The Essentials of Finding the Distinct, Unique, and Duplicate Values in Your Data

Carter Sevick MS, DoD Center for Deployment Health Research, San Diego, CA

ABSTRACT
Whether by design or by error there are data sets out there that contain variables, combinations of variables, or whole observations that are not unique. For example, duplications of a patient identifier should be expected in a medical records file, but perhaps not in a master list of patients. Unanticipated duplicates in your data can be the source of hours of headaches, which may be surprisingly easy to resolve once you apply a few well chosen techniques.

Basic knowledge of PROC SORT, and DATA step processing (including SET and MERGE) will be assumed.

INTRODUCTION
Every data set should have a primary key and every row should have a unique instance of a primary key. So goes the theory, but in real life there are imperfect data sets. Sometimes whole rows are loaded into a data set more than once. A person may be entered into a system twice, perhaps with conflicting information, such as different birth dates. If data sets have rows duplicated on their primary keys then you cannot count how many distinct individuals are represented. SAS® performs table merges brilliantly when the relationship between two tables is one-to-one, or one-to-many, but if two tables are merged with an unobserved many-to-many relationship then you may get unexpected results.

If you have duplicated data in a large data set it’s not possible to just print out the data set to find and deal with the offending rows. In this paper we will explore how to deal with simple row duplication, how to isolate rows that are unique, and isolate rows that are duplicated. Resolving conflicting data, such as the birthday example above, may not be possible given your level of access to data and the offending rows may need to be collected and sent to someone else.

THE WHOLE ROW IS DUPLICATED, GETTING THE DISTINCT LIST
You have been given the task of collecting demographic information on a list of subjects to complement a study that you are currently working on. Each subject has a four digit subject ID number named sub_id. The data set is as follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>sub_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1204</td>
</tr>
<tr>
<td>2</td>
<td>5023</td>
</tr>
<tr>
<td>3</td>
<td>3029</td>
</tr>
<tr>
<td>4</td>
<td>1106</td>
</tr>
<tr>
<td>5</td>
<td>2034</td>
</tr>
<tr>
<td>6</td>
<td>5023</td>
</tr>
</tbody>
</table>

You can see that one subject is represented twice, but remember with real life data, you will not be able to see the problem. Consequently, whenever you are confronted with a new data set the first action you should take is to check for duplicates. Fortunately, the SORT procedure has a simple option for this situation: NODUPKEY. Including this option on the procedure statement line causes the procedure to compare the BY values of the current row with those of the previous row. The current row is deleted if an exact match is found. When using PROC SORT it is always a good idea to use the OUT = option to avoid overwriting the original data set (unless you really mean to). When using NODUPKEY it is even more important to use OUT = since deleted rows are gone for good, or at least until you can get your DBA to reload your data set. To eliminate the duplicates in sub_list we can use the following code:
PROC SORT DATA = sub_list OUT = sub_list_tmp NODUPKEY;
   BY sub_id;
RUN;

NOTE: There were 6 observations read from the data set WORK.SUB_LIST.
NOTE: 1 observations with duplicate key values were deleted.
NOTE: The data set WORK.SUB_LIST_TMP has 5 observations and 1 variables.
NOTE: PROCEDURE SORT used (Total process time):
   real time         0.12 seconds
   cpu time          0.01 seconds

The log file reveals a very handy feature. Notice that on the second note SAS tells us how many rows were deleted. If you just want to check for duplicates, but do not want to create another data set, you can set OUT = _NULL_ and you will still get this information in the log.

PROC SORT DATA = sub_list OUT = _NULL_ NODUPKEY;
   BY sub_id;
RUN;

The demographics file is said to have one row per individual and contains the following fields:

<table>
<thead>
<tr>
<th>Variables in Creation Order</th>
</tr>
</thead>
<tbody>
<tr>
<td>#</td>
</tr>
<tr>
<td>-----</td>
</tr>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
<tr>
<td>3</td>
</tr>
<tr>
<td>4</td>
</tr>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

In order to access the needed information we submit the following code:

*** Merge to demographic set ***;
PROC SORT DATA = sub_list OUT = sub_list_tmp NODUPKEY;
   BY sub_id;
RUN;
PROC SORT DATA = study_demographics OUT = study_demo_tmp;
   BY sub_id;
RUN;
DATA demo_subset;
   MERGE
      sub_list_tmp (IN = a)
      study_demo_tmp;
   BY sub_id;
   IF a;
RUN;

This program is simple and straight forward, so no problems should be expected. Or should there? You must always remember the most important rule of SAS programming: "Check your log file".
DATA demo_subset;
MERGE sub_list_tmp (IN = a)
study_demo_tmp;
BY sub_id;
IF a;
RUN;

NOTE: There were 5 observations read from the data set WORK.SUB_LIST_TMP.
NOTE: There were 11 observations read from the data set WORK.STUDY_DEMO_TMP.
NOTE: DATA statement used (Total process time):
real time 0.01 seconds
cpu time 0.00 seconds

There are no error notes or warnings but if you look closely you will notice that WORK.SUB_LIST_TMP has 5 observations and the output data set has 8 observations. If each data set was truly structured as one row per subject then the output data set should also have 5 observations. Clearly, the demographics file has some duplication by subject ID. Since the output table is small, we can print it in its entirety and inspect it, though this will almost never be possible in real life. Later in this paper we will cover what to do if the table is too large to be printed.

<table>
<thead>
<tr>
<th>Obs</th>
<th>sub_id</th>
<th>gender</th>
<th>dob</th>
<th>race_ethnic</th>
<th>education</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1106</td>
<td>F</td>
<td>05SEP1973</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>1204</td>
<td>M</td>
<td>22JAN1972</td>
<td>1</td>
<td>H</td>
</tr>
<tr>
<td>3</td>
<td>1204</td>
<td>M</td>
<td>23JAN1972</td>
<td>1</td>
<td>H</td>
</tr>
<tr>
<td>4</td>
<td>1204</td>
<td>M</td>
<td>22JAN1972</td>
<td>1</td>
<td>H</td>
</tr>
<tr>
<td>5</td>
<td>2034</td>
<td>M</td>
<td>31JAN1962</td>
<td>3</td>
<td>H</td>
</tr>
<tr>
<td>6</td>
<td>3029</td>
<td>F</td>
<td>20AUG1970</td>
<td>1</td>
<td>G</td>
</tr>
<tr>
<td>7</td>
<td>5023</td>
<td>M</td>
<td>13APR1969</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>8</td>
<td>5023</td>
<td>M</td>
<td>13APR1969</td>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>

As you can see, subject 5023 has two identical rows and subject 1024 has three rows (two that are identical, and one with a different birthday). This situation is different than with the data set sub_list, then we only had a single field to worry about but here we have five. Consider the following:

PROC SORT DATA = demo_subset OUT = demo_subset_tmp NODUPKEY;
BY sub_id;
RUN;

<table>
<thead>
<tr>
<th>Obs</th>
<th>sub_id</th>
<th>gender</th>
<th>dob</th>
<th>race_ethnic</th>
<th>education</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1106</td>
<td>F</td>
<td>05SEP1973</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>1204</td>
<td>M</td>
<td>22JAN1972</td>
<td>1</td>
<td>H</td>
</tr>
<tr>
<td>3</td>
<td>2034</td>
<td>M</td>
<td>31JAN1962</td>
<td>3</td>
<td>H</td>
</tr>
<tr>
<td>4</td>
<td>3029</td>
<td>F</td>
<td>20AUG1970</td>
<td>1</td>
<td>G</td>
</tr>
<tr>
<td>5</td>
<td>5023</td>
<td>M</td>
<td>13APR1969</td>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>

If this code is run then we will get a data set (above) that has rows unique by sub_id, and subject 5023 will have all of his information preserved intact, but not 1204. Two of the rows will be deleted, leaving you with a birthday that may not be correct. If SAS is using its default sort engine then the row that is read by the SORT procedure first will be preserved and the second deleted. This can change under certain circumstances and if you need that default behavior to be insured then you should use the EQUALS option on the PROC SORT statement. Either way, the outcome is undesirable. To make sure that only whole row duplicates are deleted you must sort on all variables in the set. If you have a data set with a large number of variables then you can use the name list _ALL_. This will signal PROC SORT to include all variables in the BY statement.
PROC SORT DATA = demo_subset OUT = demo_subset_tmp NODUPKEY;
   BY _ALL_;
RUN;

Unfortunately, SAS does not let you choose the order of the BY variables, the order is determined by the order in which they appear in the data set. PROC CONTENTS displays this information in the column with the "#" header. So, the above code is equivalent to:

PROC SORT DATA = demo_subset OUT = demo_subset_tmp NODUPKEY;
   BY sub_id gender dob race_ethnic education;
RUN;

Fortunately for us, SAS is pretty smart and there is a recourse if you want the result set sorted first by dob, or one of the other variables:

PROC SORT DATA = demo_subset OUT = demo_subset_tmp NODUPKEY;
   BY dob _ALL_;
RUN;

NOTE: There were 8 observations read from the data set WORK.DEMO_SUBSET.
NOTE: Duplicate BY variable(s) specified. Duplicates will be ignored.
NOTE: 2 observations with duplicate key values were deleted.
NOTE: The data set WORK.DEMO_SUBSET_TMP has 6 observations and 5 variables.
NOTE: PROCEDURE SORT used (Total process time):
   real time           0.06 seconds
   cpu time            0.00 seconds

In the above submission, SAS actually sees the following:

PROC SORT DATA = demo_subset OUT = demo_subset_tmp NODUPKEY;
   BY dob sub_id gender dob race_ethnic education;
RUN;

But, the data set will be sorted as if the second instance of dob wasn’t there.

For the purpose of this project we will use the following code:

PROC SORT DATA = demo_subset OUT = demo_subset_tmp NODUPKEY;
   BY sub_id gender dob race_ethnic education;
RUN;

The result follows:

<table>
<thead>
<tr>
<th>Obs</th>
<th>sub_id</th>
<th>gender</th>
<th>dob</th>
<th>race_ethnic</th>
<th>education</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1106</td>
<td>F</td>
<td>05SEP1973</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>1204</td>
<td>M</td>
<td>22JAN1972</td>
<td>1</td>
<td>H</td>
</tr>
<tr>
<td>3</td>
<td>1204</td>
<td>M</td>
<td>23JAN1972</td>
<td>1</td>
<td>H</td>
</tr>
<tr>
<td>4</td>
<td>2034</td>
<td>M</td>
<td>31JAN1962</td>
<td>3</td>
<td>H</td>
</tr>
<tr>
<td>5</td>
<td>3029</td>
<td>F</td>
<td>20AUG1970</td>
<td>1</td>
<td>G</td>
</tr>
<tr>
<td>6</td>
<td>5023</td>
<td>M</td>
<td>13APR1969</td>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>

So, all of our whole row duplicates have been taken care of but we still have to deal with the fact that subject 1204 has two rows with two different birthdays. In the next section we will investigate BY group processing as a way to sort out these data.
ROWS WITH DUPLICATE PRIMARY KEYS BUT CONFLICTING DATA VALUES

BY-GROUP PROCESSING
Like many SAS procedures, you can use a BY statement in the DATA step. Just like the SAS procedures, the input data set must be presorted on the same variables that appear in the BY statement. When you include a BY statement in the DATA step, two temporary variables are created for each variable included in the BY statement. These temporary variables have the names FIRST:variable-name and LAST:variable-name and each takes values of either 1 or 0. Consider the following program and output:

```sas
data test;
  input col @@;
datalines;
  2 2 1 1 3 3 3 3;
proc sort data = test;
  by col;
data ByProcess;
  set test;
  by col;
  first = first.col;
  last = last.col;
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>col</th>
<th>first</th>
<th>last</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Remember that FIRST.col and LAST.col are temporary variables and are not visible in a PROC PRINT output. To be able to see the values that they took, they were assigned to the variables first and last, respectively. Whenever the DATA step processes the first row in a BY-group FIRST:variable-name will be given the value of 1 and 0 otherwise. Likewise, LAST:variable-name is assigned the value 1 when the DATA step is processing the last row in a BY-group, 0 otherwise. BY-group processing has a multitude of uses beyond the topic of this paper. It can be used to de-duplicate a data set, subset out duplicate rows, evaluate data across multiple rows, and many others.

USING BY PROCESSING
In order to isolate rows that are unique or duplicated on a variable or list of variables we can use BY processing. Since each row should be unique by sub_id we will use it as our BY variable. The data set demo_subset_tmp is already sorted by sub_id and so is ready for BY processing without further sorting.
DATA demo_by_sub_id;
  SET demo_subset_tmp;
  BY sub_id;

  first = FIRST.sub_id;
  last = LAST.sub_id;
RUN;

<table>
<thead>
<tr>
<th>Obs</th>
<th>sub_id</th>
<th>gender</th>
<th>dob</th>
<th>race_ethnic</th>
<th>education</th>
<th>first</th>
<th>last</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1106</td>
<td>F</td>
<td>05SEP1973</td>
<td>2</td>
<td>C</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>1204</td>
<td>M</td>
<td>22JAN1972</td>
<td>1</td>
<td>H</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>1204</td>
<td>M</td>
<td>23JAN1972</td>
<td>1</td>
<td>H</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4</td>
<td>2034</td>
<td>M</td>
<td>31JAN1962</td>
<td>3</td>
<td>H</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>3029</td>
<td>F</td>
<td>20AUG1970</td>
<td>1</td>
<td>G</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>5023</td>
<td>M</td>
<td>13APR1969</td>
<td>2</td>
<td>C</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

The above table reveals a recurring pattern: if a row is unique by the BY variables then FIRST.var and LAST.var will both be equal to one, otherwise there are at least 2 rows with the same BY values. The following two DATA steps isolate the unique and duplicated rows.

DATA unique_rows;
  SET demo_subset_tmp;
  BY sub_id;

  IF FIRST.sub_id = 1 AND LAST.sub_id = 1;
RUN;

<table>
<thead>
<tr>
<th>Obs</th>
<th>sub_id</th>
<th>gender</th>
<th>dob</th>
<th>race_ethnic</th>
<th>education</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1106</td>
<td>F</td>
<td>05SEP1973</td>
<td>2</td>
<td>C</td>
</tr>
<tr>
<td>2</td>
<td>2034</td>
<td>M</td>
<td>31JAN1962</td>
<td>3</td>
<td>H</td>
</tr>
<tr>
<td>3</td>
<td>3029</td>
<td>F</td>
<td>20AUG1970</td>
<td>1</td>
<td>G</td>
</tr>
<tr>
<td>4</td>
<td>5023</td>
<td>M</td>
<td>13APR1969</td>
<td>2</td>
<td>C</td>
</tr>
</tbody>
</table>

DATA dup_rows;
  SET demo_subset_tmp;
  BY sub_id;

  IF ~(FIRST.sub_id = 1 AND LAST.sub_id = 1);
RUN;

<table>
<thead>
<tr>
<th>Obs</th>
<th>sub_id</th>
<th>gender</th>
<th>dob</th>
<th>race_ethnic</th>
<th>education</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1204</td>
<td>M</td>
<td>22JAN1972</td>
<td>1</td>
<td>H</td>
</tr>
<tr>
<td>2</td>
<td>1204</td>
<td>M</td>
<td>23JAN1972</td>
<td>1</td>
<td>H</td>
</tr>
</tbody>
</table>

A WORD ON EFFICIENCY
If you are working with large data sets even a simple DATA step (like those above) may take a significant amount of time to run. In such circumstances anything we can do to limit the number of reads can become important. Multiple output data sets can be specified on the DATA statement and rows can be conditionally output to each set using the OUTPUT statement. The syntax for this statement is:

OUTPUT data_set_name;
It is optional to specify a data set name, if you do not then the current row will be output to all data sets names in the DATA statement. Optionally, if a data set name is specified then the current row will be output to that data set only. The following DATA step will do the job of both of the previous and demo_subset_tmp is read only once.

```sas
DATA unique_rows dup_rows;
   SET demo_subset_tmp;
   BY sub_id;

   IF FIRST.sub_id = 1 AND LAST.sub_id = 1 THEN OUTPUT unique_rows;
   ELSE OUTPUT dup_rows;
RUN;
```

CONCLUSION
At first, duplicated data seems to present us with significant challenges to our day to day data management operations, but we have seen that SAS offers several useful tools to help resolve the issues. In addition to what has been shown, PROC SORT offers several other options to help the programmer. PROC SQL is another tool that offers many routes to the same end. I would encourage everyone to investigate the SAS documentation for these PROCS.

ACKNOWLEDGMENTS
The author would like to thank his wife Myrna for her support and patience throughout this process. Also, Tyler Smith PhD., and the members of the Data Team at the DoD Center for Deployment Health Research who have been an invaluable source of inspiration, support, and assistance. The members include Besa Smith, Isabel Jacobson, Cynthia Leard, Charlene Wong, Anna Bukowinski, Molly Kelton, LtCol Christopher Phillips and Martin White. Thank you to the leadership of WUSS 2008 for making this venue possible.

RECOMMENDED READING
SAS Institute Inc. 2006
http://support.sas.com/onlinedoc/913/docMainpage.jsp

SAS Institute Inc. 2006
SAS® OnlineDoc® 9.1.3 “SAS Language Reference: Dictionary”
http://support.sas.com/onlinedoc/913/docMainpage.jsp

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Contact the author at:
Carter Sevick
Naval Health Research Center
DoD Center for Deployment Health Research
140 Sylvester Rd.
San Diego CA 92106-3521
Work Phone: (619) 767-4762
Fax: (619) 553-7601
E-mail: carter.sevick@med.navy.mil

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.
APPENDIX:
DATASETS:

DATA sub_list;
  INPUT sub_id : $4.;
  DATALINES;
1204
5023
3029
1106
2034
5023
;

DATA study_demographics;
  INPUT
    sub_id      : $4.
    gender      : $1.
    dob         : DATE9.
    race_ethnic : $1.
    education   : $1.;
  FORMAT dob DATE9.;
  LABEL
    sub_id      = "Unique identifier of a subject"
    gender      = "Gender of subject"
    dob         = "Subject's date of birth"
    race_ethnic = "Race/ethnicity of subject"
    education   = "Education level of subject";
  DATALINES;
2039 F 20NOV1971 1 G
1204 M 22JAN1972 1 H
5023 M 13APR1969 2 C
3025 M 10APR1968 2 G
1204 M 23JAN1972 1 H
3029 F 20AUG1970 1 G
1106 F 05SEP1973 2 C
2034 M 31JAN1962 3 H
5023 M 13APR1969 2 C
1204 M 22JAN1972 1 H
6101 F 08AUG1970 2 C
;