

Checks and Balances Using A Macro

Liza Thompson, GoodCents, Grayson, GA

ABSTRACT

When dealing with new data, there is always the chance for a few incorrect observations. When dealing with very large datasets, finding a few bad observations may seem a bit impossible. One may not even realize that bad observations are lurking in the dataset. GoodCents created a foolproof way to identify these bad observations and display them to the user in a way that they simply cannot ignore. Using a macro statement, a do loop, the call symput statement, and a window statement, SAS® creates a pop-up window that alerts the user of a bad record. The user has to press the Enter key on the keyboard to continue with the program or move on to the next bad record. The code can easily be changed to hold any test value, print any message, and print any portion of the record.

To find bad records, the user could simply write a new data step with the same test value in an if-statement and print the bad records to the output window. However, if additional output is printed to the output window, the bad records can easily get lost or overlooked. The pop-up window acts as a great checks-and-balances system for the SAS® user.

This simple test code can be included in previously written programs, or run by itself. The code can easily be modified to fit any user's application. The code will be presented in a way that will allow each SAS® user to create a checks-and-balances system of their own.

INTRODUCTION

Data formatting and data cleaning is the first tedious step of statistical analysis. How many times have you compiled your entire dataset, completed your entire analysis, and even started on your reporting before you realized you had a bad record in your data that is throwing off your entire analyzed results? The SAS® programmers at GoodCents do this – and do it often enough to develop the code presented in the following paper.

The GoodCents Consulting team monitors energy usage for hundreds of customers across many various utilities and regions. The equipment used to collect the data and the software used to download the energy usage data is not perfect. Often times if there is a power outage or bad connection, a bad record will be added to the usage data. The energy usage data is collected in fifteen-minute intervals, yielding thousands of records a day. Many times these bad records are not caught until the data is being analyzed and reported. The code presented in the following discussion is an easy application that can be added to any existing program, or can stand-alone as an error test, or acts as a checks and balances system.

The checks and balances program not only isolates the bad records found within the data set, but also creates a pop-up window that requires the user to scroll through each bad record. The pop-up window reminds the user of the bad records rather than allowing them to get mixed in with the high-quality output data in the output window. For this particular application, there usually are not more than a few bad records, allowing the task of scrolling through each record to be an effortless chore. For data sets with many bad records, the code can be altered so multiple records are printed at once and the user does not have to make the tedious effort to scroll through each observation.

CHECKS AND BALANCES – THE CODE

The checks and balances code can be used for many different applications. The GoodCents application of the checks the balances code will be discussed in the following paragraphs, as well as ways and places to alter the code for other uses. There are five major steps to creating the checks and balances macro code, creating the test variable, formatting the data, determining the number of bad records in the data, creating the macro, and creating the pop up window. These five steps will be discussed in detail and individually.

CREATING THE TEST VARIABLE

To begin the program, one must simply use a SAS® data step to define the data that is undergoing testing and define the value or boundary at which the data will be tested. For this application, the variable that is being tested is the customer's premise data, or whole house energy usage. Since the data is in either fifteen-minute or hourly intervals, the customer should never have a whole house electricity usage value over 20 kilowatt-hours (kWh). Therefore, a simple if-statement within the data step is used to keep only the records with flawed premise values, or premise values greater than 20 kWh. Using this SAS® code, the remaining bad data can easily be printed to the output window and recorded very easily without the use of a macro. However, often times during analysis there is a large quantity of data being printed to the output window. The bad records can easily get lost amongst the routinely printed output. The data step and creation of the test variable is shown on the following page.

```

data testerror;
set goodcents.kWHHourlyFile;
run;

data testerror2;
set testerror;
if premise>20;
run;

```

The first data step shown above can read any file of any size. The second data step above defines the test variable, the value limit, and stores only the bad records in that second dataset.

FORMATTING THE DATA

GoodCents analyzes data on a customer-by-customer basis. If one customer has multiple bad records, this customer can be treated as one bad record, rather than multiple bad records. The PROC SORT and IF FIRST. statements in the second data step format the data so that only the first bad record for each person is kept. This step can be applied to any data variable or additional code can be added here to format the data in any way the user prefers. Formatting the data in this step will prevent the user from having to scroll through numerous error windows.

This step is shown below.

```

proc sort data=testerror2;
by sitename;
run;

data testerror3 (keep=sitename);
set testerror2;
by sitename;
if first.sitename;

```

The KEEP statement used in the data step specifies which variables to keep in the new dataset that is generated. In this case, the variable "sitename", an observation id, is the only variable kept in the dataset. Other users may choose to keep the observation id and the test value.

DETERMINING THE TOTAL NUMBER OF BAD RECORDS

The next step in the program is used to determine the total number of bad records found within the data. A new variable, "total", is created by using the SAS® statement CEIL and the SAS® variable _n_. The SAS® variable _n_ is an automatic SAS® variable that indicates the number of times the data step is iterated. In this code, the data step is only iterated and reiterated when a bad record is found. Therefore, the number of iterations in the data step equals the total number of bad records. The CEIL function returns the smallest integer greater than or equal to the argument, in this case _n_, or the total number of bad records. This code is shown below.

```

data testerror4;
set testerror3;
total=ceil(_n_);
call symput ('number',total);
run;

```

The CALL SYMPUT statement is one of the tools in the SAS® language that allows for communication with macros. The CALL SYMPUT statement is used to assign the value of the variable total produced in the data step to the macro as a macro variable called number.

CREATING THE MACRO

Finally, it is time to create the macro. The macro must be defined at the beginning of the program before any of the previously mentioned code is written. For this application, we name the macro **testerror**. The code below is referred to as a call to the macro.

```
%macro testerror;
```

The %macro is standard SAS® code that defines the macro language. This call to the macro can be included in pre-existing code, or can be the start to a new program.

At the very end of this program, one needs to end and mend the macro, shown in the code below.

```

%end;
run;

%mend;
%testerror;

```

The end statement ends all previous macro statements. The mend statement closes the macro so other code or macros can be included in the program.

Since the macro was called at the beginning of the program, the variable number that was created in the testerror4 data set will work in a macro do loop. For this application, the SAS® user wanted to scroll through the pop-up window once for each customer that recorded bad data. Therefore a macro do loop was created to tell the pop-up window to display as many times as needed to show all observation ids with bad records, and print the necessary information in the window. The do loop starts with k=1 and repeats for the value of the macro variable number. This is shown below.

```
%do k=1 %to &number;

data testerror5 (keep= sitename );
set testerror4;
if _n_=&k;
call symput ('site',sitename);
%put site;
```

The variable _n_ is the total number of bad data points and &k is the macro variable for the do loop. Therefore, the do loop will continue to iterate until the macro variable &k reaches the total number of bad data points. The CALL SYMPUT statement creates a macro variable named “site” from the observation id variable named “sitename”. The macro statement %put names the variable whose value is to be written in the pop-up window.

THE POP-UP WINDOW

Within the macro do loop, a SAS® data statement define the pop-up window and instructs SAS® what to write in the pop-up window messages. The last series of data set statements, shown below, includes a macro statement %window, a window title, error, and the color definition – all of which define the pop-up window.

```
data testerror6;
set testerror5;
%window Error color=white
      #2 @2 "Error Loads Greater Than 20 for &site "
      #4 @2 'Press Enter to Continue.' ;
%display Error;
```

The %window statement tells SAS® to open a new window, name Error. The color defines the color of the window background as white. There are two statements following the window macro that specifies the exact formatting and dialog to be displayed in the window before the actual test error value, or the customer's site identification is written. The number followed by the symbol # tells SAS® which line within the window to start writing the text. The number followed by the @ symbol specifies what column to start the text, or indentation to use within the window. All text should be enclosed in quotation marks. A semicolon should only be used once all desired text is specified. An additional macro statement, % display error, notifies SAS® that the user actually wants the window displayed when an error is found.

LOG ERROR MESSAGES

If the test does not find any bad data, an error message will appear in the SAS® log. This is simply because the program is set up primarily through a series of data sets. The macro do loop will be unable to run because the replication coefficient will be equal to zero. This log error message is expected and can be considered as the notification that the data is ready for analysis, free of errors

CONCLUSION

A simple macro, the CALL SYMPUT statement, and a series of data steps can create a fool proof testing method for large datasets. This program can be applied to most data processing applications. In addition, this program can stand alone or be included in pre-existing programs. The pop-up window is a great tool in SAS® and can really help in the tedious task of data clean up.

ACKNOWLEDGMENTS

A special thank you is in order for Bill Bland, Vice President of Consulting Services with GoodCents. Bill Bland is the author of this original code.

CONTACT INFORMATION

Contact the author at:

Liza Thompson
GoodCents
1075 Cooper Road, Suite 100
Grayson, GA 30017
Work Phone: 678-836-1059
Fax: 678-344-7184
E-mail: Liza.Thompson@GoodCents.com
Web: www.GoodCents.com

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.

APPENDIX 1: SAS® CODE

```
%macro testerror;
data testerror;
set goodcents.kWHHourlyFile;
run;

data testerror2;
set testerror;
if premise>20;
run;

proc sort;
by sitename;
run;

data testerror3 (keep=sitename);
set testerror2;
by sitename;
if first.sitename;

data testerror4;
set testerror3;
total=ceil(_n_);
call symput ('number',total);
run;

%do k=1 %to &number;

data testerror5 (keep= sitename );
set testerror4;
if _n_=&k;
call symput ('site',sitename);
%put site;

data testerror6;
set testerror5;
%window Error color=white
      #2 @2 "Error Loads Greater Than 20 for &site "
      #4 @2 'Press Enter to Continue.' ;
%display Error;

%end;
run;

%mend;
%testerror;
```