

Identifying and Removing Subjects with Duplicates in Long Data Format: A Simple Approach Using PROC SORT and PROC SQL

Szu-Fu Chao, Educational Testing Service, Princeton, NJ

ABSTRACT

This paper will illustrate an efficient two-step method to identify and remove all occurrences of subjects with duplicate observations in long data format by implementing the combination of the NODUPKEY and DUPOUT options in PROC SORT and one of two queries in PROC SQL. These steps will help users produce final duplicate-free data with those problematic subjects excluded. A complete worked-through example will be presented to illustrate the entire process.

INTRODUCTION

Testing data collected from computer-based assessments often have issues with duplicate records resulting from participants' behavior during the test session or testing platform programming errors. The ability to draw critical inferences and produce high-quality recommendations to improve teaching and learning are based on data collected from tests, and therefore ensuring the cleanliness of the gathered data is essential before conducting any analyses to inform these types of recommendations. The option NODUPKEY from PROC SORT can indeed provide a good way to handle duplicates by removing redundancies and only keeping intended observations with appropriate key variables specified (Kelsey, 2005).

However, when the sample size is sufficiently large and the number of participants who have duplicate records only accounts for a small proportion (e.g., less than 3% of the total number of participants), researchers might just want to keep the unique participant records and remove anyone who has duplicate records. That is, the researcher would remove both the primary record and the associated duplicate record for this subset of examinees. There is no explicit function that can handle this task directly in SAS® software. However, this goal can be achieved easily by further applying one step in PROC SQL with the information extracted from the data generated by the option DUPOUT in PROC SORT. The goal of this paper is to describe a quick method for creating a duplicate-free data set. This approach will be demonstrated using a mock example from a computer-based assessment field test.

THE SCENARIO AND DATA BACKGROUND

As part of a large research project designed to provide effective strategies for improving reading comprehension for students in grades PreK–12, Educational Testing Service developed an innovative, scenario-based assessment referred to as the Global, Integrated, Scenario-Based Assessment (GISA). In a study conducted during the 2015–2016 school year, data were collected by delivering 12 GISA forms to approximately 2,200 students in grades 4 through 10 from 23 schools in urban, suburban, and rural locations across the United States. One of the test forms, on the topic of women's suffrage (WMSF), will be used in this example. There are 32 items on WMSF and 198 examinees took the form.

Testing data may be presented in long format, in which each row contains only one item record per subject. For the current example, an examinee without any duplicate data will have up to 32 records in the dataset. The example data initially have 5,960 rows and contain 15 variables including background

information. For simplicity, as well as privacy and confidentiality, only 5 variables are included as shown below.

	CandidateID	Form	Item	Score	ItemTime
1	RFU3KG09S0058C	GISA/WMSF	RFU_50022	0	9.756613
2	RFU3KG09S0058C	GISA/WMSF	RFU_50023	0	9.756613
3	RFU3KG09S0058C	GISA/WMSF	RFU_50024	0	9.756613
4	RFU3KG09S0058C	GISA/WMSF	RFU_50025	0	9.756613
5	RFU3KG09S0058C	GISA/WMSF	RFU_60082	0	42.76859
6	RFU3KG09S0058C	GISA/WMSF	RFU_60083	0	42.76859
7	RFU3KG09S0058C	GISA/WMSF	RFU_60084	0	42.76859
8	RFU3KG09S0058C	GISA/WMSF	RFU_60085	0	42.76859
9	RFU3KG09S0058C	GISA/WMSF	RFU_60086	0	42.76859
10	RFU3KG09S0058C	GISA/WMSF	RFU_60087	0	42.76859
11	RFU3KG09S0058C	GISA/WMSF	RFU_60088	0	42.76859
12	RFU3KG09S0058C	GISA/WMSF	RFU_60089	0	42.76859

- CandidateID: A unique ID assigned to each subject
- Form: Name of the test form
- Item: A unique item code created for each item on the form
- Score: Points received based on responses to the corresponding item
- ItemTime: Time spent in seconds on the item or section of the form

TASK: IDENTIFYING AND REMOVING ALL OBSERVATIONS OF SUBJECTS HAVING DUPLICATES

Given that students are only allowed to take the test once (per session), it is anticipated that each student should have only one record for each item. In other words, we expect to see that each ["CandidateID", "Item"] pair is unique or distinct in the data. Now, suppose a subject has some duplicate observations; that is, the student has more than one record for any given item(s). We might choose to remove only those redundancies and keep one single record per examinee per item based on some criteria. Alternatively, one might simply choose to eliminate all of the records for that particular student. This second approach, where we remove the entire set of observations for a given subject in the data, is demonstrated below.

Step 1: Identifying Subjects Having Duplicates using NODUPKEY and DUPOUT in PROC SORT

```

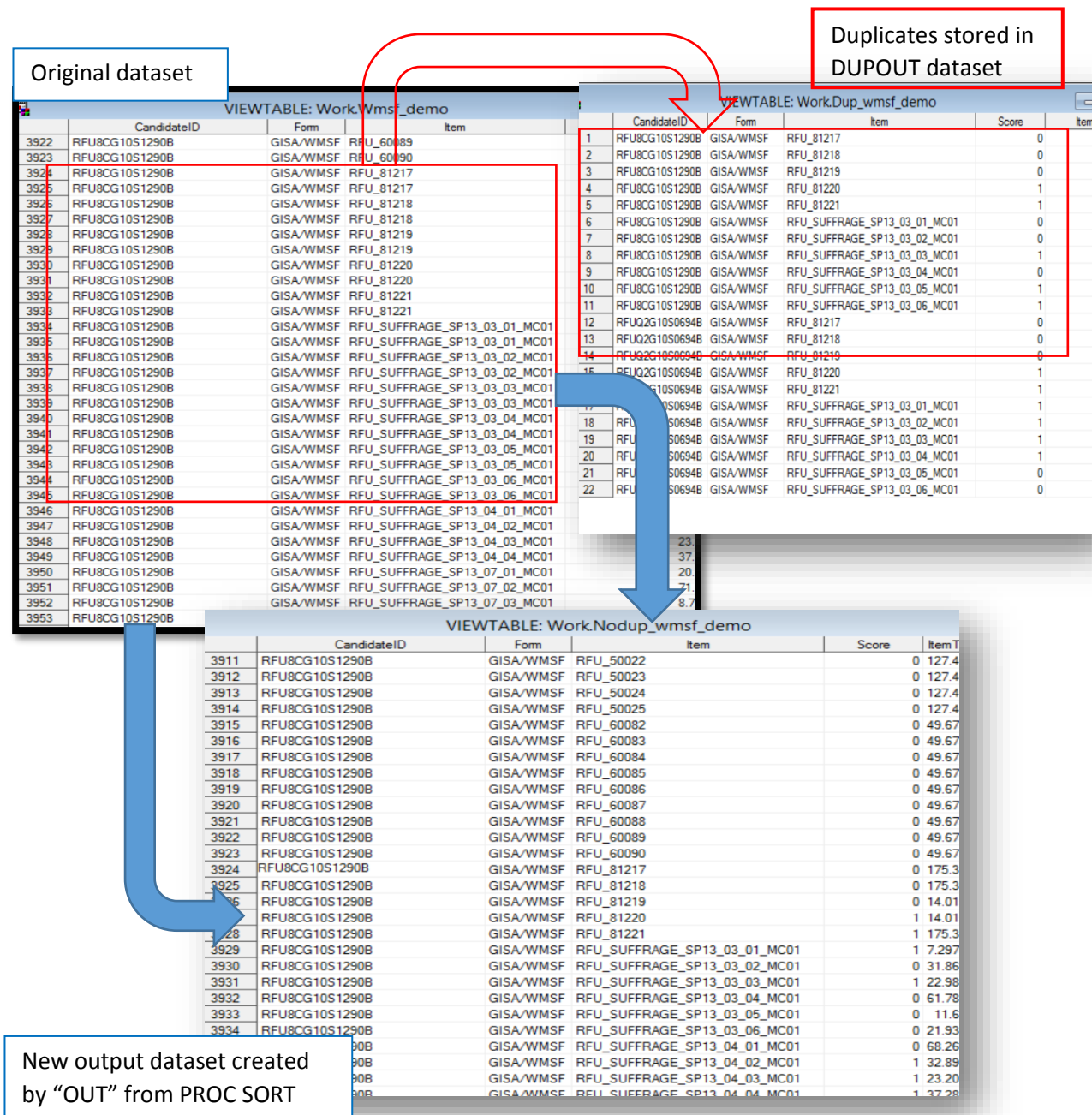
33  proc sort data=WMSF_demo dupout=dup_WMSF_demo nodupkey
34  out=nodup_WMSF_demo;
35  by CandidateID Item;
36  run;

NOTE: There were 5960 observations read from the data set WORK.WMSF_DEMO.
NOTE: 22 observations with duplicate key values were deleted.
NOTE: The data set WORK.DUP_WMSF_DEMO has 22 observations and 5 variables.
NOTE: The data set WORK.NODUP_WMSF_DEMO has 5938 observations and 5 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time          0.02 seconds
      cpu time           0.03 seconds

```

NODUPKEY in PROC SORT can be used to delete observations with duplicate BY variables values ("CandidateID" and "Item" in our case) while keeping the rest of the observations for the subjects and saving them with all others using the "OUT" statement. Additionally, one feature option in PROC SORT, DUPOUT, stores the duplicates that have been removed (Markovitz, 2006). This is actually the key to achieve the goal as we can then know exactly which subjects have duplicate records.

Now, let us look at our example. From the SAS log shown above, we see there are 22 duplicate observations; by referring to the DUPOUT dataset “Dup_wmsf_demo” as shown in the figure below, these correspond to data from two subjects (RFU8CG10S1290B and RFUQ2G10S0694B).



As mentioned previously, if we would just like to remove redundancies based on certain criteria while still keeping these subjects in the data, then the new output dataset “Nodup_wmsf_demo” would be what we are looking for.

However, what we really want now is to eliminate all observations for any subject that has duplicates in our data. This leads us to the second step.

Step 2: Removing All Observations for Subjects with Duplicate Issues from Data using PROC SQL

Now that we can easily tell which subjects have duplicate records by simply referring to the dataset created by DUPOUT, it can be quite efficient to implement PROC SQL to achieve our final goal. The code for that step is concise and straightforward, and the process time is short.

```
50 proc sql;
51   create table new_WMSF_demo as
52     select * from WMSF_demo
53     where CandidateID not in (select distinct CandidateID from dup_WMSF_demo);
NOTE: Table WORK.NEW_WMSF_DEMO created, with 5874 rows and 5 columns.

54 quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds
```

The trick here is that in the “WHERE” clause, we choose not to select subjects that appear in the dataset created by DUPOUT. An updated data set that does not include any subjects with duplicate issues can then be successfully produced. In our example, 86 observations from two subjects were removed thus the new data set now has only 5,874 rows.

If we preferred to update and overwrite the original data directly instead of creating another dataset, the “DELETE” query can be applied in PROC SQL for that purpose, as shown below.

```
115 /*Option 2: Using the "Delete" Query*/
116 proc sql;
117   delete from WMSF_demo
118   where CandidateID in (select distinct CandidateID from dup_WMSF_demo);
NOTE: 86 rows were deleted from WORK.WMSF_DEMO.

119 quit;
NOTE: PROCEDURE SQL used (Total process time):
      real time          0.01 seconds
      cpu time           0.01 seconds
```

Note that since we are applying the “DELETE” query to eliminate selected observations, the condition in the “WHERE” clause should now be what we want to remove. Also be careful when using this method as the original data cannot be recovered.

Double Check

It is always important to make sure the updated data are returned as expected. We can simply repeat Step 1 with the newly created data and check if there are any observations in the dataset created by DUPOUT. Our result is confirmed by the check shown below.

```
120 proc sort data=new_WMSF_demo dupout=dup_new_WMSF_demo nodupkey;
121   by CandidateID item;
122 run;

NOTE: There were 5874 observations read from the data set WORK.NEW_WMSF_DEMO.
NOTE: 0 observations with duplicate key values were deleted.
NOTE: The data set WORK.DUP_NEW_WMSF_DEMO has 0 observations and 5 variables.
NOTE: The data set WORK.NEW_WMSF_DEMO has 5874 observations and 5 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time          0.02 seconds
      cpu time           0.01 seconds
```

CONCLUSION

The brief procedure introduced here is very helpful as it saves time when cleaning testing data. This is a good example of how to effectively manage data collections in the educational research field. The code that has been applied is quite simple and should be easy to understand. Some SAS users may be interested in creating a macro statement for a task that needs to be executed repeatedly. In our case, since there are generally multiple forms administered in one study—meaning the same procedure needs to be applied for every form—a macro statement that incorporates the procedure demonstrated indeed makes the job easier. All in all people from different professional backgrounds who may be dealing with similar issues can benefit from the techniques demonstrated in this paper with relatively limited effort.

REFERENCES

- Kelsey, B. (2005). The Mystery of the PROC SORT Options NODUPRECS and NODUPKEY Revealed. In Proceedings of the SAS Users Group International (Paper no. 037-30), Philadelphia, PA. Retrieved from <http://www2.sas.com/proceedings/sugi30/037-30.pdf>
- Markovitz, H. (2006). Dup, Dedup, DUPOUT - New in PROC SORT. In Proceedings of the SouthEast SAS Users Group Conference (Paper no. CC14_06), Atlanta, GA. Retrieved from http://analytics.ncsu.edu/sesug/2006/CC14_06.PDF

ACKNOWLEDGMENTS

Special thanks to Jonathan Steinberg, Jonathan Weeks, Steven Holtzman, and Elizabeth Stone for their help and feedback in completing this paper. Also thanks to Ted Blew, Ed Kulick, Tenaha O'Reilly, and John Sabatini for their support.

CONTACT INFORMATION

Szu-Fu Chao
Educational Testing Service
Rosedale Rd, MS 20T Princeton, NJ 08541
609-436-4397
schao@ets.org

The views expressed are those of the author and do not necessarily represent or reflect the views of his professional organization.