1 shows the effect of the NODUP and NODUPKEY options when Code Block 2 is submitted. Using the NODUP option will result in 11 observations and the NODUPKEY option will result in 10 observations.

Code Block 2. PROC SORT using options NODUP and NODUPKEY.

PROC SORT with NODUP

```
proc sort data = Form1
    out = Patients_NODUPS
    nodup ;
    by ID ;
run ;
```

PROC SORT with NODUPKEY

```
proc sort data = Form1
    out = Patients_NODUPS
    nodupkey ;
    by ID ;
run ;
```

Table 1. The effect of submitting the PROC SORT code in Code Block 2 with options NODUP and NODUPKEY on the observations kept and removed in the output data set. Only input observations with duplicates are shown.

ID	Implanted	Device Type	History	Fluid Status	Shock Less	Total	Effect of NODUP	Effect of NODUPKEY
1	Yes	ICD	1	1	1	4	Keep	Keep
1	Yes	ICD	1	1	1	3	Keep	Remove
8	Yes	IPG	.	.	.	.	Keep	Keep
8	Yes	IPG	.	.	.	.	Remove	Remove

The method using PROC SORT with the NODUP option works very well when you are deleting exact duplicates in your data, such as ID 8 in Table 1, because it finds and eliminates them in one simple SAS procedure. When you have duplicates with conflicting information, such as ID 1 in Table 1, other methods are preferable since you will likely need to review the observations in question before removing the duplicates. SAS® version 9 contains a new SORT procedure option named DUPOUT= that puts all deleted duplicate observations into a data set. Merge this data set with FORM1 and print out all of the duplicate observations for review (Appendix A). Sometimes multiple variables make an observation unique. For example, a patient ID may be repeated between study centers, but within a study center all patient ID's should be unique. In this case the combination of patient ID and center ID make a unique observation. To handle these situations list all the key variables that define a unique observation in the BY statement and use the NODUPKEY option. The NODUP option will behave the same under these circumstances since it compares all variables in the data set and uses the BY variables only for sorting the observations. Be aware that the options NODUP and NODUPKEY can give unexpected results. See SUGI 30 paper 037-30 for a detailed description on the use of NODUP and NODUPKEY (Kelsey, 2005).

METHOD 2 – PROC FREQ THEN DATA STEP MERGE

This approach utilizes PROC FREQ to identify duplicate observations. Once the duplicates are found based on the key variables, you can merge back to the original data set to obtain other variables that differentiate the duplicates, and then produce a listing of the duplicates for review. This listing will allow you to determine which observation to keep and which observations to eliminate. Since this process requires manual review, I typically ask the people reviewing the duplicates to perform the deletions in the database. The first programming step is to run PROC FREQ on the data set FORM1 listing your key variable in the TABLE statement (Code Block 3). In this case, the key variable is ID. I use the option NOPRINT to suppress writing to the listing window and use OUT= to route the output to a new data set named FORM1_DUPIDS (Table 2). Without the qualifier "where = (Count >1)", the resulting data set FORM1_DUPIDS would contain all observations (Table 3). The duplicate observations belong to ID's where the variable COUNT is greater than 1. Using the WHERE= data step option allows you to obtain the duplicates directly in one step.

Code Block 3. Using PROC FREQ to find duplicate observations and route them into an output data set.

```
proc freq data = Form1 noprint ;
    table ID / out = Form1_DUPIDS (keep = ID Count where = (Count > 1)) ;
run ;
```

When you have multiple key variables, you can use the following syntax, but Code Block 4 is more efficient:

```
table var1 * … * varN / out = Form1_DUPIDS (keep = var1 … varN Count where = (Count > 1));
```

Table 2. Options used with PROC FREQ to create an output data set containing duplicate observations in the data set FORM1.

Syntax	Option type	Function
NOPRINT	PROC FREQ statement option	Suppresses printing PROC FREQ output to the log window.
OUT=	TABLE statement option	Routes PROC FREQ output to a named data set.
KEEP=	DATA SET option	Keeps named variables for further processing in the output data set specified by OUT=.
WHERE=	DATA SET option	Controls which observations are output to the OUT= data set. This identifies the duplicates.

Table 3. How the data set option WHERE= in Code Block 3 affects observations written to data set FORM1_DUPIDS.

Without "where = (Count >1)"	With "where = (Count >1)"
#	#

```
Obs    ID    COUNT

  1     1      2
  2     2      1
  3     3      1
  4     4      1
  5     5      1
  6     6      1
  7     7      1
  8     8      2
  9     9      1
 10    10      1
```

```
Obs    ID    COUNT

 1      1      2
 2      8      2#
```

When a data set contains multiple key variables use the coding method shown in Code Block 4 instead of what is shown in the footnote of Code Block 3 because the processing time is shorter, especially when using large data sets and many BY variables. The general form is to sort by all key variables, then run PROC FREQ including all but the last sorted key variable on the BY statement and the last sorted key variable in the TABLE statement.

Code Block 4. Using PROC FREQ to find duplicate observations and route them to an output data set with multiple key variables.

```
proc sort data = Centers ;
    by CenterID PatientID ;
run ;

proc freq data = Centers noprint ;
    by CenterID ;
    table PatientID / out = Centers_DUPS (keep = CenterID PatientID Count
                                                     where = (Count > 1)) ;
run ;
```

We now have the patients with duplicate observations contained in a data set named FORM1_DUPIDS, but we only know the identifiers corresponding to them. It is usually helpful to print additional variables to help identify any differences between the duplicate observations. Simply merge data sets FORM1 and FORM1_DUPIDS to obtain the variables of interest then print the result (Code Block 5). I like the ID statement because it gives organized and visually pleasing output.

Code Block 5. Code to merge the data set containing duplicates with the original data set and then print a listing of duplicate observations.

```
/* Merge with original data. */
proc sort data = Form1_DUPIDS ;
    by ID ;
run ;[1]

proc sort data = Form1 ;
    by ID ;
run ;[2]

data Form1_DUPS ;
    merge Form1 (in = a[3]) Form1_DUPIDS (in = b[3] drop = Count[4]) ;
    by ID ;
    if a & b then output ;
run ;

/* Print the duplicates. */
title3 'Duplicate observations in data set Form1' ;
proc print data = Form1_DUPS ;
    by ID ;
    id ID ;
run ;
```

```
title3 ;
```

[1] This PROC SORT is not necessary because PROC FREQ in Code Block 3 sorted the data.
[2] This PROC SORT could occur anywhere in the code prior to being used in a merge.
[3] The variable names using the DATA step option IN= can be any valid SAS variable name.
[4] You can elect to keep the variable COUNT to report the number of duplicates for given key variables.

The results from the PROC PRINT clearly show ID 1 has an erroneous duplicate where Total = 4 and that ID 8 has an exact duplicate (Output 1). This is an example of a report you would give to a member of the clinical study team who is responsible for reviewing and correcting the data. That person will use the report and eliminate the inappropriate duplicates from the database.

Output 1. Duplicate report listing generated by Code Blocks 3 and 5. The clinical study team uses this output to reconcile duplicates.

```
              Duplicate observations in data set Form1

                         Device                Fluid
ID     Implanted          Type      History    Status    Shockless    Total

 1        Yes             ICD          1          1          1          4
          Yes             ICD          1          1          1          3

 8        Yes             IPG          .          .          .          .
          Yes             IPG          .          .          .          .
```

METHOD 3 – PROC SORT THEN DATA STEP

This method uses FIRST.variable and LAST.variable processing in the DATA step to achieve the same results as method 2. First sort the data set FORM1 by the key variable or variables (Code Block 6). The key variables define unique observations.

Code Block 6. Sort the data set FORM1 by key variables.

```
proc sort data = Form1 ;
    by ID ;
run ;
```

For multiple key variables use: by `VAR1 … VARN ;`

Next, use the DATA step to find the duplicate observations (Code Block 7). This method of finding duplicates is driven by the temporary variables FIRST.ID and LAST.ID. A FIRST.variable and LAST.variable are created for each variable listed in the BY statement of a DATA step and take on the values 1 or 0. FIRST.ID is assigned the value of 1 for the first observation in a BY group and the value of 0 for all other observations in the BY group. LAST.ID is assigned the value of 1 for the last observation in a BY group and the value of 0 for all other observations in the BY group (SAS Institute Inc., 2004). Observations that are not duplicates will have FIRST.ID = 1 and LAST.ID = 1, so we must select observations where this is not the case. Table 4 shows what values FIRST.ID and LAST.ID have in the sample data set FORM1 when executing Code Block 7 and which observations are written to the output data set FORM1_DUPS.

Code Block 7. DATA step that outputs duplicate observations based on the BY variables.

```
data Form1_DUPS ;
    set Form1 ;
    by ID ;
    if (first.ID ne 1 or last.ID ne 1) then output ;
run ;
```

For multiple key variables, use: `by VAR1 … VARN ; if (first.VARN ne 1 or last.VARN ne 1) then output ;`

Table 4. Processing of Code Block 7 using the temporary variables FIRST.ID and LAST.ID. The observations output and not output are shown.

OBS	ID	Device Implanted	FIRST.ID	LAST.ID	Output to FORM1_DUPS
1	1	Yes	1	0	OUTPUT
2	1	Yes	0	1	OUTPUT
3	2	No	1	1	NOT
4	3	Yes	1	1	NOT
5	4	Yes	1	1	NOT
6	5	No	1	1	NOT
7	6	Yes	1	1	NOT
8	7	Yes	1	1	NOT
9	8	Yes	1	0	OUTPUT
10	8	Yes	0	1	OUTPUT
11	9	No	1	1	NOT
12	10	Yes	1	1	NOT

Finally we print the four observations in data set FORM1_DUPS. The output will be the same as Output 1. This method requires fewer lines of code than method 2 and does not create the extra work data set from PROC FREQ, but it does not compute duplicated observation counts.

METHOD 4 – SAS MACRO

The last method takes the core concepts of finding duplicates and puts them into a flexible SAS macro that can be called whenever needed. I used method 3 as the basis for a macro named %DUPCHECK (Appendix B). I created this macro because I needed to address duplicates in data sets numerous times over different programs. The macro call is much shorter than the code used in methods 2 or 3. Having a macro allows me to incorporate useful features such as controlling printed variables and creating a WORK data set with duplicates removed. Other features could be added such as putting the duplicates into a data set and further control on printing variables. Code Block 8 presents an example call of %DUPCHECK. I simply provide the data set name to check for duplicates, the variable indicating unique observations, and the variables to be printed in addition to those listed in the BYVAR parameter. The macro parameters are described in Table 5.

Code Block 8. Sample macro call for %DupCheck using the sample data set FORM1.

```
%DupCheck(Dataset = Form1,
          ByVar = ID,
          PrintVar = Implanted DeviceType History FluidStatus Shockless Total,
          Remove =  )
```

Table 5. Calling parameters for macro %DupCheck.

Parameter Name	Description
Dataset	Name of the input dataset.
ByVar	List of by variables that define unique observations. If an observation is not unique, it is considered a duplicate.
PrintVar	Variables to print on the output report in addition to those listed in the ByVar parameter.
Remove	Create an output data set with duplicates removed. Default is to not create the data set. Set to Y to create a data set named &DATASET._NODUPS.

The results of the macro are the same as methods 2 and 3, except for the customized title that is created automatically using the parameters DATASET and BYVAR (Output 2).

Output 2. Duplicate report listing generated by Code Block 8.

```
#####
      Duplicate observations in dataset FORM1 by variable(s) ID
```

```
                        Device                     Fluid
ID     Implanted         Type      History        Status      Shockless      Total

 1        Yes            ICD          1              1            1            4
          Yes            ICD          1              1            1            3

 8        Yes            IPG          .              .            .            .
          Yes            IPG          .              .            .            .
```

In this example I will use the sample data set CENTERS (Code Block 1). Unique observations are defined by a combination of variables CenterID and PatientID. These key BY variables are simply entered into the macro parameter BYVAR separated by a space. The BYVAR parameter can handle as many BY variables as the SORT procedure. Only variables CenterID, PatientID, and Sex will appear in the output as controlled by parameters BYVAR and PRINTVAR. For this example, assume %DUPCHECK is called from another program that creates a monthly summary report. I want my monthly summary program to continue processing using the data set CENTERS with duplicates removed. This is accomplished by setting the parameter REMOVE equal to Y (Code Block 9). During macro execution the data set CENTERS_NODUPS is created and named by a convention built into the macro with observation 7 removed (Table 6). I use the REMOVE option in programs that expect to merge data sets without duplicates. Now I can run the monthly summary program in its entirety, generate all output, and obtain a listing of duplicates to investigate (Output 3).

Code Block 9. Simple macro call for %DupCheck using the sample data set CENTERS.

```
%DupCheck(Dataset = Centers,
          ByVar = CenterID PatientID,
          PrintVar = Sex,
          Remove = Y)
```

Table 6. Data set CENTERS. Observation 7 will be deleted from the data set CENTERS_NODUPS because it has the same values for CenterID and PatientID as observation 6.

OBS	CenterID	PatientID	Sex
1	1	1	M
2	1	2	F
3	1	3	M
4	2	1	F
5	2	2	M
6	2	3	M
7	2	3	F
8	3	1	F
9	3	2	F
10	3	3	M

Output 3. Duplicate report listing generated by Code Block 9.

```
Duplicate observations in dataset CENTERS by variable(s) CENTERID PATIENTID

                       Center     Patient
                         ID          ID       Sex

                          2          3         M
                                               F
```

BE PROACTIVE

The methods presented for identifying and removing duplicates are reactive when used while writing programs where you expect no duplicates. I implemented a proactive approach to eliminating duplicates for a large clinical research study. I wrote a SAS program that solely identified all duplicates in our entire database using PROC SORT and the DATA step (Method 3). This program was automated to run every Monday morning using the UNIX 'cron' command and the output listing the duplicates went directly to the clinical study team's printer. They would spend time every week deleting the duplicates from the data sets, hopefully before I even knew they existed at all! The duplicate report is most effective when combined with other data check programs you have in place (Peterson, 2006). This process kept duplicates at a minimum or, at times, eliminated them altogether.

CONCLUSION

Simple SAS programs using various combinations of PROC SORT, PROC FREQ, and the DATA step can be easily implemented to aid in duplicate identification and elimination. A SAS macro simplifies the process when you find yourself repeatedly looking for duplicates in data sets while programming. It is important to remember that not all duplicates can be directly deleted, specifically when differing data is present between the observations. Often further review by clinical study team members is warranted. I advocate a proactive approach where the team routinely uses a SAS report to delete duplicates. With duplicates out of the way, you can merge with confidence and rest easy knowing your data sets are more accurate.

REFERENCES

Kelsey, Britta. "The Mystery of the PROC SORT Options NODUPRECS and NODUPKEY Revealed." *Proceedings of the Thirtieth Annual SAS® Users Group International Conference*. April 2005. Paper 037-30.

Peterson, Brett J. "A Strategy for Managing Data Integrity Using SAS®." *Proceedings of the Thirty-first Annual SAS® Users Group International Conference*. March 2006. Paper 103-31.

SAS Institute Inc. 2004, *SAS® Certification Prep Guide: Base Programming*. Cary, NC: SAS Institute Inc.

ACKNOWLEDGMENTS

Thank you to Paul Franson, Kevin Sanft, and Siddharth Jain for reviewing this work and providing constructive feedback.

CONTACT INFORMATION

Brett J. Peterson
Principal IT Developer
Statistical Programming
Cardiac Rhythm Management Clinical Research
Medtronic, Inc.
1015 Gramsie Road
Mail Stop Z240
Shoreview, MN 55126-3082
Tel: (763) 505-7391
Fax: (763) 505-7543
brett.j.peterson@medtronic.com

APPENDIX A

Produce a duplicate report using the new SAS version 9 PROC SORT option named DUPOUT=.

SAS code.

```
proc sort data = Form1
    out = Patients_NODUPS
    dupout = Patients_DUPIDS (keep = ID)
    nodupkey ;
    by ID ;
run ;

data Patients_DUPS ;
    merge Form1 (in=a) Patients_DUPIDS (in=b) ;
    by ID ;
    if a & b ;
run ;

title3 'Duplicate observations in data set Form1' ;
proc print data = Patients_DUPS ;
    by ID ;
    id ID ;
run ;
title3 ;
```

SAS Output. Duplicate report listing generated by SAS code above. The clinical study team uses this output to reconcile duplicates. This output is the same as the output generated by methods developed using SAS versions 6 and 8 as discussed in the paper.

```
              Duplicate observations in data set Form1

                    Device                Fluid
ID     Implanted     Type      History    Status     Shockless     Total

 1        Yes        ICD          1          1           1           4
          Yes        ICD          1          1           1           3

 8        Yes        IPG          .          .           .           .
          Yes        IPG          .          .           .           .
```

APPENDIX B

SAS code for macro %DupCheck.

```
%macro DupCheck(Dataset = , ByVar = , PrintVar = , Remove =  ) ;

/* Pull off the last byvar in the list.  This variable is used with first-dot
   last-dot processing to find the duplicates within a datastep. */
%let LastByVar = %upcase(%scan(&ByVar, -1, %str( ))) ;

proc sort data = &dataset out = _Sorted ; by &byvar ;
run ;

data _DupCheck1 ;
    set _Sorted ;
    by &ByVar ;
    if ^(first.&LastByVar = last.&LastByVar = 1) then output ;
run ;

title3 "Duplicate observations in dataset %sysfunc(upcase(&dataset)) by variable(s)
%sysfunc(upcase(&byvar))" ;
proc print data = _DupCheck1 noobs ;
    by &ByVar ;
    id &ByVar ;
    var &PrintVar ;
run ;
title3 ;

/* Remove duplicates based on the by variables. */
%if &Remove = Y %then %do ;

data &dataset._NoDups ;
    set _Sorted ;
    by &ByVar ;
    if first.&LastByVar then output ;
run ;

%end ;

proc datasets library = work ;
    delete _Sorted _DupCheck1 ;
quit ;

%mend DupCheck ;
```