SAS® GLOBAL FORUM 2020 #SASGF

Paper 5142-2020

Moving from Messy Data to a Clean Analytic Dataset: Common Data Management Techniques using SAS®

Raymond B. Smith, University of South Carolina; Jason A. Schoeneberger, ICF International; Bethany A. Bell, University of South Carolina

ABSTRACT

Despite the amount of quantitative research that exists in the social and behavioral sciences, many graduate programs do not offer classes focused on the multitude of steps necessary to manage quantitative datasets. Instead, this skill is often learned through trial and error with the beginning SAS user having to use multiple resources, including, but not limited to the plethora of proceedings papers from SAS Global Forum as well as regional users' groups as well as other SAS friendly resources such as UCLA's Institute for Research and Digital Education and SAS publications. Although these resources are incredibly useful when a SAS user knows what procedure he/she needs, they are less useful for the novice analyst who does not know where to begin. The focus of this paper is to help guide the novice user through common data management techniques to transform raw, messy data into an analytic dataset. This paper contains information on data management processes as basic as getting an external data set into SAS to more advanced techniques such as conducting exploratory factor analysis and creating composite variables. We illustrate the various processes using Wave 1 public use data from the National Longitudinal Study of Adolescent to Adult Health (Add Health).

Keywords: data cleaning, data management, data manipulation

INTRODUCTION

The goal of data cleaning is to ensure that the data used within the analytic process provides an accurate representation of the constructs that are being examined. Datasets usually contain numerous irregularities such as outliers or codes to indicate various forms of missingness (i.e, truly missing, skip patterns, refusals, etc.), which can significantly impact the analytic outcomes if the irregularities are unknowingly included. However, through data management and manipulation techniques, datasets can be cleaned to produce high-quality results that have incorporated better analytic decisions and outcomes due to a more representative and accurate dataset. Thus, the purpose of this paper is to provide a roadmap for data cleaning, management, and manipulation using a multitude of SAS® features and procedures. Although there are multiple ways to accomplish many tasks in SAS®, the methods and approaches shared in this paper are those that are commonly used in the social and behavioral sciences and are intended for those new to managing data.

Illustrations of SAS procedures and techniques included throughout this paper will come from Wave 1 public use data from the National Longitudinal Study of Adolescent to Adult Health (Add Health; www.cpc.unc.edu/projects/addhealth). By using publicly available data, readers can download the data and work through the examples provided. Also, although we use a large national dataset throughout the paper, please remember that the techniques included in the paper are relevant to both large national datasets as well as smaller, locally collected data. Messy data occur everywhere!!

Topics covered in this paper include (a) the importance of knowing ones data and consulting (or creating) codebooks and users manuals; (b) how to read data into SAS; (c) how to suppress formats; (d) how to examine the contents of a dataset; (e) how to merge datasets; (f) the importance of checking the log throughout the data management process; (g) how to convert missing data to SAS system missing; (h) how to create an age variable from interview and birth dates; (i) renaming variables; (j) dummy coding; (k) reverse coding; (l) how to create composite variables and assess reliability; (m) how to center variables to ensure all variables have a meaningful zero; (n) how to assess missing data mechanisms; and (0) how to create an analytic data set, including dividing data into explore and holdout samples.

KNOW THY DATA!

One of the most important steps that should be conducted prior to analyzing any dataset is reviewing the codebook and other documentation associated with the dataset that is selected for use. When working with national datasets such as Add Health, ECLS-K, NLSY, etc., codebooks and other documentation such as users' manuals and/or appendices are included when you download the data. However, if you are working with either primary data that you or a colleague collected, these materials might not exist. If that is the case, then the first step in the data management process is to create them.

Codebooks typically contain the information on how each variable in the dataset was collected – the exact wording used on the survey as well as the response options as presented in the survey and how each response option has been coded in the dataset. Codebooks often also contain a frequency table that depicts item response rates and data missingness characteristics. Appendix A contains a sample of the Add Health variables that are used throughout the paper – the information provided was taken directly from the Add Health codebook. By reviewing these items and available responses, the researcher will be informed as to whether the item is categorical or continuous, or if there are specific criteria that may exclude participants from responding. Further, the characteristics of item missingness are also included within the frequency table, which may be the result of unknown source of missing, participants refusing to answer an item, participants not knowing the answer to the question, or participants not being selected to answer a specific item (e.g., a legitimate skip).

For Add Health data, there are two codebooks – a PDF version that you get when you download the data (this is discussed more in the next section) and an online codebook – Add Health Codebook Explorer - http://www.cpc.unc.edu/projects/addhealth/documentation/ace/tool/. Whereas the Codebook Explorer is useful once you have identified the variables of interest for your study, it is less useful when you are first learning about the data and the variables that are available to you.

Users' manuals and other supplemental materials often include details on how the study was conducted (e.g., how were participants recruited/selected, what were the study inclusion criteria, etc.), copies of all questionnaires as they were administered to participants as well as information on any data management that the owners of the data have already conducted (i.e., a composite measure of socioeconomic status that was created from education and income variables in the dataset), scale reliability and validity, and when working with nationally representative data that used complex sample design, information regarding the use of analytical variables such as weights, clustering, and stratification. We do not cover details on analyzing complex survey data, however, there are many terrific papers that provide this information (e.g., Berglund, 2009; D'Agostino McGowan & Toll, 2015; Lewis, 2010)

When a researcher is working with a primary dataset and no documentation has been created, he/she should use the above information as a guideline for what to include in a codebook and users' manual. In doing so, remember that you need to create enough documentation that a person not involved in the research study can correctly use the data and be able to describe the details on all aspects of the study.

GETTING DATA INTO SAS AND CREATING A WORKING DATAFILE

Before managing your data, first you need to obtain it! Whereas Add Health data and corresponding documentation can be accessed from multiple sources (http://www.cpc.unc.edu/projects/addhealth/documentation/publicdata) from our experience, obtaining the data and documentation from the ICPSR is best: https://www.icpsr.umich.edu/icpsrweb/ICPSR/studies/21600/datadocumentation.

For this paper, if you would like to follow the examples, go to the ICPSR link and download two datafiles: DS1 Wave I: In-Home Questionnaire and DS2 Wave I: Public Use Contextual Database. The downloaded material will be in ZIP files. So, before you can access them, you need to unzip the files. Once each of the two ZIP files have been unzipped SAS C-TRANSPORT files, as well as other study documentation including users' manuals and codebooks, will appear. The SAS C-TRANSPORT files are named "21600-0001-Data.stc" (which is the In-Home Questionnaire) and "21600-0002-Data.stc" (which is the Contextual Database). The C-TRANSPORT file type is common when downloading data from national research studies. The example below provides details on how to move from a TRANSPORT file to a sas7bdat SAS datafile. Reading other types of datafiles (Excel, SPSS, Stata, CSV, etc.) can be done using the Import Wizard. Whereas we will focus on getting SAS C-TRANSPORT files read into SAS, Matlapudi & Knapp (2012) provide illustrative examples of importing other types of data into SAS.

SAS Code to read in the SAS C-TRANSPORT In-Home Questionnaire data:

```
filename in "C:\Users\bellb\Documents\SESUG/21600-0001-Data.stc";
libname out 'C:\Users\bellb\Documents\SESUG';
proc cimport file=in library=out; run;
```

In the above code, the FILENAME command is used to instruct SAS which datafile to read – you can see that the TRANSPORT file (21600-0001-Data.stc) is located on a computer on the C drive under user bellb, in the documents folder, in a subfolder called "SESUG". When using the FILENAME command you need to provide the directory location of the stc file AND the name of the datafile. Next, the LIBNAME command refers to the location on the computer where SAS will place the.sas7bdat datafile – in this example, it is the same location used in the FILENAME command, however, it does not have to be if you were to want to save the .sas7bdat file in a different folder, you can do that.

The PROC CIMPORT is the part of the code where the TRANSPORT file is converted to a SAS (.sas7bdat) file. This code directs SAS to read the file specified above in the FILENAME (FILE=in) command and to put the converted datafile in the location specified in the LIBNAME command (library=out). Using "in" and "out" as the names for the locations associated with the FILENAME and LIBNAME commands is arbitrary – you can use any "name" as long as it starts with a letter. We use "in" and "out" to help illustrate that we are reading "in" the TRANSPORT datafile and then we are "out" putting the converted file. During the PROC CIMPORT process, SAS automatically names the newly created datafile – in this example the new SAS datafile is named "da21600p1.sas". We recommend creating a copy of the datafile and giving it a more meaningful name.

SAS Code to rename a permanent SAS datafile:

```
libname AH 'C:\Users\bellb\Documents\SESUG';
options nofmterr;
```

```
    data ah.inhome;set ah.da21600p1; run;
```

In the code above, notice we list the new datafile name "ah.inhome" then the old datafile name "ah.da21600p1". Think of this like the "Save As" function in Word – you take one datafile and you save a copy of it with a new name. The OPTIONS NOFMTERR portion of the code is necessary to allow SAS to access the Add Health datafile. Like many national datasets, the Add Health data have user defined formats applied to them. You can access those format files if you want to, but, in our experience, that process is not easy given incompatibility issues between formats created in previous versions of SAS. Thus, using the NOFMTERR option suppress the formats within the SAS datafile. However, do not be concerned if you see a note about the formats in the log. Also notice that we have named the location on the computer where our SESUG folder as AH – i.e., LIBNAME AH. Again, using AH is arbitrary – you can use anything you wish as long as it starts with a letter. For us, because we are working with Add Health data, AH seemed like a good option.

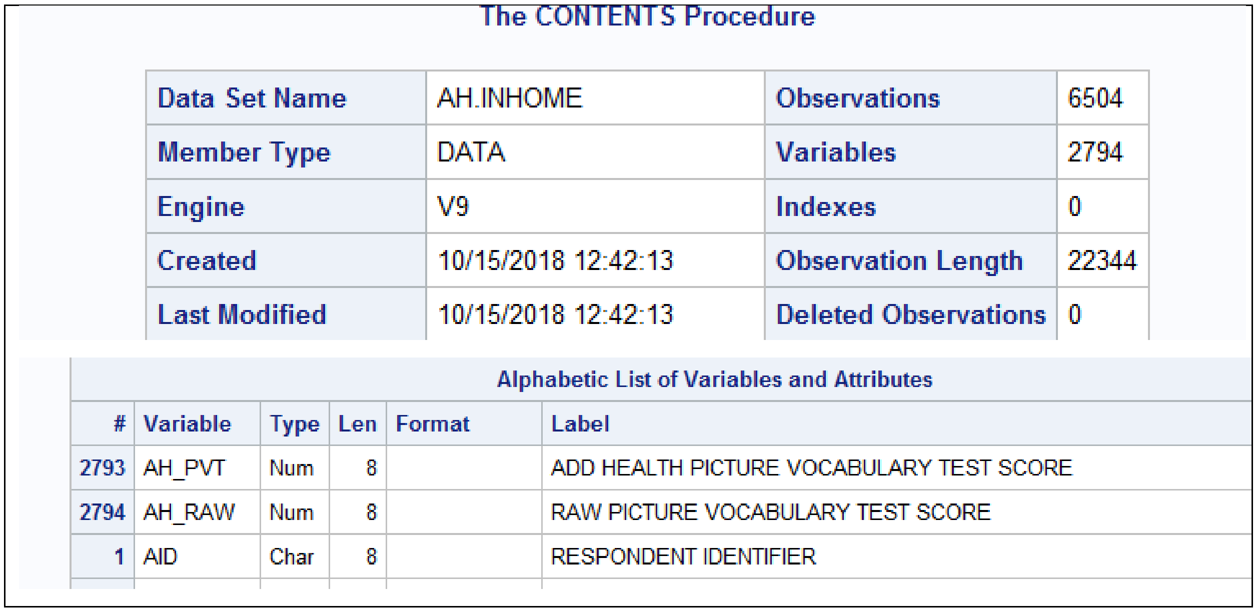
Next, we want to create one master datafile that contains both the In-Home Questionnaire data and data from the Contextual data. Before we can do that, we would repeat the steps above (i.e., converting the transport file to a SAS datafile and providing the SAS datafile with a meaningful name) using the Contextual Database instead of the In-Home Questionnaire data. To conserve space, that code is not shown, but is provided in the SAS syntax file that is available from the authors listed in the CONTACT INFORMATION section at the end of the paper. After replicating the code for the Contextual data, we now have two SAS datafiles – ah.inhome (as created using the code above) and ah.contextual (code to create not shown). The inhome datafile contains participant responses to all questionnaire items and the contextual datafile contains data about the areas in which the adolescent lives (e.g., Census data). More details on the contents of both of these datasets are provided in their respective users' manuals and codebooks, which were included in the ZIP files when the data were downloaded.

Before merging the two datafiles, you should examine the contents of each dataset. PROC CONTENTS provides the user with information on the number of observations in a dataset, the number of variables in a dataset, variable names (and labels when provided), as well as the format of each variable (i.e., character or numeric). Knowing how many observations and number of variables in each dataset is important to make sure that the merged datafile matches the details for each separate datafile.

SAS code for examining the contents of each of our two datasets is below:

```
    proc contents data = AH.inhome;
    proc contents data = AH.contextual; run;
```

Output 1 depicts partial output from PROC CONTENTS. Please note, the number of observations and variables is located in the upper righthand corner of the output.



The CONTENTS Procedure

Data Set Name	AH.INHOME	Observations	6504
Member Type	DATA	Variables	2794
Engine	V9	Indexes	0
Created	10/15/2018 12:42:13	Observation Length	22344
Last Modified	10/15/2018 12:42:13	Deleted Observations	0

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format	Label
2793	AH_PVT	Num	8		ADD HEALTH PICTURE VOCABULARY TEST SCORE
2794	AH_RAW	Num	8		RAW PICTURE VOCABULARY TEST SCORE
1	AID	Char	8		RESPONDENT IDENTIFIER

Output 1. Partial Output from PROC CONTENTS of the AH.Inhome Data

As shown in the PROC CONTENTS output (and reported in the Add Health codebook and users' manual), the Participant ID variable in Add Health data is "AID" – this variable is the key linking variable across Add Health datasets (located in the Alphabetic List of Variables and Attributes section of the output). This is important because this is the variable that will be used to merge the two datafiles into one. To accomplish this, first you must sort each dataset by AID.

SAS code to sort two data files:

```
    proc sort data = ah.inhome;by aid;
    proc sort data = ah.contextual;by aid; run;
```

The PROC SORT procedure is a command that directs SAS to order or rank observations within a dataset by the values of a designated variable. In this case, SAS is directed to order all observations by participant ID (BY aid) within each separate dataset. Figure 1 provides an example of how observations in In-home data (with variables I1, I2, I3, and I4) and observations in the Contextual data (with variables X1 & X2) are ordered by participant ID (AID). The next step is to merge the two separate datasets into a singular dataset.

In-Home Data

AID	I1	I2	I3	I4
1				
2				
3				

Contextual Data

AID	X1	X2
1		
2		
3		

Figure 1: Illustration of Two Separate Datafiles Sorted by AID

The MERGE procedure is a command that directs SAS to form a singular data set from multiple datafiles. Our example shows how to merge two files, but, you can merge as many files as necessary. As long as each datafile has been sorted by the common/linking variable first.

SAS code to merge two data sets:

```
        data ah.AH_Combined;
        merge ah.inhome ah.contextual;
        by aid; run;
```

Notice that the MERGE command happens within a data step – that is, before we tell SAS which files to merge together, we must first provide a name for the merged datafile. Above, we have named the merged datafile “Combined” and it will be stored in the AH library. After naming the merged dataset, SAS is being directed to combine (i.e., merge) ah.inhome and ah.contextual into AH_Combined. Notice that the code above has more than just the MERGE command -- the BY command is a critical part of the merging process. If you use the MERGE command without a “BY” statement, you will see an error in the log. Figure 2 shows how the two separate datasets shown in Figure 1 now appear in a single dataset. through the common variable AID (which again is the participant ID in the Add Health data).

Combined Data						
AID	I1	I2	I3	I4	X1	X2
1	░	░	░	░	▓	▓
2	░	░	░	░	▓	▓
3	░	░	░	░	▓	▓

Figure 2. Example of Newly Merged Datafile

After processing the MERGE command, the user should check the log for potential errors and complete an additional PROC CONTENTS and review the output for successful creation of the combined datafile. Output 2 depicts the number of observations and variables for each pre-consolidated dataset as well as the newly created data set. At first, you might have expected to see 2826(32+2794) variables in the combined dataset, however, remember that AID is the same variable included in each of the original two files. Thus, after merging the dataset, AID is consolidated across both datasets into a single variable within AH_Combined (i.e., So, subtract one variable from the total and you arrive at 2825 total variables). Also, notice that the number of observations is consistent across all datasets.

Inhome		Contextual		Combined	
Observations	6504	Observations	6504	Observations	6504
Variables	2794	Variables	32	Variables	2825

Output 2: Partial Output from PROC CONTENTS Used to Check the Number of Variables and Observations after Merging Two Datasets

The last step in creating a working dataset before cleaning data involves creating a dataset that contains only the variables that will be used in the analytic process. As shown above, AH_Combined has nearly 3,000 variables in it. Creating a smaller dataset that only contains variables that will be used in your analytic procedures not only speeds up processing, but also is necessary for some of the data cleaning procedures that are provided later in the paper.

Code for creating a datafile that only contains variables of interest for a study is below:

```
data AH.Combined_reduced_var;
set AH.Combined;
keep AID H1GI1M H1GI1Y BST90P19 imonth iday iyear
      H1ED19 H1ED20 H1ED21 H1ED22 H1ED23 H1ED24 AH_PVT; run;
```

The KEEP command within SAS is an option that allows users to retain selected variables when creating a new dataset. In the above code, the new dataset "ah.combined_reduced_var" is a subset of ah.combined. Instead of containing nearly 3000 variables, ah.combined_reduced_var contains 13 variables (i.e., the variables listed in the KEEP command).

Once the reduced dataset is created, review the log to determine if the new dataset has been created without error. Next, execute PROC CONTENTS on the new dataset to determine if the correct number of variables and observations have been included within the new dataset. The log will also inform you how many variables are retained in the new datafile. This verification process is conducted in the same way as verifying that the MERGE was completed correctly using the PROC CONTENTS function.

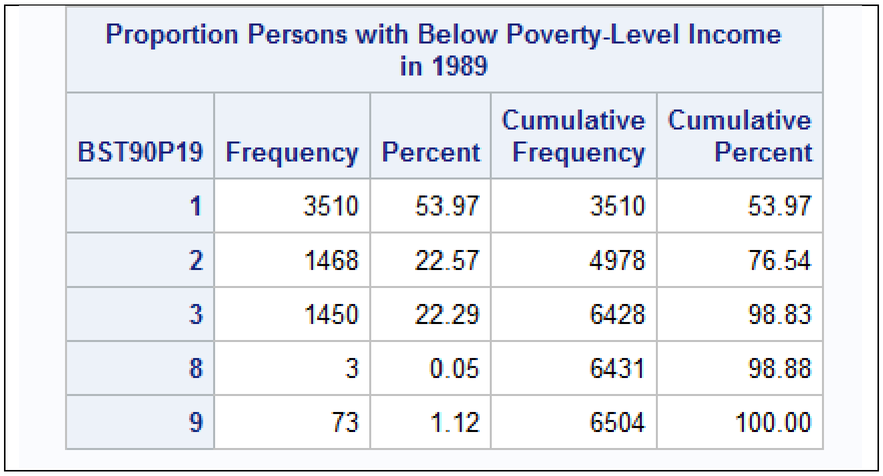
CONVERTING MISSING DATA TO SAS MISSING NOTATION

Encountering missing data is inevitable. Because missing data can be represented in numerous forms across datasets, the first step of converting missing data to a uniform representation for SAS processing is to review the codebook. A sample of items taken directly from the Add Health codebooks is provided as an Appendix at the end of the paper. In reviewing the information, you can see variables H1ED19 - H1ED24 have eight responses; 1-5 represent actual responses to the items (strongly agree to strongly disagree) whereas 6-8 represent different types of missing data (refused, legitimate skip, do not know). After reviewing the codebook, you should estimate frequency tables for each variable that will be used in their analyses – this allows you to compare the data to the information in the codebook and it provides baseline information that you will use throughout the data cleaning and management process to check that procedures execute correctly.

Code for generating frequency tables for each of the variables in the reduced dataset is below:

```
proc freq data = AH.Combined_reduced_var;
tables H1GI1M H1GI1Y BST90P19 imonth iday iyear H1ED19 H1ED20 H1ED21
H1ED22 H1ED23 H1ED24 AH_PVT; run;
```

Output 3 depicts a frequency table for BST90P19. After examining the frequency table and the codebook, we know that variable BST90P19 has five possible responses, in which only responses 1-3 represent observations that will be included within the analytic process. The other two responses (8 unstable estimates and 9 geocode missing) need to be converted into missing data using SAS system notation. You would examine the frequency tables for each variable to determine which values are present in the data and need to be converted to missing.



Proportion Persons with Below Poverty-Level Income in 1989				
BST90P19	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	3510	53.97	3510	53.97
2	1468	22.57	4978	76.54
3	1450	22.29	6428	98.83
8	3	0.05	6431	98.88
9	73	1.12	6504	100.00

Output 3. Frequency Table for BST90P19

There are two methods for converting missing data from numeric responses to ".". The first method uses the IF/THEN function, whereas the second method uses the ARRAY function.

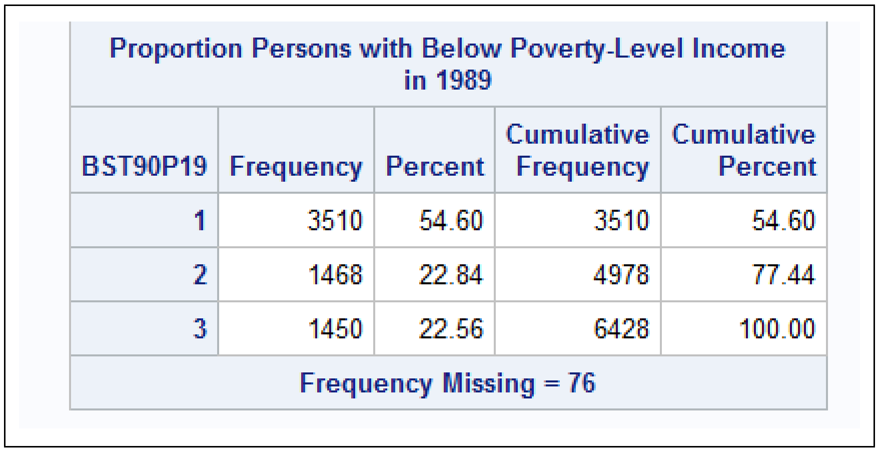
The code for using the IF/THEN function is below:

```
data missing;
set AH.Combined_reduced_var;
keep aid H1GI1M H1GI1Y BST90P19 imonth iday iyear AH_PVT;
If H1GI1M ge 96 then H1GI1M = .;
If H1GI1Y ge 96 then H1GI1Y = .;
If BST90P19 gt 3 then BST90P19 = .; run;
```

In the code above, the first thing to notice is that we created a new dataset that we have called "missing". However, unlike other datasets that we have created, this dataset is not a permanent SAS datafile. Instead, it is a temporary/working dataset named. We know the dataset is not a permanent SAS datafile because it is not linked to the AH libname. As a temporary dataset, it is important to remember that it only exist in SAS's memory – as soon as you close the SAS session you are working in, the temporary datasets are erased. You will see multiple parts of SAS code where temporary datasets are created. It is not required to use temporary datasets – anytime you need to create a new dataset, you can create permanent SAS datafiles, like we have done in the earlier examples. However, if you do that, than your libname location ends up with a lot of datasets that are not used to analyze your data.

The IF/THEN syntax is a function that SAS uses to examine an expression in order to evaluate if the variable meets specified criteria, and if it does, change the observation of the variable to the specified value. In the above code, observations that have values greater than or equal to 96, for variables H1GI1M and H1GI1Y, are converted to ".". The 'ge' portion of the code is SAS notation for greater than or equal to. The 'gt' used with variable BST90P19 is SAS notation for greater than. So, the last line of the code instructs SAS to convert all values greater than 3, for BST90P19, are converted to ".".

Output 4 depicts the frequency table for BST90P19 after the completion of the IF/THEN code above. Here you can see that the 76 observations that were originally coded as 8 and 9 now appear in the "Frequency Missing" at the bottom of the table.



Proportion Persons with Below Poverty-Level Income in 1989				
BST90P19	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	3510	54.60	3510	54.60
2	1468	22.84	4978	77.44
3	1450	22.56	6428	100.00
Frequency Missing = 76				

Output 4. Frequency Table for BST90P19 after Converting Missing Data Values

Unfortunately, when using the IF/THEN statement, each individual variable must be entered on a line of syntax for SAS processing – when you are working with 20 or 30 variables, that is a lot of coding. In those cases, you can write much less code by using the ARRAY function in SAS.

An example of using the ARRAY function is below:

```
data missing2;
set AH.Combined_reduced_var;
keep aid H1ED19 H1ED20 H1ED21 H1ED22 H1ED23 H1ED24;
      array AH [*] _numeric_;
      Do I = 1 to DIM (AH);
      If AH[I] ge 6 then AH[I] = .;
      End;
Drop I; run;
```

In the most simplistic terms, the ARRAY function designates a set of variables to be manipulated – i.e., it is a way to automate the IF/THEN logic explained above. In the code above, after creating a new dataset that will contain the output from the ARRAY (i.e., missing2) the user states ARRAY, followed by a name for the ARRAY.

In the example above, we have named this ARRAY "AH". After designating the name of the array the user directs SAS on how many elements, or variables, should be included within the array. Although the user can list each of the variable names to include in the ARRAY, we have used a more efficient approach. In the example above, after naming the array, the designation [*] _NUMERIC_ informs SAS that there are an unknown number of elements or variables ([*]), and that SAS needs to execute the AH ARRAY on all the numeric variables (_NUMERIC_) contained within the current temporary dataset missing2.

The next subscript, (DO I = 1 TO DIM (AH)), basically directs SAS to process elements from the first element or variable (DO I = 1), to the last element or variable (TO DIM) within the named (AH)array. The "I" in this line ([I]) simply marks the beginning of a loop for SAS, which commands SAS to repeat a process until all designated elements are manipulated. The actual manipulation portion of this array is similar to using IF/THEN statements. As such, the code "IF AH[I] ge 6 THEN AH[I] = ." directs SAS to convert any values greater than or equal to six (AH[I] ge 6), among the numerical variables contained in the AH array, to missing (THEN AH[I] = .). The END command directs SAS that the array is being concluded, and DROP I removes the I from the output generated from executing the AH array. Just as we did after using the IF/THEN approach to creating missing data indicators, if you use the ARRAY approach you need to run frequency tables for all of the variables that were included in the ARRAY to make sure that everything executed correctly.

Note that when using the _NUMERIC_ option it is vital to make sure that the variables in the dataset are all applicable to the requested action. For example, in the AH ARRAY, notice that we also used a KEEP statement and limited the dataset to specific variables. The variables that were not kept in this example were H1GI1M and H1GI1Y – that is because these variables had legitimate values greater than six. So, had we kept them in the dataset that was used for the ARRAY, all values greater than six would have been inappropriately converted to missing. We could have avoided that error by not using the _NUMERIC_ option and instead listing the variable names that were to be included in the ARRAY process.

However, given the way that have split the process for this example, before we move forward, we need to bring the two "missing" datasets back together.

The code for accomplishing this is below:

```
proc sort data = missing;
      by aid;
proc sort data = missing2;
      by aid;
data missing3;
      merge missing missing2;
      by aid; run;
```

As shown previously, before you can merge two datasets, you need to make sure each of them are sorted by the linking variable, which in this example is AID.

Next, in addition to checking the LOG to make sure that the datasets were merged correctly, you can double check the data by printing a few observations. This allows you to visually check the accuracy of the data.

Here, we are printing the first 10 observations [(obs = 10)]:

```
proc print data = missing3 (obs = 10); run;
```

CREATING AGE FROM INTERVIEW DATE AND BIRTHDAY

Often, databases do not include the age of a participant when the data were collected. However, age can be derived from the date the interview was conducted and birthday date of the participant.

The code for creating age from an interview date and birthday is as follows:

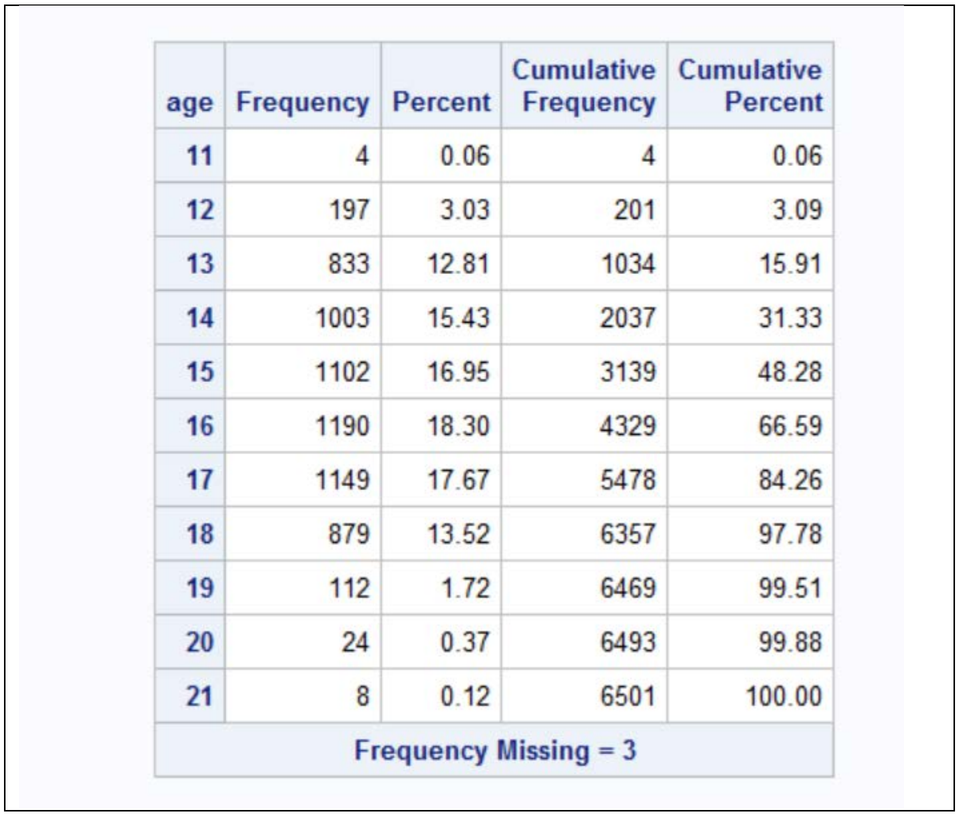
```
options nofmterr;
data clean;
set missing3;
idate=mdy(imonth,iday,iyear);
bdate=mdy(H1GI1M,15,H1GI1Y);
age=int((idate-bdate) / 365.25);
```

In the code above, we have created a temporary dataset, "clean". This is what we commonly use to indicate that data cleaning process is being conducted. First, we create the interview date using three separate variables provided in the Add Health data – imonth, iday, iyear. The MDY portion of the code is a SAS function that converts the values provided in imonth, iday, iyear to a SAS date value, which we have named idate. The participant birthday (bdate) is created similarly to the interview date, with the exception of the actual day of the birthday is not recorded within the available Add Health variables. Therefore, an average day for the month is substituted for an actual day, which is represented by fifteen (15) in this example. After single variables are created for the interview and birth dates, the next step is to create the age variable (age). This is completed by finding the integer (int) of the interview date (idate) minus the birthdate (bdate), which is then divided by the number of days in the year (365.25). We use 365.25 instead of 365 to account for leap years.

To verify that age was completed correctly, the user should check the SAS log and complete a PROC FREQ to examine the age frequencies. Output 5 depicts the age frequency table for this example. The computed age should be verified against the age of participants listed within the codebook or sampling frame within the database documentation.

The code used to generate Output 5 is below:

```
proc freq data = clean;
tables age; run;
```



age	Frequency	Percent	Cumulative Frequency	Cumulative Percent
11	4	0.06	4	0.06
12	197	3.03	201	3.09
13	833	12.81	1034	15.91
14	1003	15.43	2037	31.33
15	1102	16.95	3139	48.28
16	1190	18.30	4329	66.59
17	1149	17.67	5478	84.26
18	879	13.52	6357	97.78
19	112	1.72	6469	99.51
20	24	0.37	6493	99.88
21	8	0.12	6501	100.00

Frequency Missing = 3

Output 5. Frequency Table of the Newly Created Age Variable

RENAMING VARIABLES

Renaming variables within SAS is relatively simple. As in every new data manipulation step, the first task is to review the codebook to better understand what the variable represents. For example, the variable H1ED19 represents the survey item, "You feel close to people at your school." Ideally, the user should rename variables to something that will remind him/her as to what the variable represents. For H1ED19, the variable will be renamed as "close," as the item inquires to how close the participant feels to others within school. Users should repeat this step for every variable that is to be renamed. Obviously, renaming variables is not a required step in cleaning your data, but when working with many variables, having meaningful names is incredibly helpful and more efficient than constantly referring to the codebook to remember what variables represent.

The code for renaming variables is listed below. Please note that in the complete SAS syntax file you will see that the renaming function is executed in the same data step above when we created the age variable.

```
data clean;
………
Rename  BST90P19 = Poverty
H1ED19 = Close
H1ED20 = Belong
H1ED21 = Prejudice
H1ED22 = Happy
H1ED23 = Treat
H1ED24 = Safe
AH_PVT = vocab_score; run;
```

In this code, the RENAME function is used to direct SAS to change the names of the variables as they appear after the RENAME statement. To rename a variable, simply enter the original variable name followed by an equal sign then the new variable name. When entering multiple variables to rename, repeat this process using only white space between variables as shown above. Moving forward, the dataset no longer contains the original variable names. So, you must remember to use the new variables names.

After you have renamed the variables, again it is good practice to PROC CONTENTS (as shown below) to verify that the original variable names are no longer in the dataset and that the new variable names are.

```
proc contents data = clean; run;
```

CREATING DUMMY VARIABLES AND REVERSE CODING VARIABLES

CREATING DUMMY VARIABLES

Two common data cleaning processes are creating dummy variables and reverse coding continuous variables. Categorical variables with more than two response options need to be dummy coded so they can be used in various analytic models. Dummy variables are created by transforming a categorical variable into different variables that are coded 0 and 1. For example, data often contain a variable for respondents' race with response options of African American, American Indian/Alaska Native, Asian, white, and other. Before race could be used in regression model or other predictive analytic procedures, dummy variables would need to be created – that is, each category/response option becomes its own variable. This process can easily be accomplished using IF/THEN syntax.

For our example, we will create dummy variables from the original "poverty" variable, where poverty represents the proportion of persons living below the poverty level, where 1 = low, 2 = medium, and 3 = high. Again, we know what the variable "poverty" represents by consulting the Contextual Data Codebook.

The code for creating the dummy variables is as below.

```
data clean2;
set clean;
high_prop_in_poverty = .;
med_prop_in_poverty = .;
low_prop_in_poverty = .;
      If Poverty = 1 then low_prop_in_poverty = 1;
      If Poverty = 2 then low_prop_in_poverty = 0;
      If Poverty = 3 then low_prop_in_poverty = 0;

      If Poverty = 1 then med_prop_in_poverty = 0;
      If Poverty = 2 then med_prop_in_poverty = 1;
      If Poverty = 3 then med_prop_in_poverty = 0;

      If Poverty = 1 then high_prop_in_poverty = 0;
      If Poverty = 2 then high_prop_in_poverty = 0;
      If Poverty = 3 then high_prop_in_poverty = 1; run;

      R_Prejudice = (6 - Prejudice);run;
```

Of the variables we have included in the paper, poverty (BST90P19) is the only true categorical variable. As such, using the variable "poverty", participants are assigned to one of three new variables, based on the value they have for the original variable. Before you can assign respondents to the appropriate dummy variables, the first step is to initialize (aka create) each of the dummy variables. That is the first part of the code above where we have written "high_prop_in_poverty = .; med_prop_in_poverty = .; low_prop_in_poverty = .;". This code creates a new variable (i.e., a new column in your dataset) and assigns all participants as missing on each of the three variables. Then, we use IF/THEN statements to assign respondents to the appropriate dummy variable. In the original variable, a value of 1 meant that the neighborhood had a low proportion of residents living below the poverty level. So, in the code above, you can see that those who had a value of 1 for the original variable poverty now have a value of 1 for the new variable, low_prop_in_poverty. And zeros for the other two dummy variables. Those who had a value of 2 on the original variable are assigned a value of 1 for med_prop_in_poverty and 0s for the other dummy variables. And those who had a 3 are assigned a 1 for high_prop_in_poverty and 0s for the low and medium poverty dummy variables. For those that were missing on the original variable, poverty, they will be missing on each of the dummy variables as well. That is because they did not meet the criteria stated in any of the IF/THEN statements so the assigned value of missing, when the dummy variables were created is still in place.

Reverse coding is another common data cleaning technique. However, it is only necessary if individual items will be used to create composite variables (more on how to create composite variables in the next section). After identifying the items that will be used to create a composite variable, an analyst can determine if a variable needs to be reverse coded based on the wording of survey items and corresponding response options. For example, the original variables H1ED19 - H1ED24 all ask about adolescents' thoughts on different aspects of their school (e.g., feel safe at school, teachers treat students fairly, etc.). In reviewing each of these items (as shown in the Appendix), one can notice an overall theme for these variables -- lower values on most items represent positive connections to school and higher values represent negative connections to school. This applies to all but one of the items -- H1ED21 (which we previously renamed to Prejudice). This item asks students "Students at your school are prejudiced". A response of 1 (strongly agree) to this item reflects a negative aspect of the school and a response of 5 (strongly disagree) reflects a positive aspect of the school. Thus, in order for this item to be considered for inclusion in an overall school connectedness composite variable, the response options need to reflect the same sentiments as the other items – lower values represent positive school connection and higher values represent negative school connection. This can be accomplished by reverse coding.

The code for reverse coding is above, just after the dummy coding -- R_Prejudice = (6 – Prejudice). If more than one variable needs to be reverse coded, each line of code should be concluded with a semicolon.

When reverse coding a variable, R_ is often added to the variable name to indicate that the variable has been reversed. In order to reverse code an item, simply count the number of available responses in the original variable, add one, and then subtract the original variable. In this case, prejudice had five available responses (1 Strongly Agree, 2 Agree, 3 Neither, 4 Disagree, and 5 Strongly Disagree), so 5 +1 = 6. This is where "R_Prejudice = (6 – Prejudice)." came from. Through this process, those that originally responded 1 to this survey item (i.e., they strongly agreed that students in their school were prejudice) now have a 5. Thus, response values to this item are now in the "same direction" as the other items related to school connectedness (i.e., a value of 5 for other items indicates a student does not feel safe at school or does not feel that teachers treat students fairly).

After creating dummy variables and reverse coding variables, you want to check and make sure you're your logic was correct and that people were assigned to the correct dummy variable and that the reverse coded variable is indeed reverse coded.

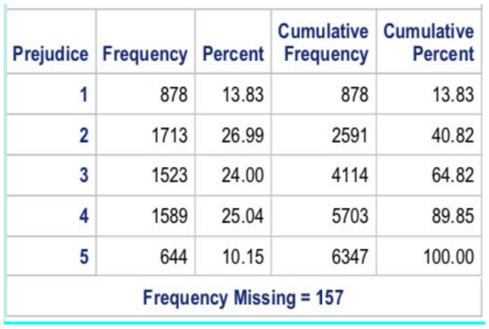
You can do this by generating frequency tables for the original variables and the newly created variables. See the code below:

```
proc freq data = clean2;
tables poverty high_prop_in_poverty
med_prop_in_poverty
low_prop_in_poverty
R_Prejudice Prejudice; run;
```

To conserve space, we only provide the output that shows the frequency tables for prejudice and R_prejudice (see Output 6). As you can see, the frequencies of responses were indeed reverse coded. The original variable had 644 participants indicate that they "strongly disagreed" with item (i.e., responses with value of 5) and in the reverse coded variable, those 644 responses are now coded as 1.

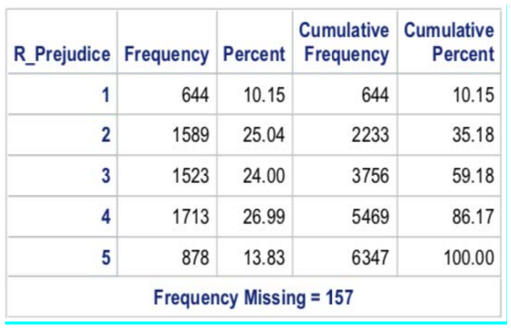
Also, notice that the above code was written as part of a data step. We created clean2 from clean. Often many data cleaning processes can be done in one very long data step, but for clarity and teaching purposes, we have broken them into a few separate data steps.

Table Prejudice



Prejudice	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	878	13.83	878	13.83
2	1713	26.99	2591	40.82
3	1523	24.00	4114	64.82
4	1589	25.04	5703	89.85
5	644	10.15	6347	100.00
Frequency Missing = 157				

Table R_Prejudice



R_Prejudice	Frequency	Percent	Cumulative Frequency	Cumulative Percent
1	644	10.15	644	10.15
2	1589	25.04	2233	35.18
3	1523	24.00	3756	59.18
4	1713	26.99	5469	86.17
5	878	13.83	6347	100.00
Frequency Missing = 157				

Output 6: PROC FREQ Output for Prejudice and R_Prejudice

CREATING COMPOSITE VARIABLES

A composite variable is a variable that is made up of two or more items, measured at the interval or ratio level, that are statistically related to each other. The creation of a composite variable is often formed to measure multidimensional concepts that are not easily observed. For example, Add Health researchers were interested in assessing adolescents' feelings of school connectedness, however, school connectedness is not a concept that is easily observed or asked about. So, instead of having one survey item that asked participants how connected he/she felt to his/her school, Add Health researchers asked a series of items that all have the underlying concept of school connectedness. We will now use the six different items related to school connectedness to create an overall measure of school connectedness. There are multiple ways that composite variables can be derived – the approach we share below is one most commonly used in the social and behavioral sciences.

EXPLORATORY FACTOR ANALYSIS

The first step to creating a composite variable is to conduct an exploratory factor analysis (EFA) on the items of interest. In the most basic terminology, EFA is a method of data reduction that seeks to identify underlying unobservable variables (i.e., school connectedness) that are reflected in observed/measured variables. Using the correlation matrix of the variables included in the EFA, this procedure identifies how closely the items being considered for the composite variable "hang" together. There are many papers published on conducting EFA in SAS (e.g., Goldberg, 1997; O'Rourke & Hatcher, 2014; Santos, Lippke, & Pope, 1998). We provide a brief overview below, however, users are encouraged to consult other resources as well for more details on the EFA process, including recommended sample sizes, etc.

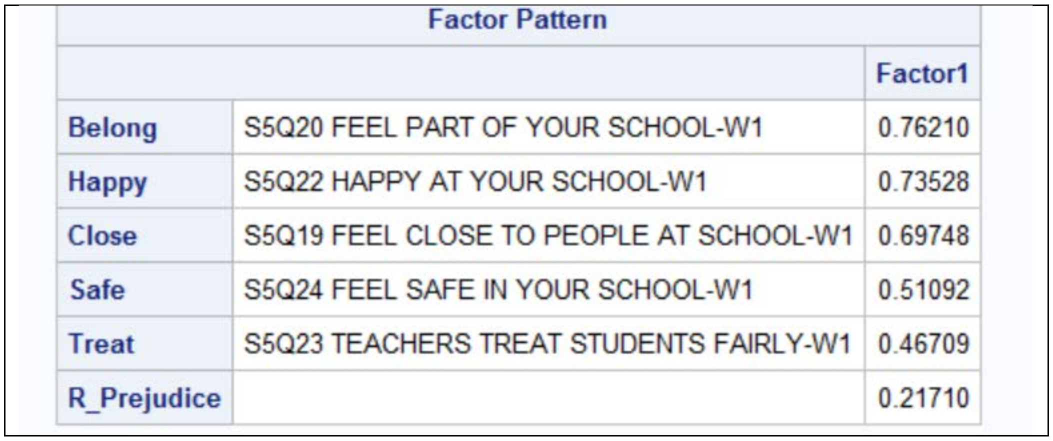
Code for completing an EFA is as follows:

```
proc factor data = clean2 method = ml priors=SMC scree mineigen=1
reorder rotate=varimax;
var Close Belong R_Prejudice Happy Treat Safe; run;
```

The syntax listed after PROC FACTOR can be rather complicated given all the options available, however explanations will be given for the those used within this example. The first option (DATA = clean2) should be familiar to the user and directs SAS to use variables within the temporary clean2 dataset. The syntax METHOD = ML specifies that the maximum likelihood of estimation should be used during statistical processing. Next, PRIORS = SMC specifies which type of communal measurement is to be used while conducting the EFA, and SMC specifically designates that the communal estimate to be used should be equal to the squared multiple correlation of each variable to all the other variables. The SCREE syntax simply directs SAS to include a scree plot within the output, and the next two commands MINEIGEN = 1 and REORDER directs SAS to use the smallest eigenvalue in which a single factor can be retained, and then to reorder the output to list the variables with the highest factor loadings first. The option of ROTATE = VARIMAX directs SAS to rotate the axes of the factors so that the results are more interpretable, while concurrently assuming none of the factors are correlated. After writing the complete PROC FACTOR syntax line, the user directs SAS to include selected variables by using VAR and adding the variables to be included in the EFA.

Deciding how many factors to include within a final product of a composite variable can be confusing, as there are multiple methods to determine the number of factors. One recommendation is to include factors with eigenvalues ≥ 1.0. The next method is to review the scree plot in the output and determine where factors tend to level off along the x-axis. Another method is to review the percentage of variance explained by each factor and determine what an acceptable percentage of contribution toward variance should be. The last technique is to review the factor loading for each variable and drop any item with factor loadings less than 0.4. Decisions are often made using a combination of the above approaches. For this, relatively simple example, we will focus on the last technique, reviewing factor loadings for variables with loadings of 0.4 or more.

Output 7 depicts the factor loadings for the items being considered for the composite variable, school connectedness. Notice only one item, R_Prejudice has a loading < 0.4. Based on these results, C_R_Prejudice should not be included within the composite variable.



Factor Pattern

		Factor1
Belong	S5Q20 FEEL PART OF YOUR SCHOOL-W1	0.76210
Happy	S5Q22 HAPPY AT YOUR SCHOOL-W1	0.73528
Close	S5Q19 FEEL CLOSE TO PEOPLE AT SCHOOL-W1	0.69748
Safe	S5Q24 FEEL SAFE IN YOUR SCHOOL-W1	0.51092
Treat	S5Q23 TEACHERS TREAT STUDENTS FAIRLY-W1	0.46709
R_Prejudice		0.21710

Output 7: Factor Loadings for Items to use for School Closeness

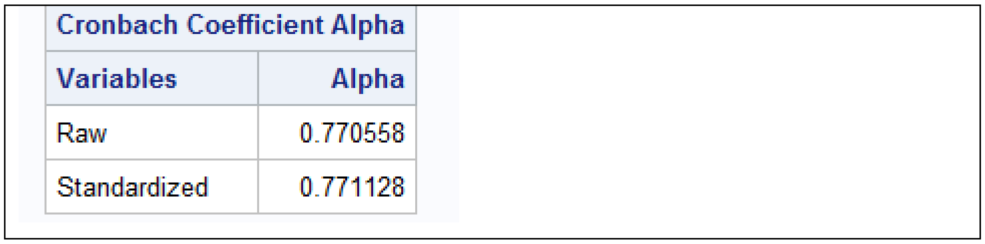
INTERNAL CONSISTENCY

After determining which items “hang” together and should be used to create a composite variable, the internal consistency of the composite variable should also be examined. The most common method for assessing this is by calculating Cronbach’s alpha, a coefficient of reliability. This helps determine how closely related a set of items are as a group. An acceptable level of internal consistency is typically α ≥ 0.70.

Code for calculating Cronbach’s alpha is below:

```
proc corr data = clean2 nosimple nomiss alpha;
var Close Belong Happy Treat Safe; run;
```

In the code above, Cronbach's alpha is calculated through PROC CORR by listing the "ALPHA" option. The NOSIMPLE option suppresses the descriptive statistics that PROC CORR generates and the NOMISS option instructs SAS to calculate Cronbach's alpha using only those with complete data on the variables listed on the VAR statement. Notice, that the variables included on the VAR statement are those that had factor loadings of 0.4 or greater in the EFA results. As shown in Output 8, these five items exhibit an acceptable level of internal consistency (α = .77). Note that you want to interpret the Raw portion of the output as it is the measure that is based upon item covariances, thus the stronger the items are inter-related the more likely construct, school connectedness, is consistent across participants. Often the Raw and Standardized will be similar but that is not always the case. Standardized alpha values are rarely used because most data do not meet the assumption that all items have the same variances. More details on Cronbach's alpha can be found in Yu (2001) & O'Rourke and Hatcher (2014).



Cronbach Coefficient Alpha	
Variables	**Alpha**
Raw	0.770558
Standardized	0.771128

Output 8. Partial Coefficient Alpha Output from PROC CORR

After completing the EFA and calculating Cronbach's alpha, the last step to creating a composite variable is to create a new variable in SAS. In this example, we want to create a new variable, School Connectedness, by taking the mean of the five items that were found to hang well together – Close, Belong, Happy, Treat, and Safe.

Code to create the composite variable is listed below:

```
data clean3;
set clean2;
School_Connectedness = mean (of Close Belong Happy Treat Safe); run;
```

After creating a new temporary dataset "clean3", the code above instructs SAS to create the variable "School_Connectedness" using the MEAN function in SAS By including "of" before the list of variables to be used in creating School_Connectedness, we are instructing SAS to calculate the mean for each person using only the items for which a person has observed data. For example, a person did not answer the item about being happy at their school, the new variable, School_Connectedness will be calculated by taking the mean of close, belong, treat, and safe.   Thus, using the "of" option allows users to create the composite variable without requiring participants to have complete data on all items used in the composite variable.

To verify that School_Connectedness was created correctly, the user should check the SAS log for any errors and also examine the descriptive statistics for the new variable, using PROC MEANS. Because running PROC MEANS is also required to center variables, we present this in more detail in the next section.

CENTERING VARIABLES

After cleaning the data and creating any necessary variables, if you plan to run predictive models (e.g., regression models) the next step in managing the data is to determine if the continuous variables have meaningful zeros. This means that for each variable that will be used as a predictor, you need to assess if zero is a plausible value and if so, is it meaningful? For continuous variables that do not have meaningful zeros, you should center those variables, thus, creating meaning for the value zero. This is important because interpretation of regression coefficients is more practical and meaningful when the variable has a meaningful zero. In our example dataset, it is plausible that we would use school connectedness, age, and the dummy variables for neighborhood poverty as predictors on adolescent achievement on a standardized vocabulary measure (vocab_score). Centering variables is a two-step process. First, you have to estimate the mean for each variable that needs to be centered, then, you use that information to create a centered version of the variable.

The SAS code for both steps is below:

```
proc means data = clean3;
var School_Connectedness age;run;

data clean4;
set clean3;
C_School_Connectedness = School_Connectedness - 2.2952418;
C_age = age - 15.5329949; run;
```

In the code above, you can see that first we created a new dataset "clean4". We had to do this because creating the centered variables has to occur in a DATA step. Second, when naming the centered variables that you are creating, it is common practice to add "C_" to the beginning of the variable name. This is a way to easily communicate that the variable has been centered. Now that the centered variables have been created, you would use C_School_Connectednes and C_Age in your predictive models. If you wanted to check to make sure that the centering code was executed correctly, you can run PROC MEANS on the new centered variables – if the centering code worked than the mean of each variable will be close to zero.

At this point you might be wondering why we did not center the other predictors – high_prop_in_poverty, med_prop_in_poverty, and low_prop_in_poverty? We did not center them because those variables are not continuous and they inherently already have a meaningful zero. Also, again, centering variables is only necessary if the variables in your dataset will be used in any type of predictive modeling. If the variables will not be used in predictive modeling, you can skip this step. Also, it is important to note that centering applies to predictor variables and not criterion variables.

CREATING AN ANALYTIC DATASET

The last step in getting your raw messy data to being ready to analyze is to create an analytic dataset (i.e., a dataset that contains those with complete data that you will use to estimate your statistical procedures). This can be done one of two ways - listwise deletion or multiple imputation. Given the complexity for correctly using either method, we do not cover either option in depth. However, at the end of this section, we do provide code that can be used to create an analytic dataset using listwise deletion. However, if you use the code without examining the missing data mechanisms, you can end up with biased results. Given the details that need to be covered to correctly handle missing data, we are writing a complete paper that addresses both of these options. Until then, remember that regardless of the method you use to handle missing data, an often-overlooked step in the process is examining the missing data mechanism – both listwise deletion and multiple imputation can yield bias results if the data do not meet the assumption of missing at random (MAR). We encourage readers to read about options for handling missing data and to educate themselves before deciding which approach to use. And, again, stay tuned for a new SAS paper that explores both listwise deletion and multiple imputation from an applied researchers' perspective.

Below is code that can be used to delete observations that do not have complete data:

```
data ah.complete_data; set clean4;
if NMISS (of aid age school_connectedness poverty vocab_score) = 0;
keep aid age c_age school_connectedness C_school_connectedness poverty
high_prop_in_poverty  med_prop_in_poverty low_prop_in_poverty
vocab_score vocab_score; run;
```

A few things to notice about the code above. First, you will notice the NMISS function – this tells SAS to include observations in the new dataset "ah.complete_data" if the sum of missingness for the items aid, age, school connectedness, poverty, and vocab_scores = 0. If an observation is missing on any of those five items they will not be included in the analytic dataset. The next part of the code above instructs SAS which variables to keep in the analytic dataset. You might notice that we are keeping more variables than we used in the NMISS function. Why did we do this?? You should see that the variables in the keep statement include the original variables (which are used in the NMISS function) as well as the manipulated versions of the original variables (i.e., c_age, c_school_connectedness). We keep both the original variables and the created variables because you need the variables in their original format to generate descriptive statistics and the created variables for your inferential statistical procedures.

CONCLUSION

We wrote this paper to help synthesize common steps that applied researchers encounter when conducting quantitative research. We consider this the first paper in a series that we plan to write over the next few years. Our hope is that novice SAS users find the content of the paper useful and that it helps users feel less intimidated when approaching quantitative research. We have posted added the complete SAS code as an appendix in this paper. We will also share the actual SAS syntax file to the Conference Repository (https://github.com/sascommunities/sas-global-forum-2020). And of course, if you have trouble accessing the full syntax, please reach out to either author listed below to obtain a copy of the file. Although the code in this paper mirrors the code in the complete file, I believe that it is best to view both so you can see how the steps flow.

REFERENCES

Berglund, P.A. (2009). Getting the Most out of the SAS® Survey Procedures: Repeated Replication Methods, Subpopulation Analysis, and Missing Data Options in SAS® v9.2. *SAS Global Forum 2009 Proceedings*. http://support.sas.com/resources/papers/proceedings09/246-2009.pdf

D'Agostino McGowan, L. & Toll, A. (2015). Using PROC SURVEYREG and PROC SURVEYLOGISTIC to Assess Potential Bias. *SAS Global Forum 2015 Proceedings*. https://support.sas.com/resources/papers/proceedings15/3320-2015.pdf

Goldberg, R.J. (1997). PROC FACTOR: How to Interpret the Output of a Real-World Example. *SouthEastern SAS Users Group 1997 Proceedings*. https://www.lexjansen.com/sesug/1997/SESUG97086.pdf

Lewis, T. (2010). Principles of Proper Inferences from Complex Survey Data. *SAS Global Forum 2010 Proceedings*. http://support.sas.com/resources/papers/proceedings10/266-2010.pdf

Matlapudi, A. & Knapp, J. (2012). SAS® system generates code for you while using Import/Export Procedure. *NorthEast SAS Users Group* 2012 Proceedings. https://www.lexjansen.com/nesug/nesug12/cc/cc20.pdf

O'Rourke, N. & Hatcher, L. (2014). Chapter 3: Assessing Scale Reliability and Cronbach's Alpha in *A step-by-step approach to using SAS for factor analysis and structural equation modeling* (2nd edition, pp. 97-106). SAS Institute.

Santos, J.R.A., Lippke, L., & Pope, P. (1998). PROC FACTOR: A Tool for Extracting Hidden Gems from a Mountain of Variables. *23rd SAS Users Group International Proceedings*. https://support.sas.com/resources/papers/proceedings/proceedings/sugi23/Stats/p240.pdf

Yu, (2001). An introduction to computing and interpreting Cronbach Coefficient Alpha in SAS. *26th SAS Users Group International Proceedings*. https://support.sas.com/resources/papers/proceedings/proceedings/sugi26/p246-26.pdf

CONTACT INFORMATION

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

Bethany A. Bell, PhD, MPH
University of South Carolina
babell@sc.edu

Jason A. Schoeneberger, PhD
ICF International
jschoeneberger@outlook.com

APPENDIX OF VARIABLES

Variable from the Wave I: Public Use Contextual Database Codebook			
Survey Item	Variable Name	Response	Frequency
Proportion persons with income in 1989 below poverty level (for persons for whom poverty status is determined)1	BST90P19	1 low 2 medium 3 high 8 unstable estimates 9 geocode missing	3510 1468 1450 3 73
Partial List of Variables from Wave I: In-Home Questionnaire, Public Use Sample Codebook			
19. (How much do you agree or disagree with the following statements:) [If SCHOOL YEAR:] You feel close to people at your school. [If SUMMER:] Last year, you felt close to people at your school.	H1ED19	1 Strongly Agree 2 Agree 3 Neither 4 Disagree 5 Strongly Disagree 6 Refused 7 Legitimate Skip 8 Don't Know	1273 3025 1214 627 226 4 128 6
20. (How much do you agree or disagree with the following:) [If SCHOOL YEAR:] You feel like you are part of your school. [If SUMMER:] Last year, you felt like you were part of your school.	H1ED20	1 Strongly Agree 2 Agree 3 Neither 4 Disagree 5 Strongly Disagree 6 Refused 7 Legitimate Skip 8 Don't Know	1677 3039 881 564 205 4 128 6
21. (How much do you agree or disagree with the following:) [If SCHOOL YEAR:] Students at your school are prejudiced. [If SUMMER:] Last year, the students at your school were prejudiced.	H1ED21	1 Strongly Agree 2 Agree 3 Neither 4 Disagree 5 Strongly Disagree 6 Refused 7 Legitimate Skip 8 Don't Know	878 1713 1523 1589 644 4 128 25
22. (How much do you agree or disagree with the following:) [If SCHOOL YEAR:] You are happy to be at your school. [If SUMMER:] Last year, you were happy to be at your school.	H1ED22	1 Strongly Agree 2 Agree 3 Neither 4 Disagree 5 Strongly Disagree 6 Refused 7 Legitimate Skip 8 Don't Know	1574 2629 1080 706 376 5 128 6
23. (How much do you agree or disagree with the following:) [If SCHOOL YEAR:] The teachers at your school treat students fairly. [If SUMMER:] Last year, the teachers at your school treated	H1ED23	1 Strongly Agree 2 Agree 3 Neither 4 Disagree 5 Strongly Disagree 6 Refused	1015 2660 1395 971 326 4

students fairly.		7 Legitimate Skip 8 Don’t Know	128 5
24. (How much do you agree or disagree with the following:) [If SCHOOL YEAR:] You feel safe in your school. [If SUMMER:] Last year, you felt safe in your school.	H1ED24	1 Strongly Agree 2 Agree 3 Neither 4 Disagree 5 Strongly Disagree 6 Refused 7 Legitimate Skip 8 Don’t Know	1622 2908 1044 577 216 4 128 5

APPENDIX OF SAS SYNTAX

```
Title 'Code for the GETTING DATA INTO SAS AND CREATING A WORKING DATAFILE
section of the SGF paper';

*Code to import SAS C-Transport file -- first for importing the Add Health
In-Home Questionnaire Data*;
filename in "C:\Users\bellb\Documents\SESUG/21600-0001-Data.stc";
libname out 'C:\Users\bellb\Documents\SESUG';
proc cimport file=in library=out;run;

*C-Transport import code for the Add Health Contextual Data*;
filename in "C:\Users\bellb\Documents\SESUG/21600-0002-Data.stc";
libname out 'C:\Users\bellb\Documents\SESUG';
proc cimport file=in library=out; run;

*The code below creates a new version of the in-home and contextual data,
using a more meaningful name -- inhome and contextual*;
libname AH 'C:\Users\bellb\Documents\SESUG';
options nofmterr;
data ah.inhome;set ah.da21600p1;
data ah.contextual;set ah.da21600p2; run;

*The code below is ueed to see the details of the datasets -- variable names,
sample size, etc.*;
proc contents data = AH.contextual;
proc contents data = AH.inhome;run;

*Code to merge the in-home and contextual datasets into one dataset.
First you sort the data then you merge the data*;
proc sort data = AH.contextual;
      by aid;
proc sort data = AH.inhome;
      by aid; run;

data AH.Combined;
      merge AH.contextual AH.inhome;
      by aid; run;
*again, checking the contents of the newly merged dataset -- good practice
      to always examine the details of newly created datasets*;
proc contents data = AH.Combined; run;

**Creating a smaller data set that only contains variables that are needed
for the study*;
data AH.Combined_reduced_var;
      set AH.Combined;
  keep AID H1GI1M H1GI1Y BST90P19
            imonth iday iyear
            H1ED19 H1ED20 H1ED21 H1ED22 H1ED23 H1ED24
            H1GI1M H1GI1Y AH_PVT; run;

*the AH.Combined dataset now becomes the master dataset. At this point,
the earlier datasets are no longer needed*;

proc contents data = AH.Combined_reduced_var; run;
```

```
Title 'Code for the MISSING DATA section of the SGFpaper';

*Running freq on all variables to examine nature of the data that will need
to be cleaned. For example, the freq tables will show all values for each
variable -- then in consultation with the Add Health codebook, you can
determine what values need to be changed to represent missing --
i.e., those that refused to answer, did not know, question was not
applicable, etc.*;

proc freq data = AH.Combined_reduced_var;
      tables
            H1GI1M H1GI1Y BST90P19
            imonth iday iyear
            H1ED19 H1ED20 H1ED21 H1ED22 H1ED23 H1ED24
            H1GI1M H1GI1Y AH_PVT; run;

*Here we create a temporary dataset and then start cleaning the data.
The data set missing is temporary because it does not have a libname attached
to it. First, we convert response values that represent missing data to SAS
missing notation.
We know what values to convert based on information in the codebooks*;

*Part 1 of converting data to missing when necessary*;

data missing;
      set AH.Combined_reduced_var;
keep aid H1GI1M H1GI1Y BST90P19 imonth iday iyear AH_PVT;
      If H1GI1M ge 96 then H1GI1M = .;
      If H1GI1Y ge 96 then H1GI1Y = .;
      If BST90P19 gt 3 the BST90P19 = .; run;

**estimating freq tables to make sure that the code above executed
correctly*;
proc freq data = missing;
tables H1GI1M H1GI1Y BST90P19; run;

*part 2 of converting data to missing when necessary - details provided in
the paper;

data missing2;
      set AH.Combined_reduced_var;
      keep aid H1ED19 H1ED20 H1ED21 H1ED22 H1ED23 H1ED24;
            array AH [*] _numeric_;
            Do I = 1 to DIM (AH);
            If AH[I] ge 6 then AH[I] = .;
            End; Drop I; run;

*checking to make sure that the code above in the array worked and the
correct values were replaced with .*;
proc freq data = missing2;
      tables _numeric_; run;

*now you need to bring the two missing data sets back together into one
dataset*;
proc sort data = missing;
      by aid;
proc sort data = missing2;
      by aid;
```

```
data missing3;
      merge missing missing2;
      by aid; run;
*in addition to checking the log to make sure that data sets were merged
correctly, you can double check the data by printing a few observations --
this allows you to visually check the accuracy of the data. Below we are
printing the first 10 observations (obs = 10);
proc print data = missing3 (obs = 10); run;

Title 'Code for the CREATING AGE FROM INTERVIEW DATE AND BIRTHDAY and
RENAMING VARIABLES
section of the SGF paper';

*the code below contains information on two data cleaning steps - creating an
age variable from interview date and birthdate and how to rename variables*;

options nofmterr;
data clean;
set missing3;
idate=mdy(imonth,iday,iyear);
bdate=mdy(H1GI1M,15,H1GI1Y);
age=int((idate-bdate) / 365.25);

Rename  BST90P19 = Poverty
H1ED19 = Close
H1ED20 = Belong
H1ED21 = Prejudice
H1ED22 = Happy
H1ED23 = Treat
H1ED24 = Safe
AH_PVT = vocab_score; run;

**now, again, check that the code above worked correctly.
PROC CONTENTS to make sure the variables now show the new names and PROC FREQ
to make sure that the age variable was created*;
proc contents data = clean; run;

proc freq data = clean;
table age; run;


Title 'Code for the CREATING DUMMY VARIABLES and REVERSE CODING section of
the SGF paper';

*Code below shows two data cleaning steps -- how to create dummy
variables and how to reverse code data*;
data clean2;
set clean;
high_prop_in_poverty = .;
med_prop_in_poverty = .;
low_prop_in_poverty = .;
            If Poverty = 1 then low_prop_in_poverty = 1;
            If Poverty = 2 then low_prop_in_poverty = 0;
            If Poverty = 3 then low_prop_in_poverty = 0;

            If Poverty = 1 then med_prop_in_poverty = 0;
            If Poverty = 2 then med_prop_in_poverty = 1;
            If Poverty = 3 then med_prop_in_poverty = 0;

            If Poverty = 1 then high_prop_in_poverty = 0;
            If Poverty = 2 then high_prop_in_poverty = 0;
            If Poverty = 3 then high_prop_in_poverty = 1;
R_Prejudice = (6 - Prejudice); run;
**checking to make sure the steps above executed correctly -- that
the dummy variables are correct and that the reverse coding worked correctly;

proc freq data = clean2;
tables high_prop_in_poverty
med_prop_in_poverty
low_prop_in_poverty
R_Prejudice Prejudice; run;

Title 'Code for the CREATING COMPOSITE VARIABLES section of the SGF paper';

*EFA code to determine if there is a latent variable "school connectedness" -
see paper for more details*;
proc factor data = clean2 method = ml priors=SMC scree mineigen=1 reorder
rotate=varimax;
      var
            Close
            Belong
            R_Prejudice
            Happy
            Treat
            Safe; run;

*checking internal consistency of the latent "school connectedness" variable
based on results from EFA above*;
proc corr data = clean2 nosimple nomiss alpha;
      var Close
            Belong
            Happy
            Treat
            Safe; run;

data clean3;
      set clean2;
            School_Connectedness = mean (of
                  Close
                  Belong
                  Happy
                  Treat
                  Safe); run;

Title 'Code for the CENTERING VARIABLES section of the SGF paper';
*generating the mean of the new composite variable -- the mean will be used
to center the variable in the next step in the code*;
proc means data = clean3;
      var School_Connectedness age; run;

data clean4;
set clean3;
C_School_Connectedness = School_Connectedness - 2.2952418;
C_age = age - 15.5329949; run;
```

```
proc contents data = clean4;
run;

Title 'Code for the CREATING ANALYTIC DATASET section of the SGF paper';

*the code below created a new data set that only includes observations with
no missing data.  this is a quick way to get an analytic dataset created,
however, as stated in the paper, other steps should also be taken.;

data ah.complete_data; set clean4;
if NMISS (of aid age school_connectedness poverty vocab_score) = 0;
keep aid age c_age school_connectedness C_school_connectedness poverty
   high_prop_in_poverty  med_prop_in_poverty low_prop_in_poverty vocab_score
   vocab_score; run;
```