Paper IT-06

# Review That You Can Do: A Guide for Systematic Review of Complex Data

Lesa Caves, RTI International, Durham, NC
Nicole Williams, RTI International, Durham, NC

## ABSTRACT

Quality control is a critical step in the process of creating and reviewing composite variables. Review of a single composite variable typically requires several iterations of multi-way crosstabs and case-level review in order to verify that the variable is programmed according to the analyst's specifications. This approach is suitable when working with simple data structures (e.g., a single dataset or multiple datasets with the same number of records per file) or when the variable is simple to program. However, when a composite variable is created from complex, multi-level data structures, it requires special care in review and quality control procedures. Analysts, with content expertise but basic SAS® programming skills, may find it difficult to adequately review the variable. In this paper, we describe a process for effectively and systematically reviewing a composite variable created from several multi-level datasets. Through this process, a programmer creates a composite variable in few data steps for efficiency, while an analyst methodically breaks the code down into multiple small data steps to create a local version of the same variable. The programmer's and analyst's versions of the variable are then compared and discrepancies are investigated.

## INTRODUCTION

When cleaning and analyzing data, one of the first things the analyst must do is determine what new variables must be created to facilitate their research. Creating composite variables (i.e. a variable that is created by combining multiple variables into one variable) allows the user to control the path of their research. In some cases, analysts must rely on the assistance of a programmer to create complex composite variables they would not otherwise be able to create. When working with programmers to create composite variables, the analyst must find a way to review the newly created variable.

This paper details a method that can be used by a programmer and analyst to program and methodically review complex composite variables. The programmer has the expert SAS skills while the analyst has the content expertise. Together, the team can create a composite variable with confidence.

## SIMPLE COMPOSITE VARIABLE REVIEW

The complexity of reviewing a composite variable depends greatly on the data structures from which it is created. A simple data structure, such as a single dataset or multiple datasets with the same number of records (i.e. on the same level), generally lends itself to simple data review because the review can be completed with a few merges and outputs. In addition, identifying discrepancies and their causes is easier because the analyst is able to output crosstabs or prints, and review each value of the composite variable to ensure proper values. As an example of a composite variable created from a simple data structure, we can look at the derivation of a composite *AGE* variable that calculates age as of January 1, 2012.

```
data a;
length AGE 8. year 4 mon $2;
input ID DOB $;

year = SUBSTR(DOB,1,4);
mon = SUBSTR(DOB,5,2);

if mon = "01" then
AGE = (2012 - year);
else
AGE = (2012 - year) - 1;

datalines;
    1      198001
    2      197205
    3      198312
    4      197909
    5      198807
    6      199006
```

```
      7      197205
      8      196811
      9      197206
      10     197503
;
run;
```

The date of birth input, *DOB*, comes from one dataset that is one record per ID. It contains the month and year of birth in YYYYMM format. A crosstab of the original *DOB* value and the derived *AGE* value can be output.

```
proc freq data = a;
tables DOB * AGE / list missing;
run;
```

Using a single crosstab, the output is sufficient for a complete review of *AGE*. The input of *DOB* and the derived *AGE* values can be compared for accuracy side by side. The analyst can go through the output, line by line, and confirm that *AGE* is calculated as intended for each case. This method of data review by multi-way crosstabs is appropriate for other composite variables that are not complex in data structure and inputs.

The FREQ Procedure

| DOB | AGE | Frequency | Percent | Cumulative Frequency | Cumulative Percent |
|---|---|---|---|---|---|
| 196811 | 43 | 1 | 10.00 | 1 | 10.00 |
| 197205 | 39 | 2 | 20.00 | 3 | 30.00 |
| 197206 | 39 | 1 | 10.00 | 4 | 40.00 |
| 197503 | 36 | 1 | 10.00 | 5 | 50.00 |
| 197909 | 32 | 1 | 10.00 | 6 | 60.00 |
| 198001 | 32 | 1 | 10.00 | 7 | 70.00 |
| 198312 | 28 | 1 | 10.00 | 8 | 80.00 |
| 198807 | 23 | 1 | 10.00 | 9 | 90.00 |
| 199006 | 21 | 1 | 10.00 | 10 | 100.00 |

**Output 1. Output from PROC FREQ Statement**

## COMPLEX TRANSCRIPT DATA

The need to create a systematic review process grew from a complex academic transcript study. The data for this study were extracted from more than 25,000 academic transcripts collected for approximately 17,000 postsecondary students. The transcripts were entered into an electronic database that translated the data contained on the transcript into a series of variables in several data sets. Several different datasets were created to accommodate the different types and unique nature of the data. There were five datasets created from the initial transcript data, and three additional datasets created with composite variables only. The datasets created are shown in Table 1 and described below.

| File | Example variables/sections | Number of records | Number of students | Average Number per student |
|---|---|---|---|---|
| Transcript | Cumulative credits earned, Cumulative GPA, Clock hours | 25,110 | 16,960 | 1.5 |
| Courses | Course name Normalized grade, Normalized credits, Honors | 636,300 | 16,900 | 37.6 |
| Institution | IPEDS ID[1], level, control, EIN number[2], Carnegie classification | 3,070 | -- | -- |
| Degree | Major, Minor, Degree date, Honors | 24,300 | 16,270 | 1.5 |
| Terms | Term start date, Term end date, Term name, Credits attempted during term | 149,460 | 16,900 | 8.8 |

| | | | | |
|---|---|---|---|---|
| Student/Schools | GPA, Number of terms enrolled, Credits earned by subject, Units needed for an award at the institution | 26,930 | 16,960 | 1.6 |
| Derived | Transcript Totals, Pre-College Information, Enrollment and Attendance, Coursework Across Institutions, and Awards | 16,960 | 16,960 | 1.0 |
| Transfer | Percentage of credits transferred, Degree program at origin and destination institution, Selectivity at origin and destination institution | 13,660 | 6,770 | 2.0 |

**Table 1. Datasets created from transcripts.**

[1] IPEDS ID is a unique identification number assigned to postsecondary institutions surveyed through the Integrated Postsecondary Education Data System (IPEDS).

[2] EIN (Employment Identification Number) number is the number assigned to an institution by the Internal Revenue Service for tax purposes.

NOTE: Numbers included here have been rounded to the nearest tenth.

- **Transcript.** The transcript dataset included items such as cumulative credits earned, cumulative grade point average (GPA), and clock hours. In addition, this dataset included the student's high school graduation date, if it was listed on the transcript. Each transcript received represented one record in the dataset. There were 25,110 records on the transcript dataset. On average, each student had approximately 1.5 records.
- **Courses.** The courses dataset included items describing each course listed on the transcript. The variables on this dataset included term start date, term end date, course name, course number, course code (based on a coding scheme designed for this study), normalized grades, normalized credits, honors, and course attributes. Each course listed on the transcript represented one record in the dataset. There were 636,300 records on the courses dataset. On average, each student had approximately 37.6 records.
- **Institution.** The institution dataset included variables defining each postsecondary institution in the study. These included IPEDS ID, level, control, EIN number, Carnegie classification, and latitude and longitude coordinates. This was the only dataset that did not contain a student ID number or a transcript ID number. Each institution in the study represented one record on this dataset.
- **Degrees.** The degree dataset included items detailing each degree included on the transcript. The variables on this dataset included major, minor, degree date, honors, and degree type. Each degree listed on the transcript represented one record in the dataset. There were 24,300 records on the degree dataset. On average, each student had approximately 1.5 records.
- **Terms.** The terms dataset included information about each term during which the student was enrolled, including term start date, term end date, term name, credits attempted during the term, credits earned during the term, term GPA, and term honors. Each term listed on the transcript represented one record in the dataset. There were 149,460 records on the term dataset. On average, each student had approximately 8.8 records.

In addition to the five datasets above that were primarily created from "raw" transcript variables, three additional datasets were created to accommodate composite variables and provide users with more versatility for analysis.

- **Student/Schools.** The student/schools dataset was created to review the relationships between students and each school they attended. Every student was paired with each school they attended, creating one record for each pair in the dataset. Variables included grade point average (GPA), number of terms enrolled, credits earned by subject, and units needed for an award at the institution. There were 26,930 records on the student/schools dataset. On average, each student had approximately 1.6 records.
- **Derived.** The derived dataset contained student-level variables. Variables were divided into 5 categories: Transcript Totals, Pre-College Information, Enrollment and Attendance, Coursework Across Institutions, and Awards. This dataset contained one record per student.
- **Transfer.** The transfer dataset contained variables describing transfer patterns among students who attended multiple postsecondary institutions. The variables in this dataset included percentage of credits transferred, degree program at origin and destination institution, and selectivity at origin and destination institution, and enrollment dates for each enrollment period at origin and destination institution. Approximately 6,680 students were included in transfer dataset. Each transfer opportunity represented one record in the dataset. There were 13,660 records on the transfer dataset. On average, each student included in this dataset had approximately 2 records.

More than 500 composite variables were created by pulling variables from the datasets detailed above. The review process described below was developed to systematically and methodically review each variable with confidence.

## COMPLEX COMPOSITE VARIABLE REVIEW

This section describes the composite variable review process in 6 steps. This process is shown in Figure 1 below.

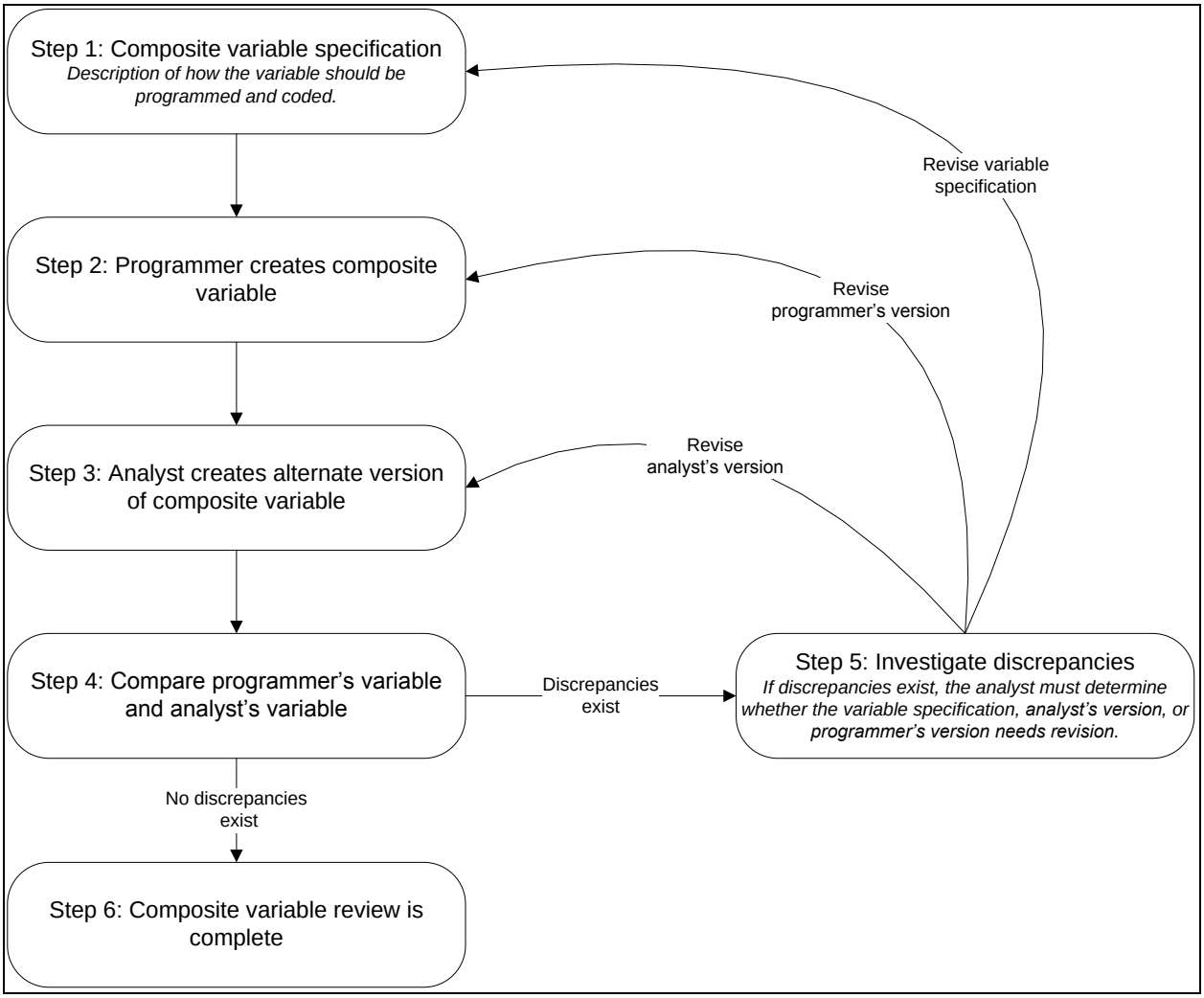


**Figure 1. Flowchart of composite variable review process.**

- **Step 1:** Composite variable specification: The analyst creates the composite variable specification, or "specs". These specs are typically written in pseudo-code and detail exactly how the variable should be programmed.
- **Step 2:** Programmer creates composite variable: The programmer creates the composite variable based on the composite variable specs.
- **Step 3:** Analyst creates an alternate version of composite variable: The analyst creates an alternate version of the variable after the programmer has created the composite variable. Whether the analyst has basic or advanced SAS skills, the analyst should create the alternate version independent of the programmer's version of the variable.

- **Step 4:** Compare programmer's variable and analyst's variable: The programmer's variable and the analyst's variable are compared to determine if any discrepancies are present.
- **Step 5:** Investigate discrepancies: If discrepancies are found in step 4, the analyst and programmer work together to resolve any discrepancies. The resolution of discrepancies can result in one of three actions: revise the variable specifications, revise the analyst's code, or revise the programmer's code.
- **Step 6:** Composite variable review is complete: The review and revision continue until the analyst and programmer are in agreement on the variable values and no discrepancies exist.

The process is illustrated below, using a composite variable, *EARNBA*, from the transcript data which calculates the elapsed time from a student's entry into a postsecondary education to earning a Bachelor's degree.

## SAMPLE DATA

For the purposes of this paper, sample input datasets are provided here. There are three datasets needed for *EARNBA*: *cases*, *degrees*, and *first_attended*.

```
data cases ;
input ID ;
datalines;
1
2
3
4
5
6
7
8
9
10
;
run ;


data degrees ;
input ID INSTID DEGPROGRAM DEGMY DEGREC ;
datalines;
1   1      2      200905 1
3   1      1      200812 1
3   2      2      200605 1
4   6      1      -9     -9
7   6      3      200705 1
7   3      1      200904 1
7   1      7      201112 1
10  2      1      200808 1
;
run ;

data first_attended  ;
input ID firstAttendedMY ;
datalines ;
1   200708
2   200609
3   200401
4   200308
5   200808
6   200307
7   200508
8   200708
9   200409
10  200409
;
```

```
run ;
```

## STEP 1: COMPOSITE VARIABLE SPECIFICATION

Before any programming begins, the analyst must create the variable specification or "spec" in order to communicate the requirements of the composite variable to the programmer. Specs for the composite variable consist of pseudo-code, expected values, source datasets (noted in parentheses), and any special notes that could be useful to the programmer. For the transcript study, the analyst enters variable specs in an application which allows all users to keep track of the progress of the composite variable. A comment is entered in the application when the variable is ready for various stages of the review process. The programmer and analyst can track progress by viewing these comments.

Once specs for *EARNBA* are entered in the application by the analyst (Figure 2), the variable is ready for initial programming.

```
EARNBA = Number of months from FIRSTATTENDEDMY to earliest DEGMY(degrees)
where DEGPROGRAM (degrees) = 1 and DEGREC (degrees) = 1;

/*Set negative values*/
If FIRSTATTENDEDMY or DEGMY (-6 -9), then EARNBA = -9.
If EARNBA is negative, then EARNBA = -6.
```

**Figure 2. Variable specifications for EARNBA**

## STEP 2: PROGRAMMER CREATES COMPOSITE VARIABLE

Once the analyst completes the specifications, or "specs" for the composite variable, the programmer can begin to create the variable. According to the specs, the programmer must use *DEGMY*, *DEGPROGRAM*, and *DEGREC* from the *degrees* dataset and *FIRSTATTENDEDMY* from the `first_attended` dataset, *FIRSTATTENDEDMY* is another composite variable that was derived earlier in the programming process.

| Variable | Source | Label |
|---|---|---|
| ID | Degree, Derived | Student case ID |
| DEGMY | Degree | Date of degree |
| DEGPROGRAM | Degree | Degree program (1 = Bachelor's) |
| DEGREC | Degree | Degree received |
| FIRSTATTENDEDMY | Derived | First attended ever month/year |

**Table 2: Variables and descriptions**

### Create datasets

The programmer creates two datasets, each containing the input variables of interest: degree date and first attendance date. The code below creates two datasets: *FIRSTATTENDEDMY* and *prog_a*. *Prog_a* contains all the cases (*ID*) that earned a Bachelor's degree. *DEGMY* is recoded from a -9 (missing) to 999999 so that missing date values will fall to the end of the sorting process.

```
data prog_a ;
set degrees;
/* Recode reserve codes */
if DEGMY = -9 then DEGMY = 999999;
if DEGREC = 1 and DEGPROGRAM = 1 ;
run ;
```

### Find the earliest degree date

Next, the programmer sorts the *prog_a* dataset by case and degree date (*ID* and *DEGMY*) then adds a data step that keeps the first record for each case, that is, the earliest degree date awarded over all Bachelor's degrees.

```
proc sort data = prog_a ;
by ID DEGMY ;
```

```
data prog_a;
set prog_a (keep = ID DEGMY) ;
by ID DEGMY ;
if first.ID then output;
run ;
```

**Calculate difference between two dates**

Now the programmer merges the two datasets containing the dates, *first_attended* and *prog_a* into a single dataset with one record per case from which the programmer can compute the difference between the two dates. This creates separate month, day, and year variables to use as input into the 'INTCK' function to compute the difference. The last two lines of code deal with reserve codes for missing data.

```
data prog_b (keep = ID EARNBA );
merge first_attended (in=in1) prog_a (in=in2) ;
by ID  ;
if in1 and in2; /* only cases that have degrees and a date to compare start */

/* reset recode since done with sorting */
if DEGMY = 999999 then DEGMY = -9;

/* Convert begin and end to mdy format */
length BEG END $ 6 ;
array orig FIRSTATTENDEDMY DEGMY;
array chvar BEG END ;
array mdyvar BEG2 END2 ;
array varYR begYR degYR ;
array varMN begMN degMN ;
array varDY begDY degDY ;

do over orig;
if orig=. then orig=-9;
end;

do over orig;
chvar =orig;
varYR = substr(chvar,1,4);
varMN = substr(chvar,5,2);
varDY = '01';
if varMN=-9 then varMN=.;
mdyvar = mdy(varMN, varDY, varYR);
end;

/* Number of full months between dates */
length EARNBA 8 ;
EARNBA = intck('month', BEG2, END2) - (day(END2) < day(BEG2));

if EARNBA < 0 then EARNBA = -6 ;
if FIRSTATTENDEDMY in (-9 -6) or DEGMY in (-9 -6) then EARNBA = -9 ;
run;
```

**Set missing values codes**

The last sort and datastep sets missing values (-9) to cases that do not have a Bachelor's degree in the degrees dataset, but are in the study.

```
proc sort data = cases ;
by ID ;

data prog_c ;
```

```
merge prog_b (in=in1) cases (in=in2);
by ID ;
if not in1 and in2 then EARNBA = -9 ;
run ;
```

The programmer now has a working version of the composite variable and the analyst can begin the next step in the review process.

## STEP 3: ANALYST CREATES ALTERNATE VERSION OF COMPOSITE VARIABLE

Once the programmer completes the composite variable, the analyst programs an alternate version of the variable using more basic SAS code than the programmer's version. The analyst methodically breaks the specs down into multiple, manageable data steps. While this may seem more cumbersome than the programmer's composite variable code, it allows the analyst to better identify issues when discrepancies are found in the review process. Using this process, the analyst can better identify potential holes or issues in the variable specifications.

### Create Bachelor's degree indicator

Initially, the analyst creates a dataset, named *analyst_a*, which creates a bachelor's degree indicator and formats the missing values for sorting.

```
*Create BA Indicator;
data analyst_a (keep = ID DEGMY DEGPROGRAM DEGREC BA);
set degrees;

*To create the bachelor's degree indicator;
if DEGPROGRAM = 1 and DEGREC = 1 then BA = 1;
else BA = 0;

*To get the missing values;
if DEGMY = -9 then DEGMY = 999999;
run;
```

### Identify earliest Bachelor's degree

Next, the analyst identifies which Bachelor's degree is the earliest for each student.

```
*Create a variable for earliest BA;
data analyst_b;
set analyst_a;
where BA = 1;
run;

proc sort data = analyst_b;
by ID DEGMY;
run;

*This is identifying the earliest BA degree;
data analyst_c(rename = (DEGMY=first_BA));
set analyst_b;
by ID DEGMY;
if first.ID then output;
run;
```

### Create Bachelor's flag

Then the analyst creates a flag to indicate whether the student received a Bachelor's degree. During this process, the analyst merges the *analyst_e* dataset with the *cases* dataset to add all *ID*s to ensure that all students will have a value for the final composite variable.

```
*Create a BA flag;
proc sort data = analyst_a out = analyst_d (keep = ID BA) nodupkey;
by ID BA;
run;
```

```
data analyst_e;
set analyst_d;
by ID BA;
if last.ID then output;
run;

proc sort data = cases out = cases_all (keep = ID) nodupkey;
by ID;
run;

data analyst_f;
merge analyst_e (in=in1) cases_all (in=in2);
by ID;
if in2;
if BA = . then BA = 0;
run;
```

**Merge with date first attended postsecondary education**

Next the analyst merges the *analyst_c* and *analyst_f* datasets with the previously created composite variable *FIRSTATTENDEDMY*, date first attended any postsecondary institution.

```
*Merge with date first attended Postsecondary Education (FIRSTATTENDEDMY);
proc sort data = analyst_c;
by ID;
run;

data FIRSTATTENDEDMY (keep = ID FIRSTATTENDEDMY);
set first_attended;
if FIRSTATTENDEDMY = -9 then FIRSTATTENDEDMY = 999999;
run;

proc sort data = analyst_f;
by ID;
run;

proc sort data = FIRSTATTENDEDMY;
by ID;
run;

data analyst_g;
merge FIRSTATTENDEDMY analyst_c analyst_f;
by ID;
run;
```

**Create Analyst Version**

The analyst can now calculate a version of *EARNBA*, *ANALYST_VERSION*, using the datasets and flags created in the steps above. Both first attended date and degree date variables are presented in YYYYMM format, so they must first be broken into year and month variables in order to calculate the elapsed time. Two interim variables are created in this step, *START_MON_JAN03* and *BA_MON_JAN03*. The elapsed time between the student's start date and bachelor's degree award date is calculated by subtracting *START_MON_JAN03* from *BA_MON_JAN03*.

```
*Create Analyst version;
data analyst_h;
set analyst_g;

if BA = 0 then TEMP1 = -3;
else if FIRSTATTENDEDMY = 999999 then TEMP1 = -9;
else if FIRST_BA = 999999 then TEMP1 = -9;

if FIRSTATTENDEDMY = 999999 then START_MONTH = .;
else START_MONTH = substr(put(FIRSTATTENDEDMY, 6.), 5, 2);
```

```
if FIRSTATTENDEDMY = 999999 then START_YEAR = .;
else START_YEAR = substr(put(FIRSTATTENDEDMY, 6.), 1, 4);

if FIRST_BA = 999999 then BA_MONTH = .;
else BA_MONTH = substr(put(FIRST_BA, 6.), 5, 2);

if FIRST_BA = 999999 then BA_year = .;
else BA_YEAR = substr(put(FIRST_BA, 6.), 1, 4) ;
run;

data analyst_i;
set analyst_h;

if START_YEAR = . then START_MON_JAN03 = .;
else START_MON_JAN03 =((START_YEAR - 2003) * 12) + START_MONTH;

if BA_YEAR = . then BA_MON_JAN03 = .;
else BA_MON_JAN03 = ((BA_YEAR - 2003) * 12) + BA_MONTH;
run;

data analyst_j(drop = start_month start_year BA_month BA_year);
set analyst_i;

if TEMP1 = -3 then ANALYST_VERSION = -3;
else if TEMP1 = -6 then ANALYST_VERSION = -6;
else if TEMP1 = -9 then ANALYST_VERSION = -9;
else if START_MON_JAN03 gt BA_MON_JAN03 then ANALYST_VERSION = -6;
else ANALYST_VERSION = BA_MON_JAN03-START_MON_JAN03;

run;
```

**Add Programmer's Version**

Now that the analyst's alternate version of the composite variable, *ANALYST_VERSION*, is created, the analyst merges the programmer's version of *EARNBA* onto the analyst dataset to allow for comparisons.

```
*Add programmer's version;
proc sort data = prog_c;
by ID;
run;

proc sort data = analyst_j ;
by ID;
run;

data prog_analyst_compare (drop = temp1);
merge prog_c (in=in1) analyst_j (in=in2);
by ID;
if EARNBA = ANALYST_VERSION then DIFFERENCE = 0;
else DIFFERENCE = 1;
run;
```

## STEP 4: COMPARE PROGRAMMER'S VARIABLE AND ANALYST'S VARIABLES

After the analyst has created their alternate version of the variable, the two versions are compared. The best way to check this is to look at a crosstab of both versions where differences are present.

```
proc freq data = prog_analyst_compare;
tables EARNBA*ANALYST_VERSION/list missing;
where DIFFERENCE = 1;
run;
```

The FREQ Procedure

| EARNBA | ANALYST_VERSION | Frequency | Percent | Cumulative Frequency | Cumulative Percent |
|---|---|---|---|---|---|
| -9 | -3 | 7 | 100.00 | 7 | 100.00 |

**Output 2. Output from PROC FREQ Statement showing 5 discrepancies between the programmer's version and the analyst's version.**

## STEP 5: INVESTIGATE DISCREPANCIES

If the analyst finds discrepancies between the programmer's and analyst's versions of the variable, then they must investigate the nature of the discrepancy. The best place to look first is in the analyst's code for the alternate version of the composite variable. The detailed, multi-step approach lends itself well to investigating discrepancies. By reviewing the discrepancies within the analyst's code, the analyst is able to find potential problems in their own code or isolate the nature of the problem in the programmer's variable. First, find a case where the values disagree.

```
proc print data = prog_analyst_compare (obs = 1);
where DIFFERENCE = 1;
var ID EARNBA ANALYST_VERSION FIRSTATTENDEDMY BA DEGPROGRAM FIRST_BA DEGREC;
run;
```

| ID | EARNBA | ANALYST_ VERSION | first Attended MY | BA | DEGPROGRAM | first_BA | DEGREC |
|---|---|---|---|---|---|---|---|
| 1 | -9 | -3 | 200708 | 0 | . | . | . |

**Output 3. Output from PROC PRINT Statement showing case-level detail for one case that has discrepant values between the programmer's version (EARNBA) and the analyst's version.**

Investigate the discrepant case in each dataset used in the analyst's data steps to ensure that the data are doing what is expected from one data step to the next.

```
proc print data = analyst_a ;
title 'analyst_a';
where ID = 1;

proc print data = analyst_b;
title 'analyst_b';
where ID = 1;

proc print data = analyst_c;
title 'analyst_c';
where ID = 1;

proc print data = analyst_d;
title 'analyst_d';
where ID = 1;

proc print data = analyst_e;
title 'analyst_e';
where ID = 1;

proc print data = cases_all ;
title 'cases_all';
where ID = 1;

proc print data = analyst_f;
title 'analyst_f';
```

```
where ID = 1;

proc print data = FIRSTATTENDEDMY ;
title 'FIRSTATTENDEDMY';
where ID = 1;

proc print data = analyst_g;
title 'analyst_g';
where ID = 1;

proc print data = analyst_h;
title 'analyst_h';
where ID = 1;

proc print data =analyst_i;
title 'analyst_i';
where ID = 1;

proc print data = analyst_j;
title 'analyst_j';
where ID = 1;

proc print data = prog_c ;
title 'prog_c';
where ID = 1;

proc print data = prog_analyst_compare;
title 'prog_analyst_compare';
where ID = 1;
run;
```

'analyst_a'

| Obs | ID | DEGPROGRAM | DEGMY | DEGREC | BA |
|---|---|---|---|---|---|
| 1 | 1 | 2 | 200905 | 1 | 0 |

'analyst_d'

| Obs | ID | BA |
|---|---|---|
| 1 | 1 | 0 |

'analyst_e'

| Obs | ID | BA |
|---|---|---|
| 1 | 1 | 0 |

'cases_all'

| Obs | ID |
|---|---|
| 1 | 1 |

'analyst_f'

| Obs | ID | BA |
|---|---|---|
| 1 | 1 | 0 |

'FIRSTATTENDEDMY'

| Obs | ID | first Attended MY |
|---|---|---|
| 1 | 1 | 200708 |

'analyst_g'

| Obs | ID | first Attended MY | DEGPROGRAM | first_BA | DEGREC | BA |
|---|---|---|---|---|---|---|
| 1 | 1 | 200708 | . | . | . | 0 |

'analyst_h'

| Obs | ID | first Attended MY | DEGPROGRAM | first_BA | DEGREC | BA | TEMP1 | START_ MONTH | START_ YEAR | BA_MONTH | BA_year |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 200708 | . | . | . | 0 | -3 | 8 | 2007 | . | . |

'analyst_i'

| Obs | ID | first Attended MY | DEGPROGRAM | first_BA | DEGREC | BA | TEMP1 | START_ MONTH | START_ YEAR | BA_MONTH | BA_year | START_ MON_JAN03 | BA_MON_ JAN03 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 200708 | . | . | . | 0 | -3 | 8 | 2007 | . | . | 56 | . |

'analyst_j'

| Obs | ID | first Attended MY | DEGPROGRAM | first_BA | DEGREC | BA | TEMP1 | START_ MON_JAN03 | BA_MON_ JAN03 | ANALYST_ VERSION |
|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 200708 | . | . | . | 0 | -3 | 56 | . | -3 |

'prog_c'

| Obs | ID | EARNBA |
|---|---|---|
| 1 | 1 | -9 |

'prog_analyst_compare'

| Obs | ID | EARNBA | ANALYST_ VERSION | DIFFERENCE | first Attended MY | DEGPROGRAM | first_BA | DEGREC | BA | START_ MON_JAN03 | BA_MON_ JAN03 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | -9 | -3 | 1 | 200708 | . | . | . | 0 | 56 | . |

**Output 4 . Output generated when reviewing case ID=1 in each dataset.**

In the process of reviewing this one specific ID in each dataset, the analyst is able to see how the case is coded at each step and identify where potential problems are occurring with the spec or programming of the variable. If an error is found in the variable spec, return to Step 1 of this review process. If an error is discovered in the analyst's

version, return to Step 3 of the review process and revise the code. If the analyst version of the variable is correct, the discrepancy is most likely occurring in the programmer's code. After going through these steps, the analyst can see that the discrepancy shown in output 2 (*in prog_analyst_compare*) is caused by an additional condition needed in the specs. The analyst must modify the specs to add the line highlighted below.

```
EARNBA = Number of months from FIRSTATTENDEDMY to earliest DEGMY(degrees)
 where DEGPROGRAM (degrees) = 1 and DEGREC = 1;

 /*Set negative values*/
 If FIRSTATTENDEDMY or DEGMY (-6 -9), then EARNBA = -9.
 If DEGPROGRAM never 1 or (DEGPROGRAM = 1 and DEGREC = 0), then EARNBA = -3.
 If EARNBA is negative, then EARNBA = -6.
```

**Figure 3. Modified specs for EARNBA**

Next, the programmer adds the new condition to the code. In this case, the code in the "Set missing values codes" step needs a change. Instead of setting *EARNBA* to a -9, the variable will be set to a -3.

```
data prog_c ;
merge prog_b (in=in1) cases (in=in2);
by ID ;
if not in1 and in2 then EARNBA = -3 ;
run ;
```

## CONDUCT REVIEW AGAIN

After the programmer has made changes to the composite variable, the variable is ready to be reviewed again by the analyst. The analyst re-runs their program that creates the alternate version of the variable. If discrepancies remain, the analyst should perform the investigation steps ("Investigate Discrepancies") again and discuss any remaining discrepancies with the programmer.

## STEP 7: COMPOSITE VARIABLE REVIEW IS COMPLETE

When no discrepancies remain, there will not be any output, only this message in the log:

```
2079  proc print data = prog_analyst_compare (obs = 1);
2080  where DIFFERENCE = 1;
2081  var ID EARNBA ANALYST_VERSION FIRSTATTENDEDMY BA DEGPROGRAM FIRST_BA DEGREC;
2082  run;

NOTE: No observations were selected from data set WORK.PROG_ANALYST_COMPARE.
NOTE: There were 0 observations read from the data set WORK.PROG_ANALYST_COMPARE.
      WHERE DIFFERENCE=1;
NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

**Figure 4. Display of the log when there are no discrepancies between the programmer's version and the analyst's version.**

## CONCLUSION

Data review is a critical process in the creation of composite variables. It ensures that the variable values are created as intended and that the programming is correct. Even with complex datasets, the analyst can methodically review composite variables easily by following the steps detailed above. Breaking the variable down into small and manageable steps gives the analyst a way to trace through the creation of the composite variable and systematically review the variable specification, input values, code, and resulting values. This process can be applied to any data review task, whether it be composite variables, internally used variables, or dataset creation.

## ACKNOWLEDGMENTS

The authors would like to thank the following coworkers for their generous assistance in the creation of this paper: Melissa Cominole, Kristin Dudley, Annaliza Nunnery, John Riccobono, Jim Rogers, Milorad Stojanovic, Nicole Tate, and Jennifer Wine.

## CONTACT INFORMATION

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

Lesa Caves
RTI International
3040 Cornwallis Rd
Durham, NC 27709
919-990-8312
919-541-7014
lcaves@rti.org

Nicole Williams
RTI International
3040 Cornwallis Rd
Durham, NC 27709
919-541-1245
919-541-7014
njwilliams@rti.org