Paper 1445-2017

Complex Merging of Emergency Department and Hospitalization Data to Create a Longitudinal Data Set

Charlotte Baker, Florida Agricultural and Mechanical University

ABSTRACT

Epidemiologists and other health scientists are often tasked with solving health problems but find collecting original data prohibitive for a multitude of reasons. For this reason, it is common to instead utilize secondary data such as that from emergency departments (ED) or inpatient hospital stays. In order to use some of these secondary data sets to study problems over time, it is necessary to link them together using common identifiers and still keep all the unique information about each ED visit or hospitalization. This paper will discuss a method that was used to combine five years worth of individual ED visits and five years worth of individual hospitalizations to create a single and (much) larger data set for longitudinal analysis.

INTRODUCTION

Hospital and emergency department data have multiple uses when assessing the health of the population. While these data are often used in a manner that is consistent with de-identified data, i.e. treating each observation as a discrete occurrence, we often know and must acknowledge that a single person could appear in this data more than one time. This means, for example, calculating the proportion of visits due to a cause rather than the incidence of an event in a year.

However, if one has access to identifiable information, it is not only possible to identify how often one frequents the emergency department or hospital in a year (or a series of years), it is also possible to better assess the rate of unique patients needing care and how many patients may be in need of differing treatment or therapy based on other characteristics. This could be useful in determining, for example, the number of times a person comes in with a particular type of cardiac related complaints and allow better follow-up with a primary care physician. This paper will show two methods for combining multiple data sets with multiple diagnostic possibilities. The methods can be tweaked in order to account for fewer or more possible diagnoses (or whatever it is that you are measuring from farm tools to bank account types).

In epidemiology, one of the best types of observational studies we can do is a cohort study. Because a cohort study collects information on a person at least two time points (at the beginning and end of a study), we get to see a time relationship between two factors in a designated group of people such as the type of housing you live in and the development of asthma. Sometimes we get to see cause and effect. We get to see how these relationships naturally develop because we are not controlling what they are being exposed to (the type of housing) or what disease they get (asthma). In some situations, however, we do not have the infrastructure, money, or time to collect information that lets us look at those time based relationships. Sometimes we are looking at rare conditions and sometimes it just is difficult to get enough people without requiring many years of data collection just to get an appropriate number of subjects to study. The methods described in this paper will show ways to take data from cross-sectional data sets and create a single data set that allows for studying data over a period of time in much the same way we would a data set that was created from a cohort study.

MERGING TECHNIQUES

Merging and concatenating data are regularly used tools in a health data analyst's toolbox. Merging can be done using a DATA step or PROC SQL and refers to putting data side by side. This might be used when the data for the same people is contained in multiple data sets, e.g. a demographic data set and a data set with work history. Concatenation can also be done with the DATA step or PROC SQL and refers to stacking observations on top of each other. This might be used when the same data is available for different groups of people, e.g. three seasons of football game statistics. Data can be both concatenated and merged in order to create a data set for analysis. A good example of this is data from the National Health and Nutrition Examination Survey (NHANES) collected by the Centers for Disease Control and

Prevention. NHANES data is collected annually and each year contains sections (separated into different data sets) for things such as demographic characteristics, questionnaire responses, and results from the physical examination and laboratory testing. To use a single year of information, data sets would need to be merged and to use data from multiple years data sets would need to be concatenated.

There are several types of merges. Two common methods are the one to one merge and the one to many merge. One to one merges mean that each unique individual in one data set has just one possible match in another data set. One to many merges mean that each unique individual in one data set has the possibility of being matched with multiple observations in another data set. In either instance there needs to be some common identifier such as a given ID number or a name or an address. This is why one to one merges and one to many merges are referred to as match-merges. Before merging, one should be careful that columns with the same name (outside of the variables being used to merge by) are in fact the same thing and formatted the same way to prevent data being overwritten.

If we do not have a unique identifier or if we know that the observations from one data set to another are independent, we often use concatenation instead of merging when working with health information. The basic requirement for this is that the variables with like information should have the same name.

In the present paper, to solve the problem at hand, data needed to be both concatenated and merged but all observations that had the same ID number needed to be on the same row instead of remaining stacked on top of each other. When we use data from health systems, such as hospital data, all patients do not have to have the same number of diagnoses or visits. This meant the solution needed to be flexible enough to fit any possible situation, especially because instead of only keeping one set of diagnoses all relevant information from every visit needed to be kept and each set of visits needed new and unique variable names. Two flexible solutions are the use of PROC TRANSPOSE or using ARRAYs and DO loops. PROC TRANSPOSE use a combination of steps to rearrange the data so that all the information for each identifier is on a row of its own. The use of ARRAYs and DO loops takes a variable by variable approach to create the new columns for each set of visits and then rearranges the data one person at a time. No matter which method is chosen, if one has multiple data sets to work with it can be much easier to achieve the desired goal if the data is concatenated beforehand to create a single data set to work with.

The final method ultimately chosen for this project was the combination of ARRAYs and DO loops due to the ease of programming even when there were a multitude of variables. Using PROC TRANSPOSE requires much more repetitive typing. When there are large data sets, such as ones with 30 diagnosis variables alone, it takes much more time to write out the steps needed to achieve the required data set.

THE MERGE

When beginning any merge, it is important to distinguish what the final data set should look like. It is important to know what variables need to be included, what should they look like, and what observations should be included and how should they be included. In this example, we used data from the National Hospital Ambulatory Medical Care Survey (NHAMCS). The data chosen was from the 2009, 2010, 2011, 2012, and 2013 emergency department data sets. The data was modified to add an identifier (solely for this demonstration) and to select a subsample of variables and observations. A total of 67 observations were selected for this example data set, representing 12 individuals. The code to achieve this model data can be found at the provided link under References. The data set contains four variables – three are diagnoses received in the emergency department (DIAG1, DIAG2, DIAG3), and one is the identifier variable (ID).

In this example, we are concerned with and want to keep the information for each visit a person had to an emergency room in the five-year period. The data, like most emergency department and hospitalization data set, did not include a variable that counted the number of variables so first we create one.

First, we need to sort our data set (hospital) by our identifying variable (id) using PROC SORT.

```
PROC SORT DATA=hospital;
BY id;
RUN;
```

Second, we use an IF-THEN-ELSE statement to tell SAS that if it is the first time it comes across that value of id then SAS should give our new variable visit a value of 1. If that is not true it should add 1 to the last value of visit. This means the second time it comes across a value for id, the value for visit will be 2 and so on. Because we use the BY statement in our DATA step, the IF-THEN-ELSE is computed separately for each unique id.

```
DATA hospital2;
SET hospital;
BY id;
IF FIRST.id THEN visit = 1;
ELSE VISIT + 1;
RUN;
```

Third, we look for the maximum number of visits possible in the data set using PROC FREQ.

```
PROC FREQ data = hospital2;
TABLES visit;
RUN;
```

Sometimes when acquiring the frequency of visits, we might notice anomalies in the number of visits. For example, if you have someone with >365 emergency department visits, that is a definite red flag that the identifier is a catchall for persons that did not provide a unique id. These likely need to be discarded before continuing based on judgement and knowledge of the conditions of interest and the general idea that we are interested in only analyzing information for individual people.

METHOD 1 – ARRAYS AND DO LOOPS

First, we sort our new data set by the variable id.

```
PROC SORT DATA = hospital2;
BY id;
RUN;
```

Next, we create multiple arrays. There should be one array representing each set of new columns that needs to be created from every unique visit for each individual person. Notice that these arrays refer to variables that do not yet exist. These will be created in the first DO loop.

```
DATA hospital3;
```

Optionally, you can add a RETAIN statement between the DATA and SET statements to set the order of the columns in the resulting data step. The variables will be in the order written.

```
RETAIN id visit year_1 diag1_1
diag2_1 diag3_1 year_2 diag1_2
diag2_2 diag3_2 year_3 diag1_3
diag2_3 diag3_3 year_4 diag1_4
diag2_4 diag3_4 year_5 diag1_5
diag2_5 diag3_5 year_6 diag1_6
diag2_6 diag3_6 year_7 diag1_7
diag2_7 diag3_7 year_8 diag1_8
diag2_8 diag3_8 year_9 diag1_9
diag2_9 diag3_9 year_10 diag1_10
diag2_10 diag3_10
;

SET hospital2;
```

So that we can use the FIRST. LAST. logic below, we need to use this BY statement.

```
BY id;
```

We know from the PROC FREQ above that there were a maximum of ten visits for any one patient. This means we need ten new first diagnosis (DIAG1), second diagnosis (DIAG2), third diagnosis (DIAG3), and year (YEAR) variables. We will later use the arrays we make here to fill in the appropriate values for these variables. Three of our original variables are character variables (DIAG1, DIAG2, and DIAG3) and one is a numeric variable. In the first array, which we named diag1s, we could have written the number 10 in the brackets but instead used an asterisk to let SAS determine how many variables were being referenced in the list diag1_1-diag1_10. We use a $ because the original variable DIAG1 is character and we want the new variables to also be character. To make sure that the new variables have an appropriate length, we set it here with the value of 10. In this example, we want the new variables to be called diag1_1 through diag1_10 where the last number is the corresponding visit number. We could have called these anything we wanted to.

```
ARRAY diag1s{*} $ 10 diag1_1-diag1_10;
ARRAY diag2s{*} $ 10 diag2_1-diag2_10;
ARRAY diag3s{*} $ 10 diag3_1-diag3_10;
ARRAY years{*} year_1-year_10;
```

For each individual person, we want to now actually create the columns and assign a default value of missing for each set of new variables. We do this using the IF-THEN statement and two DO loops. This syntax says that only once for each unique identifier (the IF-THEN and the first DO), SAS should set each new variable we named in the arrays to missing (the second DO). As with any DO loop, each of our two requires an END statement.

```
IF first.id THEN DO;
   DO i=1 to 10;
   diag1s{i}=' ';
   diag2s{i}=' ';
   diag3s{i}=' ';
   years{i}=.;
   END;
END;
```

Now we need to assign the values from the visits in the original data set to the new columns we made. Values will only be filled in if there was originally a value for that visit. If someone had five visits total, there will be missing values in the new columns corresponding to the sixth through tenth visits (e.g. diag1_6 will have a missing value). We do this with two IF-THEN statements and a DO loop. Remembering the visit variable that we made earlier, we know that for every visit someone had (no matter the year) the value of that variable must be between 1 and 10. The code will run in a loop for every value of visit. For example, the part of the code that says diag1s{visit} = diag1, SAS will assign the value of the original diag1 variable to the new diag1_1 if it is the first visit someone has had. Once we have reached the end of someone's visits, meaning that it is the last time their id appears in the data set, SAS will output the responses to the new data set (the last IF-THEN).

```
IF 1 le visit le 10 THEN DO;
diag1s{visit} = diag1;
diag2s{visit} = diag2;
diag3s{visit} = diag3;
years{visit} = year;
END;

IF last.id THEN OUTPUT;
```

Finally, we drop the i variable that was created for the first DO loop and the original variables DIAG1, DIAG2, DIAG3, and YEAR. We drop these last four because they are no longer needed. We would not want to drop variables that have not been replicated such as the id variable.

```
DROP i diag1--year;

RUN;
```

Once this DATA step is complete, the hospital3 data set will contain all the information for each individual on one row. While we created new columns in order to keep the information contained in the original four variables for every time a person appeared in our original data set, this method can be completed for more variables or fewer variables depending on the needs of your analysis.

Obs	id	visit	year_1	diag1_1	diag2_1	diag3_1	year_2	diag1_2
1	1	10	2009	Open wound of finger(s), without ment...			2009	
2	2	10	2009	Jaundice, unspecified, not of newborn			2009	Other specified organic brain syndrom...
3	3	10	2009	Migraine, unspecified without mention...	Other chronic pain		2009	Chronic airway obstruction, not elsew...
4	4	9	2009	Calculus of kidney			2010	Abdominal pain, unspecified site
5	5	9	2009	Bipolar affective disorder, depressed...			2010	Hemorrhage of gastrointestinal tract,...

Output 1. Sample of the Data Set Created by the ARRAYs and DO loop method

METHOD 2 – PROC TRANSPOSE

First, we sort our new data set by the variable id.

```
PROC SORT DATA = hospital2;
BY id;
RUN;
```

Our next step is to write our PROC TRANSPOSE statements. PROC TRANSPOSE can work with character data or numeric data but not both. We need one statement for our diagnosis variables and one statement for the year variable. In the PROC TRANSPOSE, we identify what we want the output data set that has transposed data should be called. In the first statement, we call our output data set hospitalt. We need to specify what name should be put at the beginning of the new variables that are made. We do this by using PREFIX. PROC TRANSPOSE itself will create a new variable called _NAME_ that will contain the name of the variable listed on the VAR statement. The BY statement indicates that there should be a row of output for each person for each variable listed on the VAR statement. The ID statement identifies what value will be attached to the PREFIX to create the new variable name.

Output 2 shows what part of data set looks like from the first PROC TRANSPOSE. The person who has the value of 1 as their id number had a total of ten visits over the five year period (four visible in the

output) and there is one row of information in the transposed data set for each variable – DIAG1, DIAG2, and DIAG3.

We sort the new data sets we created with the PROC TRANSPOSE to prepare for the next step.

```
PROC TRANSPOSE DATA = hospital2 OUT = hospitalt PREFIX=visit;
BY id;
VAR diag1 diag2 diag3;
ID visit;
RUN;

PROC SORT DATA = hospitalt;
BY id;
RUN;

PROC TRANSPOSE DATA = hospital2 OUT = hospitaltn PREFIX=visit;
BY id;
VAR year;
ID visit;
RUN;

PROC SORT DATA = hospitaltn;
BY id;
RUN;
```

Obs	ID	_NAME_	_LABEL_	visit1	visit2	visit3	visit4
1	1	DIAG1	Physician's diagnosis #1 - detailed	Open wound of finger(s), without ment...		Chest pain, unspecified	Personal hist venous thrombosis and embo
2	1	DIAG2	Physician's diagnosis #2 - detailed			Observation following other accident	Other chronic pain
3	1	DIAG3	Physician's diagnosis #3 - detailed				Pain in limb
4	2	DIAG1	Physician's diagnosis #1 - detailed	Jaundice, unspecified, not of newborn	Other specified organic brain syndrom...	Abdominal pain, other specified site	Sprains and strains of lumbosacral (j...
5	2	DIAG2	Physician's diagnosis #2 - detailed				
6	2	DIAG3	Physician's diagnosis #3 - detailed				

Output 2. Sample of the Data Set Created by PROC TRANSPOSE

After, we transpose the data, we need to put our two new data sets together. In our MERGE statement, we can specify which parts of each data set we want to include in the merge and how we want them included. We will end up with a data set with a single row of data for each ID number. Because of this, we need to rename the visit1 to visit10 variables to indicate what variable the information comes from and what visit it belongs to. In order to rename the columns for each variable, select the data set to merge and then identify which variables you are trying to rename by using the WHERE statement. The UPCASE is included to convert any values of _NAME_ that are not uppercase to uppercase so it will match the identification of the value found in the quotation marks. The code also includes a RENAME statement that renames each variable (such as visit3) with a new name (visit3 is renamed diag1_visit3 if the value in the _NAME_ variable is DIAG1) (Output 3).

```
DATA hospitalt2;
MERGE hospitalt (WHERE=(UPCASE(_NAME_)='DIAG1')
RENAME=(visit1=diag1_visit1 visit2=diag1_visit2
visit3=diag1_visit3 visit4=diag1_visit4 visit5=diag1_visit5
visit6=diag1_visit6 visit7=diag1_visit7
visit8=diag1_visit8 visit9=diag1_visit9 visit10=diag1_visit10))

hospitalt (WHERE=(UPCASE(_NAME_)='DIAG2') RENAME=(visit1=diag2_visit1
visit2=diag2_visit2 visit3=diag2_visit3 visit4=diag2_visit4
visit5=diag2_visit5 visit6=diag2_visit6 visit7=diag2_visit7
visit8=diag2_visit8 visit9=diag2_visit9 visit10=diag2_visit10))

hospitalt (WHERE=(UPCASE(_NAME_)='DIAG3') RENAME=(visit1=diag3_visit1
visit2=diag3_visit2
visit3=diag3_visit3 visit4=diag3_visit4 visit5=diag3_visit5
visit6=diag3_visit6 visit7=diag3_visit7
visit8=diag3_visit8 visit9=diag3_visit9 visit10=diag3_visit10))

hospitaltn (WHERE=(UPCASE(_NAME_)='YEAR') RENAME=(visit1=year_visit1
visit2=year_visit2
visit3=year_visit3 visit4=year_visit4 visit5=year_visit5
visit6=year_visit6 visit7=year_visit7
visit8=year_visit8 visit9=year_visit9 visit10=year_visit10))
;
BY id;
DROP _name_ _label_;
RUN;

PROC PRINT DATA = hospitalt2;
RUN;
```

Obs	ID	diag1_visit1	diag1_visit2	diag1_visit3	diag1_visit4
1	1	Open wound of finger(s), without ment...		Chest pain, unspecified	Personal hist venous thrombosis and embo
2	2	Jaundice, unspecified, not of newborn	Other specified organic brain syndrom...	Abdominal pain, other specified site	Sprains and strains of lumbosacral (j...
3	3	Migraine, unspecified without mention...	Chronic airway obstruction, not elsew...	Unspecified outcome of delivery	Other abnormal glucose
4	4	Calculus of kidney	Abdominal pain, unspecified site	Unspecified otitis media	Chest pain, unspecified
5	5	Bipolar affective disorder, depressed...	Hemorrhage of gastrointestinal tract,...	Diplopia	Spasm of muscle

Output 3. Sample of the Data Set Created Using the MERGE

After, we merge our data, we may want to reorder the variables with a RETAIN statement.

```
DATA hospitalt3;
RETAIN id visit year_visit1 diag1_visit1 diag2_visit1 diag3_visit1
year_visit2 diag1_visit2 diag2_visit2 diag3_visit2
year_visit3 diag1_visit3 diag2_visit3 diag3_visit3
year_visit4 diag1_visit4 diag2_visit4 diag3_visit4
year_visit5 diag1_visit5 diag2_visit5 diag3_visit5
year_visit6 diag1_visit6 diag2_visit6 diag3_visit6
year_visit7 diag1_visit7 diag2_visit7 diag3_visit7
year_visit8 diag1_visit8 diag2_visit8 diag3_visit8
year_visit9 diag1_visit9 diag2_visit9 diag3_visit9
year_visit10 diag1_visit10 diag2_visit10 diag3_visit10
;
SET hospitalt2;
RUN;
```

POSSIBLE ISSUES

If there are no identification numbers or other ways to uniquely identify people available, this method will not work. Be aware that all observations without an identifier or with an identifier that might be a catchall identifier will likely need to be dropped to accommodate the use of this method. While you lose these observations, you gain the ability to look at the data in a way to gather information such as the incidence of a condition.

Depending on the memory size of all the data sets that will be used, the number of observations, and the type of data being used (survey vs non-survey), the memory of the computer can play a role in how quickly the syntax will work. Be aware that this work could take more than a few seconds. If using survey data, be careful not to delete observations but create a domain variable to use with this method instead.

If using PROC TRANSPOSE, any formats previously applied to original data sets still work. However, with the ARRAYs and DO loops, it is necessary to apply the formats to the new variables that were created in order to see them in the output.

CONCLUSION

In this paper, we needed a method to merge data sets with a varying number of observations and where there could be multiple observation per person. Because of this, we needed to take into account a need for multiple possible columns in the final data set to scale up for more visits per person. We used a combination of ARRAYs and DO loops to achieve the final data set that met what we were aiming for. In our opinion, it was a much similar solution than using PROC TRANSPOSE especially when using data sets with more variables. Being able to rearrange our data so that we can look at information for individuals over time changes the types of analysis that can be done with the data and, therefore, increase the number of possible impacts the information can have on our understanding of health and disease prevention.

REFERENCES

Burlew, Michele M. 2009. Combining and Modifying SAS Data Sets: Examples. Second edition. Cary, NC: SAS Press.

Cody, Ron. 1999. “Transforming SAS® Data Sets Using Arrays”. *Proceedings of SUGI24.* Available at: http://www2.sas.com/proceedings/sugi24/Advtutor/p48-24.pdf

UCLA Institute for Digital Research and Education. “Match Merging Data Files in SAS”. Available at: http://stats.idre.ucla.edu/sas/modules/match-merging-data-files-in-sas/

Centers for Disease Control and Prevention. “Ambulatory Health Care Data”. Available at: https://www.cdc.gov/nchs/ahcd/ahcd_questionnaires.htm

Download code: https://github.com/drbaker11/SASGF17.git

CONTACT INFORMATION

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

Charlotte Baker
Florida Agricultural and Mechanical University Institute of Public Health
1515 S. Martin Luther King Jr Blvd
Tallahassee, FL 32307
charlotte.m.h.baker@gmail.com