

A Five-Step Quality Control Method: Checking for Unintended Changes to SAS® Datasets

Aaron Brown, South Carolina Department of Education

ABSTRACT

A SAS® programmer may need to make many edits or changes to a given data set. There is always a risk that one's code will have unintended consequences, leading to unintended changes. This paper describes a five-step quality control method that allows a programmer to quickly and systematically check for any changes in the number of records, variables in a dataset, or values in a dataset, in order to assure that the intended changes and only the intended changes occurred. This quality control method utilizes the COMPARE and MEANS procedures.

INTRODUCTION

This paper describes a five-step quality control method, for use when are modifying a data set that already exists. It can be used to confirm that all the intended changes occurred and that no unintended changes occurred. This method is particularly helpful if you have to make several modifications to a dataset, since you can use it with each modification to check that the code is working properly.

The five steps are:

1. Make a safe copy of your original data set.
2. Make the desired changes.
3. Clean-up by deleting any unnecessary variables.
4. Check that only the intended changes were made.
5. Overwrite your original dataset.

The main assumption for this method is that you have a data set and that it is already sorted in the order you want. A secondary assumption is that you have enough disk space to make temporary copies of your dataset. (When I use this method, it is generally with datasets containing 20,000 to 300,000 student records.)

As an example, we will use the dataset below (named *mydata*), sorted alphabetically by last name, and use the five-step method to create an Initials variable from the FName and LName variables. For example, John Smith would have the initials "JS."

Table 1 displays the sample data that will be used in this paper.

FName	LName
Jane	Doe
Jake	Long
John	Smith

Table 1. Contents of the *MyData* Data Set

THE FIVE-STEP METHOD: STEP 1

Step 1: make a safe copy of your data set.

Create a data set called *safecopy*, based on the original data set. Add a counter to the dataset. This data set is not otherwise modified. It is used again in Step 4.

You can also add a title to describe the change you are making.

```
*Step 1;
TITLE 'Adding in Initials';
DATA safecopy;
    SET mydata;
    ctr=_n_; *counter;
RUN;
```

THE FIVE-STEP METHOD: STEP 2

Step 2: make your changes.

Create a data set called *newdataset*, using *safecopy*. We create a new data set instead of the original to avoid overwriting anything until quality control is finished. (If your dataset is large enough that disc space is a concern, you may be tempted to overwrite the original instead. Be sure that you will be able to recreate the original in case you discover your changes have unintended results.)

Perform all necessary changes to the data using *newdataset*. This could include creating new variables, merging in values from another data set, cleaning dirty data, or dropping unnecessary records or variables. You may need to use multiple DATA steps, and you may want to use tools like the FREQ or PRINT procedure to check your work as you go. In the example below, we create the Initials variable and then, using PROC FREQ, perform a check that it was created properly.

```
*Step 2;
DATA newdataset;
    SET safecopy;

    /*code to modify the data set goes below this line*/
    LENGTH FName1 $1. LName1 $1. Initials $2.;

    FName1=SUBSTR(Fname,1,1);
    LName1=SUBSTR(LName,1,1);

    Initials=FName1 || LName1;
RUN;

PROC FREQ DATA=newdataset;
    TABLES FName1*LName1*Initials/LIST MISSING;
    TITLE2 'Optional Step: Checking Creation of Variables';
RUN;
```

THE FIVE-STEP METHOD: STEP 3

Step 3: clean-up your data set.

This step begins when *newdataset* has all the desired changes. Drop any variables that were created in Step Two that should not be kept in the final output, with the exception of the counter.

```
*Step 3;
DATA newdataset;
    SET newdataset;
    DROP FName1 LName1; *do not drop ctr;
RUN;
```

THE FIVE-STEP METHOD: STEP 4

Step 4: check that intended changes and only intended changes occur.

This step is accomplished by performing the following sub-steps:

- A) Resort by the counter, to undo any sorting that may have occurred in Step 2.
- B) Create a new counter.
- C) Compare the number of records in *safecopy* and *newdataset* via the MEANS procedure.
- D) Compare the variables and values in *safecopy* and *newdataset* via the COMPARE procedure.

Some of these steps are not always necessary. In our example, we did not sort the records so PROC SORT is not needed, nor did the number of records change, so the resetting of the counter and the use of PROC MEANS is also unneeded. However, these are included as standard code for cases when such does matter, such as cases where you need to delete records or merge data sets.

```

*Step 4;

*Sub-step A;
PROC SORT DATA=newdataset; BY ctr; RUN; *maintains original sort order;
*Sub-step B;
DATA newdataset; SET newdataset; newCtr=_n_; RUN; *create new counter;
*Sub-step C;
TITLE2 'Comparing Number of Records';
PROC MEANS DATA=safecopy MAX; VAR ctr; TITLE3 'Original Data Set'; RUN;
PROC MEANS DATA=newdataset MAX; VAR newCtr; TITLE3 'New Data Set'; RUN;
*Sub-step D;
TITLE2 'Comparison of Original and New Data Set';
PROC COMPARE DATA=safecopy COMPARE=newdataset LISTVAR;
    ID ctr;
RUN;

```

Using the MAX option in PROC MEANS causes it to only output the highest numeric value for a variable. Having a VAR statement identifying our counter causes PROC MEANS to output this value only for the counter. We can use these two PROC MEANS to easily check if the number of records either did not change or changed by the expected amount.

PROC COMPARE generates a thorough comparison of the data sets. Including the LISTVAR option causes the procedure to list any variables in one data set but not in the other. This output is used to check that only intended variables were kept or dropped. In our example, it tells us that *newdataset* contains the Initials and the newCtr variable, as shown in Output 1 below. (You should always see newCtr as a new variable. It is dropped in Step 5.)

Listing of Variables in WORK.NEWDATASET but not in WORK.SAFECOPY

Variable	Type	Length
Initials	Char	2
newCtr	Num	8

Output 1. PROC COMPARE: Comparison of Variables in the Data Sets

PROC COMPARE also states if there were any changes to values in a given record. Check that your intended changes did occur and that no unintended changes occurred. If there are no changes, you will see the message shown in Output 2.

NOTE: No unequal values were found. All values compared are exactly equal.

Output 2. PROC COMPARE: No Changes to Values

If there are changes, the procedure will state which variables changed (under the Variable column) and how many records had changes (under the Ndif column). It will then print the changes. For example, if we had accidentally set LName to blank, we would see the message displayed in Output 3.

The COMPARE Procedure					
Comparison of WORK.SAFECOPY with WORK.NEWDATASET					
(Method=EXACT)					
Variables with Unequal Values					
Variable	Type	Len	Ndif	MaxDif	MissDif
LName	CHAR	10	3		3

Value Comparison Results for Variables			
ctr	Base Value	Compare Value	
	LName	LName	
1	Doe		
2	Long		
3	Smith		

Output 3. PROC COMPARE: Details on Changes to Values

PROC COMPARE also tells you the number of records in each data set, but this information is easy to overlook amidst the other information this procedure generates. That is why PROC MEANS was used to check number of records.

THE FIVE-STEP METHOD: STEP 5

Step 5: save your changes.

This step should only be run once everything in Step 4 looks appropriate. If there were any unexpected changes revealed by Step 4, go back and fix the code used in Step 2 or change what variables are dropped in Step 3.

Once everything is clean, overwrite the original data set with the values currently in *newdataset*. You can also drop the counters during this step.

```
*Step 5;
DATA mydata; SET newdataset (DROP=ctr newCtr); RUN;
```

UTILIZING A MACRO

If you plan use this method multiple times in one program, utilizing a macro can save you time by eliminating the need to type the same code multiple times. You can put Step 4's code in an easy-to-call macro by placing the following code near the beginning of your program:

```
%MACRO QC_Check;
    PROC SORT DATA=newdataset; BY ctr; RUN; *maintain original sort order;
    DATA newdataset; SET newdataset; newCtr=_n_; RUN; *create new counter;

    TITLE2 'Comparing Number of Records';
    PROC MEANS DATA=safecopy MAX; VAR ctr; TITLE3 'Original Data Set'; RUN;
    PROC MEANS DATA=newdataset MAX; VAR newCtr; TITLE3 'New Data Set'; RUN;
    TITLE2 'Comparison of Original and New Data Set';
    PROC COMPARE DATA=safecopy compare=newdataset LISTVAR;
        ID ctr;
    RUN;
%MEND;
```

Once you create this macro, you can use this line of code to perform Step 4:

```
%QC_Check;
```

I recommend putting Step 4 in a macro because it does not change no matter what you are doing with the data, while the code for Steps 1-3 and 5 could differ depending on the original data set's name and the desired changes.

CONCLUSION

Using this Five-Step quality control method will enable programmers to systematically check modifications made to an existent data set. It is a tool to help assure cleaner data.

ACKNOWLEDGMENTS

The author thanks the SESUG chair and staff for this conference and the opportunity to present. He also thanks Dr. Imelda C. Go for introducing him to SAS programming.

CONTACT INFORMATION

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

Name: Aaron R. Brown
Enterprise: South Carolina Department of Education
Address: 1429 Senate Street
City, State ZIP: Columbia, SC 29201
Work Phone: 803-734-8858
E-mail: ARBrown@ed.sc.gov

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.