Paper 117-2012

Reordering Values within Observations: Beyond CALL SORTC(N)

Adish Jain and Kate Bachtell, NORC at the University of Chicago, Chicago, Illinois, USA

ABSTRACT

PROC SORT orders SAS® dataset observations by the values of one or more variables. CALL SORTC/N are new CALL routines in SAS 9.2 and above that reorder values within each observation (in ascending order only) for the entire dataset. In short, PROC SORT orders dataset observations vertically, while CALL SORTC(N) orders values horizontally within the observation. Although these new routines can be useful in some situations, this paper deals with a more complex data management task: sorting multiple groups of variables horizontally in descending order. In our case, these groups of variables are items for every child living in a household, which also must be linked back to household data using the child's age and position in the household.

INTRODUCTION

Making Connections is a survey conducted in 10 low-income neighborhoods across the U.S. It is part of a larger initiative funded by the Annie E. Casey Foundation designed to assess the needs of families and children and to foster supportive communities that meet those needs. NORC interviewers conducted interviews by telephone and in person using paper questionnaires. For every case, the interviewer used at least two questionnaires to gather data:

1. A "Roster Booklet," in which demographic information about each household member is enumerated in descending age order.
2. A main questionnaire, which includes survey questions about the neighborhood, mobility, services and amenities, income and assets, child well-being, and other topics.

If there are children under age 18 living in the household (HH), the interviewer also asked a series of questions about each child. Two different versions were used: one for children ages 0-6 (the "younger child booklet") and one for children ages 7-17 (the "older child booklet"). Interviewers were trained to administer these questionnaires in the same order in which the children appear in the child roster, from oldest to youngest. However, the paper-and-pencil administration of the survey made it difficult to ensure that the protocol described above was always carried out as requested. When the child-specific data were not collected in the desired order, we needed to rearrange the values in the analytic data sets so that the child data fell in the correct order by age.

This paper's focus is on the resequencing of data collected for household members under age 18 in the child roster and the related data items that are collected in the child booklet(s). We also describe how we connect related data in the child booklets by appropriately moving around the values in the group of related variables in the child booklets.

The data from these child booklets are spread across 30 loops. A loop is defined as a set of questions asked about each child living in the household. the computer-assisted data entry (CADE) instrument was programmed to allow the same set of questions to be asked regarding up to 15 children across 30 sets of looped variables) – 15 for the older child data (also referred to as "T loops 1-15") and 15 for the younger child data (also referred to as "C loops 1-15"). The CADE clerk begins entering data from a given booklet after selecting either the 7-17 path or the 0-6 path. A given case cannot have data in the corresponding loop for both an older child and a younger child. If the older child variables are populated in the first loop, for example, the younger child variables must all be "empty" in the respective first loop, or legitimately skipped.

We also need to deal with what we will call "kid administrative data" that begins each younger/older child booklet loop. These data are copied directly from the child roster and serves to prompt the data entry clerk to key the booklets in a certain order (described above). While the booklet data is split into 30 loops (15 for younger children and 15 for older children), there are only 15 loops of kid administrative data.

PROBLEM BY EXAMPLE

Here is an example based on real data in which the children were enumerated out of proper descending age order. The youngest child, Jonah, should have been listed last, rather than first.

Child Roster					
R2. Row		**R11. Please tell me the first name of each person under age 18 living in this household starting with the oldest child.**	**R12. Is NAME male or female?**	**R13. What was [NAME's] age at (his/her) last birthday?**	**IF CHILD IS 14 OR OLDER ASK:** **R14. Is this person employed or not employed?** **EMPLOYED = 1** **NOT EMPLOYED = 2**
1		Jonah	M F	1	
2		Kyle	M F	9	
3		Katie	M F	8	

Figure 1. Improper Enumeration of Children on the Child Roster

Untreated, the Child Roster data would appear as follows in the "raw" data:

Child Roster Loop 1	Child Roster Loop 2	Child Roster Loop 3	Child Roster Loop 4…
Data from Jonah's row	Data from Kyle's row	Data from Katie's row	(Empty)…

Figure 2. Untreated Sequence of Child Roster Data

Continuing this example, the kid administrative and older/younger child booklet data would appear as follows in the raw data:

Kid Admin. Loop 1	Older Child Loop 1	Younger Child Loop 1	Kid Admin. Loop 2	Older Child Loop 2	Younger Child Loop 2	Kid Admin. Loop 3	Older Child Loop 3[a]…	Younger Child loop 3[a]…
Copy of Jonah's Child Roster Data	(empty)	Data from Jonah's booklet	Copy of Kyle's Child Roster Data	Data from Kyle's booklet	(empty)	Copy of Katie's Child Roster Data	Data from Katie's booklet	(empty)

[a] *There are actually 15 loops each of younger child and older child data. Only the first three are shown here for simplicity. Loops with no data are not shown.*

Figure 3. Untreated Sequence of Child Booklet Data

GOALS

We want to create a "clean" data set that looks like this:

Child Roster Loop 1	Child Roster Loop 2	Child Roster Loop 3…	Kid Admin. Loop 1	Older Child Loop 1	Younger Child Loop 1	Kid Admin. Loop 2	Older Child Loop 2	Younger Child Loop 2	Kid Admin. Loop 3	Older Child Loop 3[a]…	Younger Child Loop 3[a]…
Oldest child (Kyle)	2nd oldest child (Katie)	3rd oldest child… (Jonah)	Copy of oldest child's Child Roster data (Kyle)	Oldest child's Booklet (Kyle)	(empty)[b]	Copy of 2nd oldest child's Child Roster data (Katie)	2nd Oldest child's Booklet (Katie)	(empty)[b]	Copy of 3rd oldest child's Child Roster data (Jonah)	(empty)[b]	3rd oldest child's Booklet… (Jonah)

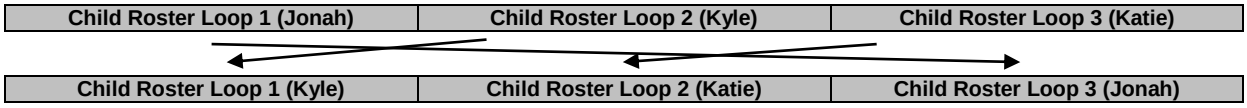
[a] *There are actually 15 loops each of younger child and older child data. Only the first three are shown here for simplicity. Loops with no data are not shown.*

[b] *Values will only be populated in older or younger child loop 1. Similarly, values will only be populated in the older or younger child loop 2 and in older or younger child loop 3.*

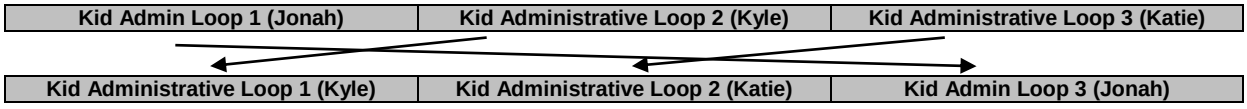
Figure 4. Desired Sequence of Child Roster, Kid Administrative, and Child Booklet Data

Figure 5 below continues the example from above and depicts the desired reordering of data in each of the four data sections. Variable names written in bold text would contain valid values in the data, while names appearing in regular font would be "empty", or legitimately skipped. The rows before the arrows represent the raw data and the rows after the arrows represent the "clean" (post-ordering) data.

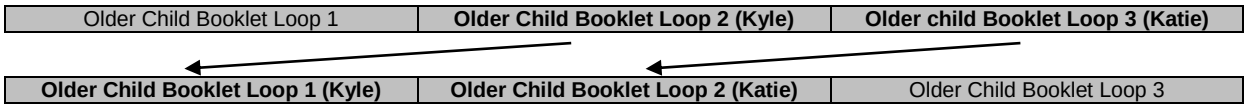
1. Here is what would need to happen for the Child Roster data:



2. Here is what would need to happen for the Kid Administrative data:



3. Here is what would need to happen for the Older child Booklet data:



4. Here is what would need to happen in the Younger Child Booklet data:

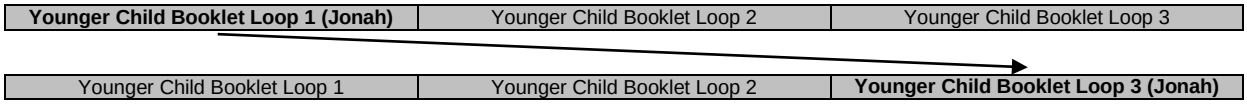


Figure 5. Reordering to Achieve Desired Sequence of Child Roster, Kid Administrative, and Child Booklet Data in Clean Data

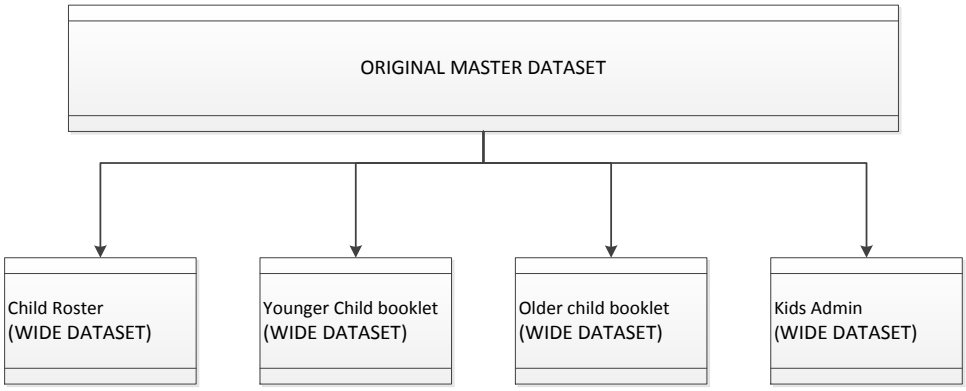
The values from the first younger child booklet loop should be moved to populate the 3rd loop of the younger child booklet data. This will leave null values in the first and second loops of the younger child booklet.

SOLUTION

The solution we devised is broken into four main steps: 1) divide the entire dataset into four smaller datasets, 2) independently process each of the four smaller datasets, 3) merge the independently processed smaller datasets, and 4) generate a quality control report to compare the original and final dataset. These steps are described in detail below.

1. Divide the entire dataset into four smaller datasets:
   - Child Roster
   - Younger Child Booklet
   - Older Child Booklet
   - Kid Administrative

   This step involves keeping the variables of interest in the respective section by using the KEEP statement. The outcome of this step is four smaller datasets. Each one has the same number of records as the original one, but with variables split into the four new datasets. Each dataset includes the original unique case identifier, along with other variables. Age is present in all these datasets, as that will be the driver variable for the reordering process.



City 1	…..	Name 2	Age 2	City 2	…..	Name 3	Age 3	City 3	…..	K Admin 1	….	Younger 1	…..	Older 1	….	K Admin 2	….	Younger 2	…..	Older 2	….	K Admin 3	….	Younger 3	…..	Older 3	….
CHI	….	KYLE	9	NYC	….	KATIE	8	SFO	….	1		1	LOOP			9				9	BRONX	8				8	BRIDGE

Gets divided into:

City 1	…..	Name 2	Age 2	City 2	…..	Name 3	Age 3	City 3	…..
CHI	…..	KYLE	9	NYC	…..	KATIE	8	SFO	…..

K Admin 2	….	K Admin 3	….
9		8	

Younger 1	…..	Younger 2	…..	Younger 3	…..
1	LOOP				

Older 1	….	Older 2	….	Older 3	….
		9	BRONX	8	BRIDGE

Figure 6. Division of Dataset into Four Smaller Datasets

2. Independently process each of the four datasets produced in Step 1. This process involves the actual horizontal sorting of values in each of the datasets. Extensive use of the SAS macro facility and data step was involved, as the process is very repeatable for group of variables and datasets. However, rather than focusing on showing the technical details of macro use, this step describes how the problem was solved at the process level. This step is explained below in sub-steps by carrying over the example we used in the previous section. Please note that only one observation from the roster is shown below, although in the actual dataset there were thousands of observations.

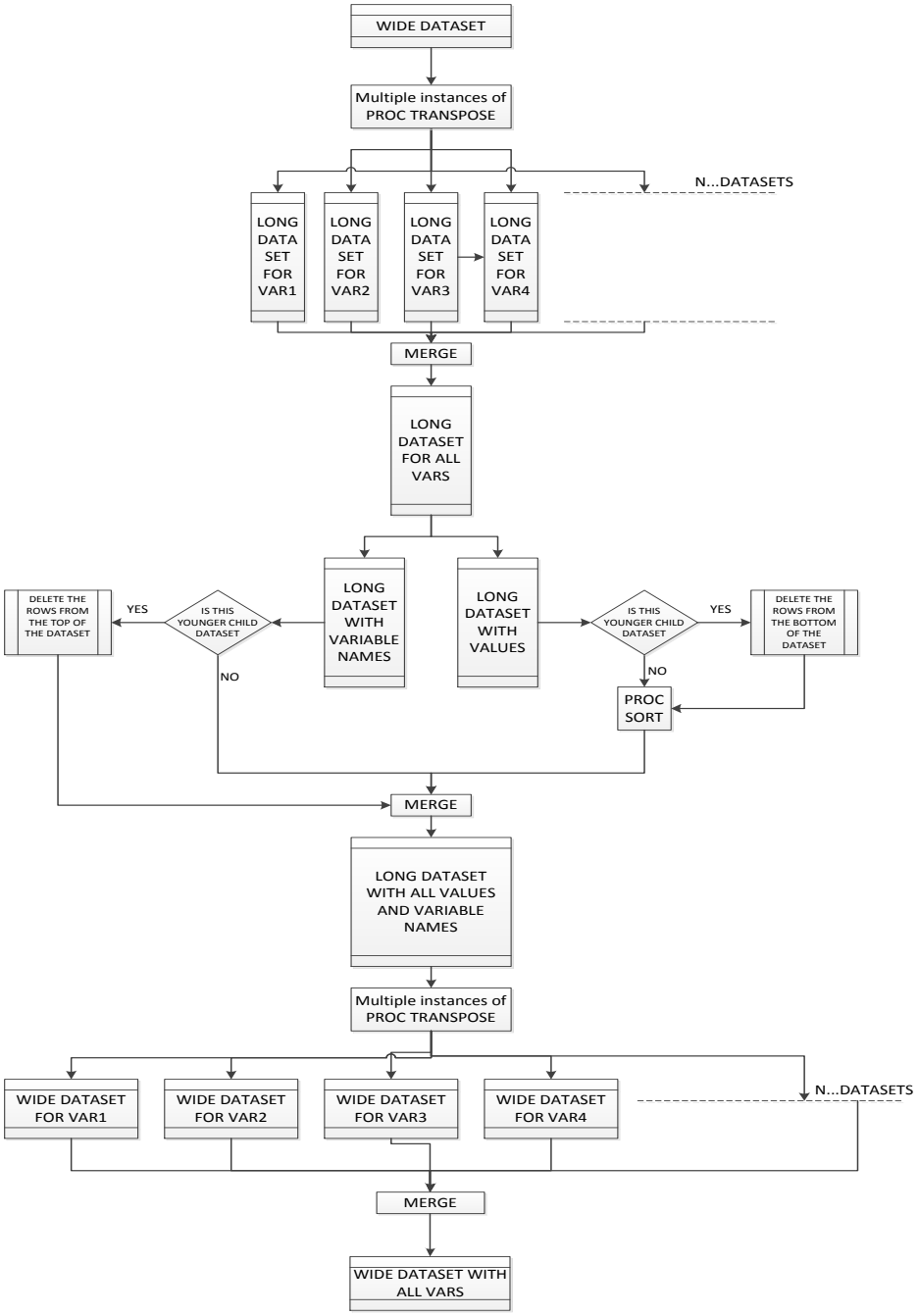


Figure 7. Independent Processing of Four Smaller Datasets

2.1. Using PROC SQL, create one macro variable that contains all the loop variables, separated by spaces, for a specific variable. For example, Loop_age1-loop_age15 is stored in one age macro variable, while loop_name1-loop_name15 is stored in a different macro variable.

2.2. Transpose each of these variables using PROC TRANSPOSE by the unique case identifier to create a long dataset. We are going from wide to long – that is, to a dataset with only three columns (unique case identifier, original variable name, value of the variable). In a sense we have collapsed the loop_name1-loop_name15 into one variable and fifteen records for each original row. The outcome of this step is that we have converted one wide dataset into multiple long datasets and collapsed the loop variables into a single variable while creating each long dataset.

City 1	…..	Name 2	Age 2	City 2	…..	Name 3	Age 3	City 3	…..
CHI	…..	KYLE	9	NYC	…..	KATIE	8	SFO	…..

Gets transposed into:

ID	Name	Variable Name
1	JONAH	NAME1
1	KYLE	NAME2
1	KATIE	NAME3

ID	Age	Variable Name
1	1	AGE1
1	9	AGE2
1	8	AGE3

ID	City	Variable Name
1	CHI	CITY1
1	NYC	CITY2
1	SFO	CITY3

…and

Younger 1	…..	Younger 2	…..	Younger 3	…..
1	LOOP				

Gets transposed into:

ID	Younger	Variable Name
1	1	YOUNGER1
1		YOUNGER2
1		YOUNGER3

ID	Address	Variable Name
1	LOOP	ADDRESS1
1		ADDRESS2
1		ADDRESS3

Figure 8. Transposing Variables to Create Long Dataset

2.3. Merge all these long datasets together by the unique case identifier using the DATA step MERGE statement, so that we have the values of each set of loop variables in one column and the variable names in another column for all the variables. The outcome of this step is one dataset.

Gets merged into:

ID	Name	Variable Name	Age	Variable Name	City	Variable Name
1	JONAH	NAME1	1	AGE1	CHI	CITY1
1	KYLE	NAME2	9	AGE2	NYC	CITY2
1	KATIE	NAME3	8	AGE3	SFO	CITY3

…and:

ID	Younger	Variable Name	Address	Variable Name
1	1	YOUNGER1	LOOP	ADDRESS1
1		YOUNGER2		ADDRESS2
1		YOUNGER3		ADDRESS3

Figure 9. Transposing Variables to Create Long Dataset

2.4. Divide this dataset into two datasets: one that includes the variables which store the original variable name, and the other that includes the values associated with these variable names. Each output data set has the same number of records as the original input data set but just a subset of the variables, as well as an ID variable that is kept on each of the data sets. Sort the dataset with values using age as the key within the unique case identifier, using PROC SORT.

Gets divided into:

ID	Name	Age	City
1	JONAH	1	CHI
1	KYLE	9	NYC
1	KATIE	8	SFO

ID	Variable Name	Variable Name	Variable Name
1	NAME1	AGE1	CITY1
1	NAME2	AGE2	CITY2
1	NAME3	AGE3	CITY3

Gets sorted into:

ID	Name	Age	City
1	KYLE	9	NYC
1	KATIE	8	SFO
1	JONAH	1	CHI

…and gets divided into:

ID	Younger	Address
1	1	LOOP
1		
1		

ID	Variable Name	Variable Name
1	YOUNGER1	ADDRESS1
1	YOUNGER2	ADDRESS2
1	YOUNGER3	ADDRESS3

…and sorted into:

ID	Younger	Address
1	1	LOOP
1		
1		

Figure 10. Dividing the Long Dataset into Separate Variable and Value Datasets

2.5. At this point we have the values sorted with the unique case identifier. We could merge the two datasets with variable name and sorted values and re-transpose the values back in the horizontal format using PROC TRANSPOSE, but we still haven't achieved the goal of having only either child or older child in a loop number. In order to achieve this we count the number of children in the older child dataset and move that many records in the younger child dataset. We will demonstrate how this is achieved in the next bullet.

- For the older child booklet dataset, start by calculating the number of older children per household from the older child booklet. This will help us determine from what position we need to start filling in the child booklet when we are working with the child booklet.
- For the younger child booklet dataset, we use the dataset that was created in a previous step, which stores the number of older children in the household. We use that dataset to move the child data in the younger child booklet dataset by the number of older children in the household. We achieve this movement by deleting the number of rows from the top within the unique case identifier from the variable name dataset in the younger child booklet dataset. This way the values are not moved but the variable names have moved up. In order to match up the number of records per the unique case identifier in both variable and value dataset we delete the bottom data records within the unique case identifier from the value dataset equal to the number of older children present in the household.

Gets sorted into after deletion:

ID	Younger	Address
1	1	LOOP

ID	Variable Name	Variable Name
1	YOUNGER3	ADDRESS3

Figure 11. Deleting Records in Both Datasets

2.6. Merge the two datasets together using the unique case identifier as the key. We have now completed the sorting/movement of data in the vertical dataset. The previous variable names now have the sorted values.

Gets merged into:

ID	Name	Age	City	Variable Name	Variable Name	Variable Name
1	KYLE	9	NYC	NAME1	AGE1	ADD1
1	KATIE	8	SFO	NAME2	AGE2	AGE2
1	JONAH	1	CHI	NAME3	AGE3	AGE3

…and into:

ID	Younger	Address	Variable Name	Variable Name
1	1	LOOP	YOUNGER3	subADD3

Figure 12. Merging the Datasets

2.7. Transpose each of these variables by the unique case identifier to create a wide dataset. From long to wide…. In a sense we have expanded the one variable into loop_name1-loop_name15 variables and 1 row for each original unique case identifier.

Gets transposed into:

ID	Name 1	Name 2	Name 3
1	KYLE	KATIE	JONAH

ID	Age 1	Age 2	Age 3
1	9	8	1

ID	City 1	City 2	City 3
1	NYC	SFO	CHI

…and into:

ID	Younger 3
1	1

ID	Address 3
1	LOOP

Figure 13. Transposing by each variable

2.8. Now merge this wide datasets per loop variables and create a combined dataset.

Gets merged into:

ID	Name 1	Name 2	Name 3	Age 1	Age 2	Age 3	City 1	City 2	City 3
1	KYLE	KATIE	JONAH	9	8	1	NYC	SFO	CHI

…and into:

ID	Younger 3	Address 3
1	1	LOOP

Figure 14. Merging the Datasets

3. Merge the individually processed datasets in step 2 to get a final dataset. This final dataset has the same number of records and variables but is sorted per our requirements.

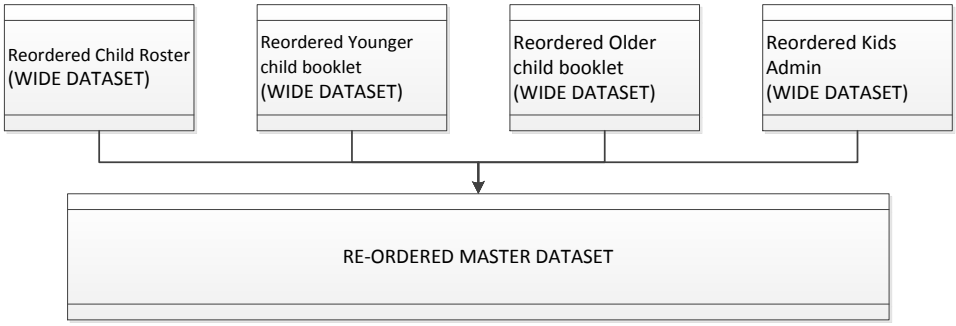


Figure 15. Final Merge of Individually Processed Datasets to Create Final Clean Dataset

4. Generate a quality control (QC) report to compare the original dataset and the final dataset to see how many cases were resequenced and to examine those values that were moved in the process of resequencing. This was generated by using PROC COMPARE on the original dataset and the resequenced dataset on a limited number of variables.

CONCLUSION

SAS software provides a variety of tools that can be used to handle complex data management problems like the one described in this paper. Although there is not a single function and/or call routine to solve this problem directly, there is a range of tools and techniques that can be combined to solve these kinds of problems. The technique described in this paper can be used by others if they have a similar “horizontal sorting” problem, where they need to move a group of variables instead of only one.

ACKNOWLEDGMENTS

We would like to thank Mike Rhoads.

RECOMMENDED READING

For more details about some of the techniques mentioned in this paper, refer to the SAS documentation for PROC SORT and the SORTC and SORTN call routines SAS help for Base SAS

CONTACT INFORMATION

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

Name: Adish Jain
Enterprise: NORC at the University of Chicago
Address: 55 E Monroe St
City, State ZIP: Chicago
Work Phone: 312-759-5071
E-mail: jain-adish@norc.org

Name: Kate Bachtell
Enterprise: NORC at the University of Chicago
Address: 55 E Monroe St
City, State ZIP: Chicago
Work Phone: 312-759-5095
E-mail: bachtell-kate@norc.org